TIR

The IPSI BgD Transactions on Internet Research

Multi-, Inter-, and Trans-disciplinary Issues in Computer Science and Engineering

A publication of IPSI Bgd Internet Research Society, New York, Frankfurt, Tokyo, Belgrade
July 2018 Volume 14 Number 2 (ISSN 1820-4503)

Special issue: „Recent Advances in DRAM and Flash Memory Architectures“
Guest Editors: Onur Mutlu, Saugata Ghose, and Rachata Ausavarungnirun

Table of Contents:

The IPSI BgD Internet Research Society

IPSI BGD TRANSACTIONS ON INTERNET RESEARCH
www.internetjournals.net

Guest Editor Introduction: Recent Advances in DRAM and Flash Memory Architectures

Onur Mutlu[1,2] Saugata Ghose[2] Rachata Ausavarungnirun[2]

[1]*ETH Zürich* [2]*Carnegie Mellon University*

Memory and storage systems are a fundamental system performance, energy, and reliability bottleneck in modern systems [1, 2, 3, 34, 35, 36]. This bottleneck is becoming increasingly severe due to (1) the very limited latency reductions in memory and storage devices over the last several years; (2) aggressive manufacturing process technology scaling and other techniques to improve memory density, such as multi-level cell technology, which increase the storage capacity of these devices, but introduce more raw bit errors and increase manufacturing process variation; (3) limited pin counts in chip packages, which prevent system designers from adding more and/or wider buses to increase bandwidth; (4) overwhelmingly data-intensive applications, which require high-bandwidth access to very large amounts of data; and (5) the increasing fraction of overall system energy consumed by memory systems and data movement. To make matters worse, it is becoming increasingly difficult to continue scaling these devices to smaller process technology nodes, and even though alternative emerging memory and storage technologies can potentially alleviate some of the shortcomings of existing memory and storage technologies, they also introduce *new* shortcomings that were previously absent. Therefore, there is a pressing need to comprehensively understand and mitigate these bottlenecks in both existing and emerging memory and storage systems and technologies.

This issue features extended summaries and retrospectives of some of the recent research done by our group, SAFARI [41, 43], on (1) understanding, characterizing, and modeling various critical properties of modern DRAM and NAND flash memory, the dominant memory and storage technologies, respectively; and (2) several new mechanisms we have proposed based on our observations from these analyses, characterization, and modeling, to tackle various key challenges in memory and storage scaling. In order to understand the sources of various bottlenecks of the dominant memory and storage technologies, these works perform rigorous studies of device-level and application-level behavior, using a combination of detailed simulation and experimental characterization of *real* memory and storage devices.

The works that perform real device characterization make use of custom FPGA-based platforms that we build to provide us with fine-grained control over the devices. We devise specific tests that perform a controlled measurement of each phenomenon that we aim to explore. Our experimental characterizations have often discovered many unexpected types of behavior in real state-of-the-art devices, and have inspired the research community to pursue further investigations (e.g., on the RowHammer phenomenon [21, 34], DRAM retention behavior [20, 30, 39], NAND flash memory error patterns [1, 2, 3, 5, 6, 7, 8, 9, 11]). In order to aid future research, we have released much of our experimental characterization data online [43, 45], and have open-sourced our DRAM characterization platform, SoftMC [19, 44].

The works that perform application and architectural analyses rely on real system characterizations and simulation to develop a rigorous understanding of the bottlenecks and to provide solutions. Our analyses have shown key scaling bottlenecks, proposed new solutions, and have inspired the research community to develop further investigations (e.g., on DRAM refresh [12, 30, 31], DRAM latency reduction [28, 29], the RowHammer phenomenon [21, 34], and in-memory data movement and computation [16, 47, 49, 50]). In order to aid future research, we have released our flexible and extensible memory system simulator, Ramulator, as open-source software [22, 42].

In each work that is featured in this issue, based on our observations and analyses from our experimental studies of real systems and applications as well as future trends and problems, we propose novel solutions that overcome many of the scaling bottlenecks that memory and storage systems face. For each of the works presented in this special issue, its corresponding article examines the work's significance in the context of modern computer systems, and discusses several new research questions and directions that each work motivates.

We start with five of our works that explore new opportunities in DRAM systems to reduce latency and/or energy consumption. As we mentioned earlier, the latency and energy consumption of DRAM have not reduced significantly in the last several years. We find that by introducing heterogeneity into DRAM architectures, or by taking advantage of the existing variation within and across DRAM modules, we can develop new mechanisms that improve DRAM access latency and/or energy efficiency.

The first paper in the issue describes Tiered-Latency DRAM (TL-DRAM), which originally appeared in HPCA 2013 [29]. This work (1) proposes a new DRAM architecture that can provide us with the performance benefits of costly reduced-latency DRAM products in a cost-effective manner, by isolating a small portion of a DRAM array so that it can behave as a low-latency DRAM buffer; and (2) exploits the low-latency in-DRAM buffer using various hardware or software mechanisms to improve overall system performance.

The second paper in the issue describes Adaptive-Latency DRAM (AL-DRAM), which originally appeared in HPCA 2015 [28]. This work experimentally characterizes (1) the large latency variation across DRAM modules and (2) the large timing margins designed to account for worst-case variation and operating conditions. Based on the findings from the characterization, the work proposes a new mechanism that can identify and safely reduce the timing margin to speed up DRAM accesses, and thus improve overall system performance and energy consumption.

The third paper in the issue describes Flexible-Latency DRAM (FLY-DRAM), which originally appeared in SIGMETRICS 2016 [15]. This work experimentally characterizes the latency variation that exists *within* each DRAM module, showing that there are regions of fast cells and regions of slow cells that exist in real DRAM modules. Based on these findings, the work proposes a new mechanism that identifies regions of fast cells and reduces the latency of DRAM operations to these regions.

The fourth paper in the issue describes Voltron, which originally appeared in SIGMETRICS 2017 [13]. This work experimentally characterizes the relationship between DRAM latency, reliability, and supply voltage, showing that these three can be traded off intelligently for various purposes. The work proposes a new mechanism that uses this relationship to dynamically reduce DRAM energy consumption within a bounded performance loss target.

The fifth paper in the issue describes SoftMC, which originally appeared in HPCA 2017 [19]. This work describes our open-source DRAM characterization infrastructure, and demonstrates its versatility for use in a wide range of DRAM research topics. SoftMC is a result of 6+ years of effort, which led to at least 11 works at top conferences, and we hope it will enable other researchers to explore the detailed behavior of existing and emerging memory architectures and develop new mechanisms and memory architectures.

Next, we look at a couple of our works that reduce data movement between the CPU and DRAM, as this movement consumes (1) a large fraction of DRAM energy and (2) much of the limited available DRAM bandwidth. We find that a large portion of DRAM bandwidth is consumed by the movement of data between DRAM and the CPU to perform simple operations such as data copy and initialization. We can instead take advantage of the underlying DRAM architecture to efficiently perform these simple operations directly within DRAM, eliminating the need to move the data to/from the CPU.

The sixth paper in the issue describes RowClone, which originally appeared in MICRO 2013 [50]. Many applications perform data copy and initialization operations, requiring only simple computation, but these operations require expensive data movement between the CPU and DRAM. This work proposes a new DRAM architecture that can internally perform bulk data copy and initialization operations at very low hardware cost, avoiding the costly data movement, and shows that doing so provides 1–2 orders of magnitude speedup and energy reduction for such operations.

The seventh paper in the issue describes low-cost interlinked subarrays (LISA), which originally appeared in HPCA 2016 [16]. This work (1) builds a general substrate that facilitates the bulk movement of data between two different rows in memory by improving the interconnectivity of DRAM arrays, and (2) demonstrates that LISA can be used to efficiently implement a number of mechanisms, such as bulk data copy/initialization, latency reduction, and fast in-DRAM caching. Each of these mechanisms provides significant performance and energy improvements.

Finally, we investigate the reliability of NAND flash memory. As NAND flash memory based solid-state drives (SSDs) are now widely-used in a large variety of modern systems (e.g., data centers [33,38,46], smartphones), there is continued demand to increase the density of SSDs while lowering the cost per bit. While manufacturers have employed several methods (e.g., aggressive manufacturing process technology scaling and multi-level cell technology), these methods have exacerbated a number of sources of raw bit errors. Due to limitations to the number of errors that can be corrected by error-correcting codes (ECC), SSDs have a limited lifetime, after which manufacturers cannot reliably retain data for a minimum guaranteed time without data loss [1, 2, 3]. Over the last decade, as a result of aggressive density scaling, the typical lifetime of an SSD has dropped by 1–2 orders of magnitude, and the various sources of raw bit errors now pose a key scaling challenge for storage [1,2,3]. As a sampling of our 7+ years of research into NAND flash memory reliability, we feature three papers that design mechanisms to significantly mitigate reliability issues and extend the limited lifetime of NAND flash memory based devices.

The eighth paper in the issue describes a new data retention study in NAND flash memory, which originally appeared in HPCA 2015 [7]. This work experimentally characterizes the susceptibility of state-of-the-art NAND flash memory to data retention errors using our FPGA-based flash memory testing infrastructure [1, 2, 3, 5], and proposes (1) a new mechanism that mitigates the impact of retention errors at runtime, which increases the lifetime of the SSD; and (2) a new mechanism that exploits retention behavior to recover data in the event of data loss, thereby improving SSD robustness.

The ninth paper in the issue describes a new read disturb study in NAND flash memory, which originally appeared in DSN 2015 [6]. This work experimentally characterizes read disturb errors in NAND flash memory, where a read operation introduces errors in unread parts of the memory. Based on the characterization, the work proposes (1) a new mechanism that mitigates read disturb errors, thereby improving the SSD lifetime; and (2) a new mechanism that exploits read disturb behavior to recover data in the event of data loss, thereby improving SSD robustness.

The last paper in the issue describes a new study on two-step programming in NAND flash memory, which originally appeared in HPCA 2017 [4]. This work demonstrates that the programming algorithm used in many state-of-the-art NAND flash memory devices can introduce previously-unknown data vulnerabilities, which can be exploited by malicious applications to perform security attacks. The work proposes three mechanisms to eliminate or mitigate these vulnerabilities, thereby improving both reliability and security.

Throughout all of these works, we find that by understanding and taking advantage of the behavior and architecture of memory and storage devices and appropriately modifying them at low cost and low overhead, we can successfully mitigate many of the scalability challenges in memory and storage devices. Even though the works presented are described in the context of DRAM and NAND flash memory, the two dominant memory and storage technologies of today, we believe many of the basic ideas and concepts can be applied or adapted to emerging memory technologies [32], e.g., phase-change memory [24, 25, 26, 40, 53, 54, 55], STT-MRAM [18, 23, 37], and memristors/RRAM [17, 51, 52]. We hope that the works featured in this special issue inspire readers to explore the presented challenges, and to develop new solutions that can enable high-performance, low-energy, low-latency, high-reliability memory and storage systems, and thus the computing systems, of the future.

Acknowledgments

The works featured in this issue, along with our related works that we reference in each featured work, are a result of the research done together with many students and collaborators over the course of the past 10+ years, whose contributions we acknowledge. In particular, we acknowledge and appreciate the dedicated effort of current and former students and postdocs in our research group, SAFARI [41, 43], who contributed to the ten featured works, including Yu Cai, Kevin Chang, Chris Fallin, Hasan Hassan, Kevin Hsieh, Ben Jaiyen, Abhijith Kashyap, Samira Khan, Yoongu Kim, Tianshi Li, Jamie Liu, Donghyuk Lee, Yixin Luo, Justin Meza, Gennady Pekhimenko, Vivek Seshadri, Lavanya Subramanian, Nandita Vijaykumar, and Abdullah Giray Yağlıkçı.

Aside from our featured works and other referenced papers from our group, where a wealth of information on modern memory and storage systems can be found, at least four Ph.D. dissertations have shaped the works that we feature in this special issue:

- Yu Cai's thesis entitled "NAND Flash Memory: Characterization, Analysis, Modeling and Mechanisms" [10],
- Donghyuk Lee's thesis entitled "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity" [27],
- Vivek Seshadri's thesis entitled "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Subsystems" [48], and
- Kevin Chang's thesis entitled "Understanding and Improving the Latency of DRAM-Based Memory Systems" [14].

We also acknowledge various funding agencies (the National Science Foundation, the Semiconductor Research Corporation, the Intel Science and Technology Center on Cloud Computing, CyLab, the CMU Data Storage Systems Center, and the NIH) and industrial partners (AMD, Facebook, Google, HP Labs, Huawei, IBM, Intel, Microsoft, NVIDIA, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware), and ETH Zürich, who have supported the featured works in this issue and other related work in our research group generously over the years.

References

[1] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, 2017.

[2] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.

[3] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.

[4] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[5] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[6] Y. Cai, Y. Luo, S. Ghose, E. F. Haratsch, K. Mai, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.

[7] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.

[8] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[9] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Flash Correct and Refresh: Retention Aware Management for Increased Lifetime," in *ICCD*, 2012.

[10] Y. Cai, "NAND Flash Memory: Characterization, Analysis, Modelling, and Mechanisms," Ph.D. dissertation, Carnegie Mellon Univ., 2012.

[11] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.

[12] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.

[13] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[14] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2017.

[15] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[16] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[17] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, 1971.

[18] X. Guo, E. Ipek, and T. Soyata, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *ISCA*, 2010.

[19] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.

[20] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[21] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[22] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[23] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.
[24] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[25] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, 2010.
[26] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, 2010.
[27] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon Univ., 2016.
[28] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[29] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[30] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[31] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[32] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[33] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A Large-Scale Study of Flash Memory Errors in the Field," in *SIGMETRICS*, 2015.
[34] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[35] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[36] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[37] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[38] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD Failures in Datacenters: What? When? and Why?" in *SYSTOR*, 2016.
[39] M. Qureshi, D. H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[40] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *ISCA*, 2009.
[41] SAFARI Research Group, http://www.ece.cmu.edu/~safari/.
[42] SAFARI Research Group, "Ramulator – GitHub Repository," https://github.com/CMU-SAFARI/ramulator.
[43] SAFARI Research Group, "SAFARI Software Tools – GitHub Repository," https://github.com/CMU-SAFARI/.
[44] SAFARI Research Group, "SoftMC – GitHub Repository," https://github.com/CMU-SAFARI/SoftMC.
[45] SAFARI Research Group, "Tools, Software, and Full Data Sets," http://www.ece.cmu.edu/~safari/tools.html.
[46] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash Reliability in Production: The Expected and the Unexpected," in *FAST*, 2016.
[47] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[48] V. Seshadri, "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2016.
[49] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
[50] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[51] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, 2008.
[52] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.
[53] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.
[54] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," *TACO*, 2014.
[55] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.

Tiered-Latency DRAM: Enabling Low-Latency Main Memory at Low Cost

Donghyuk Lee[1,2] Yoongu Kim[2] Vivek Seshadri[3,2]
Jamie Liu[4,2] Lavanya Subramanian[5,2] Onur Mutlu[6,2]

[1]*NVIDIA Research* [2]*Carnegie Mellon University*
[3]*Microsoft Research India* [4]*Google* [5]*Intel Labs* [6]*ETH Zürich*

This paper summarizes the idea of Tiered-Latency DRAM (TL-DRAM), which was published in HPCA 2013 [73], and examines the work's significance and future potential. The capacity and cost-per-bit of DRAM have historically scaled to satisfy the needs of increasingly large and complex computer systems. However, DRAM latency has remained almost constant, making memory latency the performance bottleneck in today's systems. We observe that the high access latency is not intrinsic to DRAM, but a trade-off is made to decrease the cost per bit. To mitigate the high area overhead of DRAM sensing structures, commodity DRAMs connect many DRAM cells to each sense amplifier through a wire called a bitline. These bitlines have a high parasitic capacitance due to their long length, and this bitline capacitance is the dominant source of DRAM latency. Specialized low-latency DRAMs use shorter bitlines with fewer cells, but have a higher cost-per-bit due to greater sense amplifier area overhead.

To achieve both low latency and low cost per bit, we introduce Tiered-Latency DRAM *(TL-DRAM). In TL-DRAM, each long bitline is split into two shorter segments by an isolation transistor, allowing one of the two segments to be accessed with the latency of a short-bitline DRAM without incurring a high cost per bit. We propose mechanisms that use the low-latency segment as a hardware-managed or software-managed cache. Our evaluations show that our proposed mechanisms improve both performance and energy efficiency for both single-core and multiprogrammed workloads.*

Tiered-Latency DRAM has inspired several other works on reducing DRAM latency with little to no architectural modification [20, 21, 22, 24, 37, 38, 68, 72, 116, 117, 118].

1. Problem: High DRAM Latency

Primarily due to its low cost per bit, DRAM has long been the substrate of choice for architecting main memory subsystems. In fact, DRAM's cost per bit has been decreasing at a rapid rate as DRAM process technology scales to integrate ever more DRAM cells into the same die area. As a result, each successive generation of DRAM has enabled increasingly larger-capacity main memory subsystems at low cost.

In stark contrast to the continued scaling of cost per bit, the *latency* of DRAM has remained almost constant. During the same 11-year interval in which DRAM's cost per bit decreased by a factor of 16, DRAM latency (as measured by the t_{RCD} and t_{RC} timing constraints)[1] decreased by only 30.5% and 26.3% [6, 47], respectively, as shown in Figure 1. From the perspective of the processor, an access to DRAM takes hundreds of cycles – time during which the processor may be stalled, waiting for DRAM [3, 34, 48, 92, 93, 96]. This wasted time due to stalling on DRAM leads to large performance degradation.

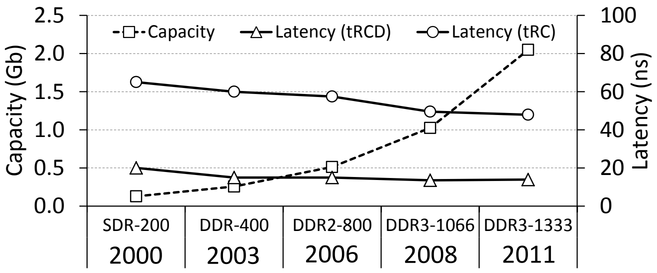


Figure 1: Change in DRAM capacity and latency over time [6, 47, 100, 111]. Reproduced from [73].

2. Key Observations and Our Goal

Bitline: Dominant Source of Latency. In DRAM, each bit is represented as electrical charge in a capacitor-based *cell.* The small size of this capacitor necessitates the use of an auxiliary structure, called a *sense amplifier*, to (1) detect the small amount of charge held by the cell and (2) amplify it to a full digital logic value. A sense amplifier is approximately one hundred times larger than a cell [107]. To amortize their large size, each sense amplifier is connected to many DRAM cells through a wire called a *bitline.*[2]

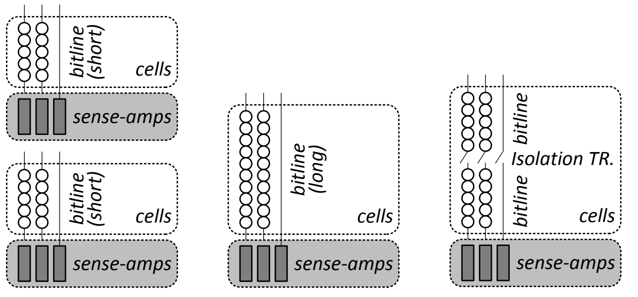
Every bitline has an associated *parasitic capacitance*, whose value is proportional to the length of the bitline. Unfortunately, the parasitic capacitance slows down DRAM operation for two reasons. First, it increases the latency of the sense amplifiers. When the parasitic capacitance is large, a cell cannot quickly create a voltage perturbation on the bitline that can be easily detected by the sense amplifier. Second, the capacitance increases the latency of charging and precharging the bitlines. Although the cell and the bitline must be restored to their

[1]The overall DRAM latency can be decomposed into individual DRAM *timing constraints.* Two of the most important timing constraints are t_{RCD} (row-to-column delay) and t_{RC} (row-cycle time).

[2]We refer the reader to our prior works for a detailed background on DRAM architecture and operation [21, 22, 23, 24, 37, 38, 54, 56, 57, 58, 59, 60, 68, 69, 71, 72, 73, 75, 76, 99, 103, 116, 117].

quiescent voltages during and after an access to a cell, such a procedure takes much longer when the parasitic capacitance of the bitline is large. Due to these two reasons, and based on a detailed latency breakdown discussed in Section 3.1 of our HPCA 2013 paper [73], we conclude that long bitlines are the dominant source of DRAM latency [44, 72, 73, 90, 91, 122].

Latency vs. Cost Trade-Off. The bitline length is a key design parameter that exposes the important trade-off between latency and die size (cost). Short bitlines (i.e., a bitline connected to only a few cells) constitute a small electrical load (parasitic capacitance), which leads to low latency. However, they require more sense amplifiers for a given DRAM capacity (Figure 2a), which leads to a large die size. In contrast, long bitlines have high latency and a small die size (Figure 2b). As a result, neither of these two approaches can optimize for *both* latency and cost per bit.



(a) Latency-optimized architecture (b) Cost-optimized architecture (c) Our proposed architecture

Figure 2: DRAM latency and cost optimization, and our proposal (TL-DRAM). Reproduced from [73].

Figure 3 shows the trade-off between DRAM latency and die size by plotting the latency (t_{RCD} and t_{RC}) and the die size for different values of cells per bitline. Existing DRAM architectures are either (1) optimized for die size (e.g., commodity DDR3 [86, 111]) and are thus low cost but high latency; or (2) optimized for latency (e.g., RLDRAM [85], FCRAM [112]) and are thus low latency but (very) high cost.

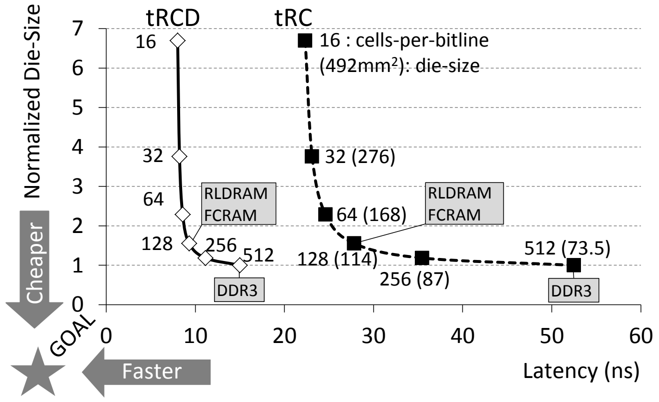
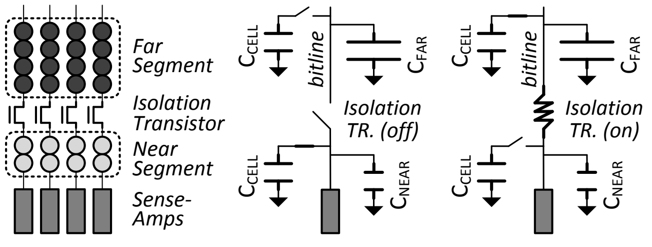


Figure 3: Bitline length: latency vs. die size. Reproduced from [73].

The goal of our HPCA 2013 paper [73] is to design a new DRAM architecture to approximate the best of both worlds (i.e., low latency *and* low cost), based on our key observation that long bitlines are the dominant source of DRAM latency.

3. Tiered-Latency DRAM

To achieve the latency advantage of short bitlines *and* the cost advantage of long bitlines, we propose the *Tiered-Latency DRAM* (TL-DRAM) architecture, which is shown in Figures 2c and 4a. The key idea of TL-DRAM is to divide the long bitline into two shorter segments using an *isolation transistor*: the *near segment* (connected directly to the sense amplifier) and the *far segment* (connected through the isolation transistor).



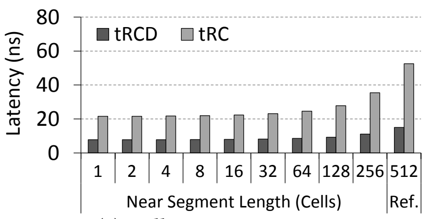
(a) Organization (b) Near segment (c) Far segment

Figure 4: TL-DRAM: accessing the near segment and the far segment. Adapted from [73].

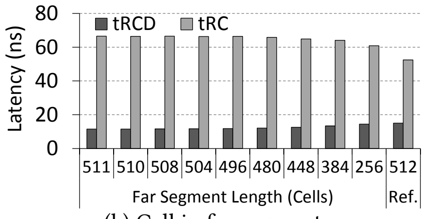
The primary role of the isolation transistor is to electrically decouple the two segments from each other. This changes the effective bitline length (and also the effective bitline capacitance) as seen by the cell and sense amplifier. Correspondingly, the latency to access a cell also changes, albeit differently depending on whether the cell is in the near or the far segment.

When accessing a cell in the near segment, the isolation transistor is turned off, disconnecting the far segment (Figure 4b). Since the cell and the sense amplifier see only the reduced bitline capacitance of the shortened near segment, they can drive the bitline voltage more easily. As a result, the bitline voltage is restored more quickly, and, thus, the latency (t_{RC}) for the near segment is significantly reduced. On the other hand, when accessing a cell in the far segment, the isolation transistor is turned on to connect the entire length of the bitline to the sense amplifier. In this case, the isolation transistor acts like a resistor inserted between the two segments (Figure 4c) and limits how quickly charge flows to the far segment. Because the far segment capacitance is charged more slowly, it takes longer for the far segment voltage to be restored, and, thus, the latency (t_{RC}) is increased for cells in the far segment.

Sensitivity to Segment Length. The lengths of the two segments are determined by where the isolation transistor is placed on the bitline. Assuming that the number of cells per bitline is fixed at 512 cells, the near segment length can range from as short as a single cell to as long as 511 cells. We perform circuit-level simulations to determine how the latency of each segment based on the number of cell in the

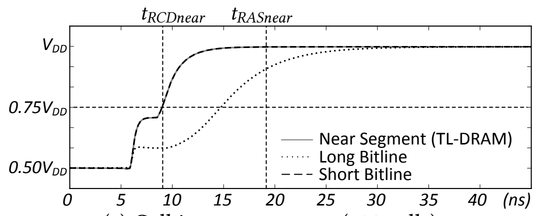


(a) Cell in near segment

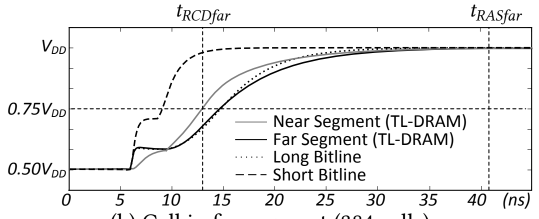


(b) Cell in far segment

Figure 5: Latency analysis. Reproduced from [73].

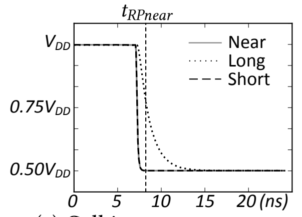


(a) Cell in near segment (128 cells)

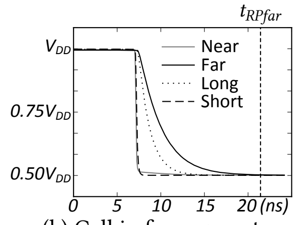


(b) Cell in far segment (384 cells)

Figure 6: Activation: bitline voltage. Reproduced from [73].



(a) Cell in near segment



(b) Cell in far segment

Figure 7: Precharging. Reproduced from [73].

segment. Figures 5a and 5b plot the latencies of the near and far segments as a function of their length, respectively. For reference, the rightmost bars in each figure are the latencies of an unsegmented long bitline whose length is 512 cells. From these figures, we draw three conclusions. First, the shorter the near segment, the lower its latencies (t_{RCD} and t_{RC}). This is expected since a shorter near segment has a lower effective bitline capacitance, allowing it to be driven to target voltages more quickly. Second, the longer the far segment, the lower the far segment's t_{RCD}. Recall from our previous discussion that the far segment's t_{RCD} depends on how quickly the *near segment* (not the far segment) can be driven. A longer far segment implies a shorter near segment (lower capacitance), which is why t_{RCD} decreases for the far segment. Third, the shorter the far segment, the smaller its t_{RC}. The far segment's t_{RC} is determined by how quickly it reaches the full voltage (V_{DD} or *0*). Regardless of the length of the far segment or the near segment, the current that trickles into it through the isolation transistor does not change significantly. Therefore, a shorter far segment (lower capacitance) reaches the full voltage more quickly.

Latency Analysis (Circuit Evaluation). We model TL-DRAM in detail using SPICE simulations. Simulation parameters are mostly derived from a publicly available 55nm DDR3 2Gb process technology file [107] which includes information such as cell and bitline capacitances and resistances, physical floorplanning, and transistor dimensions. Transistor device characteristics were derived from [98] and scaled to agree with [107]. Figures 6 and 7 show the bitline voltages during activation and precharging, respectively. The x-axis origin (time 0) in the two figures corresponds to when the subarray receives the ACTIVATE or PRECHARGE command, respectively. In addition to the voltages of the segmented bitline (near and far segments), the figures also show the voltages of two unsegmented bitlines (short and long) for reference.

First, during an access to a cell in the near segment (Figure 6a), the far segment is disconnected and is floating (hence its voltage is not shown). The bitline starts at 1/2 V_{DD}. Due to the reduced bitline capacitance of the near segment, its voltage increases almost as quickly as the voltage of a short bitline (the two curves are overlapped) during *sensing & amplification.* Since the near segment voltage reaches $0.75V_{DD}$ and V_{DD} (the *threshold* and *restored* states, respectively) quickly, its t_{RCD} and t_{RAS}, respectively, are significantly reduced compared to a long bitline. Second, during an access to a cell in the far segment (Figure 6b), we can indeed verify that the voltages of the near and the far segments increase at different rates due to the resistance of the isolation transistor, as previously explained. Compared to a long bitline, while the near segment voltage reaches $0.75V_{DD}$ more quickly, the far segment voltage reaches V_{DD} more slowly. As a result, t_{RCD} for the far segment is reduced while its t_{RAS} is increased.

While precharging the bitline after accessing a cell in the near segment (Figure 7a), the near segment reaches $0.5V_{DD}$ quickly due to the smaller capacitance, almost as quickly as the short bitline (the two curves are overlapped). On the other hand, precharging the bitline after accessing a cell in the far segment (Figure 7b) takes longer compared to the long-bitline baseline. As a result, t_{RP} is reduced for the near segment and increased for the far segment.

Summary (Latency, Power, and Die-Area). Table 1 summarizes the latency, power, and die area characteristics of TL-DRAM compared to short-bitline and long-bitline DRAMs, estimated using circuit-level SPICE simulation [98] and power/area models from Rambus [107]. Compared to commodity DRAM (long bitlines), which incurs high latency (t_{RC}) for all cells, TL-DRAM offers significantly reduced latency (t_{RC}) for cells in the near segment, while increasing the latency for cells in the far segment due to the additional resistance of the isolation transistor. In DRAM, a large fraction of the power is consumed by the bitlines. Since the near segment in TL-DRAM has a lower capacitance, it also consumes less power. On the other hand, accessing the far segment requires toggling the isolation transistors, leading to increased power consumption. Mainly due to additional isolation transistors, TL-DRAM increases die area by 3% compared to commodity DRAM. Section 4 of our HPCA 2013 paper [73] includes detailed circuit-level analyses of TL-DRAM, along with detailed area, latency, and power estimations.

	Short Bitline (Figure 2a)	Long Bitline (Figure 2b)	Segmented Bitline (Figure 2c)	
	Unsegmented	Unsegmented	Near	Far
Length (Cells)	32	512	32	480
Latency (t_{RC})	**Low** (23.1ns)	High (52.5ns)	**Low** (23.1ns)	Higher (65.8ns)
Normalized Power	**Low** (0.51)	High (1.00)	**Low** (0.51)	Higher (1.49)
Normalized Die-Size (Cost)	High (3.76)	**Lower** (1.00)	**Low** (1.03)	

Table 1: Latency, power, and die area comparison. Adapted from [73].

4. Leveraging TL-DRAM

TL-DRAM enables the design of many new memory management policies that exploit the asymmetric latency characteristics of the near and the far segments. Section 5 of our HPCA 2013 paper [73] describes four mechanisms that take advantage of TL-DRAM. Here, we describe two approaches in particular.

In the first approach, the memory controller uses the near segment as a *hardware-managed cache* for the far segment. In our HPCA 2013 paper [73], we discuss three policies for managing the near segment cache. The three policies differ in deciding when a row in the far segment is cached into the near segment and when the row is evicted. In addition, we propose a new data transfer mechanism (*Inter-Segment Data Transfer*) that efficiently migrates data between the segments by taking advantage of the fact that the bitline is a bus connected to the cells in both segments. By using this technique, the data from the source row can be transferred to the destination row over the bitlines at very low latency (additional 4ns over t_{RC}).[3] Furthermore, this Inter-Segment Data Transfer happens exclusively within a DRAM bank without utilizing the DRAM channel, allowing concurrent accesses to other banks.

In the second approach, the near segment capacity is exposed to the OS, enabling the OS to use the full DRAM capacity. We propose two concrete mechanisms, one where the memory controller uses an additional layer of indirection to map frequently-accessed pages to the near segment, and another where the OS uses static/dynamic profiling to directly map frequently-accessed pages to the near segment. In both approaches, the accesses to pages that are mapped to the near segment are served faster and with lower power than in conventional DRAM, resulting in improved system performance and energy efficiency.

We refer the reader to Section 5 of our HPCA 2013 paper [73] for a full description of use cases for TL-DRAM. Note that a very wide variety of techniques developed for cache management [105,115,119,120,132] can be adopted to manage the near segment in TL-DRAM.

5. Performance and Power Evaluation

Section 8 of our HPCA 2013 paper [73] provides a detailed evaluation of all of the above approaches to leverage TL-DRAM. Here, we present the evaluation results for only the first approach, in which the near segment is used as a hardware-managed cache managed under our best policy (*Benefit-Based Caching*), to demonstrate the advantages of our TL-DRAM substrate.

Methodology. To evaluate our mechanism, we use Ramulator [56,110], an open-source DRAM simulator, which is integrated into an in-house processor simulator. The released version of Ramulator [110] provides a model for TL-DRAM, which we hope future works use and build upon. A detailed methodology can be found in Section 7 of our HPCA 2013 paper [73].

Performance & Power Analysis. Figure 8 shows the average performance improvement and power efficiency of our proposed mechanism over the baseline with conventional DRAM, on 1-, 2- and 4-core systems. As described in Section 3, the access latency and power consumption are significantly lower for near segment accesses, but higher for far segment accesses, compared to accesses in a conventional DRAM. We observe that a large fraction (over 90% on average) of requests hit in the rows cached in the near segment, thereby accessing the near segment with low latency and low power consumption. As a result, TL-DRAM achieves significant performance improvements of 12.8%/12.3%/11.0%, and power savings of 23.6%/26.4%/28.6% in 1-/2-/4-core systems, respectively.

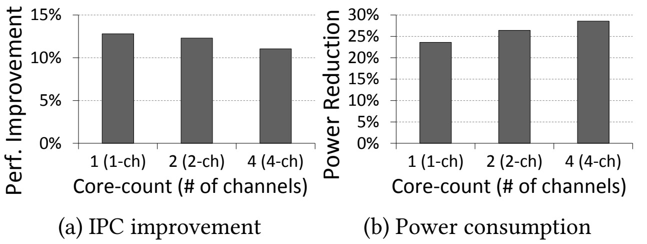


Figure 8: IPC improvement and power consumption of TL-DRAM. Adapted from [73].

Sensitivity to Near Segment Capacity. The number of rows in the near segment presents a trade-off, since increasing the near segment's size increases its capacity but also increases its access latency. Figure 9 shows the performance improvement of our proposed mechanisms over the base-

[3]A later work, RowClone [116], takes advantage of this property to enable bulk copy and initialization completely within DRAM.

line as we vary the near segment size. Initially, performance improves as the number of rows in the near segment increases, since more data can be cached. However, increasing the number of rows in the near segment beyond 32 reduces the performance benefit due to the increased capacitance and hence the higher near segment access latencies.

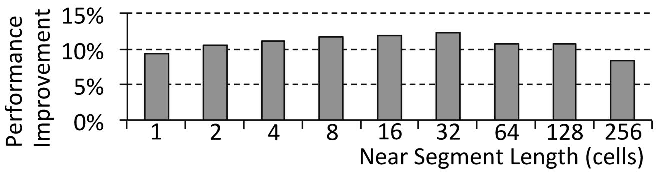


Figure 9: Effect of varying near segment capacity. Reproduced from [73].

Other Results. In our HPCA 2013 paper [73], we provide a detailed analysis of how timing parameters and power consumption vary when varying the near segment length (Sections 4 and 6.3 of [73], respectively). We also provide a comprehensive evaluation of the mechanisms we build on top of the TL-DRAM substrate for both single- and multi-core systems (Section 8 of [73]).

6. Related Work

To our knowledge, our HPCA 2013 paper [73] is the first to *i)* enable latency heterogeneity in DRAM without significantly increasing the DRAM cost per bit, and *ii)* propose hardware/software mechanisms that leverage this latency heterogeneity to improve system performance. We make the following major contributions.

A Cost-Efficient Low-Latency DRAM. Based on the key observation that long internal wires (bitlines) are the dominant source of DRAM latency, our HPCA 2013 paper [73] proposes a new DRAM architecture called Tiered-Latency DRAM (TL-DRAM). To our knowledge this is the first work to enable low-latency DRAM *without* significantly increasing the DRAM cost per bit. By adding a single isolation transistor to each bitline, we carve out a region within a DRAM chip, called the *near segment*, which is fast and energy-efficient. This comes at a modest overhead of 3% increase in DRAM die-area. While there are two prior approaches to reduce DRAM latency (using short bitlines [85, 112], adding an SRAM cache in DRAM [32, 36, 39, 142]), both of these approaches significantly increase die-area due to additional sense amplifiers or additional area for an SRAM cache, as we evaluate in our full paper [73]. Compared to these prior approaches, TL-DRAM is a much more cost-effective architecture for achieving low latency.

There are many recent works that reduce *overall memory access latency* by modifying DRAM, the DRAM-controller interface, and DRAM controllers. These works enable more parallelism and bandwidth [22, 60, 71, 116], reduce refresh counts [50, 51, 52, 53, 75, 76, 103, 134], accelerate bulk operations [23, 114, 116, 117, 118], accelerate computation in the logic layer of 3D-stacked DRAM [1,2,7,8,33,35,40,41,55,77,101,141], enable better communication between CPU and other devices through DRAM [69], leverage process variation and temperature dependency in DRAM [20, 21, 24, 70, 72], leverage design-induced variation in DRAM [68], leverage DRAM access patterns [37, 38, 123], reduce write-related latencies by better designing DRAM and DRAM control policies [26, 66, 113], and reduce overall queuing latencies in DRAM by better scheduling memory requests [29, 30, 31, 34, 42, 43, 49, 58, 59, 65, 87, 88, 89, 94, 95, 121, 126, 127, 133]. Our proposal is orthogonal to all of these approaches and can be applied in conjunction with them to achieve higher latency and energy benefits.

Inter-Segment Data Transfer. By implementing latency heterogeneity within a DRAM subarray, TL-DRAM enables efficient data transfer between the fast and slow segments by utilizing the bitlines as a wide bus. This mechanism takes advantage of the fact that both the source and destination cells share the same bitlines. Furthermore, this inter-segment migration happens only within a DRAM bank and does not utilize the DRAM channel, thereby allowing concurrent accesses to other banks over the channel. This inter-segment data transfer enables fast and efficient movement of data within DRAM, which in turn enables efficient ways of taking advantage of latency heterogeneity.

Other works that leverage latency heterogeneity in DRAM do not usually provide any efficient mechanism of inter-segment data migration between different latency segments. For example, Son et al. [124] propose a low-latency DRAM architecture that has different, fast (long bitline) and slow (short bitline) subarrays in DRAM. This approach provides the significant benefit only if latency-critical data is already allocated to the low-latency regions (the low latency subarrays). Therefore, the overall memory system performance is very sensitive to the page placement policy, and the system cannot easily adopt to changes in the access latency of pages. In contrast, our new inter-segment data transfer mechanism enables efficient relocation of pages, leading to efficient dynamic page placement and relocation based on the dynamically determined latency criticality of each page. Several more recent works [23, 114, 116, 117] take advantage of our concept of inter-segment data transfer mechanism to perform page copy/initialization and bulk bitwise operations completely within a DRAM chip.

7. Potential Long-Term Impact

Tolerating High DRAM Latency by Enabling New Layers in the Memory Hierarchy. Today, there is a large latency cliff between the on-chip last level cache and off-chip DRAM, leading to a large performance fall-off when applications start missing in the last level cache. By introducing an additional fast layer (the near segment) within the DRAM itself, TL-DRAM smoothens this latency cliff.

Note that many recent works add a DRAM cache or create heterogeneous main memories [25, 28, 62, 63, 74, 81, 82, 83, 102, 106, 108, 109, 138, 140] to smooth the latency cliff between the last level cache and a longer-latency non-volatile main memory, e.g., phase-change memory [62, 63, 64, 83, 84, 104, 106, 137, 139], STT-MRAM [61, 83, 97, 135], or RRAM/memristors [27, 125, 136], or to take advantage of the advantages of multiple different types of memories to optimize for multiple metrics. Our approach is similar at the high-level (i.e., to reduce the latency cliff at low cost by taking advantage of heterogeneity), yet we introduce the new low-latency layer *within DRAM itself* instead of adding a completely separate device. Tiered-Latency DRAM can also be used as a fast DRAM cache.

Applicability to Future Memory Devices. We show the benefits of TL-DRAM's asymmetric latencies. Considering that most memory devices adopt a similar cell organization (i.e., a two-dimensional cell array and row/column bus connections), our approach of reducing the electrical load of connecting to a bus (bitline) to achieve low access latency can be applicable to other memory devices. Furthermore, the idea of performing inter-segment data transfer can also potentially be applied to other memory devices, regardless of the memory technology. For example, we believe it is promising to examine similar approaches for emerging memory technologies like phase-change memory [62, 63, 64, 83, 84, 104, 106, 137, 139], STT-MRAM [61, 83, 97, 135], or RRAM/memristors [27, 125, 136], as well as NAND flash memory technology [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 78, 79, 80, 81].

New Research Opportunities. The TL-DRAM substrate creates new opportunities by enabling mechanisms that can leverage the latency heterogeneity offered by the substrate. We briefly describe three directions, but we believe that there are many new possibilities.

- *New ways of leveraging TL-DRAM:* TL-DRAM is a substrate that can be utilized for many applications. Although we describe two major ways of leveraging TL-DRAM in our HPCA 2013 paper [73], we believe there are more ways to leverage the TL-DRAM substrate both in hardware and software. For instance, new mechanisms could be devised to detect data that is latency critical (e.g., data that causes many threads to become serialized [31, 45, 46, 130, 131] or data that belongs to threads that are more latency-sensitive or important [4, 5, 29, 58, 59, 65, 67, 126, 127, 128, 129, 133]) or could become latency critical in the near future and allocate/prefetch such data into the near segment.
- *Opening up new design spaces with multiple tiers:* TL-DRAM can be easily extended to have multiple latency tiers by adding more isolation transistors to the bitlines, providing more latency asymmetry. Our HPCA 2013 paper [73] provides an analysis of the latency of a TL-DRAM design with three tiers, showing the spread in latency for three tiers. This enables new mechanisms both in hardware and software that can allocate data appropriately to different tiers based on their access characteristics such as locality, criticality, priority, etc.
- *Inspiring new ways of architecting latency heterogeneity within DRAM:* To our knowledge, TL-DRAM is the first to enable latency heterogeneity within DRAM, which is significantly modifying the existing DRAM architecture. We believe that this could inspire research on other possible ways of architecting latency heterogeneity within DRAM [20, 21, 24, 37, 38, 68, 70, 72] or other memory devices. Note that recent works that are after our HPCA 2013 paper clearly exploit this promising direction proposed by our paper [20, 21, 24, 37, 38, 68, 70, 72, 116].

Acknowledgments

We thank Saugata Ghose for his dedicated effort in the preparation of this article. Many thanks to Uksong Kang, Hak-soo Yu, Churoo Park, Jung-Bae Lee, and Joo Sun Choi from Samsung, and Brian Hirano from Oracle, for their helpful comments. We thank the reviewers for their feedback. We acknowledge the support of our industrial partners: AMD, HP Labs, IBM, Intel, Oracle, Qualcomm, and Samsung. This research was also partially supported by grants from the NSF (grants 0953246 and 1212962), GSRC, and the Intel URO Memory Hierarchy Program.

References

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[2] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.

[3] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a Modern Processor: Where Does Time Go?" in *VLDB*, 1999.

[4] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: achieving high performance and scalability in heterogeneous systems," in *ISCA*, 2012.

[5] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, "Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance," in *PACT*, 2015.

[6] S. Borkar and A. A. Chien, "The future of microprocessors," in *CACM*, 2011.

[7] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.

[8] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," in *IEEE CAL*, 2016.

[9] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," in *Proceedings of the IEEE*, 2017.

[10] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.

[11] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.

[12] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[13] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *DATE*, 2012.

[14] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.

[15] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.

[16] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.

[17] Y. Cai, Y. Luo, E. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.

[18] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[19] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-cell Assisted Error Correction for MLC NAND Flash Memories," in *SIGMETRICS*, 2014.

[20] K. K. Chang, "Understanding and Improving Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.

[21] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[22] K. K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.

[23] K. K. Chang, P. J. Nair, S. Ghose, D. Lee, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[24] K. K. Chang, A. G. Yaglikci, A. Agrawal, N. Chatterjee, S. Ghose, A. Kashyap, H. Hassan, D. Lee, M. O'Connor, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[25] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in *MICRO*, 2012.

[26] N. Chatterjee, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Staged Reads: Mitigating the Impact of DRAM Writes on DRAM Reads," in *HPCA*, 2012.

[27] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, 1971.

[28] G. Dhiman *et al.*, "PDRAM: A hybrid PRAM and DRAM main memory system," in *DAC*, 2009.

[29] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via Source Throttling: A Configurable and High-performance Fairness Substrate for Multi-core Memory Systems," in *ASPLOS*, 2010.

[30] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared resource management for multi-core systems," in *ISCA*, 2011.

[31] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel Application Memory Scheduling," in *MICRO*, 2011.

[32] Enhanced Memory Systems, "Enhanced SDRAM SM2604," 2002.

[33] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *HPCA*, 2016.

[34] S. Ghose, H. Lee, and J. F. Martínez, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.

[35] Q. Guo *et al.*, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WoNDP*, 2013.

[36] C. A. Hart, "CDRAM in a Unified Memory Architecture," in *Compcon*, 1994.

[37] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.

[38] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.

[39] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima, "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," in *IEEE Micro*, 1990.

[40] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.

[41] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.

[42] I. Hur and C. Lin, "Adaptive History-Based Memory Schedulers," in *MICRO*, 2004.

[43] E. Ipek *et al.*, "Self Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.

[44] JEDEC, "DDR3 SDRAM STANDARD," http://www.jedec.org/standards-documents/docs/jesd-79-3d, 2010.

[45] J. A. Joao *et al.*, "Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs," in *ISCA*, 2013.

[46] J. A. Joao, M. A. Suleman *et al.*, "Bottleneck identification and scheduling in multithreaded applications," in *ASPLOS*, 2012.

[47] T. S. Jung, "Memory technology and solutions roadmap," http://www.sec.co.kr/images/corp/ir/irevent/techforum_01.pdf, 2005.

[48] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.

[49] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era," in *MICRO*, 2011.

[50] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.

[51] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.

[52] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[53] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," in *IEEE CAL*, 2016.

[54] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.

[55] J. S. Kim, D. Senol, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.

[56] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," in *IEEE CAL*, 2015.

[57] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[58] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA*, 2010.

[59] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.

[60] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[61] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.

[62] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," in *IEEE Micro*, 2010.

[63] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory As a Scalable DRAM Alternative," in *ISCA*, 2009.

[64] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," in *CACM*, 2010.

[65] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-Aware DRAM Controllers," in *MICRO*, 2008.

[66] C. J. Lee, E. Ebrahimi, V. Narasiman, O. Mutlu, and Y. N. Patt, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," Univ. of Texas at Austin, High Performance Systems Group, Tech. Rep. TR-HPS-2010-002, 2010.

[67] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, "Improving Memory Bank-Level Parallelism in the Presence of Prefetching," in *MICRO*, 2009.

[68] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.

[69] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.

[70] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.

[71] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," in *ACM TACO*, 2016.

[72] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.

[73] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[74] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-Based Hybrid Memory Management," in *CLUSTER*, 2017.

[75] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.

[76] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.

[77] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent Data Structures for Near-Memory Computing," in *SPAA*, 2017.

[78] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.

[79] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *MSST*, 2015.

[80] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.

[81] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in *DSN*, 2014.

[82] J. Meza *et al.*, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," in *IEEE CAL*, 2012.

[83] J. Meza *et al.*, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," in *WEED*, 2013.

[84] J. Meza, J. Li, and O. Mutlu, "A case for small row buffers in non-volatile main memories," in *ICCD*, 2012.
[85] Micron, "RLDRAM 2 and 3 Specifications," http://www.micron.com/products/dram/rldram-memory.
[86] Y. Moon *et al.*, "1.2V 1.6Gb/s 56nm 6F2 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture," *ISSCC*, 2009.
[87] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *USENIX Security*, 2007.
[88] J. Mukundan and J. F. Martínez, "MORSE: Multi-Objective Reconfigurable Self-Optimizing Memory Scheduler," in *HPCA*, 2012.
[89] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing Memory Interference in Multicore Systems via Application-aware Memory Channel Partitioning," in *MICRO*, 2011.
[90] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[91] O. Mutlu, "Main Memory Scaling: Challenges and Solution Directions," in *More than Moore Technologies for Next Generation Computer Design*. Springer, 2015.
[92] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: An effective alternative to large instruction windows," in *IEEE Micro*, 2003.
[93] O. Mutlu, H. Kim, and Y. N. Patt, "Techniques for Efficient Processing in Runahead Execution Engines," in *ISCA*, 2005.
[94] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[95] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[96] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors," in *HPCA*, 2003.
[97] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[98] S. Narasimha *et al.*, "High performance 45-nm SOI technology with enhanced strain, porous low-k BEOL, and immersion lithography," in *IEDM*, 2006.
[99] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[100] D. A. Patterson, "Latency lags bandwith," in *Commun. ACM*, 2004.
[101] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.
[102] S. Phadke and S. Narayanasamy, "MLP aware heterogeneous memory system," in *DATE*, 2011.
[103] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[104] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *MICRO*, 2009.
[105] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for MLP-aware cache replacement," in *ISCA*, 2006.
[106] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[107] Rambus, "DRAM Power Model," http://www.rambus.com/energy, 2010.
[108] L. E. Ramos *et al.*, "Page placement in hybrid memory systems," in *ICS*, 2011.
[109] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu, "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems," in *MICRO*, 2015.
[110] SAFARI Research Group, "Ramulator – GitHub Repository," https://github.com/CMU-SAFARI/ramulator.
[111] Samsung, "DRAM Data Sheet," http://www.samsung.com/global/business/semiconductor/product.
[112] Y. Sato *et al.*, "Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM," in *VLSIC*, 1998.
[113] V. Seshadri, A. Bhowmick, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry, "The Dirty-Block Index," in *ISCA*, 2014.
[114] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. Kozuch, O. Mutlu, P. Gibbons, and T. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," in *IEEE CAL*, 2015.
[115] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing," in *PACT*, 2012.
[116] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[117] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[118] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses," in *MICRO*, 2015.
[119] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks," *TACO*, 2015.
[120] A. Seznec, "A Case for Two-Way Skewed-Associative Caches," in *ISCA*, 1993.
[121] J. Shao and B. T. Davis, "A Burst Scheduling Access Reordering Mechanism," in *HPCA*, 2007.
[122] S. M. Sharroush *et al.*, "Dynamic random-access memories without sense amplifiers," in *Elektrotechnik & Informationstechnik*, 2012.
[123] W. Shin, J. Yang, J. Choi, and L.-S. Kim, "NUAT: A Non-Uniform Access Time Memory Controller," in *HPCA*, 2014.
[124] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.
[125] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, 2008.
[126] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "The Blacklisting Memory Scheduler: Achieving high performance and fairness at low cost," in *ICCD*, 2014.
[127] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," in *TPDS*, 2016.
[128] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," in *MICRO*, 2015.
[129] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," in *HPCA*, 2013.
[130] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, "Accelerating critical section execution with asymmetric multi-core architectures," in *ASPLOS*, 2009.
[131] M. A. Suleman, O. Mutlu, J. A. Joao, Khubaib, and Y. N. Patt, "Data Marshaling for Multi-core Architectures," in *ISCA*, 2010.
[132] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, "A Modified Approach to Data Cache Management," in *MICRO*, 1995.
[133] H. Usui, L. Subramanian, K. K.-W. Chang, and O. Mutlu, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," in *ACM TACO*, 2016.
[134] R. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *HPCA*, 2006.
[135] J. Wang, X. Dong, and Y. Xie, "Enabling High-performance LPDDRx-compatible MRAM," in *ISLPED*, 2014.
[136] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai, "Metal–Oxide RRAM," in *Proceedings of the IEEE*, 2012.
[137] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.
[138] H. Yoon *et al.*, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.
[139] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories," in *ACM TACO*, 2014.
[140] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient DRAM Caching via Software/Hardware Cooperation," in *MICRO*, 2017.
[141] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented Programmable Processing in Memory," in *HPCA*, 2014.
[142] Z. Zhang, Z. Zhu, and X. Zhang, "Cached DRAM for ILP Processor Memory Access Latency Reduction," in *IEEE Micro*, 2001.

Adaptive-Latency DRAM: Reducing DRAM Latency by Exploiting Timing Margins

Donghyuk Lee[1,2] Yoongu Kim[2] Gennady Pekhimenko[3,2]
Samira Khan[4,2] Vivek Seshadri[5,2] Kevin Chang[6,2] Onur Mutlu[7,2]

[1]*NVIDIA Research* [2]*Carnegie Mellon University* [3]*University of Toronto*
[4]*University of Virginia* [5]*Microsoft Research India* [6]*Facebook* [7]*ETH Zürich*

This paper summarizes the idea of Adaptive-Latency DRAM (AL-DRAM), which was published in HPCA 2015 [90], and examines the work's significance and future potential. AL-DRAM is a mechanism that optimizes DRAM latency based on the DRAM module and the operating temperature, by exploiting the extra margin that is built into the DRAM timing parameters. DRAM manufacturers provide a large margin for the timing parameters as a provision against two worst-case scenarios. First, due to process variation, some outlier DRAM chips are much slower than others. Second, chips become slower at higher temperatures. The timing parameter margin ensures that the slow outlier chips operate reliably at the worst-case temperature, and hence leads to a high access latency.

Using an FPGA-based DRAM testing platform, our work first characterizes the extra margin for 115 DRAM modules from three major manufacturers. The experimental results demonstrate that it is possible to reduce four of the most critical timing parameters by a minimum/maximum of 17.3%/54.8% at 55℃ while maintaining reliable operation. AL-DRAM uses these observations to adaptively select reliable DRAM timing parameters for each DRAM module based on the module's current operating conditions. AL-DRAM does not require any changes to the DRAM chip or its interface; it only requires multiple different timing parameters to be specified and supported by the memory controller. Our real system evaluations show that AL-DRAM improves the performance of memory-intensive workloads by an average of 14% without introducing any errors.

Our characterization and proposed techniques have inspired several other works on analyzing and/or exploiting different sources of latency and performance variation within DRAM chips [30, 34, 51, 71, 89, 127].

1. Problem: High DRAM Latency

A DRAM chip is made of capacitor-based cells that represent data in the form of electrical charge. To store data in a cell, charge is injected, whereas to retrieve data from a cell, charge is extracted. Such *movement of charge* happens through a wire called *bitline*. Due to the large resistance and the large capacitance of the bitline, it takes a long time to access DRAM cells. To guarantee correct operation for every module sold, DRAM manufacturers impose a set of minimum latency restrictions on DRAM accesses, called *timing parameters* [60]. Ideally, timing parameters should provide *just enough* time for a DRAM chip to operate correctly. In practice, however, there is a very large margin in the timing parameters to ensure correct operation under *worst-case* conditions with respect to two aspects. First, due to *process variation*, some outlier cells suffer from a larger RC-delay than other cells [64, 94], and require more time to be accessed. Second, due to *temperature dependence*, DRAM cells lose more charge at high temperature [97, 171], and therefore require more time to be accessed. Due to the worst-case provisioning of the fixed timing parameters, which ensure reliable operation up to a temperature of 85℃, it takes a longer time to access most of DRAM under most operating conditions than is actually necessary for correct operation.

2. Key Observations and Our Goal

First, we observe that **most DRAM chips do *not* contain the worst-case cells that require the largest access latency.** Using an FPGA-based testing platform [52], we profile 115 real DRAM modules and observe that the slowest cell (i.e., the cell that stores the smallest amount of charge) for a typical chip is still significantly faster than the slowest cell of the worst-case chip. Our profiling exposes the large margin built into *DRAM* timing parameters. In particular, we identify four timing parameters that are the most critical during a DRAM access: `tRCD`, `tRAS`, `tWR`, and `tRP`.[1] At 55℃, we demonstrate that the parameters can be reduced by an average of 17.3%, 37.7%, 54.8%, and 35.2%, respectively, while still maintaining correctness.

Second, we observe that **most DRAM chips are *not* exposed to the worst-case temperature of 85℃.** We measure the DRAM ambient temperature in a server cluster running a very memory-intensive benchmark, and find that the temperature *never* exceeds 34℃, and never changes by more than 0.1℃ per second. Other works [48, 99] also observe that worst-case DRAM temperatures are not common, and that servers typically operate at much lower temperatures [48, 99].

Based on these two observations, we show *that* typical DRAM chips operating at typical temperatures (e.g., 55℃) are capable of operating correctly when accessed with a much smaller access latency, but are nevertheless forced to operate

[1]For a detailed background on the operation of DRAM, and an explanation of each timing parameter, we refer the reader to our prior works [30, 31, 32, 34, 51, 52, 67, 68, 69, 70, 71, 73, 75, 76, 77, 78, 88, 89, 90, 92, 93, 97, 98, 127, 146, 147].

at the largest latency of the worst-case module and operating conditions. Modules in existing systems use these worst-case latencies because existing memory controllers are equipped with only a single set of timing parameters that are dictated by the worst case.

Our goal in our HPCA 2015 paper [90] is to exploit the extra margin that is built into the DRAM timing parameters to reduce DRAM latency, and thus improve performance as well as energy consumption. To this end, we first provide a detailed analysis of *why* we can reduce DRAM timing parameters without sacrificing reliability.

3. Charge & Latency Interdependence

The operation of a DRAM cell is governed by two important parameters: *i)* the quantity of charge and *ii)* the latency it takes to move charge. These two parameters are closely related to each other. Based on SPICE simulations with a detailed DRAM model, we identify the quantitative relationship between charge and latency [90]. We briefly summarize our three key observations from these analyses here. Section 7 of our HPCA 2015 paper [90] provides a detailed analysis of our observations.

First, having more charge in a DRAM cell accelerates the *sensing* operation in the cell, especially at the beginning of sensing, enabling the opportunity to shorten the timing parameters that correspond to sensing (`tRCD` and `tRAS`). Second, when *restoring* the charge in a DRAM cell, a large amount of the time is spent on injecting the final small amount of charge into the cell. If there is already enough charge in the cell for the next access, the cell does *not* need to be fully restored. In this case, it is possible to shorten the latter part of the restoration time, creating the opportunity to shorten the timing parameters that correspond to restoration (`tRAS` and `tWR`). Third, at the end of *precharging*, i.e., setting the bitline into the initial voltage level (before accessing a cell) for the next access, a large amount of the time is spent on precharging the final small amount of bitline voltage difference from the initial level. When there is already enough charge in the cell to overcome the voltage difference in the bitline, the bitline does *not* need to be fully precharged. Thus, it is possible to shorten the final part of the precharge time, creating the opportunity to shorten the timing parameter that corresponds to precharge (`tRP`). Based on these three observations, we conclude that *timing parameters can be shortened if DRAM cells have enough charge.*

4. Adaptive-Latency DRAM

As explained in Section 3, the amount of charge in the cell right before an access to it plays a critical role in how long it takes to retrieve the correct data from the cell. In Figure 1, we illustrate the impact of process variation using two different cells: one is a *typical* cell (left column) and the other is the *worst-case* cell that deviates the most from the typical (right column). The worst-case cell initially contains less charge than the typical cell for two reasons. First, due to its *large resistance*, the worst-case cell cannot allow charge to flow inside quickly. Second, due to its *small capacitance*, the worst-case cell cannot store much charge even when it is fully charged. To accommodate such a worst-case cell, existing timing parameters are conservatively set to large values.

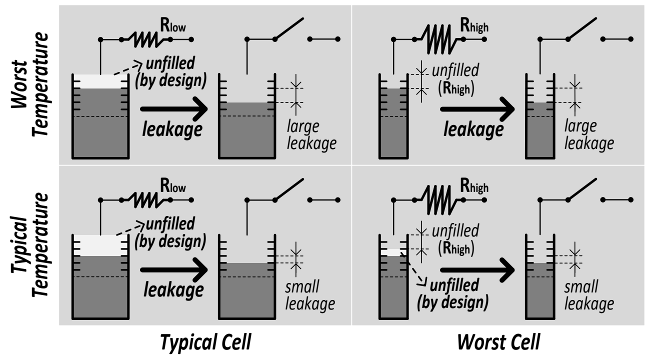


Figure 1: Effect of reduced latency: typical vs. worst-case. Reproduced from [90].

In Figure 1, we also illustrate the impact of temperature dependence using two cells at two different operating temperatures: *i)* a typical temperature (55℃, bottom row), and *ii)* the worst-case temperature (85℃, top row) supported by DRAM standards. Both typical and worst-case cells leak charge at a faster rate at the worst-case temperature. Therefore, not only does the worst-case cell have less charge to begin with, but it is left with *even less* charge at the worst temperature because it leaks charge at a faster rate (top-right in Figure 1). To accommodate the combined effect of process variation *and* temperature dependence, existing timing parameters are set to very large values. That is why the worst-case condition for correctness is specified by the top-right of Figure 1, which shows the least amount of charge stored in *the worst-case cell at the worst-case temperature* in its initial state. On top of this, DRAM manufacturers add an extra latency margin to the access time under worst-case conditions. In other words, the amount of charge in a cell under worst-case conditions is still greater than the minimum amount of charge required for correctness.

If we were to reduce the timing parameters, we would also reduce the amount of charge stored in the cells. It is important to note, however, that we are proposing to exploit *only* the *additional slack* (in terms of charge) compared to the worst case. This allows us to provide as strong of a reliability guarantee as manufacturers currently do for worst-case cells and operating conditions. In Figure 1, we illustrate the impact of reducing the timing parameters. The lightened portions inside the cells represent the amount of charge that we are giving up by using reduced timing parameters. Note that we are not giving up any charge for the worst-case cell at the worst-case temperature. Although the other three cells are *not* fully charged in their initial state, w propose to give up just enough charge from them such that they are left with a similar amount of charge as the worst case (top-right). This is because these cells are capable of either holding more charge to begin with (typical cell, left column) or holding their charge for longer (typical temperature, bottom row). Therefore, optimizing the timing parameters (based on the amount of existing charge slack) provides the opportunity to reduce overall DRAM latency while still maintaining the same reliability guarantees provided by DRAM manufacturers.

Based on these observations, we propose Adaptive-Latency DRAM (AL-DRAM), a mechanism that dynamically optimizes the timing parameters for different modules at different temperatures. AL-DRAM exploits the *additional charge slack* present in the common-case compared to the worst-case, thereby preserving the level of reliability (at least as high as the worst-case) provided by DRAM manufacturers.

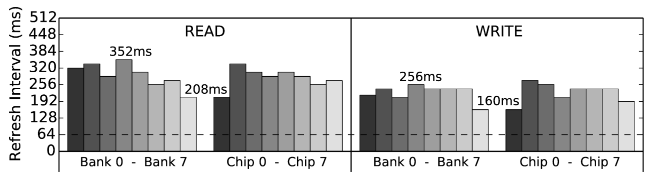
5. DRAM Latency Profiling: Experimental Analysis of 115 Modules

We present and analyze the results of our DRAM profiling experiments, performed on our FPGA-based DRAM testing infrastructure, SoftMC [52], which is also used in our various past works analyzing various DRAM characteristics [30,34,68,69,75,89,90,97,136]. In total, we analyze 115 DRAM modules from three major manufacturers, comprising 920 total DRAM chips. Our full methodology is explained in Section 6 of our HPCA 2015 paper [90].

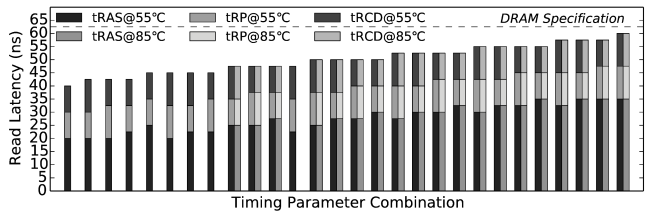
5.1. Analysis of a Representative DRAM Module

We study the possible timing parameter reductions of a DRAM module while still maintaining correctness. To guarantee reliable DRAM operation, DRAM manufacturers provide a built-in *safety margin* in retention time, also referred to as *a guardband* [2,68,97,127,166]. This way, DRAM manufacturers are able to guarantee that even the weakest cell is insured against various other modes of failure. We first measure the safety margin of a DRAM module by sweeping the refresh interval at the worst-case operating temperature (85℃), using the standard timing parameters. Figure 2a plots the maximum refresh intervals of each bank and each chip in a DRAM module for both read and write operations. We make several observations. First, the maximum error-free refresh intervals of both read and write operations are much larger than the DRAM standard (208 ms for the read and 160 ms for the write operations vs. the 64 ms standard). Second, for the smaller architectural units (banks and chips in the DRAM module), some of them operate *without* incurring errors even at much higher refresh intervals than others (as high as 352 ms for the read operations and 256 ms for the write operations). This is because the error-free retention time is determined by the worst single cell in each architectural component (i.e., bank/chip/module).

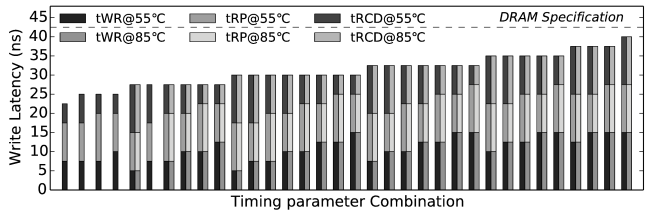
Based on this experiment, we define the *safe refresh interval* for a DRAM module as the maximum refresh interval that leads to no errors, minus an additional margin of 8 ms, which



(a) Maximum error-free refresh interval at 85℃ (bank/chip/module)



(b) Read latency (refresh Iinterval: 200 ms)



(c) Write latency (refresh interval: 152 ms)

Figure 2: Latency reductions while maintaining the safety margin of DRAM. Reproduced from [90].

is the increment at which we sweep the refresh interval. The safe refresh interval for the read and write operations are 200 ms and 152 ms, respectively. We then use the safe refresh intervals to run the tests with all possible combinations of timing parameters. For each combination, we run our tests at two temperatures: 85℃ and 55℃.

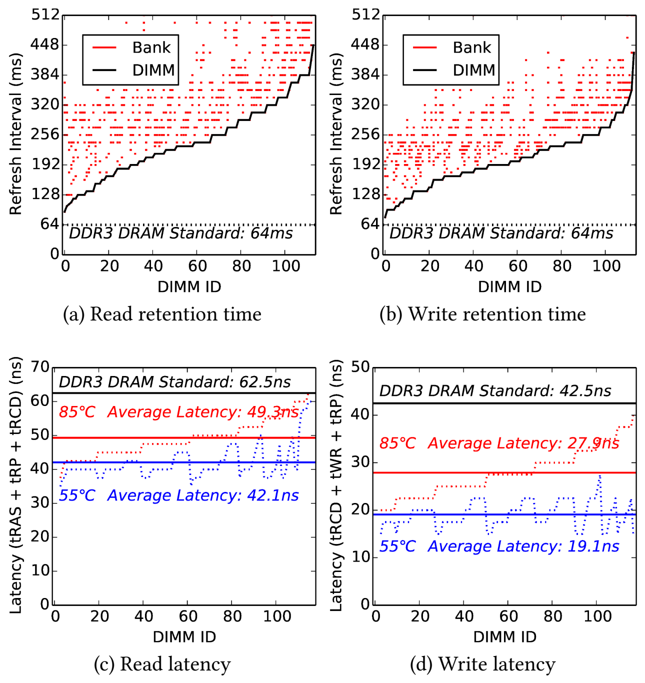
Figure 2b plots the error-free timing parameter combinations (`tRCD`, `tRAS`, and `tRP`) in the read operation test. For each combination, there are two stacked bars — the left bar for the test at 55℃ and the right bar for the test at 85℃. Missing bars indicate that the test (with that timing parameter combination at that temperature) causes errors. Figure 2c plots same data for the write operation test (`tRCD`, `tWR`, and `tRP`).

We make two observations. First, even at the highest temperature of 85℃, the DRAM module reliably operates with reduced timing parameters (24% reduction for read, and 35% reduction for write operations). Second, at the lower temperature of 55℃, the potential latency reduction is even higher (36% for read, and 47% for write operations). These latency reductions are possible *while* maintaining the safety margin of the DRAM module. From these two observations, we conclude that there is significant opportunity to reduce DRAM timing parameters *without compromising reliability.*

5.2. Analysis of 115 DRAM Modules

We have studied the effect of temperature and the potential to reduce various timing parameters at different temperatures for a single DRAM module. The same trends and observations also hold true for all of the other modules we studied. In this section, we analyze the effect of process variation by studying the results of our profiling experiments on 115 DIMMs. We also present results for intra-chip process variation by studying the process variation across different banks within each DIMM.

Figure 3a (solid line) plots the highest refresh interval that leads to correct operation across all cells at 85℃ within *each DIMM* for the read operation test. The red dots on top show the highest refresh interval that leads to correct operation across all cells within *each bank* for all 8 banks. Figure 3b plots the same data for the write operation test.



(a) Read retention time (b) Write retention time

(c) Read latency (d) Write latency

Figure 3: Analysis of 115 modules. Reproduced from [90].

We draw two conclusions. First, although there exist a few modules which *just* meet the timing parameters (with a low safety margin), a vast majority of the modules very comfortably meet the standard timing parameters (with a high safety margin). This indicates that a majority of the DIMMs have significantly higher safety margins than the worst-case module *even at the highest-acceptable operating temperature of 85℃*. Second, the effect of process variation is even higher for banks within the same DIMM, explained by the large spread in the red dots for each DIMM. Since banks within a DIMM can be accessed independently with different timing parameters, one can potentially imagine a mechanism that more aggressively reduces timing parameters at a bank granularity and not just the DIMM granularity. We leave this for future work.[2]

To study the potential of reducing timing parameters for each DIMM, we sweep all possible combinations of timing parameters (`tRCD`/`tRAS`/`tWR`/`tRP`) for all the DIMMs at both the highest acceptable operating temperature (85℃) and a more typical operating temperature (55℃). We then determine the acceptable DRAM timing parameters for each DIMM for both temperatures while maintaining its safety margin.

Figures 3c and 3d show the results of this experiment for the DRAM read and DRAM write, respectively. The y-axis plots the sum of the relevant timing parameters (`tRCD`, `tRAS`, and `tRP` for the DRAM read and `tRCD`, `tWR`, and `tRP` for the DRAM write). The solid black line shows the latency sum of the standard timing parameters (DDR3 DRAM specification). The dotted red line and the dotted blue line show the most acceptable latency parameters for each DIMM at 85℃ and 55℃, respectively. The solid red line and blue line show the average acceptable latency across all DIMMs.

We make two observations. First, even at the highest temperature of 85℃, DIMMs can reliably operate at reduced access latencies: 21.1% on average for read, and 34.4% on average for write operations. This is a direct result of the possible reductions in timing parameters `tRCD`/`tRAS`/`tWR`/ `tRP` — 15.6%/20.4%/20.6%/28.5% on average across all the DIMMs.[3] As a result, we conclude that process variation and lower temperatures enable a significant potential to reduce DRAM access latencies. Second, we observe that at lower temperatures (e.g., 55℃) the potential for latency reduction is even greater (32.7% on average for read, and 55.1% on average for write operations), where the corresponding reduction in timing parameters `tRCD`/`tRAS`/`tWR`/ `tRP` are 17.3%/37.7%/ 54.8%/35.2% on average across all the DIMMs.

We conclude that existing DRAM modules can be accessed reliably with lower access latencies, especially at lower temperatures than the worst-case temperature specified by DRAM manufacturers.

6. Real-System Evaluation

We evaluate AL-DRAM on a real system that offers dynamic software-based control over DRAM timing parameters at runtime [10,11]. We use the minimum values of the timing parameters that do *not* introduce any errors at 55℃ for any module to determine the latency reduction at 55℃. Thus, the latency is reduced by 27%/32%/33%/18% for `tRCD`/`tRAS`/`tWR`/`tRP`, respectively. Our full methodology is described in Section 8 of our HPCA 2015 paper [90].

Figure 4 shows the performance improvement of reducing the timing parameters in the evaluated memory system with one rank and one memory channel at a 55℃ operating temperature. We run a variety of different applications in two different configurations. The first one (single-core) runs only one thread, and the second one (multi-core) runs multiple applications/threads. We run each configuration 30 times (only SPEC benchmarks are executed 3 times due to their

[2]Note that our future works [30, 33, 34, 87, 89] explain this observation of latency heterogeneity within a DRAM chip.

[3]Due to space constraints, we present only the *average* potential reduction for each timing parameter. However, detailed characterization of each DIMM can be found online at the SAFARI Research Group website [91].

large execution times), and present the average performance improvement across all the runs and their standard deviation as an error bar. Based on the last-level cache misses per kilo instructions (MPKI), we categorize our applications into memory-intensive or non-intensive groups, and report the geometric mean performance improvement across all applications from each group.

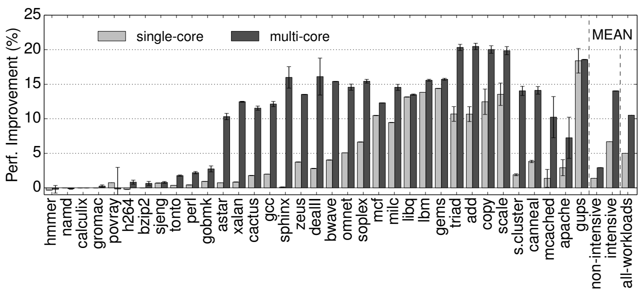


Figure 4: Real system performance improvement with AL-DRAM. Reproduced from [90].

We draw three key conclusions from Figure 4. First, AL-DRAM provides significant performance improvement over the baseline (as high as 20.5% for the very memory-bandwidth-intensive STREAM applications [109]). Second, when the memory system is under higher pressure with multi-core/multi-threaded applications, we observe significantly higher performance (than in the single-core case) across all applications from our workload pool. Third, as expected, memory-intensive applications benefit more in performance than non-memory-intensive workloads (14.0% vs. 2.9% on average). We conclude that by reducing the DRAM timing parameters using AL-DRAM, we can speed up a real system by 10.5% (on average across all 35 workloads on the multi-core/multi-thread configuration).

We also conducted reliability stress tests for our mechanism. We ran our workloads for 33 days without interruption of the lower latencies. We observed no errors and correct results.

7. Other Results and Analyses in Our Paper

Our HPCA 2015 paper [90] includes significant amount of DRAM latency analyses and system performance evaluations. We refer the reader to [90] for detailed evaluations and analyses.

- **Effect of Changing the Refresh Interval on DRAM Latency.** We evaluate DRAM latency at different refresh intervals. We observe that refreshing DRAM cells more frequently enables more DRAM latency reduction (Section 7.1 of our HPCA 2015 paper [90]).
- **Effect of Reducing Multiple Timing Parameters.** We study the potential for reducing multiple timing parameters simultaneously. Our key observation is that reducing one timing parameter leads to decreasing the opportunity to reduce another timing parameter simultaneously (Section 7.2 of our HPCA 2015 paper [90]).
- **Analysis of the Repeatability of Cell Failures.** We perform tests for five different scenarios to determine that a cell failure due to reduced latency is repeatable: *i)* same test, *ii)* test with different data patterns, *iii)* test with timing-parameter combinations, *iv)* test with different temperatures, and *v)* DRAM read/write. Most of these scenarios show that a very high fraction (more than 95%) of the erroneous cells consistently experience an error over multiple iterations of the same test (Section 7.6 of our HPCA 2015 paper [90]).
- **Performance Sensitivity Analyses.** We analyze the impact of increasing the number of ranks and channels, executing heterogeneous workloads, using different row buffer policies. We show that AL-DRAM effectively improves performance in all cases (Section 8.4 of our HPCA 2015 paper [90]).
- **Power Consumption Analysis.** We show that AL-DRAM reduces DRAM power consumption by 5.8%. This reduced power consumption is due to the reduced DRAM latencies (Section 8.4 of our HPCA 2015 paper [90]).

8. Related Work

To our knowledge, our HPCA 2015 paper is the first work to *i)* provide a detailed qualitative and empirical analysis of the relationship between *process variation* and *temperature dependence* of modern DRAM devices on the one side, and DRAM access latency on the other side (we directly attribute the relationship between the two to *the amount of charge* in cells), *ii)* experimentally characterize a large number of existing DIMMs to understand the potential of reducing DRAM timing constraints, *iii)* provide a practical mechanism that can take advantage of this potential, and *iv)* evaluate the performance benefits of this mechanism by *dynamically optimizing* DRAM timing parameters on a real system using a variety of real workloads.

Several works investigated the possibility of reducing DRAM latency by either exploiting DRAM latency variation or changing the DRAM architecture. We discuss these below.

DRAM Latency Variation. Chandrasekar et al. [29] evaluate the potential of relaxing some DRAM timing parameters to reduce DRAM latency. This work observes latency variations across DIMMs as well as for a DIMM at different operating temperatures. However, there is no explanation as to why this phenomenon exists. In contrast, our HPCA 2015 paper [90] *(i)* identifies and analyzes the root cause of latency variation in detail, *(ii)* provides a practical mechanism that can relax timing parameters, and *(iii)* provides a real system evaluation of this new mechanism, using real workloads, showing improved performance and preserved reliability.

NUAT [153] and ChargeCache [51] show that recently-refreshed rows contain more charge, and propose mechanisms to access recently-refreshed rows with reduced latency. Even though some of the observations in these works are similar to ours, the approaches to leverage them are different. AL-DRAM exploits temperature dependence in a DIMM and process variations across DIMMs, while NUAT and ChargeCache use the time difference between a row refresh and an access to the row (hence its benefits are dependent on when the row is accessed after it is refreshed). Therefore, NUAT and ChargeCache are complementary to AL-DRAM, and can potentially be combined for better performance.

Voltron [34] uses an experimental characterization of real DRAM modules to identify the relationship between the DRAM supply voltage and access latency variation. Voltron uses this relationship to identify the combination of voltage and access latency that minimizes system-level energy consumption without exceeding a user-specified threshold for the maximum acceptable performance loss.

Flexible-Latency DRAM (FLY-DRAM) [30] uses an experimental characterization of real DRAM modules to capture access latency variation across DRAM cells *within* a single DRAM chip due to manufacturing process variation. FLY-DRAM identifies that there is spatial locality in the slower cells, resulting in *fast regions* (i.e., regions where all DRAM cells can operate at significantly-reduced access latency without experiencing errors) and *slow regions* (i.e., regions where *some* of the DRAM cells *cannot* operate at significantly-reduced access latency without experiencing errors) within each chip. To take advantage of this heterogeneity in the reliable access latency of DRAM cells within a chip, FLY-DRAM (1) categorizes the cells into fast and slow regions; and (2) lowers the overall DRAM latency by accessing fast regions with a lower latency.

Design-Induced Variation-Aware DRAM (DIVA-DRAM) [89] uses an experimental characterization of real DRAM modules to identify the latency variation within a single DRAM chip that occurs due to the architectural design of the chip. For example, a cell that is further away from the row decoder requires a longer access time than a cell that is close to the row decoder. Similarly, a cell that is farther away from the wordline driver requires a larger access time than a cell that is close to the wordline driver. DIVA-DRAM uses design-induced variation to reduce the access latency to different parts of the chip.

Low-Latency DRAM Architectures. Various works [31, 32, 33, 53, 78, 92, 108, 116, 142, 146, 154, 176] propose new DRAM architectures that provide lower latency. Many of these works improve DRAM latency at the cost of either significant additional DRAM chip area (i.e., extra sense amplifiers [108, 142, 154], an additional SRAM cache [53, 176]), specialized protocols [31, 78, 92, 146] or a combination of these. Our proposed mechanism requires *no changes* to the DRAM chip and the DRAM interface, and hence has almost negligible overhead. Furthermore, AL-DRAM is largely orthogonal to these proposed designs, and can be applied in conjunction with them, providing greater cumulative reduction in latency.

Binning or Overclocking DRAM. AL-DRAM has multiple sets of DRAM timing parameters for different temperatures and dynamically optimizes the timing parameters at runtime. Therefore, AL-DRAM is different from simple binning (performed by manufacturers) or over-clocking (performed by end-users; e.g., [58, 126]) that are used to figure out the highest *static* frequency or lowest *static* timing parameters for DIMMs.

Other Methods for Lowering Memory Latency. There are many works that reduce *overall memory access latency* by modifying DRAM, the DRAM-controller interface, and DRAM controllers. These works enable more parallelism and bandwidth [3, 4, 31, 32, 78, 88, 93, 145, 146, 147, 167, 174, 178], reduce refresh counts [66, 68, 70, 97, 98, 136, 164], accelerate bulk operations [32, 145, 146, 147, 148], accelerate computation in the logic layer of 3D-stacked DRAM [5, 6, 14, 15, 50, 54, 55, 72, 100, 129, 173], enable better communication between the CPU and other devices through DRAM [93], leverage DRAM access patterns [51, 153], reduce write-related latencies by better designing DRAM and DRAM control policies [35, 83, 144], reduce overall queuing latencies in DRAM by better scheduling memory requests [12, 13, 38, 46, 49, 56, 59, 61, 65, 76, 77, 84, 85, 86, 96, 109, 110, 111, 112, 120, 121, 125, 129, 141, 152, 159, 160, 161, 162, 163, 177], employ prefetching [9, 28, 36, 37, 42, 44, 45, 47, 84, 113, 114, 115, 119, 122, 124, 128, 158], perform memory/cache compression [1, 7, 8, 39, 41, 43, 130, 131, 132, 133, 134, 151, 165, 168, 175], or perform better caching [67, 137, 138, 149, 150]. Our proposal is orthogonal to all of these approaches and can be applied in conjunction with them to achieve higher latency and energy benefits.

Experimental Studies of DRAM Chips. There are several studies that characterize various errors in DRAM. Many of these works observe how specific factors affect DRAM errors, analyzing the impact of temperature [48] and hard errors [57]. Other works have conducted studies of DRAM error rates in the field, studying failures across a large sample size [95, 106, 143, 155, 156, 157]. There are also works that have studied errors through controlled experiments, usually using FPGA-based DRAM testing infrastructures like SoftMC [52], to investigate errors due to retention time [52, 66, 68, 69, 70, 97, 98, 127, 136], disturbance from neighboring DRAM cells [62, 74, 75, 118], latency variation across/within DRAM chips [29, 30, 33, 87, 89], and supply voltage [33, 34]. None of these works extensively study latency variation across DRAM modules, which we characterize in our work.

9. Significance

Our work on AL-DRAM is the first to extensively characterize and exploit the large access latency variation that exists in modern DRAM devices. In this section, we discuss the novelty of AL-DRAM and its expected future impact on the community.

9.1. Novelty

We make the following major contributions in our HPCA 2015 paper [90]:

Addressing a Critical Real Problem, High DRAM Latency, with Low Cost. High DRAM latency is a critical bottleneck for overall system performance in a variety of modern computing systems [117, 123], especially in real large-scale server systems [63, 101]. Considering the significant difficulties in DRAM scaling [64, 117, 118, 123], the DRAM latency problem is getting worse in future systems due to process variation. Our HPCA 2015 work [90] leverages the heterogeneity created by DRAM process variation across DRAM chips and system operating conditions to mitigate the DRAM latency problem. We propose a practical mechanism, *Adaptive-Latency DRAM*, which mitigates DRAM latency with very modest hardware cost, and with *no changes* to the DRAM chip itself.

Large-Scale Latency Profiling of Modern DRAM Chips. Using our FPGA-based DRAM testing infrastructure [30, 33, 34, 52, 68, 69, 75, 87, 89, 90, 97, 127, 136], we profile 115 DRAM modules (920 DRAM chips in total) and show that there is significant timing variation between different DIMMs at different temperatures. We believe that our results are statistically significant to validate our hypothesis that the DRAM timing parameters strongly depend on the amount of cell charge. We provide a detailed characterization of each DIMM online at the SAFARI Research Group website [91]. Furthermore, we introduce our FPGA-based DRAM infrastructure and experimental methodology for DRAM profiling, which are carefully constructed to represent the worst-case conditions in power noise, bitline/wordline coupling, data patterns, and access patterns. Such information will hopefully be useful for future DRAM research.

Extensive *Real* System Evaluation of DRAM Latency. We evaluate our mechanism on a real system [10, 11] and show that our mechanism provides significant performance improvements. Reducing the timing parameters strips the excessive margin in the electrical charge stored within a DRAM cell. We show that the remaining margin is *enough* for DRAM to operate reliably. To verify the correctness of our experiments, we ran our workloads for 33 days nonstop, and examined their and the system's correctness with reduced timing parameters. Using the reduced timing parameters over the course of 33 days, our real system was able to execute 35 different workloads in both single-core and multi-core configurations while preserving correctness and being *error-free*. Note that these results do *not* absolutely guarantee that no errors can be introduced by reducing the timing parameters. However, we believe that we have demonstrated a proof-of-concept which shows that DRAM latency can be reduced at no impact on DRAM reliability. Ultimately, DRAM manufacturers can provide the reliable timing parameters for different operating conditions and modules.

9.2. Potential Long-Term Impact

Tolerating High DRAM Latency by Exploiting DRAM Intrinsic Characteristics. Today, there is a large latency cliff between the on-chip last level cache and off-chip DRAM, leading to a large performance fall-off when applications start missing in the last level cache. By enabling lower DRAM latency, our mechanism, Adaptive-Latency DRAM, smoothens this latency cliff without adding another layer into the memory hierarchy.

Applicability to Future Memory Devices. We show the benefits of the common-case timing optimization in modern DRAM devices by taking advantage of intrinsic characteristics of DRAM. Considering that most memory devices adopt a unified specification that is dictated by the worst-case operating condition, our approach that optimizes device latency for the common case can be applicable to other memory devices by leveraging the intrinsic characteristics of the technology they are built with. We believe there is significant potential for approaches that could reduce the latency of Phase Change Memory (PCM) [40, 80, 81, 82, 105, 135, 139, 140, 170, 172], STT-MRAM [79, 105], RRAM [169], and NAND flash memory [16, 17, 18, 19, 20, 21, 22, 22, 23, 24, 25, 26, 27, 102, 103, 104, 107].

New Research Opportunities. Adaptive-Latency DRAM creates new opportunities by enabling mechanisms that can leverage the heterogeneous latency offered by our mechanism. We describe a few of these briefly.

Optimizing the operating conditions for faster DRAM access: Adaptive-Latency DRAM provides different access latencies for different operating conditions. Future works can explore how the operating conditions themselves can be optimized, which can be used in conjunction with AL-DRAM to further improve the DRAM access latency. For instance, balancing DRAM accesses over multiple DRAM channels and ranks can potentially reduce the DRAM operating temperature, maximizing the benefits provided by AL-DRAM. At the system level, operating the system at a constant low temperature can enable the use of lower DRAM latencies more frequently.

Optimizing data placement to reduce overall DRAM access latency: We characterize the latency variation in different DIMMs due to process variation. Placing data based on this information and the latency criticality of data maximizes the benefits of lowering DRAM latency, by placing the data that is most sensitive to latency in the fastest DRAM chips (and, thus, providing lookups to the data with the fastest access latency).

Error correction mechanisms to further reduce DRAM latency. Error correction mechanisms allow us to lower DRAM latency even further, by correcting bit errors that occur when a small number of the DRAM operations end before the minimum charge is stored in the DRAM cell. Such mechanisms can rely on error correction to compensate for the reduced reliability of read and write operations at even lower latencies, leading to a further reduction in DRAM latency without errors. Future research that uses error correction to enable even lower latency DRAM is therefore promising as it opens a new set of trade-offs. Note that our recent work, DIVA-DRAM [89], explores this direction and finds very promising benefits.

Inspired by our characterization and proposed techniques, several recent works [30, 34, 51, 71, 89, 127] have explored many of these new research opportunities, by (1) analyzing different sources of latency and performance variation within DRAM chips, and (2) exploiting these sources of latency and performance variation to reduce access latency and/or energy consumption.

10. Conclusion

This paper summarizes our HPCA 2015 work on Adaptive-Latency DRAM (AL-DRAM), a simple and effective mechanism for dynamically tailoring the DRAM timing parameters for the current operating condition without introducing any errors. AL-DRAM takes advantage of the large latency margin available in the DRAM timing parameters for common-case operation, by dynamically the operating temperature of each DRAM module and employing timing constraints optimized for a particular module at the current temperature. AL-DRAM provides an average 14% improvement in overall system performance across a wide variety of memory-intensive applications run on a real multi-core system. We conclude that AL-DRAM is a simple and effective mechanism to reduce DRAM latency. We hope that our experimental exposure of the large margin present in the standard DRAM timing constraints will inspire other approaches to optimize DRAM chips, latencies, and parameters at low cost.

Acknowledgments

We thank Saugata Ghose for his dedicated effort in the preparation of this article. We thank the anonymous reviewers for their valuable feedback. We thank Uksong Kang, Jung-Bae Lee, and Joo Sun Choi from Samsung, and Michael Kozuch from Intel for their helpful comments. We acknowledge the support of our industrial partners: Facebook, IBM, Intel, Microsoft, Qualcomm, VMware, and Samsung. This research was partially supported by NSF (grants 0953246, 1212962, 1065112), the Semiconductor Research Corporation, and the Intel Science and Technology Center for Cloud Computing. Donghyuk Lee was supported in part by the John and Claire Bertucci Graduate Fellowship.

References

[1] B. Abali, H. Franke, D. Poff, R. Saccone, C. Schulz, L. Herger, and T. Smith, “Memory Expansion Technology (MXT): Software support and performance,” in *IBM JRD*, 2001.
[2] J.-H. Ahn *et al.*, “Adaptive Self Refresh Scheme for Battery Operated High-Density Mobile DRAM Applications,” in *ASSCC*, 2006.
[3] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, “Improving System Energy Efficiency with Memory Rank Subsetting,” in *ACM TACO*, 2012.
[4] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, “Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs,” in *IEEE CAL*, 2009.
[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing,” in *ISCA*, 2015.
[6] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture,” in *ISCA*, 2015.
[7] A. R. Alameldeen and D. A. Wood, “Adaptive Cache Compression for High-Performance Processors,” in *ISCA*, 2004.
[8] A. R. Alameldeen and D. A. Wood, “Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches,” Univ. of Wisconsin–Madison, Computer Sciences Dept., Tech. Rep. 1500, 2004.
[9] A. Alameldeen and D. Wood, “Interactions Between Compression and Prefetching in Chip Multiprocessors,” in *HPCA*, 2007.
[10] AMD, *AMD Opteron 4300 Series processors*, http://www.amd.com/en-us/products/server/4000/4300.
[11] AMD, “BKDG for AMD Family 16h Models 00h-0Fh Processors,” 2013.
[12] R. Ausavarungnirun, K. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, “Staged memory scheduling: achieving high performance and scalability in heterogeneous systems,” in *ISCA*, 2012.
[13] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, “Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance,” in *PACT*, 2015.
[14] A. Boroumand *et al.*, “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” in *ASPLOS*, 2018.
[15] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, “LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory,” in *IEEE CAL*, 2016.
[16] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives,” in *Proceedings of the IEEE*, 2017.
[17] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives,” arXiv:1706.08642 [cs.AR], 2017.
[18] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery,” arXiv:1711.11427 [cs.AR], 2017.
[19] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, “Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques,” in *HPCA*, 2017.
[20] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis,” in *DATE*, 2012.
[21] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling,” in *DATE*, 2013.
[22] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery,” in *DSN*, 2015.
[23] Y. Cai, Y. Luo, E. Haratsch, K. Mai, and O. Mutlu, “Data retention in MLC NAND flash memory: Characterization, optimization, and recovery,” in *HPCA*, 2015.
[24] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, “Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation,” in *ICCD*, 2013.
[25] Y. Cai, G. Yalcin, O. Mutlu, E. Haratsch, A. Cristal, O. Unsal, and K. Mai, “Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime,” in *ICCD*, 2012.
[26] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, “Error Analysis and Retention-Aware Error Management for NAND Flash Memory,” in *ITJ*, 2013.
[27] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, “Neighbor-cell Assisted Error Correction for MLC NAND Flash Memories,” in *SIGMETRICS*, 2014.
[28] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, “A Study of Integrated Prefetching and Caching Strategies,” in *SIGMETRICS*, 1995.
[29] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, “Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization,” in *DATE*, 2014.
[30] K. Chang, A. Kashyap, H. Hassan, S. Khan, K. Hsieh, D. Lee, S. Ghose, G. Pekhimenko, T. Li, and O. Mutlu, “Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization,” in *SIGMETRICS*, 2016.
[31] K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, “Improving DRAM performance by parallelizing refreshes with accesses,” in *HPCA*, 2014.
[32] K. Chang, P. J. Nair, S. Ghose, D. Lee, M. K. Qureshi, and O. Mutlu, “Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM,” in *HPCA*, 2016.
[33] K. K. Chang, “Understanding and Improving Latency of DRAM-Based Memory Systems,” Ph.D. dissertation, Carnegie Mellon University, 2017.
[34] K. K. Chang, A. G. Yaglikci, A. Agrawal, N. Chatterjee, S. Ghose, A. Kashyap, H. Hassan, D. Lee, M. O’Connor, and O. Mutlu, “Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms,” in *SIGMETRICS*, 2017.
[35] N. Chatterjee, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi, “Staged Reads: Mitigating the Impact of DRAM Writes on DRAM Reads,” in *HPCA*, 2012.
[36] R. Cooksey, S. Jourdan, and D. Grunwald, “A Stateless, Content-directed Data Prefetching Mechanism,” in *ASPLOS*, 2002.
[37] F. Dahlgren, M. Dubois, and P. Stenström, “Sequential Hardware Prefetching in Shared-Memory Multiprocessors,” in *IEEE TPDS*, 1995.
[38] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, “Application-to-core mapping policies to reduce memory system interference in multi-core

systems," in *HPCA*, 2013.
[39] R. de Castro, A. Lago, and M. Silva, "Adaptive compressed caching: design and implementation," in *SBAC-PAD*, 2003.
[40] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *DAC*, 2009.
[41] F. Douglis, "The Compression Cache: Using On-line Compression to Extend Physical Memory," in *Winter USENIX Conference*, 1993.
[42] J. Dundas and T. Mudge, "Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss," in *ICS*, 1997.
[43] J. Dusser, T. Piquet, and A. Seznec, "Zero-content Augmented Caches," in *ICS*, 2009.
[44] E. Ebrahimi, O. Mutlu, and Y. Patt, "Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems," in *HPCA*, 2009.
[45] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware Shared Resource Management for Multi-core Systems," in *ISCA*, 2011.
[46] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel application memory scheduling," in *MICRO*, 2011.
[47] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated Control of Multiple Prefetchers in Multi-core Systems," in *MICRO*, 2009.
[48] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature Management in Data Centers: Why Some (Might) Like It Hot," in *SIGMETRICS*, 2012.
[49] S. Ghose, H. Lee, and J. F. Martínez, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.
[50] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WoNDP*, 2014.
[51] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[52] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[53] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima, "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," in *IEEE Micro*, 1990.
[54] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
[55] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[56] I. Hur and C. Lin, "Adaptive History-Based Memory Schedulers," in *MICRO*, 2004.
[57] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," in *ASPLOS*, 2012.
[58] Intel Corp., "Intel Extreme Memory Profile (Intel XMP) DDR3 Technology," http://www.intel.com/content/www/us/en/chipsets/extreme-memory-profile-ddr3-technology-paper.html, 2009.
[59] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *ISCA*, 2008.
[60] JEDEC, *Standard No. 79-3F. DDR3 SDRAM Specification*, Jul. 2012.
[61] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting Core-Criticality for Enhanced GPU Performance," in *SIGMETRICS*, 2016.
[62] M. Jung, C. C. Rheinländer, C. Weis, and N. Wehn, "Reverse Engineering of DRAMs: Row Hammer with Crosshair," in *MEMSYS*, 2016.
[63] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.
[64] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. Choi, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[65] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era," in *MICRO*, 2011.
[66] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[67] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutlu, and D. A. Jimenez, "Improving Cache Performance by Exploiting Read-Write Disparity," in *HPCA*, 2014.
[68] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[69] S. Khan, D. Lee, C. Wilkerson, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," in *DSN*, 2016.
[70] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," in *IEEE CAL*, 2016.
[71] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[72] J. S. Kim, D. Senol, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.
[73] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," in *IEEE CAL*, 2015.
[74] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon University, 2015.
[75] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[76] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA*, 2010.
[77] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[78] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[79] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.
[80] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," in *IEEE Micro*, 2010.
[81] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory As a Scalable DRAM Alternative," in *ISCA*, 2009.
[82] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," in *CACM*, 2010.
[83] C. J. Lee, E. Ebrahimi, V. Narasiman, O. Mutlu, and Y. N. Patt, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," Univ. of Texas at Austin, High Performance Systems Group, Tech. Rep. TR-HPS-2010-002, 2010.
[84] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-Aware DRAM Controllers," in *MICRO*, 2008.
[85] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-Aware Memory Controllers," in *IEEE TC*, 2011.
[86] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, "Improving Memory Bank-level Parallelism in the Presence of Prefetching," in *MICRO*, 2009.
[87] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.
[88] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," in *ACM TACO*, 2016.
[89] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[90] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in *HPCA*, 2015.
[91] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," http://www.ece.cmu.edu/~safari/tools/aldram-hpca2015-fulldata.html.
[92] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in *HPCA*, 2013.
[93] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[94] J. Lee, K. Kim, Y. Shin, K. Lee, J. Kim, D. Kim, J. Park, and J. Lee, "Simultaneously Formed Storage Node Contact and Metal Contact Cell (SSMC) for 1Gb DRAM and Beyond," in *IEDM*, 1996.
[95] X. Li, M. C. Huang, K. Shen, and L. Chu, "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility," in *USENIX ATC*, 2010.
[96] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-Based Hybrid Memory Management," in *CLUSTER*, 2016.
[97] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[98] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[99] S. Liu, B. Leung, A. Neckar, S. Memik, G. Memik, and N. Hardavellas, "Hardware/software techniques for DRAM thermal management," in *HPCA*, 2011.
[100] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent Data Structures for Near-Memory Computing," in *SPAA*, 2017.
[101] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ISCA*, 2015.
[102] Y. Lu, J. Shu, J. Guo, S. Li, and O. Mutlu, "High-Performance and Lightweight Transaction Support in Flash-Based SSDs," in *IEEE TC*, 2015.
[103] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *MSST*, 2015.
[104] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," in *JSAC*, 2016.
[105] J. Meza, J. Li, and O. Mutlu, "A Case for Small Row Buffers in Non-Volatile Main Memories," in *ICCD, Poster Session*, 2012.
[106] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.
[107] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *SIGMETRICS*, 2015.

[108] Micron, "RLDRAM 2 and 3 Specifications," http://www.micron.com/products/dram/rldram-memory.
[109] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-core Systems," in *USENIX Security*, 2007.
[110] T. Moscibroda and O. Mutlu, "Distributed Order Scheduling and Its Application to Multi-core Dram Controllers," in *PODC*, 2008.
[111] J. Mukundan and J. F. Martínez, "MORSE: Multi-Objective Reconfigurable Self-Optimizing Memory Scheduler," in *HPCA*, 2012.
[112] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *MICRO*, 2011.
[113] O. Mutlu, H. Kim, and Y. Patt, "Address-value delta (AVD) prediction: increasing the effectiveness of runahead execution by exploiting regular memory allocation patterns," in *MICRO*, 2005.
[114] O. Mutlu, H. Kim, and Y. Patt, "Techniques for efficient processing in runahead execution engines," in *ISCA*, 2005.
[115] O. Mutlu, J. Stark, C. Wilkerson, and Y. Patt, "Runahead execution: an alternative to very large instruction windows for out-of-order processors," in *HPCA*, 2003.
[116] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *MemCon*, 2013.
[117] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[118] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[119] O. Mutlu, H. Kim, and Y. N. Patt, "Efficient Runahead Execution: Power-efficient Memory Latency Tolerance," in *IEEE Micro*, 2006.
[120] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[121] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[122] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: An effective alternative to large instruction windows," in *IEEE Micro*, 2003.
[123] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," in *SUPERFRI*, 2014.
[124] K. Nesbit, A. Dhodapkar, and J. Smith, "AC/DC: an adaptive data cache prefetcher," in *PACT*, 2004.
[125] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair Queuing Memory Systems," in *MICRO*, 2006.
[126] NVIDIA Corp., "Extreme DDR3 Performance with SLI-Ready Memory," http://www.nvidia.com/docs/IO/52280/NVIDIA_EPP2_TB.pdf, 2008.
[127] M. Patel, J. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[128] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed Prefetching and Caching," in *SOSP*, 1995.
[129] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.
[130] G. Pekhimenko, E. Bolotin, M. O'Connor, O. Mutlu, T. C. Mowry, and S. W. Keckler, "Toggle-Aware Compression for GPUs," in *IEEE CAL*, 2015.
[131] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "Toggle-Aware Bandwidth Compression for GPUs," in *HPCA*, 2016.
[132] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. P. Gibbons, M. A. Kozuch, and T. C. Mowry, "Exploiting Compressed Block Size as an Indicator of Future Reuse," in *HPCA*, 2015.
[133] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly Compressed Pages: A Low-complexity, Low-latency Main Memory Compression Framework," in *MICRO*, 2013.
[134] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches," in *PACT*, 2012.
[135] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *MICRO*, 2009.
[136] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[137] M. K. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer, "Adaptive Insertion Policies for High Performance Caching," in *ISCA*, 2007.
[138] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A Case for MLP-Aware Cache Replacement," in *ISCA*, 2006.
[139] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[140] S. Raoux *et al.*, "Phase-change random access memory: A scalable technology," in *IBM Journal of Research and Development*, 2008.
[141] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA*, 2000.
[142] Y. Sato *et al.*, "Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM," in *VLSIC*, 1998.
[143] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009.
[144] V. Seshadri, A. Bhowmick, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry, "The Dirty-Block Index," in *ISCA*, 2014.
[145] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. Kozuch, O. Mutlu, P. Gibbons, and T. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," in *IEEE CAL*, 2015.
[146] V. Seshadri *et al.*, "RowClone: Fast and Energy-efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[147] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[148] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," in *MICRO*, 2015.
[149] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," in *PACT*, 2012.
[150] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks," in *ACM TACO*, 2015.
[151] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring Unconventional Benefits from Memory Compression," in *HPCA*, 2014.
[152] J. Shao and B. T. Davis, "A Burst Scheduling Access Reordering Mechanism," in *HPCA*, 2007.
[153] W. Shin, J. Yang, J. Choi, and L.-S. Kim, "NUAT: A Non-Uniform Access Time Memory Controller," in *HPCA*, 2014.
[154] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.
[155] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory Errors in Modern Systems: The Good, the Bad, and the Ugly," in *ASPLOS*, 2015.
[156] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *SC*, 2012.
[157] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," in *SC*, 2013.
[158] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," in *HPCA*, 2007.
[159] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "The Blacklisting Memory Scheduler: Achieving high performance and fairness at low cost," in *ICCD*, 2014.
[160] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," in *TPDS*, 2016.
[161] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," in *MICRO*, 2015.
[162] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," in *HPCA*, 2013.
[163] H. Usui, L. Subramanian, K. Chang, and O. Mutlu, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," in *ACM TACO*, 2016.
[164] R. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.
[165] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," in *ISCA*, 2015.
[166] M.-J. Wang, R.-L. Jiang, J.-W. Hsia, C.-H. Wang, and J.-E. Chen, "Guardband determination for the detection of off-state and junction leakages in DRAM testing," in *Asian Test Symposium*, 2001.
[167] F. Ware and C. Hampel, "Improving Power and Data Efficiency with Threaded Memory Modules," in *ICCD*, 2006.
[168] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis, "The Case for Compressed Caching in Virtual Memory Systems," in *ATEC*, 1999.
[169] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. Chen, and M.-J. Tsai, "Metal Oxide RRAM," in *Proceedings of the IEEE*, 2012.
[170] H.-S. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," in *Proceedings of the IEEE*, 2010.
[171] D. Yaney, C. Y. Lu, R. Kohler, M. J. Kelly, and J. Nelson, "A meta-stable leakage phenomenon in DRAM charge storage - Variable hold time," in *IEDM*, 1987.
[172] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories," in *ACM TACO*, 2014.
[173] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented Programmable Processing in Memory," in *HPCA*, 2014.
[174] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation," in *ISCA*, 2014.
[175] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," in *ASPLOS*, 2000.
[176] Z. Zhang, Z. Zhu, and X. Zhang, "Cached DRAM for ILP Processor Memory Access Latency Reduction," in *IEEE Micro*, 2001.
[177] J. Zhao, O. Mutlu, and Y. Xie, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," in *MICRO*, 2014.
[178] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *MICRO*, 2008.

Flexible-Latency DRAM: Understanding and Exploiting Latency Variation in Modern DRAM Chips

Kevin K. Chang[1,2] Abhijith Kashyap[3,2] Hasan Hassan[4,2,5] Saugata Ghose[2]
Kevin Hsieh[2] Donghyuk Lee[6,2] Tianshi Li[2,7] Gennady Pekhimenko[8,2]
Samira Khan[9,2] Onur Mutlu[4,2]

[1]*Facebook* [2]*Carnegie Mellon University* [3]*NVIDIA* [4]*ETH Zürich*
[5]*TOBB University of Economics & Technology* [6]*NVIDIA Research*
[7]*Peking University* [8]*University of Toronto* [9]*University of Virginia*

This article summarizes key results of our work on experimental characterization and analysis of latency variation and latency-reliability trade-offs in modern DRAM chips, which was published in SIGMETRICS 2016 [24], and examines the work's significance and future potential. Our work is motivated to reduce the long DRAM latency, which is a critical performance bottleneck in current systems. DRAM access latency is defined by three fundamental operations that take place within the DRAM cell array: (i) activation *of a memory row, which opens the row to perform accesses;* (ii) precharge*, which prepares the cell array for the next memory access; and* (iii) restoration *of the row, which restores the values of cells in the row that were destroyed due to activation. There is significant latency variation for each of these operations across the cells of a single DRAM chip due to irregularity in the manufacturing process. As a result, some cells are* inherently *faster to access, while others are inherently slower. Unfortunately, existing systems do not exploit this variation.*

The goal of this work is to (i) *experimentally characterize and understand the latency variation across cells within a DRAM chip for these three fundamental DRAM operations, and* (ii) *develop new mechanisms that exploit our understanding of the latency variation to reliably improve performance. To this end, we comprehensively characterize 240 DRAM chips from three major vendors, and make six major new observations about latency variation within DRAM. Notably, we find that* (i) *there is large latency variation across the cells for each of the three operations;* (ii) *variation characteristics exhibit significant spatial locality: slower cells are clustered in certain regions of a DRAM chip; and* (iii) *the three fundamental operations exhibit different reliability characteristics when the latency of each operation is reduced.*

Based on our observations, we propose Flexible-LatencY DRAM (FLY-DRAM), a mechanism that exploits latency variation across DRAM cells within a DRAM chip to improve system performance. The key idea of FLY-DRAM is to exploit the spatial locality of slower cells within DRAM, and access the faster DRAM regions with reduced latencies for the fundamental operations. Our evaluations show that FLY-DRAM improves the performance of a wide range of applications by 13.3%, 17.6%, and 19.5%, on average, for each of the three different vendors' real DRAM chips, in a simulated 8-core system.

We have open sourced the data from our research online. We hope the characterization and analysis we provide opens up new research directions for both researchers and practitioners in computer architecture and systems.

1. Introduction

Over the past few decades, the long latency of memory has been a critical bottleneck in system performance. Increasing core counts, the emergence of more data-intensive and latency-critical applications, and increasingly limited bandwidth in the memory system are together leading to higher memory latency. Thus, low-latency memory operation is now even more important to improving overall system performance [30, 55, 93, 101, 102, 105, 143].

The latency of a memory request is predominantly defined by the timings of three fundamental operations: (1) *activation*, which "opens" a row of DRAM cells to access stored data, (2) *precharge*, which "closes" an activated row, and (3) *restoration*, which restores the charge level of each DRAM cell in a row to prevent data loss.[1] The latencies of these three DRAM operations, as defined by vendor specifications, have *not* improved significantly in the past 18 years, as depicted in Figure 1. This is especially true when we compare latency improvements to the capacity (128×) and bandwidth improvements (20×) [23] commodity DRAM chips experienced in the past 18 years. In fact, the activation and precharge latencies *increased* from 2013 to 2015, when DDR DRAM transitioned from the third generation (12.5ns for DDR3-1600J [51]) to the fourth generation (14.06ns for DDR4-2133P [53]). As the latencies specified by vendors have not reduced over time, the memory latency remains as a critical system performance bottleneck in many modern applications, such as big data workloads [28] and Google's warehouse-scale workloads [55].

[1]We refer the reader to our prior works [22, 24, 25, 26, 43, 44, 61, 64, 65, 66, 67, 68, 76, 77, 79, 81, 82, 86, 87, 110, 127, 128] for a detailed background on DRAM.

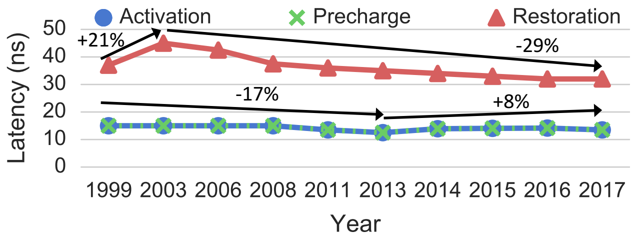


Figure 1: DRAM latency trends over time [50,51,53,97]. Adapted from [24].

2. Motivation

In this work, we observe that the three fundamental DRAM operations can *actually* complete with a much lower latency for many DRAM cells than the vendor specification, because *there is inherent latency variation present across the DRAM cells within a DRAM chip.* This is a result of manufacturing process variation, which causes the *sizes* and *strengths* of cells to be different, thus making some cells faster and other cells slower to be accessed reliably [85]. The speed gap between the fastest and slowest DRAM cells is getting worse [20, 107], as the technology node continues to scale down to sub-20nm feature sizes. Unfortunately, instead of optimizing the latency specifications for the common case, DRAM vendors use a single set of standard access latencies, called timing parameters, which provide reliable operation guarantees for the *worst case* (i.e., the slowest cells), to maximize manufacturing yield.

We experimentally demonstrate that significant latency variation is present across DRAM cells in 240 DDR3 DRAM chips from three major vendors, and that a large fraction of cells can be read reliably even if the activation/restoration/-precharge latencies are reduced significantly. By repeatedly testing these DRAM chips, we observe that access latency variation exhibits spatial locality within DRAM — slower cells cluster in certain regions of a DRAM chip. In Section 4, we propose a new mechanism, called *FLY-DRAM*, which exploits the lower latencies of DRAM regions with faster cells by introducing heterogeneous timing parameters into the memory controller. By analyzing and exploiting the latency variation that exists in DRAM cells, we can greatly reduce the DRAM access latency.

We discuss our major experimental observations in Section 3. For a detailed discussion on all of our observations, we refer the reader to our SIGMETRICS 2016 paper [24].

3. Latency Variation Analysis

To capture the effect of latency variation in modern DDR3 DRAM chips, we tune the timing parameters that control the amount of time taken for each of the fundamental DRAM operations. We developed an FPGA-based DRAM testing platform [43] that allows us to precisely control the timing parameter values and the tested DRAM location (i.e., banks, rows, and columns). A photo of the platform is shown in Figure 2. Using this platform, we characterize latency variation on a total of 30 DDR3 DRAM modules (or *DIMMs*), comprising 240 DRAM chips from three major vendors. Each chip has a 1Gb density. Thus, each of our DIMMs has a 1GB capacity. Table 1 lists the relevant information about the tested DRAM modules. Unless otherwise specified, we test modules at an ambient temperature of 20±1℃. For results using higher temperatures, we refer the reader to Section 4.5 of our SIGMETRICS 2016 paper [24].

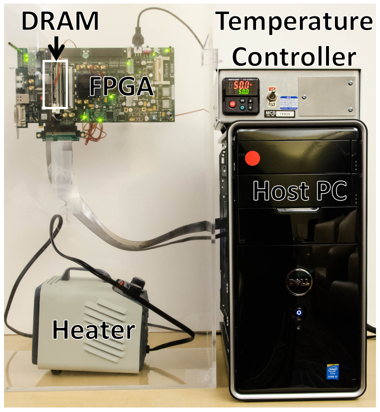


Figure 2: FPGA-based DRAM testing infrastructure. Reproduced from [24].

Vendor	Total Number of Chips	Timing (ns) (tRCD/tRP/tRAS)	Assembly Year
A (8 DIMMs)	64	13.125/13.125/35-36	2012-13
B (9 DIMMs)	72	13.75/13.75/35	2011-12
C (13 DIMMs)	104	13.75/13.75/34-36	2011-12

Table 1: Main properties of the tested DIMMs. Reproduced from [24].

In this section, we present a short summary of our key results on varying the activation, precharge, and restoration latencies, which are controlled by the tRCD, tRP, and tRAS timing parameters, respectively. For more details on the experimental results and observations, see Sections 4–6 of our SIGMETRICS 2016 paper [24].

3.1. Behavior of Timing Errors

We analyze the variation in the latencies of activation, precharge, and restoration by operating DRAM at multiple reduced latencies for each of these operations. Faster cells do *not* get affected by the reduced timings, and can be accessed reliably without any errors; however, slower cells *cannot* be read reliably with reduced latencies for the three operations, leading to bit flips. In this work, we define a *timing error* as a bit flip in a cell that occurs due to a reduced-latency access, and characterize timing errors incurred by the three DRAM operations.

Our experiments yield several **new observations** on the behavior of timing errors. When we reduce the three latencies, we observe that each latency exhibits a different level of impact on the inherently-slower cells. Lowering the activation latency (tRCD) affects *only* the cells (data) read in the first accessed cache line, but not the subsequently read cache lines from the same row. This is mainly due to two reasons. First, a READ command accesses only its corresponding sense amplifiers, without accessing the other columns. Hence, a READ's effect is isolated to its target cache line. Second, by the time a subsequent READ is issued to the same activated row, a sufficient amount of time has already passed for the row buffer to fully sense and latch in the row data. In contrast, lowering the restoration (tRAS) or precharge (tRP) latencies affects *all* cells in the activated row (see Section 5 of our SIGMETRICS 2016 paper [24] for a detailed explanation). Lowering these latencies affects the entire row because these commands operate at the row level, and they directly affect the restoration and sensing of *all* cells in the row.

We also find that the number of timing errors introduced is very sensitive to reducing the activation or precharge latency, but not that sensitive to reducing the restoration latency. We conclude that different levels of mitigation are required to address the timing errors that result from lowering each of the different DRAM operation latencies, and that reducing restoration latency to the lowest levels allowed by our infrastructure does *not* introduce timing errors in our experiments (see Section 6 in our SIGMETRICS 2016 paper [24]).

3.2. Timing Error Distribution

We briefly present the distribution of activation and precharge errors collected from all of the tests conducted on every DIMM. Figure 3 shows the box plots of the *bit error rate* (BER) observed on every DIMM as activation latency (tRCD) varies. The BER is defined as the fraction of bits with errors due to reducing tRCD in the total population of tested bits. In other words, the BER represents the fraction of cells that cannot operate reliably under the specified shortened latency. The box plot shows the maximum and minimum BER of all of our tested DIMMs as whiskers, and the box shows the quartiles of the distribution. In addition, we show *all* observation points for each specific tRCD/tRP value by overlaying them on top of their corresponding box plot. Each point shows a BER collected from one round of tests on one DIMM with a specific data pattern and tRCD value. For box plots showing the BER distribution when the precharge latency (tRP) is reduced, see Figure 12 in the original paper [24]. We make two observations from the BER distributions when reducing tRCD or tRP.

First, at tRCD or tRP values of 12.5ns and 10ns, we observe *no timing errors on any DIMM* due to reduced activation or precharge latency. This shows that the tRCD/tRP latencies of the slowest cells in our tested DIMMs likely fall between 7.5 and 10ns, which are lower than the value provided in the vendor specifications (13.125ns). DRAM vendors use the extra latency as a *guardband* to provide additional protection against process variation.

Second, there exists a large BER variation among DIMMs at tRCD of 7.5ns, and the BER variation becomes smaller as the tRCD or tRP value decreases. The number of fast cells that can operate at tRCD=7.5ns or tRP=7.5ns varies significantly across different DIMMs. These results demonstrate that there exists significant latency variation among and within DIMMs, as not all of the cells exhibit timing errors at 7.5ns.

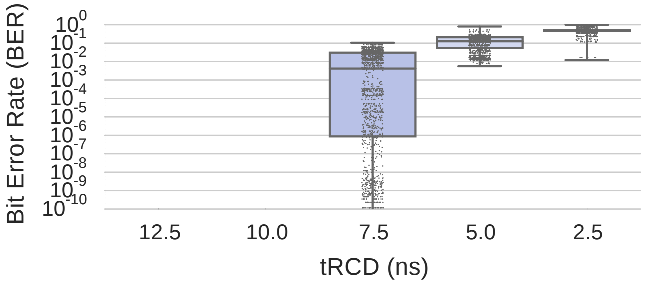
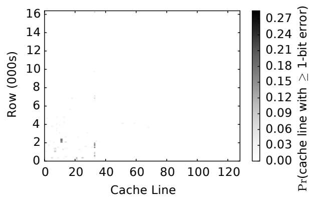


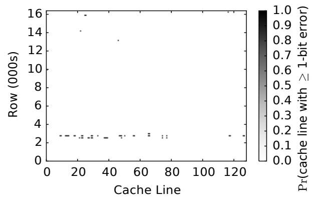
Figure 3: Bit error rate of all DIMMs with reduced tRCD. Reproduced from [24].

3.3. Spatial Locality of Timing Errors

In this section, we investigate the location and distribution of timing errors *within* a DIMM when the activation or precharge latencies are reduced. Figure 4 shows the probability of every cache line (64B) in one bank of a specific DIMM observing at least 1 bit of error with reduced activation latency (Figure 4a) or precharge latency (Figure 4b). See [24] for additional results. The x-axis and y-axis indicate the cache line number and row number (in thousands), respectively. In our tested DIMMs, a row size is 8KB, comprising 128 cache lines.



(a) Activation latency (tRCD) at 7.5ns (43% reduction).



(b) Precharge latency (tRP) at 7.5ns (43% reduction).

Figure 4: Probability of observing timing errors in one DIMM. Adapted from [24].

The main observation is that timing errors due to reducing activation or precharge latency are *not* distributed uniformly across locations within this DIMM. Timing errors tend to cluster at certain regions of cache lines. For the remaining cache lines, we observe that they do not exhibit timing errors due to reduced latency throughout the experiments. We observe similar characteristics in other DIMMs — timing errors concentrate within certain spatial regions of memory.

We hypothesize that the cause of the spatial locality of timing errors is due to the locality of variation in the fabrication process during manufacturing. Certain cache line locations can end up with less robust components, such as weaker sense amplifiers, weaker cells, or higher resistance bitlines.

3.4. Other Characterization Results

We briefly summarize our other observations on the effects of reducing timing parameters. First, we analyze the number of timing errors that occur when DRAM access latencies are reduced, and experimentally demonstrate that most of the erroneous cache lines have a single-bit error, with only a small fraction of cache lines experiencing more than one bit flip (see Section 4.7 of our SIGMETRICS 2016 paper [24]). We conclude, therefore, that using simple error-correcting codes (ECC) can correct *most* of these errors, thereby enabling lower latency for many inherently slower cells (see Section 4.8 of our SIGMETRICS 2016 paper [24] for a detailed analysis of ECC).

Second, we find that the stored data pattern in cells affects access latency variation. Certain patterns lead to more timing errors than others. For example, the bit value `1` can be read significantly more reliably at a reduced access latency than the bit value `0` (see Section 4.4 of our SIGMETRICS 2016 paper [24]). This observation is similar to the data pattern dependence observation made for retention times of DRAM cells [57, 58, 59, 60, 86, 110].

Third, we find no clear correlation between temperature and variation in cell access latency. We believe that it is not essential for latency reduction techniques that exploit such variation to be aware of the operating temperature (Section 4.5 in [24]).

4. Exploiting Latency Variation

Based on our extensive experimental characterization and new observations on latency-reliability trade-offs in modern DRAM chips, we propose a new hardware mechanism, called *Flexible-LatencY DRAM* (FLY-DRAM), to reduce DRAM latency for better system performance. FLY-DRAM exploits the key observation that *(i)* different cells can operate reliably at different DRAM latencies, and *(ii)* there is a strong correlation between the location of a cell and the lowest latency that the cell can operate reliably at. The key idea of FLY-DRAM is to *(i)* categorize the DRAM cells into fast and slow regions, *(ii)* expose this categorization to the memory controller, and *(iii)* reduce overall DRAM latency by accessing the fast regions with a lower latency.

The FLY-DRAM memory controller *(i)* loads the latency profiling results [24] into on-chip SRAM at system boot time, *(ii)* looks up the profiled latency for each memory request based on its memory address, and *(iii)* applies the corresponding latency to the request. By reducing the values of tRCD, tRAS, and tRP for some memory requests, FLY-DRAM improves overall system performance. In addition, we also propose an OS page allocator design that exploits the latency variation in DRAM to improve system performance (see Section 7.2 of our paper [24]).

There are two key design challenges of FLY-DRAM. The first challenge is determining the fraction of fast cells within a DRAM chip and the innate access latency of the fast cells. Since DRAM vendors have detailed information on their DRAM chips from the DRAM post-production tests, DRAM vendors can embed the latency profiling results in the Serial Presence Detect (SPD) circuitry (a ROM present in each DIMM) [52]. The memory controller can read the profiling results from the SPD circuitry during DRAM initialization, and apply the correct latency for each DRAM region.

The second design challenge is limiting the storage overhead of the latency profiling results. Recording the shortest latency for each cache line can incur a large storage overhead. Fortunately, the storage overhead can be reduced based on a new observation of ours. As discussed in Section 3.3, timing errors typically concentrate at certain DRAM regions. Therefore, FLY-DRAM records the shortest latency at the granularity of DRAM regions (i.e., a group of adjacent cache lines, rows, or banks). One can imagine using more sophisticated structures, such as Bloom Filters [6], to provide finer-grained latency information within a reasonable storage overhead, as shown in prior work on variable DRAM refresh intervals [87, 115].

4.1. Summary of Results

We evaluate FLY-DRAM on on an 8-core system with a wide variety of workloads by using Ramulator [64, 120], a cycle-level open-source DRAM simulator developed by our research group. Table 2 summarizes the configuration of our evaluated system. We use the standard DDR3-1333H timing parameters [51] as our baseline.

Processor	8 cores, 3.3 GHz, OoO 128-entry window
LLC	8 MB shared, 8-way set associative
DRAM	DDR3-1333H [51], open-row policy [66, 67, 118], 2 channels, 1 rank per channel, 8 banks per rank, Baseline: tRCD/tCL/tRP = 13.125ns, tRAS = 36ns

Table 2: Evaluated system configuration. Adapted from [24].

Figure 5 illustrates the system performance improvement of FLY-DRAM over the baseline (DDR3-1333) for 40 workloads. The x-axis indicates each of the evaluated DRAM configurations. $\mathtt{D}_A^2$, $\mathtt{D}_B^7$, and $\mathtt{D}_C^2$ correspond to latency profiles collected from three real DIMMs. Our SIGMETRICS 2016 paper [24] describes these real-DRAM profiles in more detail.

For these three DIMMs, FLY-DRAM improves system performance significantly, by 17.6%, 13.3%, and 19.5% on average across all 40 workloads. This is because FLY-DRAM reduces the latency of tRCD, tRP, and tRAS by 42.8%, 42.8%, and 25%, respectively, for a large fraction of cache lines. In particular, DIMM $\mathtt{D}_C^2$, which has a 99% of cells that operate reliably at low tRCD and tRP, performs within 1% of the upper-bound performance (19.7% on average), which is obtained by operating all DRAM cells at low tRCD and tRP. We conclude that FLY-DRAM is an effective mechanism to improve system performance by exploiting the widespread latency variation present across DRAM cells.

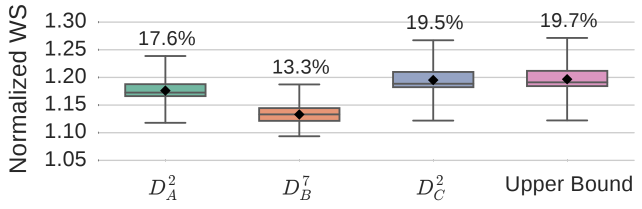


Figure 5: System performance improvement of FLY-DRAM for various DIMMs. Reproduced from [24].

As we show in our SIGMETRICS 2016 paper [24], FLY-DRAM can take advantage of an intelligent DRAM-aware page allocator that allocates frequently used and latency-critical pages in fast DRAM regions. We leave the detailed design and evaluation of such an allocator to future work.

5. Related Work

To our knowledge, this is the first work to *(i)* provide a detailed experimental characterization and analysis of latency variation for three major DRAM operations (tRCD, tRP, and tRAS) *across different cells within a DRAM chip*, *(ii)* demonstrate that a reduction in latency for each of these fundamental operations has a different impact on different cells, *(iii)* show that access latency variation exhibits spatial locality, *(iv)* demonstrate that the error rate due to reduced latencies is correlated with the stored data pattern but *not* conclusively correlated with temperature, and *(v)* propose mechanisms that take advantage of variation *within a DRAM chip* to improve system performance. We discuss the most closely related works here.

5.1. DRAM Latency Variation

Adaptive-Latency DRAM (AL-DRAM) also characterizes and exploits DRAM latency variation, but does so at a much coarser granularity [79]. This work experimentally characterizes latency variation *across* different DRAM chips under *different* operating temperatures. AL-DRAM sets a uniform operation latency for the *entire* DIMM. In contrast, our work characterizes latency variation *within each chip*, at the granularity of individual DRAM cells. Our mechanism, FLY-DRAM, can be combined with AL-DRAM to further improve performance.[2]

A recent work by Lee et al. [76] also observes latency variation within DRAM chips. The work analyzes the variation that is due to the circuit design of DRAM components, which it calls *design-induced variation*. Furthermore, it proposes a new profiling technique to identify the lowest DRAM latency without introducing errors. In this work, we provide the *first* detailed experimental characterization and analysis of the general latency variation phenomenon within real DRAM chips. Our analysis is broad and is not limited to design-induced variation. Our proposal of exploiting latency variation, FLY-DRAM can employ Lee et al.'s new profiling mechanism [76] to identify additional latency variation regions for reducing access latency.

Chandrasekar et al. study the potential of reducing some DRAM timing parameters [21]. Similar to AL-DRAM, this work observes and characterizes latency variation *across* DIMMs, whereas our work studies variation across cells *within a DRAM chip*.

5.2. DRAM Error Studies

There are several studies that characterize various errors in DRAM. Many of these works observe how specific factors affect DRAM errors, analyzing the impact of temperature [32, 79] and hard errors [48]. Other works have conducted studies of DRAM error rates in the field, studying failures across a large sample size [84, 95, 123, 132, 133]. There are also works that have studied errors through controlled experiments, investigating errors due to retention time [43, 57, 58, 59, 60, 86, 110, 115], disturbance from neighboring DRAM cells [65, 101], latency variation across/within DRAM chips [21, 76, 78, 79], and supply voltage [26]. None of these works study errors due to latency variation across the cells *within* a DRAM chip, which we extensively characterize in our work.

5.3. DRAM Latency Reduction

Several types of commodity DRAM (Micron's RL-DRAM [98] and Fujitsu's FCRAM [122]) provide low latency at the cost of high area overhead [68, 81]. Many prior works (e.g., [22, 25, 45, 68, 81, 88, 101, 102, 106, 125, 127, 128, 131, 150]) propose various architectural changes *within* DRAM chips to reduce latency. In contrast, FLY-DRAM does not require any changes to a DRAM chip. Other works [44, 75, 124, 129, 130] reduce DRAM latency by changing the memory controller, and FLY-DRAM is complementary to them.

5.4. ECC DRAM

Many memory systems incorporate ECC DIMMs, which store information used to correct data during a read operation. Prior work (e.g., [39, 54, 60, 63, 83, 140, 142, 145, 146]) proposes more flexible or more powerful ECC schemes for DRAM. While these ECC mechanisms are designed to protect against faults using standard DRAM timings, we show that they also have the potential to correct timing errors that occur due to reduced DRAM latencies. A recent work by Lee et al. [76] exploits this observation and uses ECC to correct errors that occur due to reduced latency in DRAM.

5.5. Other Latency Reduction Mechanisms

Various prior works [1, 2, 3, 5, 7, 8, 25, 31, 33, 34, 35, 36, 38, 40, 42, 46, 47, 56, 62, 69, 92, 109, 111, 112, 114, 125, 126, 128, 129, 134, 139, 149] examine processing in memory to reduce DRAM latency. Other prior works propose memory scheduling techniques, [4, 37, 49, 66, 67, 74, 99, 100, 103, 104, 135, 136, 137, 138, 141],

[2]A description of the AL-DRAM work and its impact is provided in a companion article in the very same issue of this journal [80].

which generally reduce latency to access DRAM. Our analyses and techniques can be combined with these works to enable further low-latency operation.

6. Significance

Our SIGMETRICS 2016 paper [24] presents a new experimental characterization and analysis of latency variation in modern DRAM chips. In this section, we describe the potential impact that our study can have on the research community and industry.

6.1. Potential Research Impact

Our paper develops a new way of using manufactured DRAM chips: accessing different regions of memory using each region's inherent latency instead of a homogeneous fixed standard latency for all regions of memory. We show that *(i)* there is significant latency variation within a DRAM chip, and *(ii)* it is possible to exploit the variation with simple mechanisms. We believe one key impact of our paper is demonstrating the effectiveness of designing memory optimizations based on real-world characterization. We expect that this same principle can be used to craft new memory architectures for both existing and future memory technologies, such as SRAM, PCM [71, 72, 73, 116, 117, 147, 148], STT-MRAM [27, 41, 70], or RRAM [144].

Our work exposes several opportunities for both operating systems and hardware to further optimize for memory access latency. We have open-sourced our raw characterization data, to allow other researchers to further analyze and build off of our work [120]. Other researchers can find many other ways to take advantage of the insights and the characterization data we provide. Our FLY-DRAM implementation is also available as part of the open-source release of Ramulator [64, 119].

ECC to Reduce Latency. In our paper, we analyze the distribution of timing errors (due to reduced latency) at the granularity of data beats, as conventional error-correcting codes (ECC) work at the same granularity. Our data shows that many of the erroneous data beats experience only a single-bit error, while the majority of the data beats contain no errors. Therefore, this creates an opportunity for applying ECC to correct timing errors. We also envision an opportunity for applying ECC to only certain regions of DRAM, which takes advantage of the spatial locality of timing errors exposed by our work. Lee et al. [76] provide examples of the use of ECC to reduce latency further, but they apply ECC globally to the entire DRAM chip. We believe a significant opportunity exists in customizing ECC to latency errors and different DRAM reliability issues.

Data Pattern Dependence. We find that timing errors caused by reducing activation latency are dependent on the stored data pattern. Reading bit `1` is significantly more reliable than bit `0` at reduced activation latencies. This asymmetric sensing strength can potentially be a good direction for studying DRAM reliability. Currently, DRAM commonly employs data bus inversion [53] as an encoding scheme to reduce toggle rate on the data bus, thereby saving channel power [113]. Similar encoding techniques can be developed to reduce bit `0`s and increase the overall number of `1`s in data. We believe that developing asymmetric data encodings or ECC mechanisms that favor `1`s over `0`s is a promising research direction to improve DRAM reliability.

DRAM-Aware Page Allocator. We developed a hardware mechanism (FLY-DRAM) that exploits latency variation to improve system performance in a software-transparent manner. Researchers can take better advantage of the variation by exposing the different latency regions to the software stack. In our SIGMETRICS 2016 paper [24], we discuss the potential of a DRAM-aware page allocator in the OS (Section 7.2), which can improve FLY-DRAM performance by intelligently mapping more frequently-accessed application pages to faster DRAM regions. We believe that the key idea of enabling the OS to allocate pages based on the accessed memory region's latency can be applied to other types of memory characteristics (e.g., energy efficiency or voltage [26, 29]) without needing to modify the architecture.

Applicability to Other Memory Technologies. In this work, we focus on characterizing only DRAM technology. A class of emerging memory technology is non-volatile memory (NVM), which has the capability of retaining data even when the memory is not powered. Since the memory organization of NVM mostly resembles that of DRAM [71, 96, 147], we believe that our characterization and optimization can be extended to different types of NVMs, such as PCM [71, 72, 73, 116, 117, 147, 148], STT-MRAM [27, 41, 70], or NAND flash memory [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 89, 90, 91] to further enhance their reliability or performance.

6.2. Long-Term Impact on Industry

High main memory latency remains a problem for many modern applications, such as in-memory databases (e.g., Redis [121], MemSQL [94], TimesTen [108]), Spark, Google's datacenter workloads [28, 55], and many mobile and interactive workloads. We propose two simple ideas that exploit latency variation in existing DRAM chips. Both can be adopted relatively easily in the processor architecture (i.e., the memory controller) or in the OS.

In addition to improving memory access latency, reducing the latency of the three fundamental DRAM operations also increases the effective memory bandwidth. To fully utilize the available memory bandwidth, memory controllers would have to maximize the number of READ or WRITE commands. However, due to interference between access streams within and across applications, memory controllers need to constantly open and close rows by issuing ACTIVATE and PRECHARGE commands due to an increasing number of bank conflicts [44, 68]. These commands increase the queuing latency of accesses (READ and WRITE), thus decreasing the effective memory bandwidth utilization.

As pin count is limited and increasing bus frequency is becoming more difficult (due to signal integrity issues [29]), our work offers a new alternative to help improve bandwidth utilization. By reducing the latency of DRAM operations, which fall on the critical path of DRAM access time, more accesses per second are allowed, thereby improving the overall effective bandwidth. Furthermore, improving latency and effective bandwidth also leads to lower memory energy consumption due to reduced execution time and fewer active cycles.

All these benefits (e.g., reduced latency, increased bandwidth, and reduced energy) will become much more important as applications become more data-intensive and systems become more energy-constrained in the foreseeable future [102, 105].

In conclusion, we believe that in the longer term, the idea of leveraging variation in different characteristics (e.g., latency, reliability) inside memory chips will become more beneficial for both the software and hardware industry. For example, by making CPU aware of variation behavior in memory devices, memory vendors have an incentive to sell memory with larger variation at a lower price, allowing system designers to lower costs with a small amount of additional logic in hardware. Many other opportunities to improve system performance, energy, and cost abound, which we hope the future works can build upon and exploit.

7. Conclusion

This paper provides the first experimental study that comprehensively characterizes and analyzes the latency variation within modern DRAM chips for three fundamental DRAM operations (activation, precharge, and restoration). We find that significant latency variation is present across DRAM cells in all 240 of our tested DRAM chips, and that a large fraction of cache lines can be read reliably even if the activation/restoration/precharge latencies are reduced significantly. Consequently, exploiting the latency variation in DRAM cells can greatly reduce the DRAM access latency. Based on the findings from our experimental characterization, we propose and evaluate a new mechanism, FLY-DRAM (Flexible-LatencY DRAM), which reduces DRAM latency by exploiting the inherent latency variation in DRAM cells. FLY-DRAM reduces DRAM latency by categorizing the DRAM cells into fast and slow regions, and accessing the fast regions with a reduced latency. We demonstrate that FLY-DRAM can greatly reduce DRAM latency, leading to significant system performance improvements on a variety of workloads.

We conclude that it is promising to understand and exploit the inherent latency variation within modern DRAM chips. We hope that the experimental characterization, analysis, and optimization techniques presented in this paper will enable the development of other new mechanisms that exploit the latency variation within DRAM to improve system performance and perhaps reliability.

Acknowledgments

We thank the anonymous reviewers and SAFARI group members for their feedback. We acknowledge the support of Google, Intel, NVIDIA, and Samsung. This research was supported in part by the ISTC-CC, SRC, and NSF (grants 1212962 and 1320531). Kevin Chang was supported in part by the SRCEA/Intel Fellowship.

References

[1] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[2] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[3] B. Akin *et al.*, "Data Reorganization in Memory Using 3D-stacked DRAM," in *ISCA*, 2015.
[4] R. Ausavarungnirun *et al.*, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, 2012.
[5] O. O. Babarinsa and S. Idreos, "Jafar: Near-data processing for databases," in *SIGMOD*, 2015.
[6] B. H. Bloom, "Space/Time Tradeoffs in Hash Coding with Allowable Errors," *CACM*, July 1970.
[7] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.
[8] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[9] Y. Cai *et al.*, "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation," in *DSN*, 2015.
[10] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proceedings of the IEEE*, 2017.
[11] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.
[12] Y. Cai *et al.*, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.
[13] Y. Cai *et al.*, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.
[14] Y. Cai *et al.*, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.
[15] Y. Cai *et al.*, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.
[16] Y. Cai *et al.*, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.
[17] Y. Cai *et al.*, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," in *ITJ*, 2013.
[18] Y. Cai *et al.*, "Neighbor Cell Assisted Error Correction in MLC NAND Flash Memories," in *SIGMETRICS*, 2014.
[19] Y. Cai *et al.*, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.
[20] K. Chakraborty and P. Mazumder, *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories.* Prentice Hall, 2002.
[21] K. Chandrasekar *et al.*, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.
[22] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[23] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.
[24] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[25] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[26] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[27] M. T. Chang *et al.*, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *HPCA*, 2013.
[28] R. Clapp *et al.*, "Quantifying the performance impact of memory latency and bandwidth for big data workloads," in *IISWC*, 2015.
[29] H. David *et al.*, "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC*, 2011.
[30] J. Dean and L. A. Barroso, "The Tail at Scale," *CACM*, 2013.
[31] J. Draper *et al.*, "The Architecture of the DIVA Processing-in-memory Chip," in *ICS*, 2002.
[32] N. El-Sayed *et al.*, "Temperature Management in Data Centers: Why Some (Might) Like It Hot," in *SIGMETRICS*, 2012.

[33] A. Farmahini-Farahani *et al.*, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *HPCA*, 2015.
[34] B. B. Fraguela *et al.*, "Programming the FlexRAM Parallel Intelligent Memory System," in *PPoPP*, 2003.
[35] M. Gao *et al.*, "Practical near-data processing for in-memory analytics frameworks," in *PACT*, 2015.
[36] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *HPCA*, 2016.
[37] S. Ghose *et al.*, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.
[38] M. Gokhale *et al.*, "Processing in memory: the Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.
[39] S.-L. Gong *et al.*, "CLEAN-ECC: High Reliability ECC for Adaptive Granularity Memory System," in *MICRO*, 2015.
[40] Q. Guo *et al.*, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WONDP*, 2014.
[41] X. Guo *et al.*, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *ISCA*, 2010.
[42] M. Hashemi *et al.*, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.
[43] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[44] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[45] H. Hidaka *et al.*, "The Cache DRAM Architecture," *IEEE Micro*, 1990.
[46] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[47] K. Hsieh *et al.*, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *ICCD*, 2016.
[48] A. A. Hwang *et al.*, "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," in *ASPLOS*, 2012.
[49] E. Ipek *et al.*, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.
[50] JEDEC, "DDR2 SDRAM Standard," 2009.
[51] JEDEC, "DDR3 SDRAM Standard," 2010.
[52] JEDEC, "Standard No. 21-C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules," 2011.
[53] JEDEC, "DDR4 SDRAM Standard," 2012.
[54] X. Jian *et al.*, "Low-Power, Low-Storage-Overhead Chipkill Correct via Multi-Line Error Correction," in *SC*, 2013.
[55] S. Kanev *et al.*, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.
[56] Y. Kang *et al.*, "FlexRAM: toward an advanced intelligent memory system," in *ICCD*, 1999.
[57] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[58] S. Khan *et al.*, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *CAL*, 2016.
[59] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," in *DSN*, 2016.
[60] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[61] J. S. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[62] J. S. Kim *et al.*, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.
[63] J. Kim *et al.*, "Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory," in *HPCA*, 2015.
[64] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[65] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[66] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[67] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[68] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[69] P. M. Kogge, "EXECUBE-A New Architecture for Scaleable MPPs," in *ICPP*, 1994.
[70] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.
[71] B. C. Lee *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[72] B. C. Lee *et al.*, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, vol. 53, no. 7, pp. 99–106, 2010.
[73] B. C. Lee *et al.*, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, vol. 30, no. 1, pp. 143–143, 2010.
[74] C. J. Lee *et al.*, "Prefetch-Aware DRAM Controllers," in *MICRO*, 2008.
[75] C. J. Lee *et al.*, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," Univ. of Texas at Austin, High Performance Systems Group, Tech. Rep. TR-HPS-2010-002, 2010.
[76] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[77] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[78] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.
[79] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[80] D. Lee *et al.*, "Adaptive-Latency DRAM: Reducing DRAM Latency by Exploiting Timing Margins," *IPSI Transactions on Advanced Research (TAR)*, 2018.
[81] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[82] D. Lee *et al.*, "Simultaneous Multi Layer Access: A High Bandwidth and Low Cost 3D-Stacked Memory Interface," *TACO*, 2016.
[83] S. Li *et al.*, "MAGE: Adaptive Granularity and ECC for Resilient and Power Efficient Memory Systems," in *SC*, 2012.
[84] X. Li *et al.*, "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility," in *USENIX ATC*, 2010.
[85] Y. Li *et al.*, "DRAM Yield Analysis and Optimization by a Statistical Design Approach," in *IEEE TCSI*, 2011.
[86] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[87] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[88] S.-L. Lu *et al.*, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in *MICRO*, 2015.
[89] Y. Luo *et al.*, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *MSST*, 2015.
[90] Y. Luo *et al.*, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.
[91] Y. Luo *et al.*, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.
[92] K. Mai *et al.*, "Smart memories: a modular reconfigurable architecture," in *ISCA*, 2000.
[93] S. A. McKee, "Reflections on the memory wall," in *CF*, 2004.
[94] MemSQL, Inc., "MemSQL," https://www.memsql.com.
[95] J. Meza *et al.*, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.
[96] J. Meza *et al.*, "A Case for Small Row Buffers in Non-Volatile Main Memories," in *ICCD Poster Session*, 2012.
[97] Micron Technology, Inc., "128Mb: x4, x8, x16 Automotive SDRAM," 1999.
[98] Micron Technology, Inc., "576Mb: x18, x36 RLDRAM3," 2011.
[99] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-core Systems," in *USENIX Security*, 2007.
[100] S. P. Muralidhara *et al.*, "Reducing Memory Interference in Multicore Systems via Application-aware Memory Channel Partitioning," in *MICRO*, 2011.
[101] O. Mutlu, "The RowHammer problem and other issues we may face as memory becomes denser," in *DATE*, 2017.
[102] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.
[103] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[104] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[105] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[106] S. O *et al.*, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *ISCA*, 2014.
[107] M. Onabajo and J. Silva-Martinez, *Analog Circuit Design for Process Variation-Resilient Systems-on-a-Chip*. Springer, 2012.
[108] Oracle, "Oracle TimesTen In-Memory Database," https://www.oracle.com/database/timesten-in-memory-database/index.html.
[109] M. Oskin *et al.*, "Active pages: a computation model for intelligent memory," in *ISCA*, 1998.
[110] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[111] D. Patterson *et al.*, "A Case for Intelligent RAM," *IEEE Micro*, 1997.
[112] A. Pattnaik *et al.*, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," in *PACT*, 2016.
[113] G. Pekhimenko *et al.*, "A Case for Toggle-Aware Compression for GPU Systems," in *HPCA*, 2016.
[114] S. H. Pugsley *et al.*, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *ISPASS*, 2014.
[115] M. K. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[116] M. K. Qureshi *et al.*, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *MICRO*, 2009.
[117] M. K. Qureshi *et al.*, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[118] S. Rixner *et al.*, "Memory Access Scheduling," in *ISCA*, 2000.

[119] SAFARI Research Group, “Ramulator – GitHub Repository,” https://github.com/CMU-SAFARI/ramulator.

[120] SAFARI Research Group, “SAFARI Software Tools – GitHub Repository,” https://github.com/CMU-SAFARI.

[121] S. Sanfilippo, “Redis,” https://redis.io.

[122] Y. Sato *et al.*, “Fast cycle RAM (FCRAM): A 20-ns Random Row Access, Pipe-Lined Operating DRAM,” in *VLSIC*, 1998.

[123] B. Schroeder *et al.*, “DRAM Errors in the Wild: A Large-Scale Field Study,” in *SIGMETRICS*, 2009.

[124] V. Seshadri *et al.*, “The Dirty-Block Index,” in *ISCA*, 2014.

[125] V. Seshadri *et al.*, “Fast Bulk Bitwise AND and OR in DRAM,” *CAL*, 2015.

[126] V. Seshadri, “Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems,” Ph.D. dissertation, Carnegie Mellon University, 2016.

[127] V. Seshadri *et al.*, “RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization,” in *MICRO*, 2013.

[128] V. Seshadri *et al.*, “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology,” in *MICRO*, 2017.

[129] V. Seshadri *et al.*, “Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses,” in *MICRO*, 2015.

[130] W. Shin *et al.*, “NUAT: A Non-Uniform Access Time Memory Controller,” in *HPCA*, 2014.

[131] Y. H. Son *et al.*, “Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations,” in *ISCA*, 2013.

[132] V. Sridharan *et al.*, “Memory Errors in Modern Systems: The Good, The Bad, and The Ugly,” in *ASPLOS*, 2015.

[133] V. Sridharan and D. Liberty, “A Study of DRAM Failures in the Field,” in *SC*, 2012.

[134] H. S. Stone, “A Logic-in-Memory Computer,” *IEEE TC*, 1970.

[135] L. Subramanian *et al.*, “BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling,” in *IEEE TPDS*, 2016.

[136] L. Subramanian *et al.*, “The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost,” in *ICCD*, 2014.

[137] L. Subramanian *et al.*, “Mise: Providing performance predictability and improving fairness in shared main memory systems,” in *HPCA*, 2013.

[138] L. Subramanian *et al.*, “The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-application Interference at Shared Caches and Main Memory,” in *MICRO*, 2015.

[139] Z. Sura *et al.*, “Data access optimization in a processing-in-memory system,” in *CF*, 2015.

[140] A. N. Udipi *et al.*, “LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems,” in *ISCA*, 2012.

[141] H. Usui *et al.*, “DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators,” *TACO*, vol. 12, no. 4, pp. 65:1–65:28, 2016.

[142] C. Wilkerson *et al.*, “Reducing Cache Power with Low-cost, Multi-bit Error-correcting Codes,” in *ISCA*, 2010.

[143] M. V. Wilkes, “The Memory Gap and the Future of High Performance Memories,” *SIGARCH CAN*, 2001.

[144] H.-S. P. Wong *et al.*, “Metal-Oxide RRAM,” *Proc. IEEE*, 2012.

[145] D. H. Yoon *et al.*, “BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs,” in *ISCA*, 2012.

[146] D. H. Yoon and M. Erez, “Virtualized ECC: Flexible Reliability in Main Memory,” in *ASPLOS*, 2010.

[147] H. Yoon *et al.*, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” in *ICCD*, 2012.

[148] H. Yoon *et al.*, “Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories,” *TACO*, vol. 11, no. 4, pp. 40:1–40:25, 2014.

[149] D. Zhang *et al.*, “TOP-PIM: Throughput-Oriented Programmable Processing in Memory,” in *HPDC*, 2014.

[150] T. Zhang *et al.*, “Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-grained Activation,” in *ISCA*, 2014.

Voltron: Understanding and Exploiting the Voltage–Latency–Reliability Trade-Offs in Modern DRAM Chips to Improve Energy Efficiency

Kevin K. Chang[1,2] Abdullah Giray Yağlıkçı[2] Saugata Ghose[2] Aditya Agrawal[3]
Niladrish Chatterjee[3] Abhijith Kashyap[4,2] Donghyuk Lee[3]
Mike O'Connor[3,5] Hasan Hassan[6] Onur Mutlu[6]

[1]*Facebook* [2]*Carnegie Mellon University* [3]*NVIDIA Research*
[4]*NVIDIA* [5]*The University of Texas at Austin* [6]*ETH Zürich*

This paper summarizes our work on experimental characterization and analysis of reduced-voltage operation in modern DRAM chips, which was published in SIGMETRICS 2017 [29], and examines the work's significance and future potential. This work is motivated to reduce the energy consumption of DRAM, which is a critical concern in modern computing systems. Improvements in manufacturing process technology have allowed DRAM vendors to lower the DRAM supply voltage conservatively, which reduces some of the DRAM energy consumption. We would like to reduce the DRAM supply voltage more aggressively, to further reduce energy. Aggressive supply voltage reduction requires a thorough understanding of the effect voltage scaling has on DRAM access latency and DRAM reliability.

We take a comprehensive approach to understanding and exploiting the latency and reliability characteristics of modern DRAM when the supply voltage is lowered below the nominal voltage level specified by DRAM standards. Using an open-source FPGA-based testing platform based on SoftMC [54], we perform an experimental study of 124 real DDR3L (low-voltage) DRAM chips manufactured recently by three major DRAM vendors. We find that reducing the supply voltage below a certain point introduces bit errors in the data, and we comprehensively characterize the behavior of these errors. We discover that these errors can be avoided by increasing the latency of three major DRAM operations (activation, restoration, and precharge). We perform detailed DRAM circuit simulations to validate and explain our experimental findings. We also characterize the various relationships between reduced supply voltage and error locations, stored data patterns, DRAM temperature, and data retention.

Based on our observations, we propose a new DRAM energy reduction mechanism, called Voltron. *The key idea of Voltron is to use a performance model to determine by how much we can reduce the supply voltage without introducing errors and without exceeding a user-specified threshold for performance loss. Our evaluations show that Voltron reduces the average DRAM and system energy consumption by 10.5% and 7.3%, respectively, while limiting the average system performance loss to only 1.8%, for a variety of memory-intensive quad-core workloads. We also show that Voltron significantly outperforms prior dynamic voltage and frequency scaling mechanisms for DRAM. We believe our experimental characterization and findings can pave the way for new mechanisms that exploit DRAM voltage to improve power, performance, energy, and reliability.*

1. Motivation

In a wide range of modern computing systems, spanning from warehouse-scale data centers to mobile platforms, energy consumption is a first-order concern [39, 56, 65, 105, 107]. In these systems, the energy consumed by the DRAM-based main memory system constitutes a significant fraction of the total energy. For example, experimental studies of production systems have shown that DRAM consumes 40% of the total energy in servers [56, 140] and 40% of the total power in graphics cards [115].

Improvements in manufacturing process technology have allowed DRAM vendors to lower the DRAM supply voltage conservatively, which reduces some of the DRAM energy consumption [59, 60, 61]. In this work, we would like to reduce DRAM energy by *further reducing DRAM supply voltage*. Vendors choose a conservatively high supply voltage, to provide a *guardband* that allows DRAM chips with worst-case process variation to operate without errors under the worst-case operating conditions [36]. The exact amount of supply voltage guardband varies across chips, and lowering the voltage below the guardband can result in erroneous or even undefined behavior [29]. Therefore, we need to understand how DRAM chips behave during reduced-voltage operation. To our knowledge, no previously published work examines the effect of using a wide range of different supply voltage values on the reliability, latency, and retention characteristics of DRAM chips.

Our goal in our SIGMETRICS 2017 paper [29] is to *(i)* characterize and understand the relationship between supply voltage reduction and various characteristics of DRAM, including DRAM reliability, latency, and data retention; and *(ii)* use the insights derived from this characterization and understanding to design a new mechanism that can aggressively lower the supply voltage to reduce DRAM energy consumption while keeping performance loss under a bound.

To this end, we build an FPGA-based testing platform based on SoftMC [54] that allows us to tune the DRAM supply voltage and change DRAM timing parameters (i.e., the amount of time the memory controller waits for a DRAM operation to complete). We perform an experimental study on 124 real 4Gb DDR3L (low-voltage) DRAM chips manufactured recently (between 2014 and 2016) by three major DRAM vendors. Our extensive experimental characterization yields four major observations on how DRAM latency, reliability, and data retention are affected by reduced voltage.

Based on our experimental observations, we propose a new low-cost DRAM energy reduction mechanism called *Voltron*. The key idea of Voltron is to use a performance model to determine by how much we can reduce the DRAM array voltage at runtime without introducing errors and without exceeding a user-specified threshold for acceptable performance loss.

2. Characterization of DRAM Under Reduced Supply Voltage

In this section, we briefly summarize our four major observations from our detailed experimental characterization of 31 commodity DRAM modules, also called DIMMs, from three vendors, when the DIMMs operate under reduced supply voltage (i.e., below the nominal voltage level of 1.35V). Each DIMM comprises 4 DDR3L DRAM chips, totaling to 124 chips for 31 DIMMs. Each chip has a 4Gb density. Thus, each of our DIMMs has a 2GB capacity. Table 1 describes the relevant information about the tested DIMMs. For a complete discussion on all of our observations and experimental methodology, we refer the reader to our SIGMETRICS 2017 paper [29].

Vendor	Total Number of Chips	Timing (ns) (tRCD/tRP/tRAS)	Assembly Year
A (10 DIMMs)	40	13.75/13.75/35	2015-16
B (12 DIMMs)	48	13.75/13.75/35	2014-15
C (9 DIMMs)	36	13.75/13.75/35	2015

Table 1: Main properties of the tested DIMMs. Reproduced from [29].

2.1. DRAM Reliability as Voltage Decreases

We first study the reliability of DRAM chips under low voltage, which was not studied by prior works on DRAM voltage scaling (e.g., [36]; see Section 4 for a detailed discussion of these works). Figure 1 shows the fraction of cache lines that experience at least 1 bit of error (i.e., *1 bit flip*) in each DIMM (represented by each curve), categorized based on vendor.

We observe that we can reliably access data when DRAM supply voltage is lowered below the nominal voltage level, *until a certain voltage value*, V_{min}, which is the minimum voltage level at which no bit errors occur. Furthermore, we find that we can reduce the voltage below V_{min} to attain further energy savings, but that errors start occurring in some

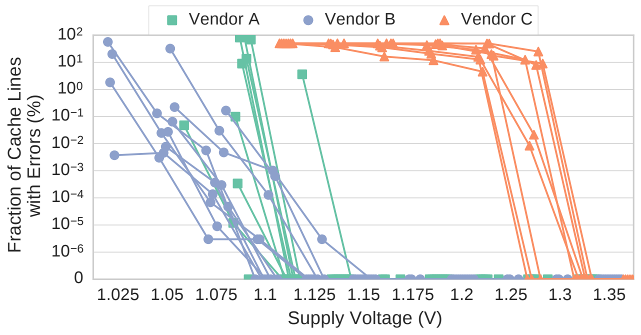


Figure 1: The fraction of erroneous cache lines in each DIMM as we reduce the supply voltage, with a fixed latency. Reproduced from [29].

of the data read from memory. However, not all cache lines exhibit errors for all supply voltage values below V_{min}. Instead, the number of erroneous cache lines for each DIMM increases as we reduce the voltage further below V_{min}. Specifically, Vendor A's DIMMs experience a near-exponential increase in errors as the supply voltage reduces below V_{min}. This is mainly due to the *manufacturing process* [90] and *architectural variation* [87], which introduces strength and size variation across the different DRAM cells within a chip.

We make two major conclusions: *(i)* the variation of errors due to reduced-voltage operation across vendors is very significant; and *(ii)* in most cases, there is a significant margin in the voltage specification, i.e., V_{min} for each chip is significantly lower than the manufacturer-specified supply voltage value.

2.2. Longer Access Latency Mitigates Voltage-Induced Errors

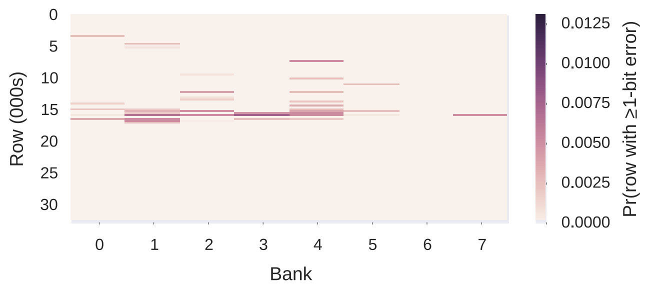
We observe that while reducing the voltage below V_{min} introduces bit errors in the data, we can prevent these errors if we increase the timing parameters of three major DRAM operations, i.e., activation, restoration, and precharge [27,29,55,87,90].[1] When the supply voltage is reduced, the DRAM cell capacitor charge takes a longer time to change, thereby causing these DRAM operations to become slower to complete. Errors are introduced into the data when the memory controller does *not* account for this slowdown in the DRAM operations. We find that if the memory controller allocates extra time for these operations to finish when the supply voltage is below V_{min}, errors no longer occur. We validate, analyze, and explain this behavior using SPICE simulation of a detailed circuit-level model, which we have openly released online [124]. Sections 4.1 and 4.2 of our SIGMETRICS 2017 paper [29] provide our extensive circuit-level analyses, validated using data from real DRAM chips.

[1]We refer the reader to our prior works [26, 27, 28, 29, 54, 55, 72, 75, 77, 78, 79, 80, 87, 88, 90, 91, 92, 96, 97, 112, 128, 129] for a detailed background on DRAM.

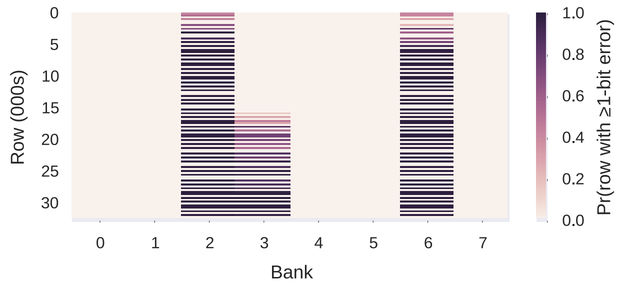
2.3. Spatial Locality of Errors

While reducing the supply voltage induces errors when the DRAM latency is *not* long enough, we also show that *not* all DRAM locations experience errors at all supply voltage levels. To understand the locality of the errors induced by a low supply voltage, we show the probability of each DRAM row in a DIMM experiencing at least one bit of error across all experiments.

Figure 2 shows the probability of each row experiencing at least a one-bit error due to reduced voltage in the two representative DIMMs. For each DIMM, we choose the supply voltage at which errors start appearing (i.e., the voltage level one step below V_{min}), and we do *not* increase the DRAM access latency (i.e., keep it at 10ns for both tRCD and tRP, which are the activation and precharge timing parameters, respectively). The x-axis and y-axis indicate the bank number and row number (in thousands), respectively. Our tested DIMMs are divided into eight banks, and each bank consists of 32K rows of cells. Additional results showing the error locations at different voltage levels are in our SIGMETRICS 2017 paper [29].



(a) DIMM B_6 of vendor B at 1.05V.



(b) DIMM C_2 of vendor C at 1.20V.

Figure 2: The probability of error occurrence for two representative DIMMs, categorized into different rows and banks, due to reduced voltage. Reproduced from [29].

The major observation is that when only a small number of errors occur due to reduced supply voltage, these errors tend to *cluster* physically in certain *regions* of a DRAM chip, as opposed to being randomly distributed throughout the chip.[2] This observation implies that when we reduce the supply voltage to the DRAM array, we need to increase the fundamental operation latencies for *only* the regions where errors can occur.

[2]We believe this observation is due to both process and architectural variation across different regions in the DRAM chip.

2.4. Impact on Refresh Rate

Commodity DRAM chips guarantee that all cells can safely retain data for 64ms, after which the cells are *refreshed* to replenish charge that leaks out of the capacitors [26, 96, 97]. We observe that the effect of the supply voltage on retention times is *not* statistically significant. Even when we reduce the supply voltage from 1.35V to 1.15V (i.e., a 15% reduction), the rate at which charge leaks from the capacitors is so slow that no data is lost during the 64ms refresh interval at both 20℃ and 70℃. Therefore, we conclude that using a reduced supply voltage does not require any changes to the standard refresh interval at 20℃ and 70℃. Detailed results are in Section 4.6 of our SIGMETRICS 2017 paper [29].

2.5. Other Experimental Observations

We refer the reader to our SIGMETRICS 2017 paper [29] for more details on the other two key observations. First, we find that the most commonly-used ECC scheme, SECDED [66, 99, 132], is unlikely to alleviate errors induced by a low supply voltage. This is because lowering voltage increases the fraction of data that contains more than two bits of errors, exceeding the one-bit correction capability of SECDED (see Section 4.4 of our SIGMETRICS 2017 paper [29]). Second, temperature affects the reliable access latency at low supply voltage levels and the effect is very vendor-dependent (see Section 4.5 of our SIGMETRICS 2017 paper [29]). Out of the three major vendors whose DIMMs we evaluate, DIMMs from two vendors require longer activation and precharge latencies to operate reliably at high temperature under low supply voltage. The main reason is that DRAM chips become slower at higher temperature [24, 87, 90].

3. Exploiting Reduced-Voltage Behavior

Based on the extensive understanding we have developed on reduced-voltage operation of real DRAM chips, we propose a new mechanism called *Voltron*, which reduces DRAM energy without sacrificing memory throughput. Voltron exploits the fundamental observation that reducing the supply voltage to DRAM requires increasing the latency of the three DRAM operations in order to prevent errors. Using this observation, the key idea of Voltron is to use a performance model to determine by how much to reduce the DRAM supply voltage, without introducing errors and without exceeding a user-specified threshold for performance loss. Voltron consists of two main components: *(i) array voltage scaling* and *(ii) performance-aware voltage control.*

3.1. Components of Voltron

Array Voltage Scaling. Unlike prior works, Voltron does *not* reduce the voltage of the *peripheral circuitry*, which is responsible for transferring commands and data between the memory controller and the DRAM chip. If Voltron were to reduce the voltage of the peripheral circuitry, we would *have to* also reduce the operating frequency of DRAM. A reduction in the operating frequency reduces the memory data throughput, which can significantly degrade the performance of applications that require high memory bandwidth. Instead, Voltron reduces the voltage supplied to *only* the DRAM array without changing the voltage supplied to the peripheral circuitry, thereby allowing the DRAM channel to maintain a high frequency while reducing the power consumption of the DRAM array. To prevent errors from occurring during reduced-voltage operation, Voltron increases the latency of the three DRAM operations (activation, restoration, and precharge) based our observation in Section 2.2.

Performance-Aware Voltage Control. Array voltage scaling provides system users with the ability to decrease DRAM array voltage (V_{array}) to reduce DRAM power. Employing a lower V_{array} provides greater power savings, but at the cost of longer DRAM access latency, which leads to larger performance degradation. This trade-off varies widely across different applications, as each application has a different tolerance to the increased memory latency. This raises the question of how to pick a "suitable" array voltage level for different applications as a system user or designer. For our evaluations, we say that an array voltage level is suitable if it does not degrade system performance by more than a user-specified threshold. Our goal is to provide a simple technique that can automatically select a suitable V_{array} value for different applications. To this end, we propose *performance-aware voltage control*, a power–performance management policy that selects a minimum V_{array} which satisfies a desired performance constraint. The key observation is that an application's performance loss (due to increased memory latency) scales linearly with the application's memory demand (e.g., memory intensity). Based on this empirical observation we make, we build a *performance loss predictor* that leverages a linear model to predict an application's performance loss based on its characteristics and the effect of different voltage level choices at runtime. Using the performance loss predictor, Voltron finds a value of V_{array} that can keep the predicted performance within the user-specified target at runtime. We refer the reader to Section 5.2 of our SIGMETRICS 2017 paper [29] for more detail and for an evaluation of the performance model alone.

3.2. Evaluation

We evaluate the system-level energy and performance impact of Voltron using Ramulator [75, 124], integrated with McPAT [93] and DRAMPower [25] for modeling the energy consumption of both the processor and DRAM. Our workloads consist of 27 benchmarks from SPEC CPU2006 [134] and YCSB [34]. We evaluate Voltron with a target performance loss of 5%. Voltron executes the performance-aware voltage control mechanism once every four million cycles. We refer the reader to Section 6.1 of our SIGMETRICS 2017 paper [29] for more detail on the system configuration and workloads. We qualitatively and quantitatively compare Voltron to *MemDVFS*, a dynamic DRAM frequency and voltage scaling mechanism proposed by prior work [36].

Figure 3 shows the system energy savings and the system performance (i.e., weighted speedup [43, 131]) loss due to MemDVFS and Voltron, compared to a baseline DRAM with a supply voltage of 1.35V. The graph uses box plots to show the distribution among all workloads that are categorized as either non-memory-intensive or memory-intensive. The memory intensity is determined based on the commonly-used metric MPKI (last-level cache misses per kilo-instruction). We categorize an application as memory intensive when its MPKI is greater than or equal to 15. We make two observations.

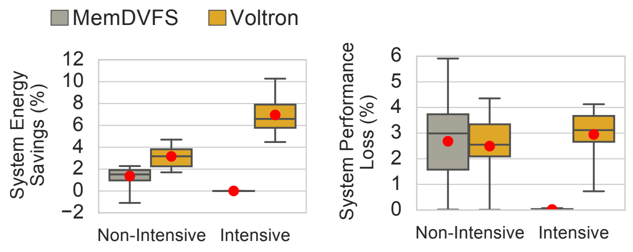


Figure 3: Energy (left) and performance (right) comparison between Voltron and MemDVFS on non-memory-intensive and memory-intensive workloads. Adapted from [29].

First, Voltron is effective and saves more energy than MemDVFS. MemDVFS has almost zero effect on memory-intensive workloads. This is because MemDVFS avoids scaling DRAM frequency (and hence voltage) when an application's memory bandwidth utilization is above a fixed threshold. Reducing the frequency can result in a large performance loss since the memory-intensive workloads require high memory throughput. As memory-intensive applications have high memory bandwidth consumption that easily exceeds the fixed threshold used by MemDVFS, MemDVFS *cannot* perform frequency and voltage scaling during most of the execution time. In contrast, Voltron reduces system energy by 7.0% on average for memory-intensive workloads. Thus, we demonstrate that Voltron is an effective mechanism that improves system energy efficiency not only on non-memory-intensive applications, but also (especially) on memory-intensive workloads where prior work was unable to do so.

Second, as shown in Figure 3 (right), Voltron consistently selects a V_{array} value that satisfies the performance loss bound of 5% across all workloads. Voltron incurs an average (maximum) performance loss of 2.5% (4.4%) and 2.9% (4.1%) for non-memory-intensive and memory-intensive workloads, respectively. This demonstrates that our performance model enables Voltron to select a low voltage value that saves energy while bounding performance loss based on the user's requirement.

Our SIGMETRICS 2017 paper contains extensive performance and energy analysis of the Voltron mechanism in Sections 6.2 to 6.8 [29]. In particular, we show that if we exploit spatial locality of errors (Section 2.3), we can improve the performance benefits of Voltron, reducing the average performance loss for memory-intensive workloads to 1.8% (see Section 6.5 of our SIGMETRICS 2017 paper [29]). We refer the reader to these sections for a detailed evaluation of Voltron.

4. Related Work

To our knowledge, this is the first work to *(i)* experimentally characterize the reliability and performance of modern low-power DRAM chips under different supply voltages, and *(ii)* introduce a new mechanism that reduces DRAM energy while retaining high memory data throughput by adjusting the DRAM array voltage. We briefly discuss other prior work in DRAM energy reduction.

DRAM Frequency and Voltage Scaling. Many prior works propose to reduce DRAM energy by adjusting the memory channel frequency and/or the DRAM supply voltage dynamically. Deng et al. [39] propose MemScale, which scales the frequency of DRAM at runtime based on a performance predictor of an in-order processor. Other work focuses on developing management policies to improve system energy efficiency by coordinating DRAM *DFS* with DVFS on the CPU [12,37,38] or GPU [115]. In addition to frequency scaling, David et al. [36] propose to scale the DRAM supply voltage along with the memory channel frequency, based on the memory bandwidth utilization of applications.

In contrast to all these works, our work focuses on a detailed experimental characterization of real DRAM chips as the supply voltage varies. Our study provides fundamental observations for potential mechanisms that can mitigate DRAM and system energy consumption. Furthermore, frequency scaling hurts memory throughput, and thus significantly degrades the system performance of especially memory-intensive workloads (see Section 2.4 in our SIGMETRICS 2017 paper [29] for our quantitative analysis). We demonstrate the importance and benefits of exploiting our experimental observations by proposing Voltron, one new example mechanism that uses our observations to reduce DRAM and system energy without sacrificing memory throughput.

Low-Power Modes for DRAM. Modern DRAM chips support various low-power standby modes. Entering and exiting these modes incurs some amount of latency, which delays memory requests that must be serviced. To increase the opportunities to exploit these low-power modes, several prior works propose mechanisms that increase periods of memory idleness through data placement (e.g., [44, 83]) and memory traffic reshaping (e.g., [2, 9, 14, 40, 100]). Exploiting low-power modes is orthogonal to our work on studying the impact of reduced-voltage operation in DRAM. Furthermore, low-power modes have a smaller effect on memory-intensive workloads, which exhibit little idleness in memory accesses, whereas, as we show in Section 3.2, our mechanism is especially effective on memory-intensive workloads.

Low-Power DDR DRAM Chips. Low-power DDR (LPDDR) [59, 61, 112] is a specific type of DRAM that is optimized for low-power systems like mobile devices. To reduce power consumption, LPDDRx (currently in its 4th generation) employs a few major design changes that differ from conventional DDRx chips. First, LPDDRx uses a low-voltage swing I/O interface that consumes 40% less I/O power than DDR4 DRAM [33]. Second, it supports additional low-power modes with a lower supply voltage. Since the LPDDRx array design remains the same as DDRx, our observations on the correlation between access latency and array voltage are applicable to LPDDRx DRAM as well. Voltron, our proposal, can provide significant benefits in LPDDRx, since array energy consumption is significantly *higher* than the energy consumption of peripheral circuitry in LPDDRx chips [33]. We leave the detailed evaluation of LPDDRx chips for future work since our current experimental platform is not capable of evaluating them. Two recent experimental works [72, 112] examine the retention time behavior of LPDDRx chips and find it to be similar to DDRx chips.

Low-Power DRAM Architectures. Prior works (e.g., [31, 35, 137, 150]) propose to modify the DRAM chip architecture to reduce the `ACTIVATE` power by activating only a fraction of a row instead of the entire row. Another common technique, called sub-ranking or mini-ranks, reduces dynamic DRAM power by accessing data from a subset of chips from a DRAM module [139, 145, 152]. A couple of prior works [102, 144] propose DRAM module architectures that integrate many low-frequency LPDDR chips to enable DRAM power reduction. These proposed changes to DRAM chips or DIMMs are orthogonal to our work.

Reducing Refresh Power. In modern DRAM chips, although different DRAM cells have widely different retention times [74, 96, 112], memory controllers conservatively refresh *all* of the cells based on the retention time of a small fraction of weak cells, which have the longest retention time out of all of the cells. To reduce DRAM refresh power, many prior works (e.g., [3, 11, 13, 68, 69, 70, 71, 95, 96, 97, 106, 108, 110, 112, 119, 138]) propose mechanisms to reduce unnecessary refresh operations, and, thus, refresh power, by characterizing the retention time profile (i.e., the data retention behavior of each cell) within the DRAM chips. However, these techniques do not reduce the power of *other* DRAM operations, and these prior works do *not* provide an experimental characterization of the effect of reduced voltage levels on data retention time.

Improving DRAM Energy Efficiency by Reducing Latency or Improving Parallelism. Various prior works (e.g., [26, 28, 54, 55, 80, 87, 88, 89, 90, 91, 92, 107, 128, 129, 130]) improve DRAM energy efficiency by reducing the execution time through techniques that reduce the DRAM access latency or improve parallelism between memory requests. These mechanisms are orthogonal to ours, because they do not reduce the voltage level of DRAM.

Improving Energy Efficiency by Processing in Memory. Various prior works [4, 5, 6, 10, 16, 17, 28, 41, 45, 46, 48, 49, 50, 51, 53, 57, 58, 67, 73, 81, 101, 111, 113, 114, 118, 126, 127, 129, 130, 135, 136, 149] examine processing in memory to improve energy efficiency. Our analyses and techniques can be combined with these works to enable low-voltage operation in processing-in-memory engines.

Experimental Studies of DRAM Chips. Recent works experimentally investigate various reliability, data retention, and latency characteristics of modern DRAM chips [24, 27, 54, 63, 64, 70, 71, 76, 77, 87, 89, 90, 96, 97, 104, 112, 125, 132, 133] usually using FPGA-based DRAM testing infrastructures, like SoftMC [54], or using large-scale data from the field. None of these works study these characteristics under reduced-voltage operation, which we do in this paper.

Reduced-Voltage Operation in SRAM Caches. Prior works propose different techniques to enable SRAM caches to operate under reduced voltage levels (e.g., [7, 8, 32, 123, 141, 142]). These works are orthogonal to our experimental study because we focus on understanding and enabling reduced-voltage operation in DRAM, which is a significantly different memory technology than SRAM.

5. Significance

Our SIGMETRICS 2017 paper [29] presents a new set of detailed experimental characterization and analyses on the voltage-latency-reliability trade-offs in modern DRAM chips. In this section, we describe the potential impact that our study can bring to the research community and industry.

5.1. Potential Industry Impact

We believe our experimental characterization results and proposed mechanism can have significant impact in fast-growing data centers as well as mobile systems, where DRAM power consumption is growing due to higher demand for memory capacity for certain types of service (e.g., memcached). To reduce the energy and power consumed by DRAM, DRAM manufacturers have been decreasing the supply voltage of DRAM chips with newer DRAM standards (e.g., DDR4) or low-voltage variants of DDR, such as LPDDR4 (Low-Power DDR4) and DDR3L (DDR3 Low-voltage). However, the supply voltage reduction has been conservative with each new DDR standard, which takes years to be adopted by the vendors and the market. For example, since the release of DDR3L (1.35V) in 2010, the supply voltage has reduced by *only* 11% with the latest DDR4 standard (1.2V) released in 2014. Furthermore, since the release of DDR4 in 2014, the supply voltage for most commodity DDR4 chips has remained at 1.2V. As a result, further reducing DRAM supply voltage below the standard voltage, as we do in our SIGMETRICS 2017 paper [29], can be a very effective way of reducing DRAM power consumption. However, to do so, we need to carefully and rigorously understand how DRAM chips behave under reduced-voltage operation.

To enable the development of new mechanisms that leverage reduce-voltage operation in DRAM, we provide the first set of comprehensive experimental results on the effect of using a wide range of different supply voltage values on the reliability, latency, and retention characteristics of DRAM chips. In this work, we demonstrate how we can use our experimental data to design a new mechanism, Voltron (Section 3), which reduces DRAM energy consumption through voltage reduction. Therefore, we believe that understanding and leveraging reduced-voltage operation will help industry improve the energy efficiency of memory subsystems.

5.2. Potential Research Impact

Our paper sheds new light on the feasibility of enabling reduced-voltage operation in manufactured DRAM chips. One important research question that our work raises is *how do modern DRAM chips behave under a wide range of supply voltage levels?* Existing systems are limited to a few DRAM power states, which prevent DRAM from serving memory accesses when it enters a low-power state. However, in our work, we show that it is possible to operate commodity DRAM chips under a wide range of supply voltage levels while still being able to serve memory accesses under a different set of trade-offs. To facilitate further research initiative to exploit reduced-voltage operation in DRAM chips, we have open-sourced our characterization results, FPGA-based testing platform [54], and DRAM SPICE circuit model (for validation) in our GitHub repository [124]. We believe that these tools can be extended for other research objectives besides studying voltage reduction in DRAM. One potential direction is to leverage our results to design mechanisms that reduce DRAM latency by operating DRAM at a higher supply voltage.

5.3. Applicability to Other Memory Technologies

We believe the high-level ideas of our work can be leveraged in the context of other memory technologies, such as NAND flash memory [19, 20, 21], PCM [84, 85, 86, 103, 120, 121, 146, 147], STT-MRAM [30, 52, 82, 103, 109], RRAM [143], or hybrid memory systems [1, 15, 42, 47, 62, 94, 98, 103, 116, 117, 121, 122, 146, 148, 151]. A recent work on NAND flash memory, for example, proposes reducing the pass-through voltage [18, 19, 20, 21] to reduce read disturb errors, which in turn saves energy. We refer the reader to past works on NAND flash memory for a more detailed analysis of reliability-voltage trade-offs [18, 19, 20, 21, 22, 23]. We hope our work inspires characterization and understanding of reduced-voltage operation in other memory technologies, with the goal of enabling a more energy-efficient system design.

6. Conclusion

Our SIGMETRICS 2017 paper [29] provides the first experimental study that comprehensively characterizes and analyzes the behavior of DRAM chips when the supply voltage is reduced below its nominal value. We demonstrate, using 124 DDR3L DRAM chips, that the DRAM supply voltage can be reliably reduced to a certain level, beyond which errors arise within the data. We then experimentally demonstrate the relationship between the supply voltage and the latency of the fundamental DRAM operations (activation, restoration, and precharge). We show that bit errors caused by reduced-voltage operation can be eliminated by increasing the latency of the three fundamental DRAM operations. By changing the memory controller configuration to allow for the longer latency of these operations, we can thus *further* lower the supply voltage without inducing errors in the data. We also experimentally characterize the relationship between reduced supply voltage and error locations, stored data patterns, temperature, and data retention.

Based on these observations, we propose and evaluate Voltron, a low-cost energy reduction mechanism that reduces DRAM energy *without* affecting memory data throughput. Voltron reduces the supply voltage for *only* the DRAM array, while maintaining the nominal voltage for the peripheral circuitry to continue operating the memory channel at a high frequency. Voltron uses a new piecewise linear performance model to find the array supply voltage that maximizes the system energy reduction within a given performance loss target. Our experimental evaluations across a wide variety of workloads demonstrate that Voltron significantly reduces system energy consumption with only very modest performance loss.

We conclude that it is very promising to understand and exploit reduced-voltage operation in modern DRAM chips. We hope that the experimental characterization, analysis, and optimization techniques presented in our SIGMETRICS 2017 paper will enable the development of other new mechanisms that can effectively exploit the trade-offs between voltage, reliability, and latency in DRAM to improve system performance, efficiency, and/or reliability. We also hope that our paper's studies inspire new experimental studies to understand reduced-voltage operation in other memory technologies, such as NAND flash memory, PCM, and STT-MRAM.

Acknowledgments

We thank the anonymous reviewers of SIGMETRICS 2017 and SAFARI group members for their feedback. We acknowledge the support of Google, Intel, NVIDIA, Samsung, VMware, and the United States Department of Energy. This research was supported in part by the ISTC-CC, SRC, and NSF (grants 1212962 and 1320531). Kevin Chang was supported in part by an SRCEA/Intel Fellowship.

References

[1] N. Agarwal and T. F. Wenisch, "Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory," in *ASPLOS*, 2017.
[2] N. Aggarwal *et al.*, "Power-Efficient DRAM Speculation," in *HPCA*, 2008.
[3] A. Agrawal *et al.*, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules," in *HPCA*, 2014.
[4] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[5] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[6] B. Akin *et al.*, "Data Reorganization in Memory Using 3D-stacked DRAM," in *ISCA*, 2015.
[7] A. R. Alameldeen *et al.*, "Adaptive Cache Design to Enable Reliable Low-Voltage Operation," *IEEE TC*, 2011.
[8] A. R. Alameldeen *et al.*, "Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes," in *ISCA*, 2011.
[9] A. M. Amin and Z. A. Chishti, "Rank-aware Cache Replacement and Write Buffering to Improve DRAM Energy Efficiency," in *ISLPED*, 2010.
[10] O. O. Babarinsa and S. Idreos, "Jafar: Near-data processing for databases," in *SIGMOD*, 2015.
[11] S. Baek *et al.*, "Refresh Now and Then," *IEEE TC*, vol. 63, no. 12, pp. 3114–3126, 2014.
[12] R. Begum *et al.*, "Energy-Performance Trade-offs on Energy-Constrained Devices with Multi-component DVFS," in *IISWC*, 2015.
[13] I. Bhati *et al.*, "Flexible Auto-refresh: Enabling Scalable and Energy-efficient DRAM Refresh Reductions," in *ISCA*, 2015.
[14] M. Bi *et al.*, "Delay-Hiding Energy Management Mechanisms for DRAM," in *HPCA*, 2010.
[15] S. Bock *et al.*, "Concurrent Migration of Multiple Pages in Software-Managed Hybrid Main Memory," in *ICCD*, 2016.
[16] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.
[17] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[18] Y. Cai *et al.*, "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation," in *DSN*, 2015.
[19] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proceedings of the IEEE*, 2017.
[20] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.
[21] Y. Cai *et al.*, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.
[22] Y. Cai *et al.*, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.
[23] Y. Cai *et al.*, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.
[24] K. Chandrasekar *et al.*, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.
[25] K. Chandrasekar *et al.*, "DRAMPower: Open-source DRAM Power & Energy Estimation Tool," http://www.drampower.info.
[26] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[27] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[28] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[29] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[30] M. T. Chang *et al.*, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *HPCA*, 2013.
[31] N. Chatterjee *et al.*, "Architecting an Energy-Efficient DRAM System for GPUs," in *HPCA*, 2017.
[32] Z. Chishti *et al.*, "Improving Cache Lifetime Reliability at Ultra-Low Voltages," in *MICRO*, 2009.
[33] J. Choi, "LPDDR4: Evolution for new Mobile World," in *MEMCON*, 2013. Available: http://www.memcon.com/pdfs/proceedings2013/track1/LPDDR4_Evolution_for_a_New_Mobile_World.pdf
[34] B. F. Cooper *et al.*, "Benchmarking Cloud Serving Systems with YCSB," in *SOCC*, 2010.
[35] E. Cooper-Balis and B. Jacob, "Fine-Grained Activation for Power Reduction in DRAM," *IEEE Micro*, vol. 30, no. 3, pp. 34–47, 2010.
[36] H. David *et al.*, "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC*, 2011.
[37] Q. Deng *et al.*, "CoScale: Coordinating CPU and Memory System DVFS in Server Systems," in *MICRO*, 2012.
[38] Q. Deng *et al.*, "MultiScale: Memory System DVFS with Multiple Memory Controllers," in *ISLPED*, 2012.
[39] Q. Deng *et al.*, "MemScale: Active Low-power Modes for Main Memory," in *ASPLOS*, 2011.
[40] B. Diniz *et al.*, "Limiting the Power Consumption of Main Memory," in *ISCA*, 2007.
[41] J. Draper *et al.*, "The Architecture of the DIVA Processing-in-memory Chip," in *ICS*, 2002.
[42] S. R. Dulloor *et al.*, "Data Tiering in Heterogeneous Memory Systems," in *EuroSys*, 2016.

[43] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.
[44] X. Fan *et al.*, "Memory Controller Policies for DRAM Power Management," in *ISLPED*, 2001.
[45] A. Farmahini-Farahani *et al.*, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *HPCA*, 2015.
[46] B. B. Fraguela *et al.*, "Programming the FlexRAM Parallel Intelligent Memory System," in *PPoPP*, 2003.
[47] K. Gai *et al.*, "Smart Energy-Aware Data Allocation for Heterogeneous Memory," in *HPCC*, 2016.
[48] M. Gao *et al.*, "Practical near-data processing for in-memory analytics frameworks," in *PACT*, 2015.
[49] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in *HPCA*, 2016.
[50] M. Gokhale *et al.*, "Processing in memory: the Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.
[51] Q. Guo *et al.*, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WONDP*, 2014.
[52] X. Guo *et al.*, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *ISCA*, 2010.
[53] M. Hashemi *et al.*, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.
[54] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[55] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[56] U. Höelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan & Claypool, 2009.
[57] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[58] K. Hsieh *et al.*, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *ICCD*, 2016.
[59] JEDEC, "Low Power Double Data Rate 3 (LPDDR3)," 2012.
[60] JEDEC, "Addendum No.1 to JESD79-3 - 1.35V DDR3L-800, DDR3L-1066, DDR3L-1333, DDR3L-1600, and DDR3L-1866," 2013.
[61] JEDEC, "Low Power Double Data Rate 4 (LPDDR4)," 2014.
[62] X. Jiang *et al.*, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *HPCA*, 2010.
[63] M. Jung *et al.*, "A New Bank Sensitive DRAMPower Model for Efficient Design Space Exploration," in *PATMOS*, 2016.
[64] M. Jung *et al.*, "Reverse Engineering of DRAMs: Row Hammer with Crosshair," in *MEMSYS*, 2016.
[65] R. Kalla *et al.*, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, no. 2, pp. 7–15, 2010.
[66] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[67] Y. Kang *et al.*, "FlexRAM: toward an advanced intelligent memory system," in *ICCD*, 1999.
[68] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[69] S. Khan *et al.*, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *CAL*, 2016.
[70] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," in *DSN*, 2016.
[71] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[72] J. S. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[73] J. S. Kim *et al.*, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.
[74] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *EDL*, vol. 30, no. 8, pp. 846–848, 2009.
[75] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[76] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon University, 2015.
[77] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[78] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[79] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[80] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[81] P. M. Kogge, "EXECUBE-A New Architecture for Scaleable MPPs," in *ICPP*, 1994.
[82] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.
[83] A. R. Lebeck *et al.*, "Power Aware Page Allocation," in *ASPLOS*, 2000.
[84] B. C. Lee *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[85] B. C. Lee *et al.*, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, vol. 53, no. 7, pp. 99–106, 2010.
[86] B. C. Lee *et al.*, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, vol. 30, no. 1, pp. 143–143, 2010.
[87] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[88] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[89] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.
[90] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[91] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[92] D. Lee *et al.*, "Simultaneous Multi Layer Access: A High Bandwidth and Low Cost 3D-Stacked Memory Interface," *TACO*, 2016.
[93] S. Li *et al.*, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO*, 2009.
[94] Y. Li *et al.*, "Utility-Based Hybrid Memory Management," in *CLUSTER*, 2017.
[95] C. H. Lin *et al.*, "SECRET: Selective Error Correction for Refresh Energy Reduction in DRAMs," in *ICCD*, 2012.
[96] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[97] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[98] L. Liu *et al.*, "Memos: A Full Hierarchy Hybrid Memory Management Framework," in *ICCD*, 2016.
[99] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in *DSN*, 2014.
[100] C. Lyuh and T. Kim, "Memory Access Scheduling and Binding Considering Energy Minimization in Multi-Bank Memory Systems," in *DAC*, 2004.
[101] K. Mai *et al.*, "Smart memories: a modular reconfigurable architecture," in *ISCA*, 2000.
[102] K. T. Malladi *et al.*, "Towards Energy-Proportional Datacenter Memory with Mobile DRAM," in *ISCA*, 2012.
[103] J. Meza *et al.*, "A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[104] J. Meza *et al.*, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.
[105] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.
[106] O. Mutlu, "The RowHammer problem and other issues we may face as memory becomes denser," in *DATE*, 2017.
[107] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.
[108] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[109] H. Naeimi *et al.*, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[110] T. Ohsawa *et al.*, "Optimizing the DRAM Refresh Count for Merged DRAM/Logic LSIs," in *ISLPED*, 1998.
[111] M. Oskin *et al.*, "Active pages: a computation model for intelligent memory," in *ISCA*, 1998.
[112] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[113] D. Patterson *et al.*, "A Case for Intelligent RAM," *IEEE Micro*, 1997.
[114] A. Pattnaik *et al.*, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," in *PACT*, 2016.
[115] I. Paul *et al.*, "Harmonia: Balancing Compute and Memory Power in High-performance GPUs," in *ISCA*, 2015.
[116] A. J. Peña and P. Balaji, "Toward the Efficient Use of Multiple Explicitly Managed Memory Subsystems," in *CLUSTER*, 2014.
[117] S. Phadke and S. Narayanasamy, "MLP aware heterogeneous memory system," in *DATE*, 2011.
[118] S. H. Pugsley *et al.*, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads," in *ISPASS*, 2014.
[119] M. K. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[120] M. K. Qureshi *et al.*, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *MICRO*, 2009.
[121] M. K. Qureshi *et al.*, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[122] L. E. Ramos *et al.*, "Page Placement in Hybrid Memory Systems," in *ICS*, 2011.
[123] D. Roberts *et al.*, "On-Chip Cache Device Scaling Limits and Effective Fault Repair Techniques in Future Nanoscale Technology," in *DSD*, 2007.
[124] SAFARI Research Group, "SAFARI Software Tools – GitHub Repository," https://github.com/CMU-SAFARI.
[125] B. Schroeder *et al.*, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009.
[126] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.

[127] V. Seshadri, “Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems,” Ph.D. dissertation, Carnegie Mellon University, 2016.
[128] V. Seshadri *et al.*, “RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization,” in *MICRO*, 2013.
[129] V. Seshadri *et al.*, “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology,” in *MICRO*, 2017.
[130] V. Seshadri *et al.*, “Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses,” in *MICRO*, 2015.
[131] A. Snavely and D. Tullsen, “Symbiotic Jobscheduling for a Simultaneous Multithreading Processor,” in *ASPLOS*, 2000.
[132] V. Sridharan *et al.*, “Memory Errors in Modern Systems: The Good, The Bad, and The Ugly,” in *ASPLOS*, 2015.
[133] V. Sridharan and D. Liberty, “A Study of DRAM Failures in the Field,” in *SC*, 2012.
[134] Standard Performance Evaluation Corp., “SPEC CPU2006 Benchmarks,” http://www.spec.org/cpu2006.
[135] H. S. Stone, “A Logic-in-Memory Computer,” *IEEE TC*, 1970.
[136] Z. Sura *et al.*, “Data access optimization in a processing-in-memory system,” in *CF*, 2015.
[137] A. N. Udipi *et al.*, “Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores,” in *ISCA*, 2010.
[138] R. Venkatesan *et al.*, “Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM,” in *HPCA*, 2006.
[139] F. A. Ware and C. Hampel, “Improving Power and Data Efficiency with Threaded Memory Modules,” in *ICCD*, 2006.
[140] M. Ware *et al.*, “Architecting for Power Management: The IBM® POWER7™ Approach,” in *HPCA*, 2010.
[141] C. Wilkerson *et al.*, “Trading Off Cache Capacity for Reliability to Enable Low Voltage Operation,” in *ISCA*, 2008.
[142] C. Wilkerson *et al.*, “Trading Off Cache Capacity for Low-Voltage Operation,” *IEEE Micro*, 2009.
[143] H.-S. P. Wong *et al.*, “Metal-Oxide RRAM,” *Proc. IEEE*, 2012.
[144] D. H. Yoon *et al.*, “BOOM: Enabling Mobile Memory Based Low-power Server DIMMs,” in *ISCA*, 2012.
[145] D. H. Yoon *et al.*, “Adaptive Granularity Memory Systems: A Tradeoff Between Storage Efficiency and Throughput,” in *ISCA*, 2011.
[146] H. Yoon *et al.*, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” in *ICCD*, 2012.
[147] H. Yoon *et al.*, “Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories,” *TACO*, vol. 11, no. 4, pp. 40:1–40:25, 2014.
[148] X. Yu *et al.*, “Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation,” in *MICRO*, 2017.
[149] D. Zhang *et al.*, “TOP-PIM: Throughput-Oriented Programmable Processing in Memory,” in *HPDC*, 2014.
[150] T. Zhang *et al.*, “Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-grained Activation,” in *ISCA*, 2014.
[151] W. Zhang and T. Li, “Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures,” in *PACT*, Raleigh, NC, September 2009, pp. 101–112.
[152] H. Zheng *et al.*, “Mini-rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency,” in *MICRO*, 2008.

SoftMC: Practical DRAM Characterization Using an FPGA-Based Infrastructure

Hasan Hassan[1,2,3] Nandita Vijaykumar[2] Samira Khan[4,2]
Saugata Ghose[2] Kevin Chang[5,2] Gennady Pekhimenko[6,2]
Donghyuk Lee[7,2] Oguz Ergin[3] Onur Mutlu[1,2]

[1]*ETH Zürich* [2]*Carnegie Mellon University* [3]*TOBB University of Economics & Technology*
[4]*University of Virginia* [5]*Facebook* [6]*Microsoft Research* [7]*NVIDIA Research*

This paper summarizes the SoftMC DRAM characterization infrastructure, which was published in HPCA 2017 [44], and examines the work's significance and future potential. DRAM is the primary technology used for main memory in modern systems. Unfortunately, as DRAM scales down to smaller technology nodes, it faces key challenges in both data integrity and latency, which strongly affect overall system reliability and performance. To develop reliable and high-performance DRAM-based main memory in future systems, it is critical to characterize, understand, and analyze various aspects (e.g., reliability, latency) of modern DRAM chips. To enable this, there is a strong need for a publicly-available DRAM testing infrastructure that can flexibly and efficiently test DRAM chips in a manner accessible to both software and hardware developers.

This work develops the first such infrastructure, SoftMC (Soft Memory Controller)*, an FPGA-based testing platform that can control and test memory modules designed for the commonly-used DDR (Double Data Rate) interface. SoftMC has two key properties:* (i) *it provides* flexibility *to thoroughly control memory behavior or to implement a wide range of mechanisms using DDR commands; and* (ii) *it is* easy to use *as it provides a simple and intuitive high-level programming interface for users, completely hiding the low-level details of the FPGA.*

We demonstrate the capability, flexibility, and programming ease of SoftMC with two example use cases. First, we implement a test that characterizes the retention time of DRAM cells. Experimental results we obtain using SoftMC are consistent with the findings of prior studies on retention time in modern DRAM, which serves as a validation of our infrastructure. Second, we validate two recently-proposed mechanisms, which rely on accessing recently-refreshed or recently-accessed DRAM cells faster than other DRAM cells. Using our infrastructure, we show that the expected latency reduction effect of these mechanisms is not observable in existing DRAM chips,which demonstrates the usefulness of SoftMC in testing new ideas on existing memory modules.

Various versions of the SoftMC platform have enabled many of our other DRAM characterization studies [26, 29, 60, 61, 62, 68, 80, 84, 88, 117]. We discuss several other use cases of SoftMC, including the ability to characterize emerging non-volatile memory modules that obey the DDR standard. We hope that our open-source release of SoftMC fills a gap in the space of publicly-available experimental memory testing infrastructures and inspires new studies, ideas, and methodologies in memory system design.

1. Understanding DRAM Characteristics

DRAM (Dynamic Random Access Memory) is the predominant technology used to build main memory systems of modern computers. The continued scaling of DRAM process technology has enabled tremendous growth in DRAM density in the last few decades, leading to higher capacity main memories. Unfortunately, as the process technology node scales down to the sub-20 nm feature size range, DRAM technology faces key challenges that critically impact its reliability and performance [102, 103, 106].

The fundamental challenge with scaling DRAM cells into smaller technology nodes arises from the way DRAM stores data in cells. A DRAM cell consists of a transistor and a capacitor. Data is stored as charge in the capacitor. A DRAM cell cannot retain its data permanently as this capacitor leaks its charge gradually over time. To maintain correct data in DRAM, each cell is periodically refreshed to replenish the charge in the capacitor [87]. At smaller technology nodes, it is becoming increasingly difficult to store and retain enough charge in a cell, causing various reliability and performance issues [27,63,87,88]. Ensuring reliable operation of the DRAM cells is a key challenge in future technology nodes [55, 60, 66, 87, 88, 93, 99, 102, 103, 112].

The fundamental problem of retaining data with less charge in smaller cells directly impacts the reliability and performance of DRAM cells. First, smaller cells placed in close proximity make cells more susceptible to various types of interference. This potentially disrupts DRAM operation by flipping bits in DRAM, resulting in major reliability issues [68, 95, 108, 121, 126, 135, 136], which can lead to system failure [95, 126] or security breaches [10, 41, 68, 120, 127, 128, 144, 148]. Second, it takes longer time to access a cell with less charge [43, 80], and write latency increases as the access transistor size reduces [55]. Thus, smaller cells directly impact DRAM latency, as DRAM access latency is determined by the worst-case (i.e., *slowest*) cell in any acceptable chip [24, 29, 80]. DRAM access latency has not significantly improved with technology scaling in the past two decades [7, 25, 26, 54, 81, 82, 102], and, in fact, some latencies are expected to increase [55], making memory latency an increasingly critical system performance bottleneck.

As such, there is a significant need for new mechanisms that improve the reliability and performance of DRAM-based main memory systems. In order to design, evaluate, and validate many such mechanisms, it is important to accurately characterize, analyze, and understand DRAM (cell) behavior in terms of reliability and latency. For such an understanding to be accurate, it is critical that the characterization and analysis be based on the *experimental* studies of *real DRAM chips*, since a large number of factors (e.g., various types of cell-to-cell interference [68, 108, 121], inter- and intra-die process variation [24, 26, 29, 65, 80, 84, 109, 112], random effects [45, 60, 88, 117, 123, 137, 149], operating conditions [29, 65, 80, 86, 88, 112], internal organization [46, 61, 88], stored data patterns [61, 62, 88]) concurrently impact the reliability and latency of cells. Many of these phenomena and their interactions cannot be properly modeled (e.g., in simulation or using analytical methods) without rigorous experimental characterization and analysis of real DRAM chips. The need for such experimental characterization and analysis, with the goal of building the understanding necessary to improve the reliability and performance of future DRAM-based main memories at various levels (both software and hardware), motivates the need for a publicly-available DRAM testing infrastructure that can enable system users and designers to characterize real DRAM chips.

2. Experimental DRAM Characterization

Two key features are desirable from an experimental memory testing infrastructure. First, the infrastructure should be *flexible* enough to test any DRAM operation (supported by the commonly-used DRAM interfaces, e.g., the standard Double Data Rate, or DDR, interface) to characterize cell behavior or evaluate the impact of a mechanism (e.g., adopting different refresh rates for different cells [60, 62, 63, 87, 112, 117, 145]) on real DRAM chips. Second, the infrastructure should be *easy to use*, such that it is possible for both software and hardware developers to implement new tests or mechanisms without spending significant time and effort. For example, a testing infrastructure that requires circuit-level implementation, detailed knowledge of the physical implementation of DRAM data transfer protocols over the memory channel, or low-level FPGA-programming to modify the infrastructure would severely limit the usability of such a platform to a limited number of experts.

Our HPCA 2017 paper [44] designs, prototypes, and demonstrates the basic capabilities of such a flexible and easy-to-use experimental DRAM testing infrastructure, called *SoftMC (Soft Memory Controller)*. SoftMC is an open-source FPGA-based DRAM testing infrastructure, consisting of a programmable memory controller that can control and test memory modules designed for the commonly-used DDR (Double Data Rate) interface. To this end, SoftMC implements *all* low-level DRAM operations (i.e., DDR commands) available in a typical memory controller (e.g., opening a row in a bank, reading a specific column address, performing a refresh operation, enforcing various timing constraints between commands). Using these low-level operations, SoftMC can test and characterize any (existing or new) DRAM mechanism that uses the existing DDR interface. SoftMC provides a simple and intuitive high-level programming interface that completely hides the low-level details of the FPGA from users. Users implement their test routines or mechanisms in a high-level language that automatically gets translated into the low-level SoftMC memory controller operations in the FPGA.

3. Overview of SoftMC

A publicly-available DRAM testing infrastructure should have two key features to ensure widespread adoption among architects and designers: *(i)* flexibility and *(ii)* ease of use.

Flexibility. A DRAM chip is typically accessed by issuing a set of DRAM commands in a particular sequence with a strict delay between the commands (specified by the timing parameters in the datasheet of the DRAM chip/module). A DRAM testing infrastructure should implement all low-level DRAM operations with tunable timing parameters without any restriction on the ordering of DRAM commands. Such a design enables flexibility at two levels. First, it enables comprehensive testing of *any* DRAM operation with the ability to customize the length of each timing constraint. For example, we can implement a retention test with different refresh intervals to characterize the distribution of retention time in modern DRAM chips (as done in [60, 87, 112]). Such a characterization can enable new mechanisms to reduce the number of refresh operations in DRAM, leading to performance and power efficiency improvements. Second, it enables testing of DRAM chips with high-level test programs, which can consist of *any combination of DRAM operations and timings*. Such flexibility is extremely powerful to test the impact of existing or new DRAM mechanisms in real DRAM chips.

Ease of Use. A DRAM testing infrastructure should provide a simple and intuitive programming interface that minimizes programming effort and time. An interface that hides the details of the underlying implementation is accessible to a wide range of users. With such a high-level abstraction, even users that lack hardware design experience should be able to develop DRAM tests.

Figure 1 shows our temperature-controller setup for testing DRAM modules. The components of SoftMC operate on the *host machine* and the *FPGA*. On the host machine, the *SoftMC API* provides a high-level software interface (in C++) for developing a test program that generates DRAM commands and sends them to the FPGA. On the FPGA, *SoftMC hardware* is responsible for handling the commands sent by the host machine. The SoftMC hardware issues the DRAM commands in order and with the timing parameters as defined in the test program developed using the SoftMC API. SoftMC also implements a PCIe driver for high-speed communication between the host machine and the FPGA. The user only needs to focus on defining a routine for testing the DRAM.

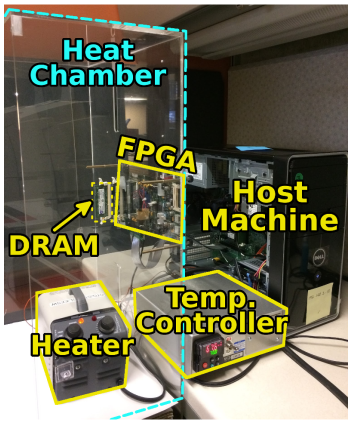


Figure 1: Our SoftMC infrastructure. Reproduced from [44].

A detailed description of the interface, design, and operation of SoftMC can be found in our HPCA 2017 paper [44]. The source code for SoftMC can be freely downloaded from [125].

4. Example Use Cases

Using our SoftMC prototype, we perform two case studies on randomly-selected real DRAM chips from three major manufacturers. First, we discuss how a simple retention test can be implemented using SoftMC, and present the experimental results of that test (Section 4.1). Second, we demonstrate how SoftMC can be leveraged to test the expected effect of two recently-proposed mechanisms [43, 134] that aim to reduce DRAM access latency (Section 4.2). Both use cases demonstrate the flexibility and ease of use of SoftMC.

4.1. Retention Time Distribution Study

This test aims to characterize data retention time in different DRAM modules. The retention time of a cell can be determined by testing the cell with different refresh intervals. The cell fails at a refresh interval that is greater than its retention time. In this test, we gradually increase the refresh interval from the default 64 ms and count the number of bytes that have an incorrect value at each refresh interval.

4.1.1. Evaluating Retention Time with SoftMC. We perform a simple test to measure the retention time of the cells in a DRAM chip. Our test consists of three steps: *(i)* We write a reference data pattern (e.g. all zeros, or all ones) to an entire row. *(ii)* We wait for the specified refresh interval, so that the row is idle for that time and all cells gradually leak charge. *(iii)* We read data back from the same row and compare it against the reference pattern that we wrote in the first step. Any mismatch indicates that the cell could not hold its data for that duration, resulting in a bit flip. We count the number of bytes that have bit flips for each test.

We repeat this procedure for all rows in the DRAM module. The read and write operations in the test are issued with the standard timing parameters, to make sure that the only timing change that affects the reliability of the cells is the change in the refresh interval.

Writing Data to DRAM. In Program 1, we present the implementation of the first part of our retention time test, where we write data to a row, using the SoftMC API. First, to activate the row, we insert the instruction generated by the *genACT()* function to an instance of the *InstructionSequence* (Lines 1-2). This function is followed by a *genWAIT()* function (Line 3) that ensures that the activation completes with the standard timing parameter `tRCD`. Second, we issue write instructions to write the data pattern in each column of the row. This is implemented in a loop, where, in each iteration, we call *genWR()* (Line 5), followed by a call to *genWAIT()* function (Line 6) that ensures proper delay between two `WRITE` operations. After writing to all columns of the row, we insert another delay (Line 8) to account for the *write recovery* time `tWR`. Third, once we have written to all columns, we close the row by precharging it. This is done by the *genPRE()* function (Line 9), followed by a *genWAIT()* function with standard `tRP` timing.[1] Finally, we call the *genEND()* function to indicate the end of the instruction sequence, and send the test program to the FPGA by calling the *execute()* function.

```
1  InstructionSequence iseq;
2  iseq.insert(genACT(bank, row));
3  iseq.insert(genWAIT(tRCD));
4  for(int col = 0; col < COLUMNS; col++){
5      iseq.insert(genWR(bank, col, data));
6      iseq.insert(genWAIT(tBL));
7  }
8  iseq.insert(genWAIT(tCL + tWR));
9  iseq.insert(genPRE(bank));
10 iseq.insert(genWAIT(tRP));
11 iseq.insert(genEND());
12 iseq.execute(fpga);
```

Program 1: Writing data to a row using the SoftMC API. Reproduced from [44].

Employing a Specific Refresh Interval. Using SoftMC, we can implement the target refresh interval in two ways. We can use the auto-refresh support provided by the SoftMC hardware, by setting the `tREFI` parameter to our target value, and letting the FPGA take care of the refresh operations. Alternatively, we can disable auto-refresh, and manually control the refresh operations from the software. In this case, the user is responsible for issuing refresh operations at the right time. In this retention test, we disable auto-refresh and use a software clock to determine when we should read back data from the row (i.e., refresh the row).

Reading Data from DRAM. Reading data back from the DRAM requires steps similar to DRAM writes (presented in

[1]For details on DRAM timing parameters and internal DRAM operation, we refer the reader to our prior works [26, 27, 28, 29, 43, 44, 65, 68, 69, 70, 71, 72, 80, 81, 83, 84, 85, 87, 88, 112, 130, 131].

Program 1). The only difference is that, instead of issuing a `WRITE` command, we need to issue a `READ` command and enforce read-related timing parameters. In the SoftMC API, this is done by calling the *genRD()* function in place of the *genWR()* function, and specifying the appropriate read-related timing parameters. After the read operation is done, the FPGA sends back the data read from the DRAM module, and the user can access that data using the *fpga_recv()* function provided by the driver.

Note that the complete code to implement our full retention test (i.e., writing a data pattern to a DRAM module, waiting for the target retention time, reading the data back from the DRAM module, and checking the data for errors) in SoftMC takes *only* approximately 200 lines of C code, in the form shown in Program 1. Based on the intuitive code implementation of the retention test, we conclude that it requires minimal effort to write test programs using the SoftMC API. Our full test is provided in our open-source release of SoftMC [125].

4.1.2. Results. We perform the retention time test at room temperature, using 24 DRAM chips from three major manufacturers. We vary the refresh interval from 64 ms to 8192 ms, exponentially. Figure 2 shows the results for the test, where the x-axis shows the refresh interval in milliseconds, and the y-axis shows the number of erroneous bytes found in each interval. We make two major observations.

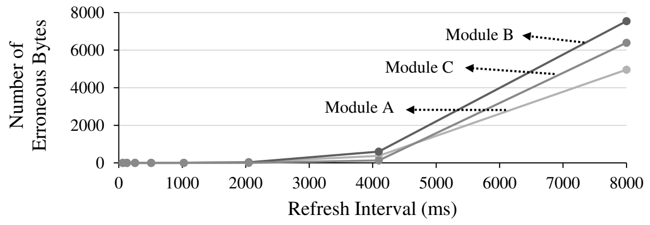


Figure 2: Number of erroneous bytes observed in retention time tests. Reproduced from [44].

(i) We do not observe any retention failures until we test with a refresh interval of 1 s. This shows that there is a large safety margin for the refresh interval in modern DRAM chips, which is conservatively set to 64 ms by the DDR standard.[2]

(ii) We observe that the number of failures increases exponentially with the increase in refresh interval.

Other experimental studies on retention time of DRAM cells have reported similar observations as ours [42, 47, 60, 67, 80, 80, 88, 112]. We conclude that SoftMC can easily reproduce experimental DRAM results, validating the correctness of our testing infrastructure and showing its flexibility and ease of use.

[2]DRAM manufacturers perform retention tests that are similar to ours (but with proprietary in-house infrastructures that are not disclosed). Their results are similar to ours [26, 42, 60, 67, 80, 88, 112], showing significant margin for the refresh interval. This margin is added to ensure reliable DRAM operation for the worst-case operating conditions (i.e., worst case temperature and voltage levels) and for worst-case cells, as has been shown by prior works [26, 42, 60, 67, 80, 88, 112].

4.2. Evaluating the Expected Effect of Two Recently-Proposed Mechanisms in Existing DRAM Chips

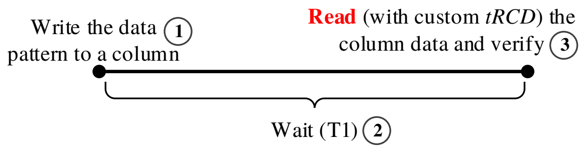
Two recently-proposed mechanisms, ChargeCache [43] and NUAT [134], provide low-latency access to highly-charged DRAM cells. They both are based on the key idea that a highly-charged cell can be accessed faster than a cell with less charge [80]. ChargeCache observes that cells belonging to *recently-accessed* DRAM rows are in a highly-charged state and that such rows are likely to be accessed again in the near future. ChargeCache exploits the highly-charged state of these recently-accessed rows to lower the latency for later accesses to them. NUAT observes that *recently-refreshed* cells are in highly-charged state, and thus it lowers the latency for accesses to recently-refreshed rows. Prior to activating a DRAM row, both ChargeCache and NUAT determine whether the target row is in a highly-charged state. If so, the memory controller uses reduced `tRCD` and `tRAS` timing parameters to perform a low latency access.

In this section, we evaluate whether or not the expected latency reduction effect of these two works is observable in existing DRAM modules, using SoftMC. We first describe our methodology for evaluating the improvement in the `tRCD` and `tRAS` timing parameters. We then show the results we obtain using SoftMC, and discuss our observations.

4.2.1. Evaluating DRAM Latency with SoftMC. In our experiments, we use 24 DDR3 chips (i.e., three SO-DIMMs [53]) from three major manufacturers. To stress DRAM reliability and maximize the amount of cell charge leakage, we raise the test temperature to 80°C (significantly higher than the common-case operating range of 35-55°C [80]) by enclosing our FPGA infrastructure in a temperature-controlled heat chamber (see Figure 1). For all experiments, the temperature within the heat chamber was maintained within 0.5°C of the target 80°C temperature.

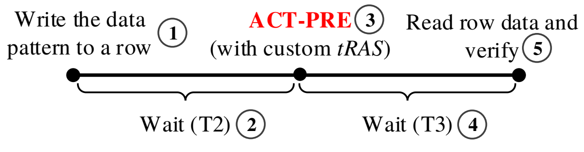
To study the impact of charge variation in cells on access latency, which is dominated by the `tRCD` and `tRAS` timing parameters [26, 69, 80, 81], we perform experiments on existing DRAM chips to test the headroom for reducing these parameters. In our experiments, we vary one of the two timing parameters, and test whether the original data can be read back correctly with the reduced timing. If the data that is read out contains errors, this indicates that the timing parameter *cannot* be reduced to the tested value without inducing errors in the data. We perform the tests using a variety of data patterns (e.g., 0x00, 0xFF, 0xAA, 0x55) because 1) different DRAM cells store information (i.e., 0 or 1) in different states (i.e., charged or empty) [88] and 2) we would like to stress DRAM reliability by increasing the interference between adjacent bitlines [60, 61, 62, 63, 88, 112]. We also perform tests using different refresh intervals, to study whether the variation in charge leakage increases significantly if the time between refreshes increases.

tRCD Test. We measure how highly-charged cells affect the tRCD timing parameter (i.e., how long the controller needs to wait after a row activation command is sent to safely perform read and write operations on the row), by using a custom tRCD value to read data from a row to which we previously wrote a reference data pattern. We adjust the time between writing a reference data pattern and performing the read, to vary the amount of charge stored within the cells of a row. In Figure 3a, we show the command sequence that we use to test whether recently-refreshed DRAM cells can be accessed with a lower tRCD, compared to cells that are close to the end of the refresh interval. We perform the write and read operations to each DRAM row one column at a time, to ensure that each read incurs the tRCD latency. First (① in Figure 3a), we perform a reference write to the DRAM column under test by issuing ACTIVATE, WRITE, and PRECHARGE successively with the *default* DRAM timing parameters. Next (②), we wait for the duration of a time interval (**T1**), which is the refresh interval in practice, to vary the charge contained in the cells. When we wait longer, we expect the target cells to have less charge at the end of the interval. We cover a wide range of wait intervals, evaluating values between 1 and 512 ms. Finally (③), we read the data from the column that we previously wrote to and compare it with the reference pattern. We perform the read with the custom tRCD value for that specific test. We evaluate tRCD values ranging from 3 to 6 (default) cycles. Since a tRCD of 3 cycles produced errors in *every* run, we did not perform any experiments with a lower tRCD.



Write the data pattern to a column ①
Read (with custom tRCD) the column data and verify ③
Wait (T1) ②

(a) tRCD Test



Write the data pattern to a row ①
ACT-PRE ③ (with custom tRAS)
Read row data and verify ⑤
Wait (T2) ②
Wait (T3) ④

(b) tRAS Test

Figure 3: Timelines that illustrate the methodology for testing the improvement of (a) tRCD and (b) tRAS on highly-charged DRAM cells. Reproduced from [44].

We process multiple rows in an interleaved manner (i.e., we write to multiple rows, wait, and then verify their data one after another) in order to further stress the reliability of DRAM [80]. We repeat this process for all DRAM rows to evaluate the entire memory module.

tRAS Test. We measure the effect of accessing highly-charged rows on the tRAS timing parameter (i.e., the time that the controller needs to wait after a row activation command is sent to safely start precharging the row) by issuing the ACTIVATE and PRECHARGE commands, with a custom tRAS value, to a row. We check if that row still contains the same data that it held before the ACTIVATE-PRECHARGE command pair was issued. Figure 3b illustrates the methodology for testing the effect of the refresh interval on tRAS. First (①), we write the reference data pattern to the selected DRAM row with the default timing parameters. Different from the tRCD test, we write to *every column* in the open row (before switching to another row) to save cycles by eliminating a significant amount of ACTIVATE and PRECHARGE commands, thereby reducing the testing time. Next (②), we wait for the duration of time interval **T2**, during which the DRAM cells lose a certain amount of charge. To refresh the cells (③), we issue an ACTIVATE-PRECHARGE command pair associated with a custom tRAS value. When the ACTIVATE-PRECHARGE pair is issued, the charge in the cells of the target DRAM row may not be fully restored if the wait time is too long or the tRAS value is too short, potentially leading to loss of data. Next (④), we wait again for a period of time **T3** to allow the cells to leak a portion of their charge. Finally (⑤), we read the row using the default timing parameters and test whether it still retains the correct data. Similar to the tRCD test, to stress the reliability of DRAM, we simultaneously perform the tRAS test on multiple DRAM rows.

We would expect, from this experiment, that the data is likely to maintain its integrity when evaluating reduced tRAS with *shorter* wait times (**T2**). This is because when **T2** is short, a DRAM cell would lose only a *small* amount of its charge. Thus, there would be more room for reducing tRAS, as the cell would already contain a *higher* amount of charge prior to the row activation. The higher amount of charge would allow us to safely reduce tRAS by a larger amount. In contrast, we would expect failures to be more likely when using a reduced tRAS with a *longer* wait time, because the cells would have a low amount of charge that is not enough to reliably reduce tRAS.

4.2.2. Results. We analyze the results of the tRCD and tRAS tests, for 24 real DRAM chips from different vendors, using the test programs detailed in Section 4.2.1. We evaluate tRCD values ranging from 3 to 6 cycles, and tRAS values ranging from 2 to 14 cycles, where the maximum number for each is the default timing parameter value. For both tests, we evaluate refresh intervals between 8 and 512 ms and measure the number of observed errors during each experiment.

Figures 4 and 5 depict the results for the tRCD test and the tRAS test, respectively, for three DRAM modules (each from a different DRAM vendor). We make three major observations:

(i) Within the duration of the standard refresh interval (64 ms), DRAM cells do not leak a sufficient amount of charge to have a negative impact on DRAM access latency.[3] For refresh intervals less than or equal to 64 ms, we observe little to no

[3]Other studies have shown methods to take advantage of the fact that latencies can be reduced without incurring errors [26, 80].

variation in the number of errors induced. Within this refresh interval range, depending on the tRCD or tRAS value, the errors generated are either zero or a constant number. We make the same observation in both the tRCD and tRAS tests for all three DRAM modules.

For all the modules tested, using different data patterns and stressing DRAM operation with temperatures significantly higher than the common-case operating conditions, we can significantly reduce tRCD and tRAS parameters, without observing any errors. We observe errors only when tRCD and tRAS parameters are too small to correctly perform the DRAM access, regardless of the charge amount of the accessed cells.

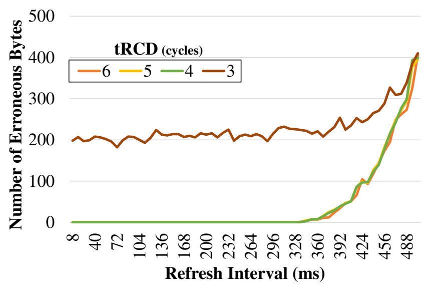
(ii) The large safety margin employed by the manufacturers protects DRAM against errors even when accessing DRAM cells with low latency. We observe no change in the number of induced errors for tRCD values less than the default of 6 cycles (down to 4 cycles in modules A and B, and 5 cycles in module C). We observe a similar trend in the tRAS test: tRAS can be reduced from the default value of 14 cycles to 5 cycles without increasing the number of induced errors for *any* refresh interval.

We conclude that even at temperatures much higher than typical operating conditions, there exists a large safety margin for access latency in existing DRAM chips. This demonstrates that DRAM cells are much *stronger* than their datasheet timing specifications indicate.[4] In other words, the timing margin in most DRAM cells is very large, given the existing timing parameters.

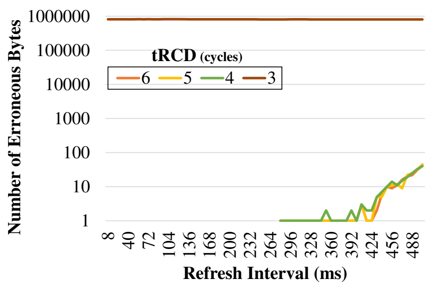
(iii) The expected effect of ChargeCache and NUAT, that highly-charged cells can be accessed with lower latency, is slightly observable only when very long refresh intervals are used. For each of the tests, we observe a significant increase in the number of errors at refresh intervals that are much higher than the typical refresh interval of 64 ms, demonstrating the variation in charge held by each of the DRAM cells. Based on the assumptions made by ChargeCache and NUAT, we expect that when lower values of tRCD and tRAS are employed, the error rate should increase more rapidly. However, we find that for all but the minimum values of tRCD and tRAS (and for tRCD = 4 for module C), the tRCD and tRAS latencies have almost no impact on the error rate.

We believe that the reason we cannot observe the expected latency reduction effect of ChargeCache and NUAT on existing DRAM modules is due to the internal behavior of existing DRAM chips that does not allow latencies to be reduced beyond a certain point: we cannot *externally control* when the sense amplifier gets enabled, since this is dictated with a fixed latency internally, regardless of the charge amount in the cell. The sense amplifiers are enabled only after charge sharing, which starts by enabling the wordline and lasts until sufficient amount of charge flows from the activated cell into the bitline [28, 69, 81, 129, 130, 131], is expected to complete. Within existing DRAM chips, the expected charge sharing latency (i.e., the time when the sense amplifiers get enabled) is *not* represented by a timing parameter managed by the memory

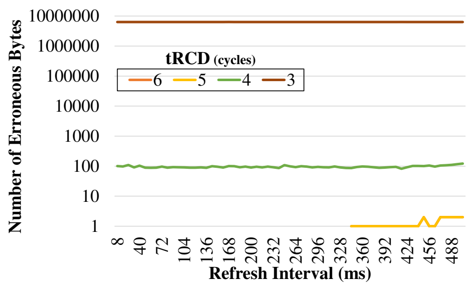
[4]Similar observations were made by prior work [24, 26, 80].



(a) Module A

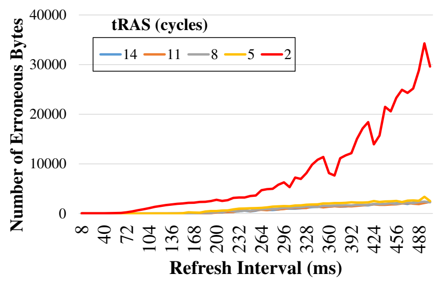


(b) Module B

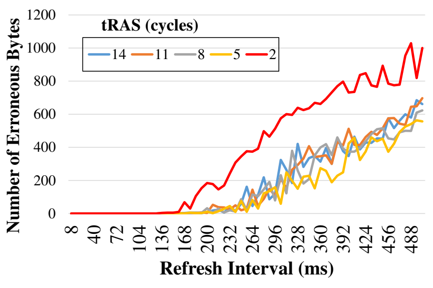


(c) Module C

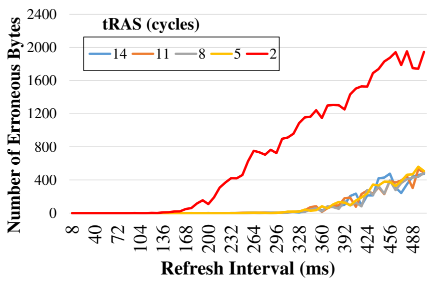
Figure 4: Effect of reducing tRCD on the number of errors at various refresh intervals. Reproduced from [44]



(a) Module A



(b) Module B



(c) Module C

Figure 5: Effect of reducing tRAS on the number of errors at various refresh intervals. Reproduced from [44].

controller. Instead, the latency is controlled internally within the DRAM using a fixed value [58, 143]. ChargeCache and NUAT require that charge sharing completes in less time, and the sense amplifiers get enabled faster for a highly-charged cell. However, since existing DRAM chips provide no way to control the time it takes to enable the sense amplifiers, we cannot harness the potential latency reduction possible for highly-charged cells [143]. Reducing `tRCD` affects the time spent *only after charge sharing*, at which point the bitline voltages exhibit similar behavior regardless of the amount of charge initially stored within the cell. Consequently, we are unable to observe the expected latency reduction effect of ChargeCache and NUAT by simply reducing `tRCD`, even though we believe that the mechanisms are sound and can reduce latency (assuming the behavior of DRAM chips is modified). If the DDR interface exposes a method of controlling the time it takes to enable the sense amplifiers in the future, SoftMC can be easily modified to use the method and fully evaluate the latency reduction effect of ChargeCache and NUAT.

Summary. Overall, we make two major conclusions from the implementation and experimental results of our DRAM latency experiments. First, SoftMC provides a simple and easy-to-use interface to quickly implement tests that characterize modern DRAM chips. Second, SoftMC is an effective tool to validate or refute the expected effect of existing or new mechanisms on existing DRAM chips.

5. Related Work

No prior DRAM testing infrastructure provides both *flexibility* and *ease of use* properties, which are critical for enabling widespread adoption of the infrastructure. Three different kinds of tools/infrastructure are available today for characterizing the behavior of real DRAM chips. As we will describe, each kind of tool has some shortcomings. SoftMC eliminates *all* of these shortcomings and provides the first open-source DRAM testing infrastructure that is publicly available [125].

Commercial Testing Infrastructures. A large number of commercial DRAM testing platforms (e.g., [1, 39, 110, 142]) are available in the market. Such platforms are optimized for test throughput (i.e., to test as many DRAM chips as possible in a given time period), and generally apply a *fixed test pattern* to the units under test. Thus, since they lack support for flexibility in defining the test routine, these infrastructures are not suitable for detailed DRAM characterization where the goal is to investigate new issues and new ideas. Furthermore, such testing equipment is usually quite expensive, which makes these infrastructures an impractical option for research in academia. Industry may also have internal DRAM development and testing tools, but, to our knowledge, these are proprietary and are unlikely to be made openly available.

We design SoftMC to be a low-cost (i.e., free) and flexible open-source alternative to commercial testing equipment that can enable new research directions and mechanisms. For example, prior work [151] recently proposed a random command pattern generator to validate DRAM chips against uncommon yet supported (according to JEDEC specifications) DDR command patterns. Using the test patterns on commercial test equipment, this work demonstrates that specific sequences of commands introduce failures in current DRAM chips (e.g., an `ACTIVATE` followed by a `PRECHARGE`, without any `READ` or `WRITE` commands in between, results in *future* accesses reading incorrect data in some DRAM devices). SoftMC flexibly supports the ability to issue an arbitrary command sequence, and therefore can be used as a low-cost method for validating DRAM chips against problems that arise due to command ordering.

FPGA-Based Testing Infrastructures. Several prior works propose FPGA-based DRAM testing infrastructures [47, 50, 59]. Unfortunately, all of them lack flexibility and/or a simple user interface, and none are open-source. The FPGA-based infrastructure proposed by Huang et al. [50] provides a high-level interface for developing DRAM tests, but the interface is limited to defining only data patterns and march algorithms for the tests. Hou et al. [47] propose an FPGA-based test platform whose capability is limited to analyzing only the data retention time of the DRAM cells. Another work [59] develops a custom memory testing board with an FPGA chip, specifically designed to test memories at a very high data rate. However, it requires low-level knowledge to develop FPGA programs, and even then offers only limited flexibility in defining a test routine. On the other hand, SoftMC provides *full control over all DRAM commands* using a high-level *software interface*, and it is open-source.

PARDIS [6] is a reconfigurable logic (e.g., FPGA) based programmable memory controller meant to be implemented inside microprocessor chips. PARDIS is capable of optimizing memory scheduling algorithms, refresh operations, etc. at run-time based on application characteristics, and can improve system performance and efficiency. However, it does not provide programmability for DRAM commands and timing parameters, and therefore cannot be used for detailed DRAM characterization.

Built-In Self Test (BIST). A BIST mechanism (e.g, [5, 52, 114, 115, 150, 152]) is implemented inside the DRAM chip to enable fixed test patterns and algorithms. Using such an approach, DRAM tests can be performed faster than with other testing platforms. However, BIST has two major flexibility issues, since the testing logic is hard-coded into the hardware: *(i)* BIST offers only a limited number of tests that are fixed at hardware design time. *(ii)* A limited set of DRAM chips, which come with BIST support, can be tested. In contrast, SoftMC allows for the implementation of a wide range of DRAM test routines and supports any off-the-shelf DRAM chip that is compatible with the DDR interface.

Other Related Work. Although no prior work provides an open-source DRAM testing infrastructure similar to SoftMC, infrastructures for testing other types of memories have been developed. Cai et al. [11, 12, 13, 15] develop a platform for characterizing NAND flash memory. They propose a flash controller, implemented on an FPGA, to quickly characterize error patterns of existing flash memory chips. They expose the functions of the flash translation layer (i.e., the flash chip interface) to the software developer via the host machine connected to the FPGA board, similar to how we expose the DDR interface to the user in SoftMC. Many works [11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 38, 89, 90, 91] use this flash memory testing infrastructure to study various aspects of flash chips.

Our prior works [26,60,61,62,68,80,84,88] develop and use FPGA-based infrastructures for a wide range of DRAM studies. Liu et al. [88] and Khan et al. [60] analyze the data retention behavior of modern DRAM chips and proposed mechanisms for mitigating retention failures. Khan et al. [61, 62] study data-dependent failures in DRAM, and developed techniques for efficiently detecting and handling them. Lee et al. [80, 84] analyze latency characteristics of modern DRAM chips and propose mechanisms for latency reduction. Kim et al. [68] discover a new reliability issue in existing DRAM, called *RowHammer*, which can lead to security breaches [41,103,120, 127,128,144,148]. Chang et al. [26] use SoftMC to characterize latency variation across DRAM cells for fundamental DRAM operations (e.g., activation, precharge). SoftMC evolved out of these previous infrastructures, to address the need to make the infrastructure flexible and easy to use.

Recently, Chang et al. [29] extend SoftMC with the capability to change the array voltage of DRAM chips, such that SoftMC can be used to evaluate the trade-offs between voltage, latency, and reliability in modern DRAM chips.

Sukhwani et al. propose ConTutto [141], which is a recent work that builds an FPGA-based platform for evaluating different memory technologies and new mechanisms on existing server systems. ConTutto is an extender board, which plugs into the DDR3 module slot of a server machine. On the board, an FPGA chip manages the communication between the server machine and the memory, which is connected to the other end of the ConTutto board. Using ConTutto, any type of memory that can be attached to the ConTutto board can potentially be used in existing systems, as part of main memory, by using the FPGA as a translator between the two interfaces, i.e., between the DDR3 interface to the server and the interface of the memory attached to the ConTutto board. Although ConTutto can be used as a prototyping platform to evaluate different memory technologies and mechanisms on existing systems, it is *not* practical or flexible enough to use for testing memories for two reasons. First, the operating system needs to ensure that it does not allocate application data to the memory that is being tested, as the data could be destroyed during a testing procedure. Second, the memory that is connected to ConTutto is accessed using load/store instructions, which does *not* provide the flexibility of testing the memory at the memory command level. In contrast, (1) the memory in SoftMC is not a part of the main memory of the host machine, and (2) SoftMC provides a high-level software interface for directly issuing commands to the memory. These design choices enable many tests that are not otherwise possible or practical to implement using load/store instructions.

We conclude that prior work lacks either the flexibility or the ease-of-use properties that are critical for performing detailed DRAM characterization. To fill the gap left by current infrastructures, we introduce an open-source DRAM testing infrastructure, SoftMC, that fulfills these two properties.

6. Significance

Computing systems typically use DRAM-based memories as main memory since DRAM provides large capacity and high performance. As the process technology scales down, DRAM technology faces challenges that impact its reliability and performance [102, 103]. Our HPCA 2017 paper [44] introduces SoftMC, a new DRAM characterization infrastructure that is flexible and practical to use. We release SoftMC as a publicly-available open-source tool [125]. In this section, we discuss the significance of our work by describing its novelty and long-term impact. We also discuss various future research directions in which SoftMC can be extended and applied.

6.1. Novelty

As we describe in Section 5, no prior DRAM testing infrastructure provides both *flexibility* and *ease of use* properties, which are critical for enabling widespread adoption of the infrastructure. Three different kinds of tools/infrastructures are available today for characterizing DRAM behavior, where each kind of tool has some shortcomings. We discuss these tools and their shortcomings in Section 5. In contrast to all these works, SoftMC allows for the implementation of a wide range of DRAM test routines and supports any off-the-shelf DRAM chip that is compatible with the DDR interface. SoftMC is also the first DRAM characterization tool that is freely available to public [118].

6.2. Research Directions Enabled by SoftMC

We believe SoftMC can enable many new studies of the behavior of DRAM and other memories. We briefly describe several examples in this section.

Enabling New Studies of DRAM Scaling and Failures. The SoftMC DRAM testing infrastructure can test any DRAM mechanism consisting of low-level DDR commands. Therefore, it enables a wide range of characterization and analysis studies of real DRAM modules that would otherwise not have been possible without such an infrastructure. We discuss three such example research directions.

First, as DRAM scales down to smaller technology nodes, it faces key challenges in both reliability and latency [26, 29, 55, 61, 62, 63, 66, 87, 88, 93, 99, 102, 103]. Unfortunately, there is no comprehensive experimental study that characterizes and analyzes the trends in DRAM cell operations and behavior with technology scaling across various DRAM generations. The SoftMC infrastructure can help us answer various questions to this end: How are the cell characteristics, reliability, and latency changing with different generations of technology nodes? Do all DRAM operations and cells get affected by scaling at the same rate? Which DRAM operations are getting worse?

Second, aging-related failures in DRAM can potentially affect the reliability and availability of systems in the field [95, 102, 106, 126]. However, the causes, characteristics, and impact of *aging* in real DRAM devices have remained largely unstudied. Using SoftMC, it is possible to devise controlled experiments to analyze and characterize DRAM aging. The SoftMC infrastructure can help us answer questions such as: How prevalent are aging-related failures? What types of usage accelerate aging? How can we design architectural techniques that can slow down the aging process?

Third, prior works show that the failure rate of DRAM modules in large data centers is significant, largely affecting the cost and downtime in data centers [92, 95, 126, 136]. Unfortunately, there is no study that analyzes DRAM modules that have failed in the field to determine the common causes of failure. Our SoftMC infrastructure can test faulty DRAM modules and help answer various research questions: What are the dominant types of DRAM failures at runtime? Are failures correlated to any location or specific structure in DRAM? Do all chips from the same generation exhibit the same failure characteristics? Do failures repeat?

Characterization of Non-Volatile Memory. The SoftMC infrastructure can test any chip compatible with the DDR interface. Such a design makes the scope of the chips that can be tested by SoftMC go well beyond just DRAM. With the emergence of byte-addressable non-volatile memories (e.g., phase-change memory [75, 76, 77, 94, 116, 119, 122, 146, 153], STT-MRAM [57, 74, 94, 107], RRAM/memristors [4, 30, 139, 147]), several vendors are working towards manufacturing DDR-compatible non-volatile memory chips at a large scale [36, 96]. When these chips become commercially available, it will be critical to characterize and analyze them in order to understand, exploit, and/or correct their behavior. We believe that SoftMC can be seamlessly used to characterize these chips, and can help enable future mechanisms for NVM.

SoftMC will hopefully enable other works that build on it in various ways. For example, future work can extend the infrastructure to enable researchers to analyze memory scheduling (e.g., [34, 40, 51, 70, 71, 78, 79, 97, 98, 100, 101, 104, 105, 124, 140, 154]) and memory power management [31, 32] mechanisms, and allow them to develop new mechanisms using a programmable memory controller and real workloads. SoftMC can also be used as a substrate for developing in-memory computation platforms and evaluating mechanisms for in-memory computation (e.g., [2, 3, 8, 9, 33, 35, 37, 48, 49, 56, 64, 73, 111, 113, 130, 131, 132, 133, 138]).

We conclude that characterization with SoftMC enables a wide range of research directions in DDR-compatible memory chips (DRAM or NVM), leading to better understanding of these technologies and helping to develop mechanisms that improve the reliability and performance of future memory systems.

7. Conclusion

This work introduces the first publicly-available FPGA-based DRAM testing infrastructure, *SoftMC* (Soft Memory Controller), which provides a programmable memory controller with a flexible and easy-to-use software interface. SoftMC enables the flexibility to test any standard DRAM operation and any (existing or new) mechanism comprising of such operations. It provides an intuitive high-level software interface for the user to invoke low-level DRAM operations, in order to minimize programming effort and time. We provide a prototype implementation of SoftMC, and we have released it publicly as a freely-available open-source tool [125].

We demonstrate the capability, flexibility, and programming ease of SoftMC by implementing two example use cases. Our experimental analyses demonstrate the effectiveness of SoftMC as a new tool to *(i)* perform detailed characterization of various DRAM parameters (e.g., refresh interval and access latency) as well as the relationships between them, and *(ii)* test the expected effects of existing or new mechanisms (e.g., whether or not highly-charged cells can be accessed faster in existing DRAM chips). We believe and hope that SoftMC, with its flexibility and ease of use, can enable many other studies, ideas and methodologies in the design of future memory systems, by making memory control and characterization easily accessible to a wide range of software and hardware developers.

Acknowledgments

We thank the reviewers, the SAFARI group members, and Shigeki Tomishima from Intel for their feedback. We acknowledge the generous support of Google, Intel, NVIDIA, Samsung, and VMware. This work is supported in part by NSF grants 1212962, 1320531, and 1409723, the Intel Science and Technology Center for Cloud Computing, and the Semiconductor Research Corporation.

References

[1] Advantest, "V6000 Memory Platform," https://www.advantest.com/products/ic-test-systems/v6000-memory.
[2] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[3] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[4] H. Akinaga and H. Shima, "Resistive Random Access Memory (ReRAM) Based on Metal Oxides," *Proc. IEEE*, 2010.
[5] P. Bernardi, M. Grosso, M. S. Reorda, and Y. Zhang, "A Programmable BIST for DRAM Testing and Diagnosis," in *ITC*, 2010.
[6] M. N. Bojnordi and E. Ipek, "PARDIS: A Programmable Memory Controller for the DDRx Interfacing Standards," in *ISCA*, 2012.

[7] S. Borkar and A. A. Chien, "The Future of Microprocessors," *CACM*, 2011.
[8] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[9] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *IEEE CAL*, 2017.
[10] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *SP*, 2016.
[11] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.
[12] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.
[13] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, 2017.
[14] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.
[15] Y. Cai, E. F. Haratsch, M. McCartney, and K. Mai, "FPGA-Based Solid-State Drive Prototyping Platform," in *FCCM*, 2011.
[16] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.
[17] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.
[18] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.
[19] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.
[20] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.
[21] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.
[22] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," *ITJ*, 2013.
[23] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," in *SIGMETRICS*, 2014.
[24] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.
[25] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2017.
[26] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[27] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[28] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[29] K. K. Chang, A. G. Yaălikçi, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," *SIGMETRICS*, 2017.
[30] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, 1971.
[31] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC*, 2011.
[32] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active Low-Power Modes for Main Memory," in *ASPLOS*, 2011.
[33] J. Draper *et al.*, "The Architecture of the DIVA Processing-in-Memory Chip," in *ICS*, 2002.
[34] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel Application Memory Scheduling," in *MICRO*, 2011.
[35] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational RAM: Implementing Processors in Memory," *IEEE DT*, 1999.
[36] EverSpin, "ST-MRAM," https://www.everspin.com/mram-replaces-dram.
[37] B. B. Fraguela, J. Renau, P. Feautrier, D. Padua, and J. Torrellas, "Programming the FlexRAM Parallel Intelligent Memory System," in *PPoPP*, 2003.
[38] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, "Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices," in *DFRWS EU*, 2017.
[39] FuturePlus, "FS2800 DDR Detective," http://www.futureplus.com/DDR-Detective-Standalone/summary-2800.html.
[40] S. Ghose, H. Lee, and J. F. Martínez, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.
[41] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.
[42] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *Electron Devices*, 1998.
[43] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[44] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[45] P. Hazucha and C. Svensson, "Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate," *TNS*, 2000.
[46] H. Hidaka, K. Fujishima, Y. Matsuda, M. Asakura, and T. Yoshihara, "Twisted Bit-Line Architectures for Multi-Megabit DRAMs," *JSSC*, 1989.
[47] C. Hou, J.-F. Li, C.-Y. Lo, D.-M. Kwai, Y.-F. Chou, and C.-W. Wu, "An FPGA-Based Test Platform for Analyzing Data Retention Time Distribution of DRAMs," in *VLSI-DAT*, 2013.
[48] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[49] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.
[50] J. Huang, C.-K. Ong, K.-T. Cheng, and C.-W. Wu, "An FPGA-Based Re-Configurable Functional Tester for Memory Chips," in *ATS*, 2000.
[51] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.
[52] K. Itoh, *VLSI Memory Chip Design.* Springer Science & Business Media, 2013.
[53] JEDEC, "204-Pin DDR3 SDRAM Unbuffered SODIMM Design Specification," 2014. https://www.jedec.org/standards-documents/docs/module-42018
[54] T. S. Jung, "Memory Technology and Solutions Roadmap," http://www.sec.co.kr/images/corp/ir/irevent/techforum_01.pdf, 2005.
[55] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. Choi, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[56] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an Advanced Intelligent Memory System," in *ICCD*, 2012.
[57] T. Kawahara *et al.*, "2 Mb SPRAM (SPin-Transfer Torque RAM) with bit-by-bit bi-directional current write and parallelizing-direction current read," *JSSC*, 2008.
[58] B. Keeth, *DRAM Circuit Design: Fundamental and High-Speed Topics.* John Wiley & Sons, 2008.
[59] D. Keezer, T. Chen, T. Moon, D. Stonecypher, A. Chatterjee, H. Choi, S. Kim, and H. Yoo, "An FPGA-Based ATE Extension Module for Low-Cost Multi-GHz Memory Test," in *ETS*, 2015.
[60] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[61] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[62] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *CAL*, 2016.
[63] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[64] J. S. Kim, D. Senol, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.
[65] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[66] K. Kim, "Technology for Sub-50nm DRAM and NAND Flash Manufacturing," in *IEDM*, 2005.
[67] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *EDL*, 2009.
[68] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[69] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[70] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[71] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MI-*

CRO, 2010.
[72] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A Fast and Extensible DRAM Simulator,” in *CAL*, 2015.
[73] P. M. Kogge, “EXECUBE-a New Architecture for Scaleable MPPs,” in *ICPP*, 1994.
[74] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, “Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative,” in *ISPASS*, 2013.
[75] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, “Phase Change Technology and the Future of Main Memory,” *IEEE Micro*, 2010.
[76] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting Phase Change Memory as a Scalable DRAM Alternative,” in *ISCA*, 2009.
[77] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Phase Change Memory Architecture and the Quest for Scalability,” *CACM*, 2010.
[78] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, “Improving Memory Bank-Level Parallelism in the Presence of Prefetching,” in *MICRO*, 2009.
[79] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, “Prefetch-Aware DRAM Controllers,” in *MICRO*, 2008.
[80] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, “Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case,” in *HPCA*, 2015.
[81] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, “Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture,” in *HPCA*, 2013.
[82] D. Lee, “Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity,” Ph.D. dissertation, Carnegie Mellon Univ., 2016.
[83] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, “Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost,” in *TACO*, 2016.
[84] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, “Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms,” *SIGMETRICS*, 2017.
[85] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, “Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM,” in *PACT*, 2015.
[86] M. Lee and K. W. Park, “A Mechanism for Dependence of Refresh Time on Data Pattern in DRAM,” *EDL*, 2010.
[87] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “RAIDR: Retention-Aware Intelligent DRAM Refresh,” in *ISCA*, 2012.
[88] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms,” in *ISCA*, 2013.
[89] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, “WARM: Improving NAND Flash Memory Lifetime with Write-Hotness Aware Retention Management,” in *MSST*, 2015.
[90] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, “Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory,” *JSAC*, 2016.
[91] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, “HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness,” in *HPCA*, 2018.
[92] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, “Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory,” in *DSN*, 2014.
[93] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, “Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM),” *IBM JRD*, 2002.
[94] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, “A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory,” in *WEED*, 2013.
[95] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, “Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field,” in *DSN*, 2015.
[96] Micron, “3D XPoint Memory,” http://www.micron.com/about/innovations/3d-xpoint-technology, 2016.
[97] T. Moscibroda and O. Mutlu, “Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems,” in *USENIX Security*, 2007.
[98] T. Moscibroda and O. Mutlu, “Distributed Order Scheduling and Its Application to Multi-core DRAM Controllers,” in *PODC*, 2008.
[99] W. Mueller *et al.*, “Challenges for the DRAM Cell Scaling to 40nm,” in *IEDM*, 2005.
[100] J. Mukundan and J. F. Martinez, “MORSE: Multi-objective Reconfigurable Self-optimizing Memory Scheduler,” in *HPCA*, 2012.
[101] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, “Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning,” in *MICRO*, 2011.
[102] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” *IMW*, 2013.
[103] O. Mutlu, “The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser,” in *DATE*, 2017.
[104] O. Mutlu and T. Moscibroda, “Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors,” in *MICRO*, 2007.
[105] O. Mutlu and T. Moscibroda, “Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems,” in *ISCA*, 2008.
[106] O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” *SUPERFRI*, 2014.
[107] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, “STT-RAM Scaling and Retention Failure,” *Intel Technology Journal*, 2013.
[108] Y. Nakagome, M. Aoki, S. Ikenaga, M. Horiguchi, S. Kimura, Y. Kawamoto, and K. Itoh, “The Impact of Data-Line Interference Noise on DRAM Scaling,” *JSSC*, 1988.
[109] S. Nassif, “Delay Variability: Sources, Impacts and Trends,” in *ISSCC*, 2000.
[110] Nickel Electronics, “DRAM Memory Testing,” https://www.nickelelectronics.com/memory-testing/.
[111] M. Oskin, F. T. Chong, and T. Sherwood, “Active Pages: A Computation Model for Intelligent Memory,” in *ISCA*, 1998.
[112] M. Patel, J. S. Kim, and O. Mutlu, “The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions,” in *ISCA*, 2017.
[113] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, “A Case for Intelligent RAM,” *IEEE Micro*, 1997.
[114] B. Querbach, R. Khanna, D. Blankenbeckler, Y. Zhang, R. T. Anderson, D. G. Ellis, Z. T. Schoenborn, S. Deyati, and P. Chiang, “A Reusable BIST with Software Assisted Repair Technology for Improved Memory and IO Debug, Validation and Test Time,” in *ITC*, 2014.
[115] B. Querbach, R. Khanna, S. Puligundla, D. Blankenbeckler, J. Crop, and P. Chiang, “Architecture of a Reusable BIST Engine for Detection and Auto Correction of Memory Failures and for IO Debug, Validation, Link Training, and Power Optimization on 14nm SOC,” *IEEE D&T*, 2016.
[116] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable High Performance Main Memory System Using Phase-Change Memory Technology,” in *ISCA*, 2009.
[117] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, “AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems,” in *DSN*, 2015.
[118] Ramulator Source Code, https://github.com/CMU-SAFARI/ramulator.
[119] S. Raoux *et al.*, “Phase-Change Random Access Memory: A Scalable Technology,” *IBM JRD*, 2008.
[120] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, “Flip Feng Shui: Hammering a Needle in the Software Stack,” in *USENIX Security*, 2016.
[121] M. Redeker, B. F. Cockburn, and D. G. Elliott, “An Investigation into Crosstalk Noise in DRAM Structures,” in *MTDT*, 2002.
[122] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu, “ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems,” in *MICRO*, 2015.
[123] P. J. Restle, J. W. Park, and B. F. Lloyd, “DRAM Variable Retention Time,” in *IEDM*, 1992.
[124] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory Access Scheduling,” in *ISCA*, 2000.
[125] SAFARI Research Group, “SoftMC – GitHub Repository,” https://github.com/CMU-SAFARI/SoftMC.
[126] B. Schroeder, E. Pinheiro, and W.-D. Weber, “DRAM Errors in the Wild: A Large-Scale Field Study,” in *SIGMETRICS*, 2009.
[127] M. Seaborn and T. Dullien, “Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges,” https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.
[128] M. Seaborn and T. Dullien, “Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges,” in *Black Hat*, 2015.
[129] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, “Fast Bulk Bitwise AND and OR in DRAM,” in *CAL*, 2015.
[130] V. Seshadri *et al.*, “RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization,” in *MICRO*, 2013.
[131] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, “Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology,” in *MICRO*, 2017.
[132] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, “Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses,” in *MICRO*, 2015.
[133] D. E. Shaw, S. J. Stolfo, H. Ibrahim, B. Hillyer, G. Wiederhold, and J. Andrews, “The NON-VON Database Machine: A Brief Overview,” *IEEE DEB*, 1981.
[134] W. Shin, J. Yang, J. Choi, and L.-S. Kim, “NUAT: A Non-Uniform Access Time Memory Controller,” in *HPCA*, 2014.
[135] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, “Memory Errors in Modern Systems: The Good, the Bad, and the Ugly,” in *ASPLOS*, 2015.
[136] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, “Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults,” in *SC*, 2013.
[137] G. Srinivasan, P. Murley, and H. Tang, “Accurate, Predictive Modeling of Soft Error Rate Due to Cosmic Rays and Chip Alpha Radiation,” in *IRPS*, 1994.
[138] H. S. Stone, “A Logic-in-Memory Computer,” *IEEE TC*, 1970.
[139] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The Missing Memristor Found,” *Nature*, 2008.
[140] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, “The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost,” in

ICCD, 2014.

[141] B. Sukhwani, T. Roewer, C. L. Haymes, K.-H. Kim, A. J. McPadden, D. M. Dreps, D. Sanner, J. Van Lunteren, and S. Asaad, "ConTutto: A Novel FPGA-Based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor," in *MICRO*, 2017.

[142] Teradyne, "Magnum Memory Test System," http://www.teradyne.com/products/semiconductor-test/magnum.

[143] S. Tomishima, Personal Communication, Dec. 2016.

[144] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.

[145] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.

[146] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.

[147] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.

[148] Y. Xiao, X. Zhang, Y. Zhang, and M. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.

[149] D. S. Yaney, C.-Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson, "A Meta-Stable Leakage Phenomenon in DRAM Charge Storage - Variable Hold Time," in *IEDM*, 1987.

[150] C. Yang, J.-F. Li, Y.-C. Yu, K.-T. Wu, C.-Y. Lo, C.-H. Chen, J.-S. Lai, D.-M. Kwai, and Y.-F. Chou, "A Hybrid Built-In Self-Test Scheme for DRAMs," in *VLSI-DAT*, 2015.

[151] H. Yang, S.-H. Kuo, T.-H. Huang, C.-H. Chen, C. Lin, and M. C.-T. Chao, "Random Pattern Generation for Post-Silicon Validation of DDR3 SDRAM," in *VTS*, 2015.

[152] Y. You and J. Hayes, "A Self-Testing Dynamic RAM Chip," *TED*, 1985.

[153] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.

[154] W. K. Zuravleff and T. Robinson, "Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order," US Patent No. 5,630,096, 1997.

RowClone: Accelerating Data Movement and Initialization Using DRAM

Vivek Seshadri[1,2] Yoongu Kim[2] Chris Fallin[2] Donghyuk Lee[3,2]
Rachata Ausavarungnirun[2] Gennady Pekhimenko[4,2] Yixin Luo[2]
Onur Mutlu[5,2] Phillip B. Gibbons[2,6] Michael A. Kozuch[6] Todd C. Mowry[2]

[1]*Microsoft Research India* [2]*Carnegie Mellon University* [3]*NVIDIA Research*
[4]*University of Toronto* [5]*ETH Zürich* [6]*Intel Labs*

This paper summarizes the idea of RowClone, which was published in MICRO 2013 [151], and examines the work's significance and future potential. In existing systems, to perform any bulk data movement operation (copy or initialization), the data has to first be read into the on-chip processor, all the way into the L1 cache, and the result of the operation must be written back to main memory. This is despite the fact that these operations do not involve any actual computation. RowClone exploits the organization and operation of commodity DRAM to perform these operations completely inside DRAM using two mechanisms. The first mechanism, Fast Parallel Mode, copies data between two rows inside the same DRAM subarray by issuing back-to-back activate commands to the source and the destination row. The second mechanism, Pipelined Serial Mode, transfers cache lines between two banks using the shared internal bus. RowClone significantly reduces the raw latency and energy consumption of bulk data copy and initialization. This reduction directly translates to improvement in performance and energy efficiency of systems running copy or initialization-intensive workloads.

Our proposed technique has inspired significant research on various ways to perform operations in memory and reduce data movement between the CPU and DRAM [2, 25, 69, 76, 102, 103, 153, 154, 157, 162].

1. Problem: Bulk Data Movement

The main memory subsystem is an increasingly more significant limiter of system performance and energy efficiency [123, 124] for at least two reasons. First, the available memory bandwidth between the processor and main memory is not growing and nor is it expected to grow commensurately with the compute bandwidth available in modern multi-core processors [61, 64]. Second, a significant fraction (20% to 42%) of the energy required to access data from memory is consumed in driving the high-speed bus connecting the processor and memory [149] (calculated using [112]). Therefore, judicious use of the available memory bandwidth is critical to ensure both high system performance and energy efficiency.

In this work, we focus our attention on optimizing two important classes of bandwidth-intensive memory operations that frequently occur in modern systems: 1) *bulk data copy*—copying a large quantity of data from one location in physical memory to another, and 2) *bulk data initialization*—initializing a large quantity of data to a specific value. We refer to these two operations as *bulk data movement operations.* Prior research [68, 131, 147] has shown that operating systems and data center workloads spend a significant portion of their time performing bulk data movement operations. Therefore, accelerating these operations will likely improve system performance. In fact, the x86 ISA has recently introduced instructions to provide enhanced performance for bulk copy and initialization (ERMSB [60]), highlighting the importance of bulk operations.

The main reason bulk data movement operations degrade system performance and energy efficiency is that they require large amounts of data to be transferred back and forth on the memory bus. This large data transfer has three shortcomings. First, because the data is transferred one cache line at a time across the bus, these operations incur high latency, directly degrading the performance of the application performing the operation. Second, transferring a large amount of data on the bus interferes with the memory accesses of other concurrently-running applications, degrading their performance as well. Finally, the large data transfer contributes to a significant fraction of the energy consumed by these bulk movement operations.

While bulk data movement operations also degrade performance by hogging the CPU and potentially polluting the on-chip caches, prior works [66, 192] have proposed simple solutions to address these problems by adding support for such operations in the memory controller. However, the techniques proposed by these works do *not* eliminate the need to transfer data over the memory bus, which is a increasingly more critical bottleneck for performance in modern systems.

2. RowClone: Fast In-DRAM Copy

The fact that both bulk data copy and initialization do *not* require any computation on the part of the processor enables the opportunity to perform these operations *completely* inside DRAM. Our MICRO 2013 paper [151] presents a new mechanism, RowClone, which exploits the internal organization and operation of DRAM to perform bulk data copy/initialization quickly and efficiently inside DRAM.

Figure 1 illustrates the organization of a DRAM chip. The chip contains multiple banks, each of which is divided into subarrays, and each subarray in turn consists of multiple rows of DRAM cells. Each subarray contains a *row buffer*, which is used to extract the data from the DRAM cells. Data transfer between the DRAM cells and the row buffer happen at a row granularity, i.e., even to read a single byte from a row, the chip copies the entire row of data from the DRAM cells to the corresponding row buffer.[1]

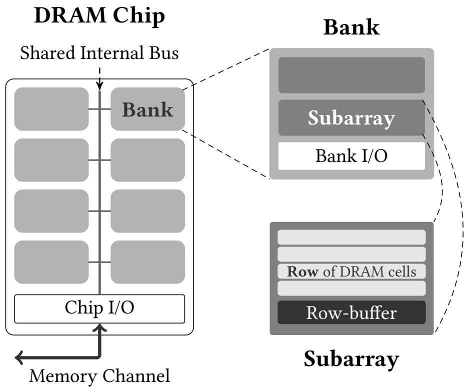


Figure 1: DRAM chip microarchitecture. Reproduced from [151].

2.1. RowClone Mechanisms

RowClone consists of two mechanisms: (1) *Fast Parallel Mode* (FPM), which is used to copy data from one row to another row in the *same* subarray; and (2) *Pipelined Serial Mode* (PSM), which is used to copy data from one row to another row in a *different* subarray or bank. We briefly discuss how each mechanism performs bulk data copy and bulk data initialization. Section 3 of our MICRO 2013 paper [151] provides a detailed implementation and discussion of FPM and PSM.

Fast Parallel Mode (FPM). FPM uses the high internal bandwidth offered by DRAM to quickly and efficiently copy data between two rows within the same subarray in two simple steps. First, FPM copies the data from the source row to the local row buffer of the subarray. Second, FPM copies the data from the row buffer to the destination row. To perform the copy, FPM simply issues two back-to-back `ACTIVATE` commands to the bank, first with the source row address and the second with the destination row address. Implementing this in existing DRAM chips requires almost negligible changes. These small changes are to the peripheral logic that controls back-to-back `ACTIVATE`s.

FPM imposes two constraints on the copy operation. First, it requires the source and the destination row to be within the same subarray. Second, it copies the entire row's worth of data. It cannot partially copy data from one row to another. Despite these constraints, FPM can be used to accelerate many operations in modern systems (Section 3).

Pipelined Serial Mode (PSM). PSM accelerates copy operations between rows in *different* banks/subarrays, As shown in Figure 1, each DRAM chip uses a shared internal bus to transfer data between the bank and the memory channel (for both reads and writes). PSM exploits this fact to overlap the latency of the read and write operations involved in a copy. To implement PSM, we propose a new DRAM command called `TRANSFER`. `TRANSFER` is equivalent to appropriately overlapping `READ` to the source bank and `WRITE` to the destination bank. However, unlike `READ` or `WRITE`, `TRANSFER` does not transfer the data on to the memory channel, saving significant amounts of energy.

Bulk Data Initialization. For bulk initialization, RowClone initializes one row of the destination with the required data and then initializes the remaining rows by copying the data from the pre-initialized row using the appropriate bulk copy mechanism described above. For bulk zeroing (which happens frequently), our mechanism reserves a single row in each subarray, which is pre-initialized to zero. This enables the memory controller to use FPM to zero out any row in the system. We refer the reader to Section 3.4 of our MICRO 2013 paper [151] for more details on performing bulk data initialization with RowClone.

2.2. Latency and Energy Benefits

Table 1 shows the reduction in latency and energy consumption due to our mechanisms for different cases of 4KB copy and zeroing operations. To be fair to the baseline, the results include only the energy consumed by the DRAM and the DRAM channel. We draw two conclusions from our results.

Table 1: DRAM latency and memory energy reductions due to RowClone. Adapted from [151].

		Latency		Memory Energy	
	Mechanism	(ns)	(↓)	(μJ)	(↓)
Copy	Baseline	1046	1.0x	3.6	1.0x
	FPM	90	**11.6x**	0.04	**74.4x**
	Inter-Bank - PSM	540	1.9x	1.1	3.2x
	Intra-Bank - PSM	1050	1.0x	2.5	1.5x
Zero	Baseline	546	1.0x	2.0	1.0x
	FPM	90	**6.0x**	0.05	**41.5x**

First, FPM significantly improves both the latency and the energy consumed by bulk data operations — 11.6x and 6x reduction in latency of 4KB copy and zeroing, and 74.4x and 41.5x reduction in memory energy of 4KB copy and zeroing. Second, although PSM does not provide as much benefit as FPM, it still reduces the latency and energy of a 4KB inter-bank copy by 1.9x and 3.2x, while providing a more generally applicable mechanism. As we show in Section 4, these latency and energy benefits translate to significant improvements in both overall system performance and energy efficiency.

2.3. End-to-End System Design

To fully extract the potential benefits of RowClone, changes are required to the ISA, processor microarchitecture, and the system software. First, we introduce two new instructions to the ISA, namely, `memcopy` and `meminit`, which enable the

[1]We refer the reader to our prior works [25, 26, 27, 28, 53, 54, 77, 78, 79, 80, 81, 82, 96, 97, 98, 99, 100, 108, 109, 132, 151, 154] for a detailed background on DRAM.

software to indicate occurrences of bulk data operations to the processor. Second, for each instance of the `memcopy/meminit` instruction, the processor microarchitecture determines if the operation can be partially/fully accelerated by RowClone and issues appropriate commands to the memory controller. While existing mechanisms to handle Direct Memory Access requests can be used to ensure cache coherence with RowClone, we also propose two simple mechanisms, called *in-cache copy* and *clean zero cache line insertion*, to further reduce memory bandwidth requirements and improve performance. We call this optimized version of RowClone, which includes in-cache copy and clean zero cache line insertion, *RowClone-ZI*. Third, to maximize the use of FPM, we make the system software aware of subarrays and the minimum granularity of copy (required by FPM). Section 4 of our MICRO 2013 paper [151] describes these changes in detail.

3. Applications

RowClone can be used to accelerate any bulk copy and initialization operation to improve both system performance and energy efficiency. We quantitatively evaluate the efficacy of RowClone by using it to accelerate two primitives widely used by modern system software: 1) Copy-on-Write and 2) Bulk Zeroing. We first describe these primitives, and then discuss several applications that frequently trigger the primitives.

3.1. Primitives Accelerated by RowClone

Copy-on-Write (CoW) is a technique used by most modern operating systems (OS) to postpone an expensive copy operation until it is actually needed. When data of one virtual page needs to be copied to another, instead of creating a copy, the OS points both virtual pages to the same physical page (source) and marks the page as read-only. In the future, when one of the sharers attempts to write to the page, the OS allocates a new physical page (destination) for the writer and copies the contents of the source page to the newly allocated page. Fortunately, prior to allocating the destination page, the OS already knows the location of the source physical page. Therefore, it can ensure that the destination is allocated in the same subarray as the source, thereby enabling the processor to use FPM to perform the copy.

Bulk Zeroing (BuZ) is an operation where a large block of memory is zeroed out. Our mechanism maintains a reserved row that is fully initialized to zero in each subarray. For each row in the destination region to be zeroed out, the processor uses FPM to copy the data from the reserved zero-row of the corresponding subarray to the destination row.

3.2. Applications That Use CoW/BuZ

We now describe seven example applications or use-cases that extensively use the CoW or BuZ operations. Note that these are just a small number of example scenarios that incur a large number of copy and initialization operations. Some other applications and scenarios are provided in one of our more recent works [155]. Recent work from Google [68] shows that a considerable fraction of execution time is spent on `memset` and `memcpy` system calls in Google's data center workloads.

Process Forking. `fork` is a frequently-used system call in modern operating systems (OS). When a process (parent) calls `fork`, it creates a new process (child) with the exact same memory image and execution state as the parent. This semantics of `fork` makes it useful for different scenarios. Common uses of the `fork` system call are to 1) create new processes, and 2) create stateful threads from a single parent thread in multi-threaded programs. One main limitation of `fork` is that it results in a CoW operation whenever the child/parent updates a shared page. Hence, despite its wide usage, as a result of the large number of copy operations triggered by `fork`, it remains one of the most expensive system calls in terms of memory performance [150].

Initializing Large Data Structures. Initializing large data structures often triggers Bulk Zeroing. In fact, many managed languages (e.g., C#, Java, PHP) require zero initialization of variables to ensure memory safety [185]. In such cases, to reduce the overhead of zeroing, memory is zeroed-out in bulk.

Secure Deallocation. Most operating systems (e.g., Linux [18], Windows [148], Mac OS X [166]) zero out pages newly allocated to a process. This is done to prevent malicious processes from gaining access to the data that previously belonged to other processes or the kernel itself. Not doing so can potentially lead to security vulnerabilities, as shown by prior works [31, 41, 51, 52].

Process Checkpointing. Checkpointing is an operation during which a consistent version of a process state is backed-up, so that the process can be restored from that state in the future. This checkpoint-restore primitive is useful in many cases including high-performance computing servers [15], software debugging with reduced overhead [168], hardware-level fault and bug tolerance mechanisms [33, 34, 105, 106, 107], and speculative OS optimizations to improve performance [24, 182]. However, to ensure that the checkpoint is consistent (i.e., the original process does not update data while the checkpointing is in progress), the pages of the process are marked with copy-on-write. As a result, checkpointing often results in a large number of CoW operations.

Virtual Machine Cloning/Deduplication. Virtual machine (VM) cloning [88] is a technique to significantly reduce the startup cost of VMs in a cloud computing server. Similarly, deduplication is a technique employed by modern hypervisors [180] to reduce the overall memory capacity requirements of VMs. With this technique, different VMs share physical pages that contain the same data. Similar to forking, both these operations likely result in a large number of CoW operations for pages shared across VMs [155].

Page Migration. Bank conflicts, i.e., concurrent requests to different rows within the same bank, typically result in reduced row buffer hit rate and hence degrade both system performance and energy efficiency [80]. Prior work [175] proposed techniques to mitigate bank conflicts using page migration. The PSM mode of RowClone can be used in conjunction with such techniques to 1) significantly reduce the migration latency and 2) make the migrations more energy-efficient.

CPU-GPU Communication. In many current and future processors, the GPU is or is expected to be integrated on the same chip with the CPU. Even in such systems where the CPU and GPU share the same off-chip memory, the off-chip memory is partitioned between the two devices. As a consequence, whenever a CPU program wants to offload some computation to the GPU, it has to copy all the necessary data from the CPU address space to the GPU address space [62]. When the GPU computation is finished, all the data needs to be copied back to the CPU address space. This copying involves a significant overhead. By spreading out the GPU address space over all subarrays and mapping the application data appropriately, RowClone can significantly speed up these copy operations. Note that communication between different processors and accelerators in a heterogeneous system-on-chip (SoC) is done similarly to the CPU-GPU communication and can also be accelerated by RowClone.

4. Results

In this section, we briefly summarize our evaluation of RowClone. We evaluate three configurations: *Baseline*, an unmodified main memory subsystem that cannot perform bulk data copy or initialization within memory; *RowClone*, which uses the FPM and PSM mechanisms described in Section 2.1; and *RowClone-ZI*, an optimized version of RowClone that includes the two optimizations discussed in Section 2.3. Section 6 of our MICRO 2013 paper [151] discusses our full evaluation methodology, including details on the simulator, system configuration, and benchmarks used for our evaluations.

4.1. Single-Core Evaluations

Figure 2 shows the performance improvement and reduction in DRAM energy consumption due to RowClone-ZI compared to the baseline for six copy- and initialization-intensive benchmarks. As we observe from the figure, these applications improve significantly with RowClone-ZI. Compared with Baseline, RowClone-ZI improves the IPC by up to 43%, while reducing DRAM energy consumption by up to 67%.

Section 7 of our MICRO 2013 paper [151] provides more detailed single-core results, including (1) the individual performance of the FPM and PSM mechanisms using a fork benchmark (Section 7.2 of [151]); (2) a breakdown of memory traffic for each application into read, write, copy, and initialization operations (Section 7.3 of [151]); (3) the performance,

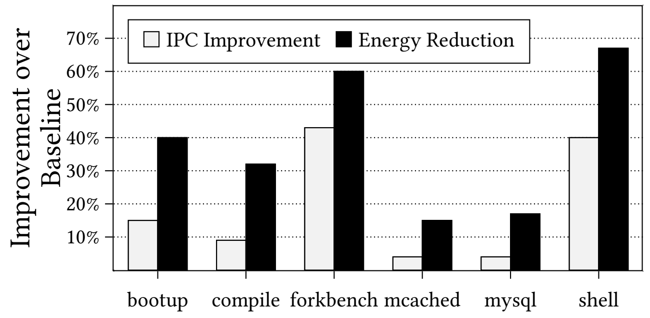


Figure 2: Performance improvement and energy reduction of RowClone-ZI compared to a baseline memory subsystem without bulk copy support.

energy, and bandwidth improvements of both RowClone and RowClone-ZI (Section 7.3 of [151]); and (4) a comparison of RowClone to a memory-controller-based DMA approach for data copy and initialization, similar to [192] (Section 7.5 of [151]).

4.2. Multi-Core Evaluations

As RowClone performs bulk data operations completely within DRAM, it significantly reduces the memory bandwidth consumed by these operations. As a result, RowClone can benefit other applications that are running concurrently on the same system, even if these applications do not perform bulk data operations themselves. We evaluate this benefit of RowClone by running our copy/initialization-intensive applications alongside memory-intensive applications from the SPEC CPU2006 benchmark suite [169] (i.e., those applications with last-level cache misses per kilo-instruction, or MPKI, greater than 1). Table 2 lists the set of applications used for our multi-programmed workloads.

Table 2: List of benchmarks used for multi-core evaluation. Reproduced from [151].

Copy/Initialization-intensive benchmarks
bootup, compile, forkbench, mcached, mysql, shell
Memory-intensive benchmarks from SPEC CPU2006
bzip2, gcc, mcf, milc, zeusmp, gromacs, cactusADM, leslie3d, namd, gobmk, dealII, soplex, hmmer, sjeng, GemsFDTD, libquantum, h264ref, lbm, omnetpp, astar, wrf, sphinx3, xalancbmk

We generate multi-programmed workloads for two-core, four-core and eight-core systems. In each workload, half of the cores run copy/initialization-intensive benchmarks, while the remaining cores run memory-intensive SPEC benchmarks. Benchmarks from each category are chosen at random.

Figure 3 plots the performance improvement due to RowClone and RowClone-ZI for the 50 four-core workloads that we evaluate (sorted based on the performance improvement due to RowClone-ZI). Two conclusions are in order. First, although RowClone degrades performance of certain four-core workloads (with *compile*, *mcached* or *mysql* benchmarks), it significantly improves performance for all other workloads (by 10% across all workloads). Second, RowClone-ZI eliminates the performance degradation due to RowClone and consistently outperforms both the baseline and RowClone for all workloads (20% on average).

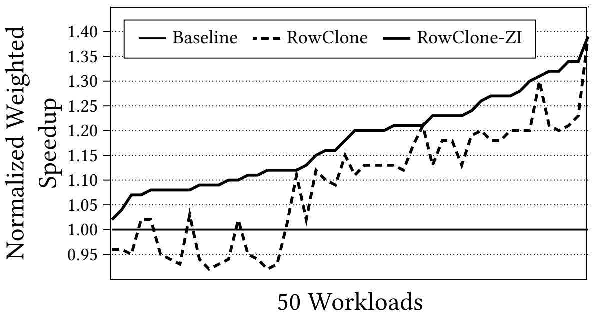


Figure 3: System performance improvement of RowClone for four-core workloads. Reproduced from [151].

To provide more insight into the benefits of RowClone on multi-core systems, we classify our copy/initialization-intensive benchmarks into two categories: 1) Moderately copy/initialization-intensive (*compile*, *mcached*, and *mysql*) and highly copy/initialization-intensive (*bootup*, *forkbench*, and *shell*). Figure 4 shows the average improvement in weighted speedup for the different multi-core workloads, categorized based on the number of highly copy/initialization-intensive benchmarks. As the trends indicate, RowClone's performance improvement increases with increasing number of such benchmarks for all three multi-core systems, indicating the effectiveness of RowClone in accelerating bulk copy/initialization operations.

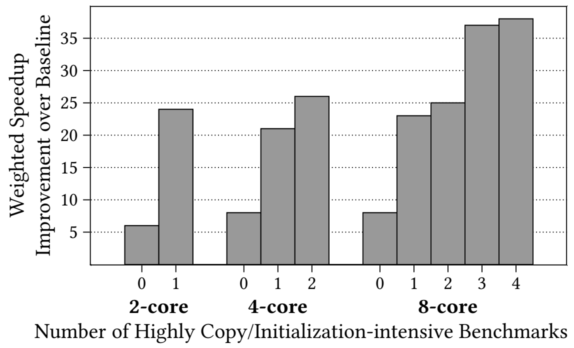


Figure 4: Effect of increasing copy/initialization intensity. Reproduced from [151].

We conclude that RowClone is an effective mechanism to improve system performance, energy efficiency and bandwidth efficiency of future, bandwidth-constrained multi-core systems.

5. Related Work

To our knowledge, this is the first paper to propose a concrete mechanism to perform bulk data copy and initialization operations completely in DRAM. In this section, we discuss related work and qualitatively compare them to RowClone. Other treatments of related works can be found in [156, 158, 159].

Patents on Data Copy in DRAM. Several patents [3, 48, 113, 114] propose the abstract notion that the row buffer in DRAM can be used to copy data from one row to another. These patents have four major drawbacks. First, they do not provide any concrete mechanism to perform the copy operation. Second, while using the row buffer to copy data between two rows is possible only when the two rows are within the same subarray, these patents make no such distinction. Third, these patents do not discuss the support required from the other layers of the system to realize a working system. Fourth, these patents do not provide any concrete evaluation to show the benefits of performing copy operations in DRAM. In contrast, RowClone is more generally applicable, and our MICRO 2013 paper [151] discusses the concrete changes required to *all layers* of the system stack, from the DRAM architecture to the system software, to enable bulk data copy.

Offloading Copy/Initialization Operations. Prior works [66, 192] propose mechanisms to 1) offload bulk data copy/initialization operations to a separate engine; 2) reduce the impact of pipeline stalls (by waking up instructions dependent on a copy operation as soon as the necessary blocks are copied without waiting for the entire copy operation to complete); and 3) reduce cache pollution by using hints from software to decide whether to cache blocks involved in the copy or initialization. While Section 7.5 of our MICRO 2013 paper [151] shows the effectiveness of RowClone compared to offloading bulk data operations to a separate engine, techniques to reduce pipeline stalls and cache pollution [66] can be naturally combined with RowClone to further improve performance.

Low-cost Interlinked Sub-Arrays (LISA) [25] proposes to connect adjacent subarrays inside a DRAM bank using a set of isolation transistors. Using this structure, LISA proposes mechanisms to efficiently copy data across rows in different subarrays within the same bank. LISA and RowClone can be combined to perform all bulk copy and initialization operations efficiently inside DRAM. However, unlike LISA, RowClone does *not* require any changes to the DRAM array.

The Compute Cache [2] performs copy, zero, and bitwise operations completely inside the on-chip SRAM cache. Like RowClone, the Compute Cache exploits the fact that many cells are connected to the same bitline to efficiently perform these operations across cells connected to the same bitline. Again, depending on the location of the data, RowClone and Compute Cache can be combined to further improve system performance and efficiency.

Bulk Memory Initialization. Jarrod et al. [63] propose a mechanism for avoiding the memory access required to fetch uninitialized blocks on a store miss. They use a specialized cache to keep track of uninitialized regions of memory. RowClone can potentially be combined with this mechanism. While Jarrod et al.'s approach can be used to reduce bandwidth consumption for irregular initialization (initializing different pages with different values), RowClone can be used to push regular initialization (e.g., initializing multiple pages with the same values) to DRAM, thereby freeing up the CPU to perform other useful operations.

Yang et al. [185] propose to reduce the cost of zero initialization by 1) using non-temporal store instructions to avoid cache pollution, and 2) using idle cores/threads to perform zeroing ahead of time. While the proposed optimizations reduce the negative performance impact of zeroing, their mechanism does *not* reduce memory bandwidth consumption of the bulk zeroing operations. In contrast, RowClone significantly reduces the memory bandwidth consumption and the associated energy overhead.

Processing-in-Memory. Recent works propose mechanisms that exploit the internal organization and operation of DRAM [102, 153, 154], SRAM [2, 69], phase-change memory (PCM) [103], or memristors [162] to perform bulk bitwise Boolean algebra and/or simple arithmetic operations. One such mechanism, called Ambit [153, 154], uses a number of row copy and initialization operations to perform Boolean algebra using DRAM. Ambit makes use of RowClone to efficiently perform these row copy and initialization operations. Another mechanism, the Compute Cache [2], can perform copy and initialization operations within SRAM. Other mechanisms for in-memory Boolean algebra or arithmetic [69, 102, 103, 162] can be trivially used to perform data copy and initialization operations (e.g., a data copy can be performed by performing a bulk addition, where the row to be copied is added to a row of all zeroes).

Various prior works (e.g., [6,7,16,17,49,55,56,76,83,110,133, 135,188]) have investigated mechanisms to add logic circuitry closer to memory to perform bandwidth-intensive computations (e.g., SIMD vector operations) more efficiently. The main limitation of such approaches is that adding logic to or near DRAM significantly increases the cost of main memory. In contrast, RowClone exploits the *existing* internal organization and operation of DRAM to perform bandwidth-intensive copy and initialization operations quickly and efficiently with low cost.

Other Methods for Lowering Memory Latency. There are many works that improve the performance of applications by reducing the *overall memory access latency*. These works enable more parallelism and bandwidth [4,5,27,80,97,100,153, 154,181,189,193], exploit latency variation within DRAM [23, 26,28,96,98,99], reduce refresh counts [71,72,74,75,108,109, 141, 178], enable better communication between the CPU and other devices through DRAM [100], leverage DRAM access patterns to reduce access latency [54, 165], reduce write-related latencies by better designing DRAM and DRAM control policies [30,92,152], reduce overall queuing latencies in DRAM by better scheduling memory requests [13,14,37, 45, 47, 57, 61, 67, 70, 78, 79, 93, 94, 95, 104, 115, 116, 117, 118, 125,126,130,135,146,164,171,172,173,174,177,191], employ prefetching [12, 22, 35, 36, 40, 43, 44, 46, 93, 119, 120, 121, 122, 127, 129, 134, 167], perform memory/cache compression [1, 10, 11, 38, 39, 42, 136, 137, 138, 139, 140, 163, 179, 183, 190], or perform better caching [73, 142, 144, 160, 161]. RowClone is orthogonal to all of these approaches, and can be combined with any of them with them to achieve higher latency and energy benefits.

6. Significance

Our MICRO 2013 paper [151] proposes RowClone, a simple mechanism to export bulk copy and initialization operations to DRAM. In this section, we describe the novelty of our approach, the long term impact of our proposed techniques, and new research directions triggered by our work.

6.1. Novelty

Prior works investigate mechanisms to add logic closer to memory to perform bandwidth-intensive operations more efficiently. Although this approach has the potential to be used for a wide range of applications, it has two shortcomings. First, adding logic to DRAM increases the cost of DRAM significantly. Second, this approach does *not* reduce the bandwidth requirement of simple bulk copy/initialization operations.

In contrast, our work is the first (to our knowledge) to propose mechanisms that exploit the internal organization and operation of DRAM to perform bandwidth-intensive copy and initialization operations quickly and efficiently *in* DRAM. The changes required by our mechanism in the DRAM chip are limited to the peripheral logic and are very modest, with a DRAM die area overhead of only 0.2%. With this small overhead, our mechanisms significantly reduce the latency, bandwidth, and energy consumed by bulk data operations.

6.2. Long-Term Impact

We believe four trends in current and future systems make our proposed solutions even more relevant. We discuss each trend, and how RowClone can be applied in the context of the trend.

Increasingly Limited Memory Bandwidth. Processor manufactures are integrating more and more cores on a single chip, thereby significantly increasing the compute capability of the processing chip. However, due to (1) the high cost associated with increasing pin counts and (2) limitations in DRAM scalability, the available memory bandwidth is not expected to grow at the same rate [61, 64]. This makes mechanisms like RowClone, which significantly reduce the overall memory bandwidth utilization of the system, likely even more important in future systems.

Increasing Use of Hardware Accelerators. Many modern processors already integrate the GPU on the same die as the CPU. With emerging systems moving towards a system-on-chip (SoC) model, many components/accelerators (called *agents*) are integrated on the same die as the CPU, and share the off-chip memory [176, 177]. To reduce the complexity of managing these agents, each agent is given its own share of the physical address space, and agents typically communicate with each other by copying data in bulk across the individual device address spaces. By enabling faster bulk data copies, we expect RowClone to significantly reduce the communication latency between different agents without increasing the complexity of the system.

Increasing Use of Virtualization. Modern systems (especially data centers and cloud computers) are increasingly employing virtualization to improve the utilization, security, and availability of systems and services. As described in our MICRO 2013 paper [151], the use of techniques such as VM cloning and deduplication [88, 180] to reduce the memory capacity requirements will likely increase the number of copy operations and zeroing operations (to protect data across VMs). RowClone can improve the performance and energy efficiency of such systems by performing these copy/initialization operations efficiently.

Ease of Adoption. Given the low implementation complexity of RowClone, it can be easily adopted in existing systems. RowClone is not limited only to DDR DRAMs. It can be used with 3D-stacked DRAM technologies [97, 111] such as the Hybrid Memory Cube [58, 59] and High Bandwidth Memory [65], which are gaining increasing interest among researchers, DRAM manufacturers, and system designers [6, 7, 82].

6.3. New Research Directions

Our proposed approach to performing bulk data copy and initialization in DRAM inspires several important research directions (and hopefully many more that others will imagine). We describe a few of them below.

One important research question that our work raises is *how can one redesign system software (e.g., operating system, hypervisors) and application software to take better advantage of RowClone?* Existing systems assume that copies are expensive and hence trade off complexity for performance. However, with RowClone, it may be possible to design simpler yet high performance systems by rethinking software design in the presence of very fast bulk copy and initialization.

Our MICRO 2013 paper [151] proposes low-cost mechanisms to export bulk copy and initialization to DRAM. These are by no means the only bandwidth-intensive operations. There are other operations that unnecessarily move data between the main memory and the processor, which can be optimized using low-cost mechanisms. Therefore, another natural research question is *what other bandwidth-intensive operations can be exported to main memory using low-cost mechanisms?* We believe RowClone can inspire similar mechanisms for other such operations. For example, one of our recent works [157] proposes an efficient method to perform gather/scatter operations in DRAM. Another of our recent works proposes mechanisms to perform bulk bitwise operations in DRAM [153, 154], building upon and taking advantage of RowClone.

Recently, there has been increased interest in emerging non-volatile memory technologies (e.g., PCM [89, 90, 91, 143, 145, 184, 186, 187], STT-MRAM [29, 50, 84, 128], memristors [32, 170]). Given this trend, *exploring the feasibility of extending RowClone to these new memory technologies* is a relevant and important research direction. For example, two recent works [103, 162] use the principles discussed in RowClone to perform bulk Boolean algebra and arithmetic operations within emerging memories. Similarly, exploring the idea of RowClone in other storage/memory technologies, e.g., NAND flash memory [19, 20, 21], is promising.

Given that memory bandwidth is expected to become an even more scarce resource in future systems, answers to these research questions have the potential to greatly mitigate bandwidth contention, and, thus, significantly improve both the performance and energy efficiency of these systems.

6.4. Works Building on RowClone

RowClone has inspired a number of followup works that propose 1) new mechanisms to perform bulk operations inside various memory technologies (e.g., DRAM [25, 102], SRAM [2, 69], PCM [103], memristors [162]), and 2) mechanisms that exploit RowClone to speedup other operations (e.g., in-DRAM bulk bitwise operations [76, 153, 154]). A survey of related works is provided in [159].

One of our recent works, Ambit [153, 154], proposes a mechanism to perform bulk bitwise operations completely inside DRAM. Ambit operations involve a number of row copy and initialization operations. Ambit uses RowClone to perform these operations quickly and efficiently inside DRAM. In fact, RowClone is essential for Ambit to obtain the performance and energy efficiency improvements. Other recent works that perform bulk bitwise Boolean algebra and/or simple arithmetic operations [2, 8, 9, 69, 85, 86, 87, 101, 102, 103, 162] exploit the organization and operation of memory arrays, akin to RowClone, and can be used to perform bulk data copy and initialization operations.

Data movement is expected to become an even more critical problem in future systems. We believe RowClone can inspire other works that propose mechanisms to reduce data movement, thereby enabling higher system performance and energy efficiency.

7. Conclusion

Our MICRO 2013 paper [151] proposes RowClone, a mechanism that performs bulk data copy and initialization operations completely inside DRAM. RowClone consists of two mechanisms, Fast Parallel Mode and Pipelined Serial Mode, that are used to copy data using existing peripheral structures within DRAM, requiring no changes to the DRAM cell array. By enabling efficient bulk data copy and initialization, RowClone provides significant performance and DRAM energy improvements that are between one to two orders of magnitude higher compared to existing systems.

RowClone is one of the first steps towards reducing unnecessary data movement between the processor and the main memory using a low-cost in-memory approach. Current trends in system design indicate that our approach will be more relevant to future, bandwidth-limited systems. We hope that our work triggers research that leads to 1) simpler and more efficient software design and 2) extensions of our approach to other operations and memory technologies, with the goal of continuing to greatly improve system performance and energy efficiency.

Acknowledgments

We thank Saugata Ghose for his dedicated effort in the preparation of this article. We acknowledge the support of AMD, IBM, Intel, Oracle, Qualcomm, and Samsung. This research was partially supported by the NSF (grants 0953246, 1147397, and 1212962), the Intel University Research Office Memory Hierarchy Program, the Intel Science and Technology Center for Cloud Computing, and the Semiconductor Research Corporation.

References

[1] B. Abali, H. Franke, D. Poff, R. Saccone, C. Schulz, L. Herger, and T. Smith, "Memory Expansion Technology (MXT): Software support and performance," in *IBM JRD*, 2001.
[2] S. Aga *et al.*, "Compute Caches," in *HPCA*, 2017.
[3] J. Ahn, "Memory device having page copy mode," U.S. Patent 5,886,944, 1999.
[4] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Improving System Energy Efficiency with Memory Rank Subsetting," in *ACM TACO*, 2012.
[5] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," in *IEEE CAL*, 2009.
[6] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[7] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[8] A. Akerib, O. Agam, E. Ehrman, and M. Meyassed, "Using Storage Cells to Perform Computation," U.S. Patent 8,908,465, 2014.
[9] A. Akerib and E. Ehrman, "In-Memory Computational Device," U.S. Patent 9,653,166, 2015.
[10] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *ISCA*, 2004.
[11] A. R. Alameldeen and D. A. Wood, "Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches," Univ. of Wisconsin–Madison, Computer Sciences Dept., Tech. Rep. 1500, 2004.
[12] A. Alameldeen and D. Wood, "Interactions Between Compression and Prefetching in Chip Multiprocessors," in *HPCA*, 2007.
[13] R. Ausavarungnirun, K. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: achieving high performance and scalability in heterogeneous systems," in *ISCA*, 2012.
[14] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, "Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance," in *PACT*, 2015.
[15] J. Bent *et al.*, "PLFS: A checkpoint filesystem for parallel applications," in *SC*, 2009.
[16] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[17] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," in *IEEE CAL*, 2016.
[18] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*. O'Reilly Media, 2005, p. 388.
[19] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, 2017.
[20] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.
[21] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.
[22] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, "A Study of Integrated Prefetching and Caching Strategies," in *SIGMETRICS*, 1995.
[23] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization," in *DATE*, 2014.
[24] F. Chang and G. A. Gibson, "Automatic I/O hint generation through speculative execution," in *OSDI*, 1999.
[25] K. K. Chang *et al.*, "Low-cost Inter-linked Subarrays (LISA): Enabling Fast Inter-subarray Data Movement in DRAM," in *HPCA*, 2016.
[26] K. K. Chang, A. Kashyap, H. Hassan, S. Khan, K. Hsieh, D. Lee, S. Ghose, G. Pekhimenko, T. Li, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[27] K. K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses ," in *HPCA*, 2014.
[28] K. K. Chang, A. G. Yaglikci, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[29] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *HPCA*, 2013.
[30] N. Chatterjee, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Staged Reads: Mitigating the Impact of DRAM Writes on DRAM Reads," in *HPCA*, 2012.
[31] J. Chow *et al.*, "Shredding Your Garbage: Reducing data lifetime through secure deallocation," in *USENIX SS*, 2005.
[32] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, Sep. 1971.
[33] K. Constantinides *et al.*, "Software-Based Online Detection of Hardware Defects: Mechanisms, architectural support, and evaluation," in *MICRO*, 2007.
[34] K. Constantinides *et al.*, "Online Design Bug Detection: RTL analysis, flexible mechanisms, and evaluation," in *MICRO*, 2008.
[35] R. Cooksey, S. Jourdan, and D. Grunwald, "A Stateless, Content-directed Data Prefetching Mechanism," in *ASPLOS*, 2002.
[36] F. Dahlgren, M. Dubois, and P. Stenström, "Sequential Hardware Prefetching in Shared-Memory Multiprocessors," in *IEEE TPDS*, 1995.
[37] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-core mapping policies to reduce memory system interference in multi-core systems," in *HPCA*, 2013.
[38] R. de Castro, A. Lago, and M. Silva, "Adaptive compressed caching: design and implementation," in *SBAC-PAD*, 2003.
[39] F. Douglis, "The Compression Cache: Using On-line Compression to Extend Physical Memory," in *Winter USENIX Conference*, 1993.
[40] J. Dundas and T. Mudge, "Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss," in *ICS*, 1997.
[41] A. M. Dunn *et al.*, "Eternal Sunshine of the Spotless Machine: Protecting privacy with ephemeral channels," in *OSDI*, 2012.
[42] J. Dusser, T. Piquet, and A. Seznec, "Zero-content Augmented Caches," in *ICS*, 2009.
[43] E. Ebrahimi, O. Mutlu, and Y. Patt, "Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems," in *HPCA*, 2009.
[44] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware Shared Resource Management for Multi-core Systems," in *ISCA*, 2011.
[45] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt, "Parallel application memory scheduling," in *MICRO*, 2011.
[46] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated Control of Multiple Prefetchers in Multi-core Systems," in *MICRO*, 2009.
[47] S. Ghose, H. Lee, and J. F. Martínez, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.
[48] P. B. Gillingham and R. Torrance, "DRAM page copy method," U.S. Patent 5,625,601, 1997.
[49] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WoNDP*, 2014.
[50] X. Guo, E. İpek, and T. Soyata, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *ISCA*, 2009.
[51] J. A. Halderman *et al.*, "Lest We Remember: Cold boot attacks on encryption keys," in *USENIX SS*, 2008.
[52] K. Harrison and S. Xu, "Protecting cryptographic keys from memory disclosure attacks," in *DSN*, 2007.
[53] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[54] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.

[55] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, “Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation,” in *ICCD*, 2016.
[56] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O’Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, “Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems,” in *ISCA*, 2016.
[57] I. Hur and C. Lin, “Adaptive History-Based Memory Schedulers,” in *MICRO*, 2004.
[58] Hybrid Memory Cube Consortium, “HMC Specification 1.1,” 2013.
[59] Hybrid Memory Cube Consortium, “HMC Specification 2.0,” 2014.
[60] Intel, “Intel 64 and IA-32 Architectures Optimization Reference Manual,” Apr. 2012.
[61] E. Ipek *et al.*, “Self Optimizing Memory Controllers: A Reinforcement Learning Approach,” in *ISCA*, 2008.
[62] T. B. Jablin *et al.*, “Automatic CPU-GPU communication management and optimization,” in *PLDI*, 2011.
[63] L. A. Jarrod *et al.*, “Avoiding Initialization Misses to the Heap,” in *ISCA*, 2002.
[64] JEDEC, “Server memory roadmap,” http://www.jedec.org/sites/default/files/Ricki_Dee_Williams.pdf.
[65] JEDEC, “High Bandwidth Memory (HBM) DRAM,” Standard No. JESD235, 2013.
[66] X. Jiang *et al.*, “Architecture support for improving bulk memory copying and initialization performance,” in *PACT*, 2009.
[67] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, “Exploiting Core-Criticality for Enhanced GPU Performance,” in *SIGMETRICS*, 2016.
[68] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, “Profiling a Warehouse-Scale Computer,” in *ISCA*, 2015.
[69] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, “An Energy-Efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM,” in *ICASSP*, 2014.
[70] D. Kaseridis, J. Stuecheli, and L. K. John, “Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era,” in *MICRO*, 2011.
[71] S. Khan *et al.*, “Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content,” in *MICRO*, 2017.
[72] S. Khan, D. Lee, and O. Mutlu, “PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM,” in *DSN*, 2016.
[73] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutlu, and D. A. Jimenez, “Improving Cache Performance by Exploiting Read-Write Disparity,” in *HPCA*, 2014.
[74] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, “The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study,” in *SIGMETRICS*, 2014.
[75] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, “A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM,” in *IEEE CAL*, 2016.
[76] J. S. Kim *et al.*, “Genome Read In-Memory (GRIM) Filter: Fast Location Filtering in DNA Read Mapping Using Emerging Memory Technologies,” in *APBC*, 2018.
[77] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, “The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices,” in *HPCA*, 2018.
[78] Y. Kim *et al.*, “ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers,” in *HPCA*, 2010.
[79] Y. Kim *et al.*, “Thread Cluster Memory Scheduling: Exploiting differences in memory access behavior,” in *MICRO*, 2010.
[80] Y. Kim *et al.*, “A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM,” in *ISCA*, 2012.
[81] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in *ISCA*, 2014.
[82] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A Fast and Extensible DRAM Simulator,” *CAL*, 2015.
[83] P. M. Kogge, “EXECUBE - A new architecture for scaleable MPPs,” in *ICPP*, 1994.
[84] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, “Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative,” in *ISPASS*, 2013.
[85] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, “MAGIC—Memristor-Aided Logic,” *IEEE TCAS II: Express Briefs*, 2014.
[86] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, “Memristor-Based IMPLY Logic Design Procedure,” in *ICCD*, 2011.
[87] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, “Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies,” *TVLSI*, 2014.
[88] H. A. Lagar-Cavilla *et al.*, “SnowFlock: Rapid virtual machine cloning for cloud computing,” in *EuroSys*, 2009.
[89] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting Phase Change Memory as a Scalable DRAM Alternative,” in *ISCA*, 2009.
[90] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Phase Change Memory Architecture and the Quest for Scalability,” *CACM*, 2010.
[91] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, “Phase-Change Technology and the Future of Main Memory,” *IEEE Micro*, 2010.
[92] C. J. Lee, E. Ebrahimi, V. Narasiman, O. Mutlu, and Y. N. Patt, “DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems,” Univ. of Texas at Austin, High Performance Systems Group, Tech. Rep. TR-HPS-2010-002, 2010.
[93] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, “Prefetch-Aware DRAM Controllers,” in *MICRO*, 2008.
[94] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, “Prefetch-Aware Memory Controllers,” in *IEEE TC*, 2011.
[95] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, “Improving Memory Bank-level Parallelism in the Presence of Prefetching,” in *MICRO*, 2009.
[96] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, “Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture,” in *HPCA*, 2013.
[97] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, “Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost,” in *ACM TACO*, 2016.
[98] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, “Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms,” in *SIGMETRICS*, 2017.
[99] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. K. Chang, and O. Mutlu, “Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case,” in *HPCA*, 2015.
[100] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, “Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM,” in *PACT*, 2015.
[101] Y. Levy, J. Bruck, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaakobi, and S. Kvatinsky, “Logic Operations in Memory Using a Memristive Akers Array,” *Microelectronics Journal*, 2014.
[102] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, “DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator,” in *MICRO*, 2017.
[103] S. Li *et al.*, “Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories,” in *DAC*, 2016.
[104] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, “Utility-Based Hybrid Memory Management,” in *CLUSTER*, 2017.
[105] Y. Li, S. Makar, and S. Mitra, “CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns,” in *DATE*, 2008.
[106] Y. Li, O. Mutlu, D. S. Gardner, and S. Mitra, “Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips,” in *VTS*, 2010.
[107] Y. Li, O. Mutlu, and S. Mitra, “Operating System Scheduling for Efficient Online Self-Test in Robust Systems",” in *ICCAD*, 2009.
[108] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms,” in *ISCA*, 2013.
[109] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “RAIDR: Retention-aware Intelligent DRAM Refresh,” in *ISCA*, 2012.
[110] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, “Concurrent Data Structures for Near-Memory Computing,” in *SPAA*, 2017.
[111] G. H. Loh, “3D-Stacked Memory Architectures for Multi-Core Processors,” in *ISCA*, 2008.
[112] Micron, “DDR3 SDRAM system-power calculator,” 2011.
[113] D. M. Morgan and M. A. Shore, “DRAMs having on-chip row copy circuits for use in testing and video imaging and method for operating same,” U.S. Patent 5,440,517, 1995.
[114] K. Mori, “Semiconductor memory device including copy circuit,” U.S. Patent 5,854,771, 1998.
[115] T. Moscibroda and O. Mutlu, “Memory Performance Attacks: Denial of Memory Service in Multi-core Systems,” in *USENIX Security*, 2007.
[116] T. Moscibroda and O. Mutlu, “Distributed Order Scheduling and Its Application to Multi-core Dram Controllers,” in *PODC*, 2008.
[117] J. Mukundan and J. F. Martínez, “MORSE: Multi-Objective Reconfigurable Self-Optimizing Memory Scheduler,” in *HPCA*, 2012.
[118] S. P. Muralidhara *et al.*, “Reducing memory interference in multi-core systems via application-aware memory channel partitioning,” in *MICRO*, 2011.
[119] O. Mutlu *et al.*, “Efficient Runahead Execution: Power-efficient memory latency tolerance,” *IEEE Micro*, vol. 26, no. 1, 2006.
[120] O. Mutlu, H. Kim, and Y. Patt, “Address-value delta (AVD) prediction: increasing the effectiveness of runahead execution by exploiting regular memory allocation patterns,” in *MICRO*, 2005.
[121] O. Mutlu, H. Kim, and Y. Patt, “Techniques for efficient processing in runahead execution engines,” in *ISCA*, 2005.
[122] O. Mutlu, J. Stark, C. Wilkerson, and Y. Patt, “Runahead execution: an alternative to very large instruction windows for out-of-order processors,” in *HPCA*, 2003.
[123] O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” *SUPERFRI*, 2014.
[124] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” in *IMW*, 2013.
[125] O. Mutlu and T. Moscibroda, “Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors,” in *MICRO*, 2007.
[126] O. Mutlu and T. Moscibroda, “Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems,” in *ISCA*, 2008.
[127] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, “Runahead execution: An effective alternative to large instruction windows,” in *IEEE Micro*, 2003.
[128] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, “STT-RAM Scaling and Retention Failure,” *Intel Technol. J.*, May 2013.

[129] K. Nesbit, A. Dhodapkar, and J. Smith, "AC/DC: an adaptive data cache prefetcher," in *PACT*, 2004.
[130] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair Queuing Memory Systems," in *MICRO*, 2006.
[131] J. K. Ousterhout, "Why aren't operating systems getting faster as fast as hardware?" in *USENIX STC*, 1990.
[132] M. Patel, J. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[133] D. Patterson *et al.*, "A case for Intelligent RAM," *IEEE Micro*, 1997.
[134] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed Prefetching and Caching," in *SOSP*, 1995.
[135] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.
[136] G. Pekhimenko, E. Bolotin, M. O'Connor, O. Mutlu, T. C. Mowry, and S. W. Keckler, "Toggle-Aware Compression for GPUs," in *IEEE CAL*, 2015.
[137] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, "Toggle-Aware Bandwidth Compression for GPUs," in *HPCA*, 2016.
[138] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. P. Gibbons, M. A. Kozuch, and T. C. Mowry, "Exploiting Compressed Block Size as an Indicator of Future Reuse," in *HPCA*, 2015.
[139] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly Compressed Pages: A Low-complexity, Low-latency Main Memory Compression Framework," in *MICRO*, 2013.
[140] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches," in *PACT*, 2012.
[141] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[142] M. K. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer, "Adaptive Insertion Policies for High Performance Caching," in *ISCA*, 2007.
[143] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *MICRO*, 2009.
[144] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A Case for MLP-Aware Cache Replacement," in *ISCA*, 2006.
[145] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[146] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA*, 2000.
[147] M. Rosenblum *et al.*, "The impact of architectural trends on operating system performance," in *SOSP*, 1995.
[148] M. E. Russinovich *et al.*, *Windows Internals*. Microsoft Press, 2009, p. 701.
[149] G. Sandhu, "DRAM scaling and bandwidth challenges," in *WETI*, 2012.
[150] R. F. Sauers *et al.*, *HP-UX 11i Tuning and Performance*. Prentice Hall, 2004, ch. 8. Memory Bottlenecks.
[151] V. Seshadri *et al.*, "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *MICRO*, 2013.
[152] V. Seshadri *et al.*, "The Dirty-Block Index," in *ISCA*, 2014.
[153] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," in *IEEE CAL*, 2015.
[154] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[155] V. Seshadri, G. Pekhimenko, O. Ruwase, O. Mutlu, P. B. Gibbons, M. A. Kozuch, T. C. Mowry, and T. Chilimbi, "Page Overlays: An Enhanced Virtual Memory Framework to Enable Fine-Grained Memory Management," in *ISCA*, 2015.
[156] V. Seshadri, "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2016.
[157] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses," in *MICRO*, 2015.
[158] V. Seshadri and O. Mutlu, "The Processing Using Memory Paradigm: In-DRAM Bulk Copy, Initialization, Bitwise AND and OR," arXiv:1610.09603 [cs:AR], 2016.
[159] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers, Volume 106*, 2017.
[160] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," in *PACT*, 2012.
[161] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks," in *TACO*, 2015.
[162] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[163] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring Unconventional Benefits from Memory Compression," in *HPCA*, 2014.
[164] J. Shao and B. T. Davis, "A Burst Scheduling Access Reordering Mechanism," in *HPCA*, 2007.
[165] W. Shin, J. Yang, J. Choi, and L.-S. Kim, "NUAT: A Non-Uniform Access Time Memory Controller," in *HPCA*, 2014.
[166] A. Singh, *Mac OS X Internals: A Systems Approach*. Addison-Wesley Professional, 2006.
[167] S. Srinath *et al.*, "Feedback Directed Prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *HPCA*, 2007.
[168] S. M. Srinivasan *et al.*, "Flashback: A lightweight extension for rollback and deterministic replay for software debugging," in *USENIX ATC*, 2004.
[169] Standard Performance Evaluation Corporation, "SPEC CPU2006," http://www.spec.org/cpu2006.
[170] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, May 2008.
[171] L. Subramanian *et al.*, "MISE: Providing performance predictability and improving fairness in shared main memory systems," in *HPCA*, 2013.
[172] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "The Blacklisting Memory Scheduler: Achieving high performance and fairness at low cost," in *ICCD*, 2014.
[173] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," in *TPDS*, 2016.
[174] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," in *MICRO*, 2015.
[175] K. Sudan *et al.*, "Micro-pages: Increasing DRAM efficiency with locality-aware data placement," in *ASPLOS*, 2010.
[176] M. A. Suleman, O. Mutlu, J. A. Joao, Khubaib, and Y. N. Patt, "Data Marshaling for Multi-Core Architectures," in *ISCA*, 2010.
[177] H. Usui, L. Subramanian, K. Chang, and O. Mutlu, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," in *ACM TACO*, 2016.
[178] R. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.
[179] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," in *ISCA*, 2015.
[180] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *OSDI*, 2002.
[181] F. Ware and C. Hampel, "Improving Power and Data Efficiency with Threaded Memory Modules," in *ICCD*, 2006.
[182] B. Wester *et al.*, "Operating system support for application-specific speculation," in *EuroSys*, 2011.
[183] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis, "The Case for Compressed Caching in Virtual Memory Systems," in *ATEC*, 1999.
[184] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, Dec. 2010.
[185] X. Yang *et al.*, "Why Nothing Matters: The impact of zeroing," in *OOPSLA*, 2011.
[186] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.
[187] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories," *TACO*, 2014.
[188] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented Programmable Processing in Memory," in *HPCA*, 2014.
[189] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation," in *ISCA*, 2014.
[190] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," in *ASPLOS*, 2000.
[191] J. Zhao, O. Mutlu, and Y. Xie, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," in *MICRO*, 2014.
[192] L. Zhao *et al.*, "Hardware support for bulk data movement in server platforms," in *ICCD*, 2005.
[193] H. Zheng *et al.*, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *MICRO*, 2008.

LISA: Increasing Internal Connectivity in DRAM for Fast Data Movement and Low Latency

Kevin K. Chang[1,2] Prashant J. Nair[3,4] Saugata Ghose[2]
Donghyuk Lee[5,2] Moinuddin K. Qureshi[4] Onur Mutlu[6,2]

[1]*Facebook* [2]*Carnegie Mellon University* [3]*IBM Research*
[4]*Georgia Institute of Technology* [5]*NVIDIA Research* [6]*ETH Zürich*

This paper summarizes the idea of Low-Cost Interlinked Subarrays (LISA), which was published in HPCA 2016 [10], and examines the work's significance and future potential. Our HPCA 2016 paper introduces a new DRAM design that enables fast and energy-efficient bulk data movement across subarrays in a DRAM chip. While bulk data movement is a key operation in many applications and operating systems, we observe that contemporary systems perform this movement inefficiently, by transferring data from DRAM to the processor, and then back to DRAM, across a narrow off-chip channel. The use of this narrow channel for bulk data movement results in high latency and energy consumption. Prior work proposes to avoid these high costs by exploiting the existing *wide internal DRAM bandwidth for bulk data movement, but the limited connectivity of wires within DRAM allows fast data movement within only a single DRAM subarray. Each subarray is only a few megabytes in size, greatly restricting the range over which fast bulk data movement can happen within DRAM.*

Our HPCA 2016 paper proposes a new DRAM substrate, Low-Cost Inter-Linked Subarrays (LISA), whose goal is to enable fast and efficient data movement across a large range of memory at low cost. LISA adds low-cost connections between *adjacent subarrays. By using these connections to interconnect the existing internal wires (*bitlines*) of adjacent subarrays, LISA enables wide-bandwidth data transfer across multiple subarrays with little (only 0.8%) DRAM area overhead. As a DRAM substrate, LISA is versatile, enabling a variety of new applications. We describe and evaluate three such applications in detail: (1) fast inter-subarray bulk data copy, (2) in-DRAM caching using a DRAM architecture whose rows have heterogeneous access latencies, and (3) accelerated bitline precharging by linking multiple precharge units together. Our extensive evaluations show that each of LISA's three applications significantly improves performance and memory energy efficiency, and their combined benefit is higher than the benefit of each alone, on a variety of workloads and system configurations.*

1. Introduction

Bulk data movement, the movement of thousands or millions of bytes between two memory locations, is a common operation performed by an increasing number of real-world applications (e.g., [6, 37, 57, 58, 74, 82, 85, 88, 89, 94, 99, 110]). Therefore, it has been the target of several architectural optimizations (e.g., [4, 6, 35, 40, 58, 70, 86, 88, 103, 110]). In fact, bulk data movement is important enough that modern commercial processors are adding specialized support to improve its performance, such as the ERMSB instruction recently added to the x86 ISA [28].

In today's systems, to perform a bulk data movement between two locations in memory, the data needs to go through the processor *even though both the source and destination are within memory*. To perform the movement, the data is first read out one cache line at a time from the source location in memory into the processor caches, over a pin-limited off-chip channel (typically 64 bits wide). Then, the data is written back to memory, again one cache line at a time over the pin-limited channel, into the destination location. By going through the processor, this data movement incurs a significant penalty in terms of latency and energy consumption.

To address the inefficiencies of traversing the pin-limited channel, a number of mechanisms have been proposed to accelerate bulk data movement (e.g., [35, 63, 88, 110]). The state-of-the-art mechanism, RowClone [88], performs data movement *completely within a DRAM chip*, avoiding costly data transfers over the pin-limited memory channel. However, its effectiveness is limited because RowClone can enable *fast* data movement *only* when the source and destination are within the same DRAM *subarray*. A DRAM chip is divided into multiple *banks* (typically 8), each of which is further split into many *subarrays* (16 to 64) [45], shown in Figure 1a, to ensure reasonable read and write latencies at high density [8, 32, 33, 45, 101].[1] Each subarray is a two-dimensional array with hundreds of rows of DRAM cells, and contains only a few megabytes of data (e.g., 4MB in a rank of eight 1Gb DDR3 DRAM chips with 32 subarrays per bank). While two DRAM rows in the *same* subarray are connected via a wide (e.g., 8K bits) bitline interface, rows in *different* subarrays are connected via only a *narrow 64-bit data bus* within the DRAM chip (Figure 1a). Therefore, even for previously-proposed in-DRAM data movement mechanisms such as RowClone [88], *inter-subarray* bulk data movement incurs long latency and high memory energy consumption even though data does *not* move out of the DRAM chip.

[1]We refer the reader to our prior works [8, 9, 10, 11, 21, 22, 39, 41, 42, 43, 44, 45, 54, 55, 56, 57, 58, 60, 61, 75, 88, 89] for a detailed background on DRAM.

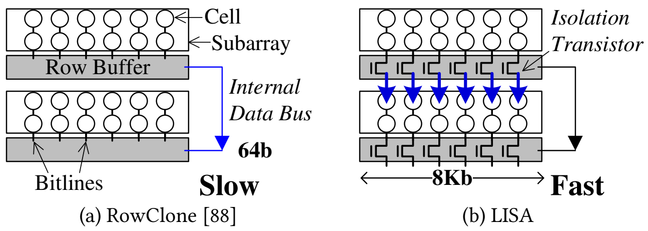


Figure 1: Transferring data between subarrays using the internal data bus takes a long time in state-of-the-art DRAM design, RowClone [88] (a). Our work, LISA, enables fast inter-subarray data movement with a low-cost substrate (b). Reproduced from [10].

While it is clear that fast *inter-subarray* data movement can have several applications that improve system performance and memory energy efficiency [6, 37, 55, 74, 82, 85, 88, 89, 110], there is currently no mechanism that performs such data movement quickly and efficiently. This is because *no wide datapath exists today between subarrays* within the same bank (i.e., the connectivity of subarrays is low in modern DRAM). **Our goal** is to design a low-cost DRAM substrate that enables fast and energy-efficient data movement *across subarrays*.

2. Low-Cost Inter-Linked Subarrays (LISA)

We make two key observations that allow us to improve the connectivity of subarrays within each bank in modern DRAM. First, accessing data in DRAM causes the transfer of an entire row of DRAM cells to a buffer (i.e., the *row buffer*, where the row data temporarily resides while it is read or written) via the subarray's *bitlines*. Each bitline connects a column of cells to the row buffer, interconnecting every row within the same subarray (Figure 1a). Therefore, the bitlines essentially serve as a very wide bus that transfers a *row's worth of data* (e.g., 8K bits in a chip) at once. Second, subarrays within the same bank are placed in close proximity to each other. Thus, the bitlines of a subarray are very close to (but are not currently connected to) the bitlines of neighboring subarrays (as shown in Figure 1a).

Key Idea. Based on these two observations, we introduce a new DRAM substrate, called *Low-cost Inter-linked SubArrays* (*LISA*). LISA enables *low-latency, high-bandwidth inter-subarray connectivity* by linking neighboring subarrays' bitlines together with *isolation transistors*, as illustrated in Figure 1b. We use the new inter-subarray connection in LISA to develop a new DRAM operation, *row buffer movement (RBM)*, which moves data that is latched in an activated row buffer in one subarray into an inactive row buffer in another subarray, without having to send data through the narrow internal data bus in DRAM. RBM exploits the fact that the activated row buffer has enough drive strength to induce charge perturbation within the idle (i.e., *precharged*) bitlines of neighboring subarrays, allowing the destination row buffer to sense and latch this data when the isolation transistors are enabled. We describe the detailed operation of RBM in our HPCA 2016 paper [10].

By using a rigorous DRAM circuit model that conforms to the JEDEC standards [32] and ITRS specifications [30, 31], we show that RBM performs *row buffer movement* at 26x the bandwidth of a modern 64-bit DDR4-2400 memory channel (500 GB/s vs. 19.2 GB/s), even after we conservatively add a large (60%) timing margin to account for process and temperature variation.

Die Area Overhead. To evaluate the area overhead of adding isolation transistors, we use area values from prior work, which adds isolation transistors to disconnect bitlines from sense amplifiers [73]. That work shows that adding an isolation transistor to every bitline incurs a total of 0.8% die area overhead in a 28nm DRAM process technology. Similar to prior work that adds isolation transistors to DRAM [57, 73], our LISA substrate also requires additional control logic outside the DRAM banks to control the isolation transistors, which incurs a small amount of area and is non-intrusive to the cell arrays.

3. Applications of LISA

We exploit LISA's *fast inter-subarray movement capability* to enable many applications that can improve system performance and energy efficiency. In our HPCA 2016 paper [10], we implement and evaluate three applications of LISA, which significantly improve system performance in different ways.

3.1. Rapid Inter-Subarray Bulk Data Copying (LISA-RISC)

Due to the narrow memory channel width, bulk copy operations used by applications and operating systems are performance limiters in today's systems [35, 37, 55, 88, 110]. These operations are commonly performed due to the `memcpy` and `memmov`. Recent work reported that these two operations consume 4-5% of *all of* Google's data center cycles [37], making them an important target for lightweight hardware acceleration.

Our goal is to design a new mechanism that enables *low-latency* and *energy-efficient* memory copy between rows *in different subarrays* within the same bank. To this end, we propose a new in-DRAM copy mechanism that uses LISA to exploit the high-bandwidth links between subarrays. The key idea, step by step, is to: (1) activate a source row in a subarray; (2) rapidly transfer the data in the activated source row buffers to the destination subarray's row buffers, through LISA's RBM operation; and (3) activate the destination row, which enables the contents of the destination row buffers to be latched into the destination row. We call this inter-subarray row-to-row copy mechanism *LISA-Rapid Inter-Subarray Copy* (LISA-RISC).

3.1.1. DRAM Latency and Energy Consumption. Figure 2 shows the DRAM latency and DRAM energy consumption of `memcpy` (i.e, the baseline system), RowClone [88] (state-of-the-art work), and LISA-RISC for copying a row of data (8KB). The exact latency and energy numbers are listed

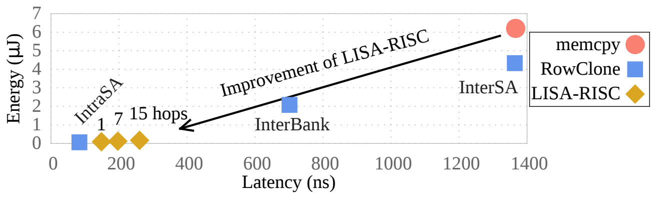


Figure 2: Latency and DRAM energy of 8KB copy. Reproduced from [10].

Copy Commands (8KB)	**Latency** (ns)	**Energy** (µJ)
`memcpy` (via mem. channel)	1366.25	6.2
RC-InterSA / Bank / IntraSA	1363.75 / 701.25 / 83.75	4.33 / 2.08 / 0.06
LISA-RISC (1 / 7 / 15 hops)	148.5 / 196.5 / 260.5	0.09 / 0.12 / 0.17

Table 1: Copy latency and DRAM energy. Reproduced from [10].

in Table 1. For LISA-RISC, we define a *hop* as the number of subarrays that LISA-RISC needs to copy data *across* to move the data from the source subarray to the destination subarray. For example, if the source and destination subarrays are adjacent to each other, the number of hops is 1. The DRAM chips we evaluate have 16 subarrays per bank, so the maximum number of hops is 15.

We make two observations from these numbers. First, although inter-subarray RowClone (*RC-InterSA*) incurs similar latencies as `memcpy`, it consumes 1.43x less energy, as it does *not* transfer data over the channel and DRAM I/O for each copy operation. However, as we discuss in Section 4.1 of our HPCA 2016 paper [10], RC-InterSA incurs a higher system performance penalty because it is a *blocking* long-latency memory command. Second, copying between subarrays using LISA reduces the copy latency by 9x and copy energy by 48x compared to RowClone, even though the total latency of LISA-RISC grows linearly with the hop count. An additional benefit of using LISA-RISC is that its inter-subarray copy operations are performed *completely inside a bank*. As the internal DRAM data bus is untouched, *other* banks can *concurrently* serve memory requests, exploiting bank-level parallelism.

3.1.2. Evaluation. We briefly summarize the system performance improvement due to LISA-RISC on a quad-core system. We evaluate our system using Ramulator [41, 83], an open-source cycle-accurate DRAM simulator, driven by traces generated from Pin [64]. Our workload evaluation results show that LISA-RISC outperforms RowClone and `memcpy`: its average performance improvement and energy reduction over the best performing inter-subarray copy mechanism (i.e., `memcpy`) are 66.2% and 55.4%, respectively, on a quad-core system, across 50 workloads that perform bulk copies. We refer the reader to Section 9 of our HPCA 2016 paper [10] for detailed evaluation and analysis.

3.2. In-DRAM Caching Using Heterogeneous Subarrays (LISA-VILLA)

Our second application aims to reduce the DRAM access latency for frequently-accessed (hot) data. We propose to introduce heterogeneity *within a bank* by designing *heterogeneous-latency subarrays*. We call this heterogeneous DRAM design *VarIabLe LAtency DRAM* (VILLA-DRAM). To design a low-cost fast subarray, we take an approach similar to prior work, attaching fewer cells to each bitline to reduce the parasitic capacitance and resistance. This reduces the latency of the three fundamental DRAM operations–*activation*, *precharge*, and *restoration*–when accessing data in the fast subarrays [57, 67, 94]. *Activation* "opens" a row of DRAM cells to access stored data. *Precharge* "closes" an activated row. *Restoration* restores the charge level of each DRAM cell in a row to prevent data loss. Together, these three operations predominantly define the latency of a memory request [8, 9, 10, 11, 21, 22, 39, 41, 42, 43, 44, 45, 54, 55, 56, 57, 58, 60, 61, 75, 88, 89]. In this work, we focus on managing the fast subarrays in hardware, as doing so offers better adaptivity to *dynamic* changes in the hot data set.

In order to take advantage of VILLA-DRAM, we rely on LISA-RISC to rapidly copy rows across subarrays, which significantly reduces the caching latency. We call this synergistic design, which builds VILLA-DRAM using LISA, *LISA-VILLA*. Nonetheless, the cost of transferring data to a fast subarray is still non-negligible, especially if the fast subarray is far from the subarray where the data to be cached resides. Therefore, an intelligent cost-aware mechanism is required to make astute decisions on which data to cache and when.

3.2.1. Caching Policy for LISA-VILLA. We design a simple epoch-based caching policy to evaluate the benefits of caching a row in LISA-VILLA. Every epoch, we track the number of accesses to rows by using a set of 1024 saturating counters for each bank.[2] The counter values are halved every epoch to prevent staleness. At the end of an epoch, we mark the 16 most frequently-accessed rows as *hot*, and cache them when they are accessed the next time. For our cache replacement policy, we use the *benefit-based caching* policy proposed by Lee et al. [57]. Specifically, it uses a benefit counter for each row cached in the fast subarray: whenever a cached row is accessed, its counter is incremented. The row with the least benefit is replaced when a new row needs to be inserted. Note that a large body of work proposes various caching policies (e.g., [20, 23, 26, 34, 38, 59, 66, 78, 79, 87, 91, 100, 104, 106]), each of which can potentially be used with LISA-VILLA.

3.2.2. Evaluation. Figure 3 shows the system performance improvement of LISA-VILLA over a baseline without any fast subarrays in a four-core system. It also shows the hit rate in VILLA-DRAM, i.e., the fraction of accesses that hit in the fast subarrays. We make two main observations. First, by

[2]The hardware cost of these counters is low, requiring only 6KB of storage in the memory controller (see Section 7.1 of our HPCA 2016 paper [10]).

exploiting LISA-RISC to quickly cache data in VILLA-DRAM, LISA-VILLA improves system performance for a wide variety of workloads — by up to 16.1%, with a geometric mean of 5.1%. This is mainly due to reduced DRAM latency of accesses that hit in the fast subarrays. The performance improvement heavily correlates with the VILLA cache hit rate. Second, the VILLA-DRAM design, which consists of heterogeneous subarrays, is not practical without LISA. Figure 3 shows that using RC-InterSA (i.e., RowClone copying data across subarrays) to move data into the cache *reduces* performance by 52.3% due to slow data movement, which overshadows the benefits of caching. The results indicate that LISA is an important substrate to enable not only fast bulk data copy, but also a fast in-DRAM caching scheme.

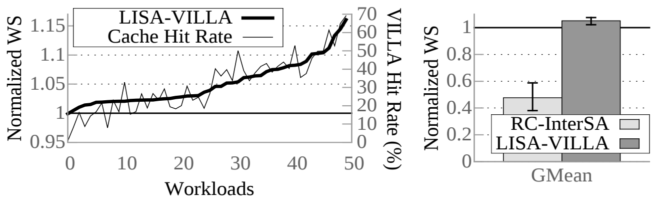


Figure 3: Performance improvement and hit rate with LISA-VILLA, and performance comparison to using RC-InterSA with VILLA-DRAM. Reproduced from [10].

3.3. Fast Precharge Using Linked Precharge Units (LISA-LIP)

Our third application aims to accelerate the process of precharge. The precharge time for a subarray is determined by the drive strength of the precharge unit (i.e., a circuitry in a subarray's row buffer for precharging the connected subarray). We observe that in modern DRAM, while a subarray is being precharged, the precharge units (PUs) of *other* subarrays remain idle.

We propose to exploit these idle PUs to accelerate a precharge operation by connecting them to the subarray that is being precharged. Our mechanism, *LISA-LInked Precharge* (LISA-LIP), precharges a subarray using *two* sets of PUs: one from the row buffer that is being precharged, and a second set from a neighboring subarray's row buffer (which is already in the precharged state), by enabling the links between the two subarrays.

To evaluate the accelerated precharge process, we use the same DRAM circuit model described in Section 2 and simulate the linked precharge operation in SPICE. Our SPICE simulation reports that LISA-LIP significantly reduces the precharge latency by 2.6x compared to the baseline (5ns vs. 13ns). Our system evaluation shows that LISA-LIP improves performance by 10.3% on average, across 50 four-core workloads. We refer the reader to Section 6 of our HPCA 2016 paper [10] for a detailed analysis of LISA-LIP.

3.4. Evaluation: Putting Everything Together

As all of the three proposed applications are complementary to each other, we evaluate the effect of putting them together on a four-core system. Figure 4 shows the system performance improvement of adding LISA-VILLA to LISA-RISC, as well as combining all three optimizations, compared to our baseline using `memcpy` and standard DDR3-1600 memory across 50 workloads. We refer the reader to our full paper [10] for the detailed configuration and workloads. We draw several key conclusions. First, the performance benefits from each scheme are additive. On average, adding LISA-VILLA improves performance by 16.5% over LISA-RISC alone, and adding LISA-LIP further provides an 8.8% gain over LISA-(RISC+VILLA). Second, although LISA-RISC alone provides a majority of the performance improvement over the baseline (59.6% on average), the use of both LISA-VILLA and LISA-LIP further improves performance, resulting in an average performance gain of 94.8% and memory energy reduction (not plotted) of 49.0%. Taken together, these results indicate that LISA is an effective substrate that enables a wide range of high-performance and energy-efficient applications in the DRAM system.

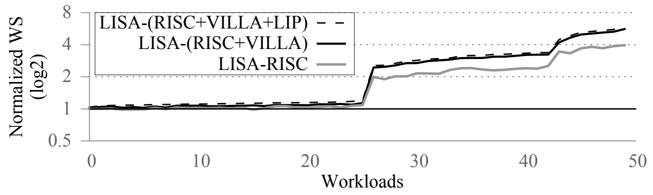


Figure 4: Combined weighted speedup (WS) [14, 93] improvement of LISA applications. Reproduced from [10].

We conclude that LISA is an effective substrate that can greatly improve system performance and reduce system energy consumption by synergistically enabling multiple different applications. Our HPCA 2016 paper [10] provides many more experimental results and analyses confirming this finding.

4. Related Work

To our knowledge, this is the first work to propose a DRAM substrate that supports fast data movement between subarrays in the same bank, which enables a wide variety of applications for DRAM systems. We now discuss prior works that focus on each of the optimizations that LISA enables.

4.1. Bulk Data Transfer Mechanisms

Prior works [7, 16, 17, 36, 108] propose to add scratchpad memories to reduce CPU pressure during bulk data transfers, which can also enable sophisticated data movement (e.g., scatter-gather [90]), but they still require data to first be moved on-chip. A patent proposes a DRAM design that can copy a page across memory blocks [84], but lacks concrete analysis and evaluation of the underlying copy operations. Intel I/O Acceleration Technology [27] allows for memory-to-memory DMA transfers *across a network*, but cannot transfer data within main memory.

Zhao et al. [110] propose to add a bulk data movement engine inside the memory controller to speed up bulk-copy operations. Jiang et al. [35] design a different copy engine, placed within the cache controller, to alleviate pipeline and cache stalls that occur when these transfers take place. However, these works do not directly address the problem of data movement across the narrow memory channel.

A concurrent work by Lu et al. [63] proposes a heterogeneous DRAM design similar to VILLA-DRAM, called DAS-DRAM, but with a very different data movement mechanism from LISA. It introduces a row of *migration cells* into each subarray to move rows across subarrays. Unfortunately, the latency of DAS-DRAM is not scalable with movement distance, because it requires writing the migrating row into each intermediate subarray's migration cells before the row reaches its destination, which prolongs data transfer latency. In contrast, LISA provides a *direct path* to transfer data *between row buffers* between adjacent subarrays without requiring intermediate data writes into any subarray.

4.2. Cached DRAM

Several prior works (e.g., [20,23,26,38,109]) propose to add a small SRAM cache to a DRAM chip to lower the access latency for data that is kept in the SRAM cache (e.g., frequently or recently used data). There are two main disadvantages of these works. First, adding an SRAM cache into a DRAM chip is very intrusive: it incurs a high area overhead (38.8% for 64KB in a 2Gb DRAM chip) and design complexity [45,57]. Second, transferring data from DRAM to SRAM uses a narrow global data bus, internal to the DRAM chip, which is typically 64-bit wide. Thus, installing data into the DRAM cache incurs high latency. Compared to these works, our LISA-VILLA design enables low latency without significant area overhead or complexity.

4.3. Heterogeneous-Latency DRAM

Prior works propose DRAM architectures that provide heterogeneous latency either *spatially* (dependent on *where* in the memory an access targets) or *temporally* (dependent on *when* an access occurs).

Spatial Heterogeneity. Prior work introduces spatial heterogeneity into DRAM, where one region has a fast access latency but fewer DRAM rows, while the other has a slower access latency but many more rows [57, 94]. Recent works show that latency heterogeneity inherent in DRAM chips due to process or design-induced variation can also naturally enable such heterogeneous-latency substrates [9, 54]. The fast region in DRAM can be utilized as a caching area, for the frequently or recently accessed data. We briefly describe two state-of-the-art works that offer different heterogeneous-latency DRAM designs.

CHARM [94] introduces heterogeneity *within a rank* by designing a few fast banks with (1) shorter bitlines for faster data sensing, and (2) closer placement to the chip I/O for faster data transfers. To exploit these low-latency banks, CHARM uses an OS-managed mechanism to *statically* map hot data to these banks, based on profiled information from the compiler or programmers. Unfortunately, this approach *cannot adapt* to program phase changes, limiting its performance gains. If it were to adopt dynamic hot data management, CHARM would incur high migration costs over the narrow 64-bit bus that internally connects the fast and slow banks.

TL-DRAM [57] provides heterogeneity *within a subarray* by dividing it into fast (near) and slow (far) segments that have short and long bitlines, respectively, using isolation transistors. The fast segment can be managed as an OS-transparent hardware cache. The main disadvantage is that it needs to cache each hot row in *two near segments* as each subarray uses two row buffers on *opposite ends* to sense data in the open-bitline architecture (as discussed in our HPCA 2016 paper [10]). This prevents TL-DRAM from using the full near segment capacity. As we can see, neither CHARM nor TL-DRAM strike a good design balance for heterogeneous-latency DRAM. Our proposal, LISA-VILLA, is a new heterogeneous DRAM design that offers fast data movement with a low-cost and easy-to-implement design.

Temporal Heterogeneity. Prior work observes that DRAM latency can vary depending on *when* an access occurs. The key observation is that a *recently-accessed or refreshed* row has nearly full electrical charge in the cells, and thus the following access to the same row can be performed faster [21, 22, 92]. We briefly describe two state-of-the-art works that focus on providing heterogeneous latency temporally.

ChargeCache [22] enables faster access to *recently-accessed* rows in DRAM by tracking the addresses of recently-accessed rows. NUAT [92] enables accesses to recently-refreshed rows at low latency because these rows are already highly-charged. In contrast to ChargeCache and NUAT, LISA does not require data to be recently-accessed/refreshed in order to reduce DRAM latency. Adaptive-Latency DRAM (AL-DRAM) [56] adapts the DRAM latency of each DRAM module to temperature, observing that each module can be operated faster at lower temperatures. LISA is orthogonal to AL-DRAM. The ideas of LISA can be employed in conjunction with works that exploit the temporal heterogeneity of DRAM latency.

4.4. Other Latency Reduction Mechanisms

Many prior works propose memory scheduling techniques, which generally reduce latency to access DRAM [3, 13, 15, 29, 43, 44, 51, 52, 53, 68, 69, 71, 72, 96, 97, 98, 102]. Other works propose mechanisms to perform in-memory computation to reduce data movement and access latency [1, 2, 5, 6, 18, 24, 25, 40, 46, 62, 76, 77, 88, 89, 95, 107]. LISA is complementary to these works, and it can work synergistically with in-memory computation mechanisms by enabling fast aggregation of data.

5. Significance

Our HPCA 2016 paper [10] proposes a new DRAM substrate that significantly improves the performance and efficiency of bulk data movement in modern systems. In this section, we briefly discuss the expected future impact of our work, and discuss several research directions that our work motivates.

5.1. Potential Industry Impact

We believe that our LISA substrate can have a large impact on mobile systems as well as data centers that consume a significant amount of cycle time performing bulk data movement. A recent study [37] by Google reports that `memcpy()` and `memmove()` library functions alone represent 4-5% of their data center cycles even though Google has a significant workload diversity running within their data centers. Another recent study shows that 62.7% of system energy is spent on data movement on consumer devices (e.g., smartphones, wearable devices, web-based computers such as Chromebooks) [6]. In this work, we demonstrate that one potential application of using the LISA substrate is to accelerate `memcpy()` and `memmove()`, as discussed in Section 3.1. Our detailed DRAM circuit model reports that LISA reduces the latency and DRAM energy of these functions by 9x and 69x compared to today's systems, respectively. Hence, we expect LISA can improve the efficiency and performance of *both* mobile and data center systems.

5.2. Future Research Directions

This work opens up several avenues of future research directions. In this section, we describe several directions that can enable researchers to tackle other problems related to memory systems based on the LISA substrate.

Reducing Subarray Conflicts via Remapping. When two memory requests access two different rows in the same bank, they have to be served serially, even if they are to different subarrays. To mitigate such *bank conflicts*, Kim et al. [45] propose *subarray-level parallelism (SALP)*, which enables multiple subarrays to remain activated at the same time. However, if two accesses are to the same subarray, they still have to be served serially. This problem is exacerbated when frequently-accessed rows reside in the same subarray. To help alleviate such *subarray conflicts*, LISA can enable a simple mechanism that efficiently remaps or moves the conflicting rows to different subarrays by exploiting fast RBM operations.

Enabling LISA to Perform 1-to-N Memory Copy or Move Operations. A typical `memcpy` or `memmove` call only allows the data to be copied from one source location to one destination location. To copy or move data from one source location to multiple different destinations, repeated calls are required. The problem is that such repeated calls incur long latency and high bandwidth consumption. One potential application that can be enabled by LISA is performing `memcpy` or `memmove` from one source location to *multiple destinations* completely in DRAM without requiring multiple calls of these operations.

By using LISA, we observe that moving data from the source subarray to the destination subarray latches the source row's data in all the intermediate subarrays' row buffer. As a result, activating these intermediate subarrays would copy their row buffers' data into the specified row within these subarrays. By extending LISA to perform multi-point (1-to-N) copy or move operations, we can significantly increase system performance of several commonly-used system operations. For example, forking multiple child processes can utilize 1-to-N copy operations to efficiently copy memory pages from the parent's address space to all the children. As another example, LISA can extend the range of in-DRAM bulk bitwise operations [85, 89]. Thus, LISA can efficiently enable architectural support to a new, useful system and programming primitive: 1-to-N bulk memory copy/movement.

In-Memory Computation with LISA. One important requirement of efficient in-memory computation is being able to move data from its stored location to the computation units with very low latency and energy. We believe using the LISA substrate can enable a new in-memory computation framework. The idea is to add a small computation unit inside each or a subset of banks, and connect these computation units to the neighboring subarrays which store the data. Doing so allows the system to utilize LISA to move bulk data from the subarrays to the computation units with low latency and low area overhead.

Extending LISA to Non-Volatile Memory. In this work, we only focus on the DRAM technology. A class of emerging memory technology is non-volatile memory (NVM), which has the capability of retaining data without power supply. We believe that the LISA substrate can be extended to NVM (e.g., PCM [48, 49, 50, 80, 81, 104, 105] and STT-MRAM [12, 19, 47]) since the memory organization of NVM mostly resembles that of DRAM. A potential application of LISA in NVM is an efficient file copy operation that does not incur costly I/O data transfer. We believe LISA can provide further benefits when main memory becomes persistent [65].

6. Conclusion

We present a new DRAM substrate, *low-cost inter-linked subarrays (LISA)*, that expedites bulk data movement across subarrays in DRAM. LISA achieves this by creating a new high-bandwidth datapath at low cost between subarrays, via the insertion of a small number of isolation transistors. We describe and evaluate three applications that are enabled by LISA. First, LISA significantly reduces the latency and memory energy consumption of bulk copy operations between subarrays over state-of-the-art mechanisms [88]. Second, LISA enables an effective in-DRAM caching scheme on a new heterogeneous DRAM organization, which uses fast subarrays for caching hot data in every bank. Third, we reduce precharge latency by connecting two precharge units of adjacent subarrays together using LISA. We experimentally show that the three applications of LISA greatly improve system performance and memory energy efficiency when used individually or together, across a variety of workloads and system configurations.

We conclude that LISA is an effective substrate that enables several effective applications. We believe that this substrate, which enables low-cost interconnections between DRAM subarrays, can pave the way for other applications that can further improve system performance and energy efficiency through fast data movement in DRAM. We greatly encourage future work to 1) investigate new applications and benefits of LISA, and 2) develop new low-cost interconnection substrates within a DRAM chip to improve internal connectivity and data transfer ability.

Acknowledgments

We thank the anonymous reviewers and SAFARI group members for their helpful feedback. We acknowledge the support of Google, Intel, NVIDIA, Samsung, and VMware. This research was supported in part by the ISTC-CC, SRC, CFAR, and NSF (grants 1212962, 1319587, and 1320531). Kevin Chang was supported in part by the SRCEA/Intel Fellowship.

References

[1] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[2] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[3] R. Ausavarungnirun *et al.*, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, 2012.
[4] S. Blagodurov *et al.*, "A Case for NUMA-Aware Contention Management on Multicore Systems," in *USENIX ATC*, 2011.
[5] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.
[6] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[7] J. Carter *et al.*, "Impulse: Building a Smarter Memory Controller," in *HPCA*, 1999.
[8] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[9] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[10] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[11] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[12] M. T. Chang *et al.*, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *HPCA*, 2013.
[13] E. Ebrahimi *et al.*, "Parallel Application Memory Scheduling," in *MICRO*, 2011.
[14] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.
[15] S. Ghose *et al.*, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.
[16] M. Gschwind, "Chip Multiprocessing and the Cell Broadband Engine," in *CF*, 2006.
[17] J. Gummaraju *et al.*, "Architectural Support for the Stream Execution Model on General-Purpose Processors," in *PACT*, 2007.
[18] Q. Guo *et al.*, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WONDP*, 2014.
[19] X. Guo *et al.*, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *ISCA*, 2010.
[20] C. A. Hart, "CDRAM in a Unified Memory Architecture," in *Intl. Computer Conference*, 1994.
[21] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[22] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[23] H. Hidaka *et al.*, "The Cache DRAM Architecture," *IEEE Micro*, 1990.
[24] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.
[25] K. Hsieh *et al.*, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *ICCD*, 2016.
[26] W.-C. Hsu and J. E. Smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," in *ISCA*, 1993.
[27] Intel Corp., "Intel®I/O Acceleration Technology," http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html.
[28] Intel Corp., "Intel 64 and IA-32 Architectures Optimization Reference Manual," 2012.
[29] E. Ipek *et al.*, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.
[30] ITRS, http://www.itrs.net/ITRS1999-2014Mtgs,Presentations&Links/2013ITRS/2013Tables/FEP_2013Tables.xlsx, 2013.
[31] ITRS, http://www.itrs.net/ITRS1999-2014Mtgs,Presentations&Links/2013ITRS/2013Tables/Interconnect_2013Tables.xlsx, 2013.
[32] JEDEC, "DDR3 SDRAM Standard," 2010.
[33] JEDEC, "DDR4 SDRAM Standard," 2012.
[34] X. Jiang *et al.*, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *HPCA*, 2010.
[35] X. Jiang *et al.*, "Architecture Support for Improving Bulk Memory Copying and Initialization Performance," in *PACT*, 2009.
[36] J. A. Kahle *et al.*, "Introduction to the Cell Multiprocessor," *IBM JRD*, 2005.
[37] S. Kanev *et al.*, "Profiling a Warehouse-Scale Computer," in *ISCA*, 2015.
[38] G. Kedem and R. P. Koganti, "WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines," Duke Univ. Dept. of Computer Science, Tech. Rep. CS-1997-03, 1997.
[39] J. S. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[40] J. S. Kim *et al.*, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.
[41] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[42] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[43] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[44] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[45] Y. Kim *et al.*, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[46] P. M. Kogge, "EXECUBE-A New Architecture for Scaleable MPPs," in *ICPP*, 1994.
[47] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.
[48] B. C. Lee *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[49] B. C. Lee *et al.*, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, vol. 53, no. 7, pp. 99–106, 2010.
[50] B. C. Lee *et al.*, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, vol. 30, no. 1, pp. 143–143, 2010.
[51] C. J. Lee *et al.*, "Prefetch-Aware DRAM Controllers," in *MICRO*, 2008.
[52] C. J. Lee *et al.*, "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," Univ. of Texas at Austin, High Performance Systems Group, Tech. Rep. TR-HPS-2010-002, 2010.
[53] C. J. Lee *et al.*, "Improving Memory Bank-Level Parallelism in the Presence of Prefetching," in *MICRO*, 2009.
[54] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[55] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[56] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[57] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[58] D. Lee *et al.*, "Simultaneous Multi Layer Access: A High Bandwidth and Low Cost 3D-Stacked Memory Interface," *TACO*, 2016.
[59] Y. Li *et al.*, "Utility-Based Hybrid Memory Management," in *CLUSTER*, 2017.
[60] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[61] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[62] Z. Liu *et al.*, "Concurrent Data Structures for Near-Memory Computing," in *SPAA*, 2017.
[63] S.-L. Lu *et al.*, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in *MICRO*, 2015.
[64] C.-K. Luk *et al.*, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI*, 2005.
[65] J. Meza *et al.*, "A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[66] J. Meza *et al.*, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," *CAL*, 2012.
[67] Micron Technology, Inc., "576Mb: x18, x36 RLDRAM3," 2011.

[68] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-core Systems," in *USENIX Security*, 2007.
[69] J. Mukundan and J. F. Martinez, "MORSE: Multi-objective Reconfigurable Self-Optimizing Memory Scheduler," in *HPCA*, 2012.
[70] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.
[71] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.
[72] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.
[73] S. O *et al.*, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *ISCA*, 2014.
[74] J. K. Ousterhout, "Why Aren't Operating Systems Getting Faster as Fast as Hardware?" in *USENIX Summer Conf.*, 1990.
[75] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[76] D. Patterson *et al.*, "A Case for Intelligent RAM," *IEEE Micro*, 1997.
[77] A. Pattnaik *et al.*, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," in *PACT*, 2016.
[78] M. Qureshi *et al.*, "A Case for MLP-Aware Cache Replacement," in *ISCA*, 2006.
[79] M. K. Qureshi *et al.*, "Adaptive Insertion Policies for High-Performance Caching," in *ISCA*, 2007.
[80] M. K. Qureshi *et al.*, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling," in *MICRO*, 2009.
[81] M. K. Qureshi *et al.*, "Scalable High Performance Main Memory System Using Phase-change Memory Technology," in *ISCA*, 2009.
[82] M. Rosenblum *et al.*, "The Impact of Architectural Trends on Operating System Performance," in *SOSP*, 1995.
[83] SAFARI Research Group, "Ramulator – GitHub Repository," https://github.com/CMU-SAFARI/ramulator.
[84] S.-Y. Seo, "Methods of Copying a Page in a Memory Device and Methods of Managing Pages in a Memory System," U.S. Patent Application 20140185395, 2014.
[85] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
[86] V. Seshadri *et al.*, "Page overlays: An enhanced virtual memory framework to enable fine-grained memory management," in *ISCA*, 2015.
[87] V. Seshadri *et al.*, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," in *PACT*, 2012.
[88] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[89] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[90] V. Seshadri *et al.*, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses," in *MICRO*, 2015.
[91] V. Seshadri *et al.*, "Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks," *TACO*, vol. 11, no. 4, pp. 51:1–51:22, 2015.
[92] W. Shin *et al.*, "NUAT: A Non-Uniform Access Time Memory Controller," in *HPCA*, 2014.
[93] A. Snavely and D. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor," in *ASPLOS*, 2000.
[94] Y. H. Son *et al.*, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.
[95] H. S. Stone, "A Logic-in-Memory Computer," *IEEE TC*, 1970.
[96] L. Subramanian *et al.*, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," in *IEEE TPDS*, 2016.
[97] L. Subramanian *et al.*, "The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost," in *ICCD*, 2014.
[98] L. Subramanian *et al.*, "Mise: Providing performance predictability and improving fairness in shared main memory systems," in *HPCA*, 2013.
[99] K. Sudan *et al.*, "Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement," in *ASPLOS*, 2010.
[100] G. Tyson *et al.*, "A Modified Approach to Data Cache Management," in *MICRO*, 1995.
[101] A. N. Udipi *et al.*, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *ISCA*, 2010.
[102] H. Usui *et al.*, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," *TACO*, vol. 12, no. 4, pp. 65:1–65:28, 2016.
[103] S. Wong *et al.*, "A Hardware Cache memcpy Accelerator," in *FPT*, 2006.
[104] H. Yoon *et al.*, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.
[105] H. Yoon *et al.*, "Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories," *TACO*, vol. 11, no. 4, pp. 40:1–40:25, 2014.
[106] X. Yu *et al.*, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," in *MICRO*, 2017.
[107] D. Zhang *et al.*, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," in *HPDC*, 2014.
[108] L. Zhang *et al.*, "The Impulse Memory Controller," *IEEE TC*, vol. 50, no. 11, pp. 1117–1132, 2001.
[109] Z. Zhang *et al.*, "Cached DRAM for ILP Processor Memory Access Latency Reduction," *IEEE Micro*, vol. 21, no. 4, Jul. 2001.
[110] L. Zhao *et al.*, "Hardware Support for Bulk Data Movement in Server Platforms," in *ICCD*, 2005.

Experimental Characterization, Optimization, and Recovery of Data Retention Errors in MLC NAND Flash Memory

Yu Cai[1] Yixin Luo[1] Erich F. Haratsch[2] Ken Mai[1] Saugata Ghose[1] Onur Mutlu[3,1]

[1]*Carnegie Mellon University* [2]*Seagate Technology* [3]*ETH Zürich*

This paper summarizes our work on experimentally characterizing, mitigating, and recovering data retention errors in multi-level cell (MLC) NAND flash memory, which was published in HPCA 2015 [10], and examines the work's significance and future potential. Retention errors, caused by charge leakage over time, are the dominant source of flash memory errors. Understanding, characterizing, and reducing retention errors can significantly improve NAND flash memory reliability and endurance. In this work, we first characterize, with real 2Y-nm MLC NAND flash chips, how the threshold voltage distribution of flash memory changes with different retention ages – the length of time since a flash cell was programmed. We observe from our characterization results that 1) the optimal read reference voltage of a flash cell, using which the data can be read with the lowest raw bit error rate (RBER), systematically changes with its retention age, and 2) different regions of flash memory can have different retention ages, and hence different optimal read reference voltages.

Based on our findings, we propose two new techniques. First, Retention Optimized Reading (ROR) adaptively learns and applies the optimal read reference voltage for each flash memory block online. The key idea of ROR is to periodically learn a tight upper bound of the optimal read reference voltage, and from there approach the optimal read reference voltage. Our evaluations show that ROR can extend flash memory lifetime by 64% and reduce average error correction latency by 10.1%, with only 768 KB storage overhead in flash memory for a 512 GB flash-based SSD. Second, Retention Failure Recovery (RFR) recovers data with uncorrectable errors offline by identifying and probabilistically correcting flash cells with retention errors. Our evaluation shows that RFR reduces RBER by 50%, which essentially doubles the error correction capability, and thus can effectively recover data from otherwise uncorrectable flash errors.

1. Introduction

Over the past decade, the capacity of NAND flash memory has been increasing continuously, as a result of aggressive process scaling and the advent of *multi-level cell* (MLC) technology. This trend has enabled NAND flash memory to replace spinning disks for a wide range of applications – from high performance clusters and large-scale data centers to consumer PCs, laptops, and mobile devices. Unfortunately, as flash density increases, flash memory cells become more vulnerable to various types of device and circuit level noise [3, 4, 5, 8, 86] – e.g., retention noise [3, 4, 5, 8, 12, 13, 70, 80, 91], read disturb noise [3, 4, 5, 6, 15, 91], cell-to-cell program interference noise [3, 4, 5, 6, 8, 11, 14], and program/erase (P/E) cycling noise [3, 4, 5, 8, 9]. These are sources of errors that can significantly degrade NAND flash memory reliability.

A traditional solution to overcome flash errors, regardless of their source, is to use error-correcting codes (ECC) [3, 4, 5, 30, 66]. By storing a certain amount of redundant bits per unit data, ECC can detect and correct a limited number of raw bit errors. With the help of ECC, flash memory can hide these errors from the users until the number of errors per unit data exceeds the correction capability of the ECC. Flash memory designers have been relying on stronger ECC to compensate for lifetime reductions due to technology scaling. However, stronger ECC, which has higher capacity and implementation overhead, has diminishing returns on the amount of flash lifetime improvement [12, 13]. As such, we intend to look for more efficient ways of reducing flash errors.

Retention errors, caused by charge leakage over time after a flash cell is programmed, are the dominant source of flash memory errors [3, 4, 5, 8, 12, 13, 109]. The amount of charge stored in a flash memory cell determines the threshold voltage level of the cell, which in turn represents *the logical data value* stored in the cell. As illustrated in Figure 1, the threshold voltage (V_{th}) range of a 2-bit MLC NAND flash cell is divided into four regions by three read reference voltages, V_a, V_b, and V_c. The region in which the threshold voltage of a flash cell falls represents the cell's current state, which can be ER (or erased), P1, P2, or P3. Each state decodes into a 2-bit value that is stored in the flash cell (e.g., 11, 10, 00, or 01).[1]

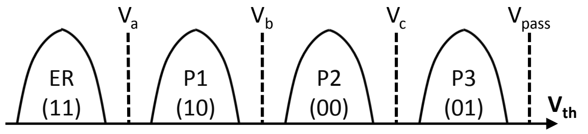


Figure 1: Threshold voltage distribution in 2-bit MLC NAND flash memory. Stored data values are represented as the tuple (LSB, MSB). Reproduced from [15].

As the manufacturing process technology for NAND flash memory scales to smaller feature sizes, the capacitance of a flash cell, and the number of electrons stored in the cell, decrease. State-of-the-art MLC flash memory cells can store only ∼100 electrons [10, 81]. Gaining or losing several electrons in a flash cell can significantly change the cell's voltage

[1]A detailed background on NAND flash memory design and operation, and on data retention errors in NAND flash memory, can be found in our prior works [3, 4, 5, 11, 12].

level and eventually alter the state of the cell. In addition, MLC technology reduces the size of the *threshold voltage window* [9], i.e., the span of threshold voltage values corresponding to each logical state, in order to store more states in a single cell. This also makes the state of a cell more likely to shift due to charge loss caused by retention noise. As such, for NAND flash memory, retention errors are one of the most important limiting factors of more aggressive process scaling and MLC technology.

One way to reduce retention errors is to periodically read, correct, and reprogram the flash memory before the number of errors accumulated over time exceed the error correction capability of the ECC, i.e., the maximum number of raw bit errors tolerable by the ECC [12, 13, 69, 90]. However, this *flash correct and refresh* (FCR) technique has two major limitations: 1) FCR uses a fixed read reference voltage to read data under different retention ages, which is suboptimal, and 2) FCR requires the flash controller to be consistently powered on so that errors can be corrected, limiting its applicability to enterprise deployments that have always-on power supplies.

In our HPCA 2015 paper [10], we pursue a better understanding of retention error behavior to improve NAND flash reliability and lifetime, and find better (and complementary) ways to mitigate flash retention errors. We characterize 1) the distortion of threshold voltage distribution at different *retention ages*, i.e., the idle time after the data is programmed to the flash memory, for state-of-the-art 2Y-nm (20- to 24-nm) NAND flash memory chips at room temperature, and 2) the retention age distribution of flash pages using disk traces taken from real workloads. Our key findings are:

1. Due to threshold voltage distribution distortion, the *optimal read reference voltages* of flash cells, at which the minimum raw bit error rate (RBER) can be achieved, systematically shift to lower values as retention age increases.
2. Pages within the same flash block (the granularity at which flash memory can be erased) tend to have similar retention ages and hence similar optimal read reference voltages, whereas pages across different flash blocks have different optimal read reference voltages.

Based on our findings, we propose two mechanisms to mitigate data retention errors. First, we propose an *online* technique called *Retention Optimized Reading* (ROR). They key idea of ROR is to reduce the raw bit error rate by adaptively learning and applying the optimal read reference voltage for each flash block. Our evaluations show that ROR extends flash lifetime by 64% and reduces average error correction latency by 10.1%, with only 768 KB storage overhead for a 512 GB flash-based SSD. Second, we propose an *offline* error recovery technique called *Retention Failure Recovery* (RFR). The key idea of RFR is to identify fast- and slow-leaking cells and probabilistically determine the original value of an erroneous cell based on its leakage-speed property and its threshold voltage. Our evaluations show that RFR can effectively reduce the average raw bit error rate (RBER) by 50%, essentially doubling the error correction capability of flash memory, and allowing for the recovery of data otherwise uncorrectable by ECC.

We first summarize our experimental characterization results (Section 2), and then introduce the Retention Optimized Reading (Section 3) and Retention Failure Recovery (Section 4) techniques.

2. Flash Data Retention Characterization

We use an FPGA-based flash memory testing platform to characterize real state- of-the-art 2Y-nm NAND flash memory chips [7, 8]. As absolute threshold voltage values are proprietary information to NAND flash vendors, we present our results using normalized voltages, where the nominal maximum value of V_{th} is equal to 512 in our normalized scale, and where 0 represents GND. Section 3.1 of our HPCA 2015 paper [10] provides a detailed description of our experimental methodology.

Figure 2 shows the threshold voltage distribution of flash memory at different retention ages for 8,000 P/E cycles. We make two observations from the figure. First, for the higher-voltage states (P2 and P3), their threshold voltage distributions systematically shift to lower voltage values as the retention age grows. Second, the distributions of each state become wider with higher retention age, and that the distributions of states at higher voltage (e.g., P3) shift faster than those of states at lower voltage (e.g., P1).

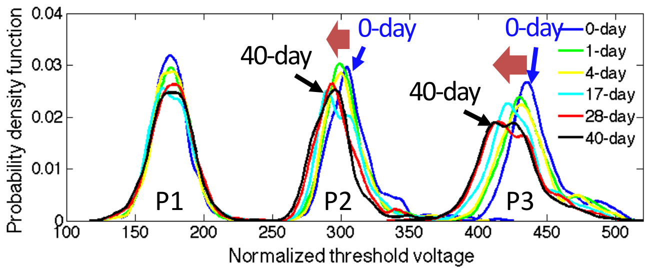


Figure 2: Threshold voltage distribution of 2Y-nm MLC NAND flash memory vs. retention age, at 8K P/E cycles under room temperature. Reproduced from [10].

We find that these changes due to retention leakage have an impact to the *optimal read reference voltage* (OPT), which is the read reference voltage between two states that minimizes the raw bit error rate (RBER). Figure 3 shows the optimal read reference voltage over retention age. We make two observations from the figure. First, Figure 3a shows a slightly decreasing trend of P1–P2 OPT (the optimal read reference voltage used to distinguish between cells in the P1 state and cells in the P2 state) over retention age. Second, we observe that P2–P3 OPT decreases much more rapidly with retention age than P1–P2 OPT, as shown in Figure 3b.

As the distributions continue to shift with growing retention age, the OPT for one retention age will be different than the OPT for a different age, suggesting that a dynamically changing OPT is ideal. To quantify how the choice of read

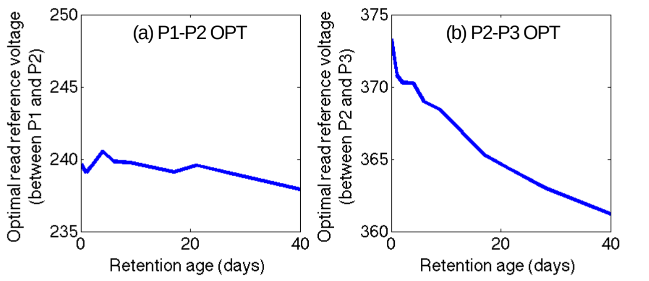


Figure 3: Effect of retention age on the optimal read reference voltage between (a) the P1 and P2 states, and (b) the P2 and P3 states. Reproduced from [10].

reference voltage affects RBER, we apply the optimal read reference voltages (OPTs) determined for {0, 1, 2, 6, 9, 17, 21, 28}-day retention ages to read 28-day-old data. Figure 4 shows the RBER obtained when reading the 28-day-old data with different OPTs, normalized to the RBER obtained when reading the data with the 28-day OPT. This figure shows that picking the correct value of OPT for each retention age results in a lower RBER. In turn, this allows us to *extend* the lifetime (i.e., the number of P/E cycles the device can tolerate) of the NAND flash memory if we always use the correct OPT based on the retention age of the data that is being read.

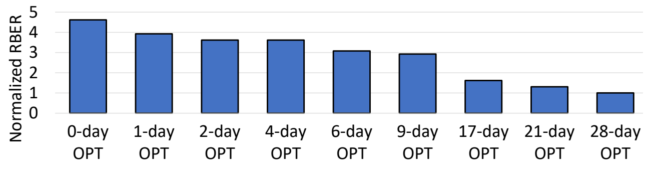


Figure 4: Normalized RBER when reading 28-day-old data with different optimal read reference voltages (normalized to 28-day OPT). Reproduced from [10].

In Section 3 of our HPCA 2015 paper [10], we perform several other experimental characterization studies of flash memory data retention behavior, and make the following eight new findings:

1. The threshold voltage distributions of the P2 and P3 states systematically shift to lower voltages with retention age.
2. The threshold voltage distribution of each state becomes wider with higher retention age.
3. The threshold voltage distribution of a higher-voltage state shifts faster than that of a lower-voltage state.
4. Both P1–P2 OPT and P2–P3 OPT become smaller over retention age.
5. P2–P3 OPT changes more significantly over retention age than P1–P2 OPT.
6. The optimal read reference voltage corresponding to one retention age is suboptimal (i.e., it results in a higher RBER) for reading data with a different retention age.
7. RBER becomes lower when the retention age for which the used read reference voltage is optimized becomes closer to the actual retention age of the data.
8. The lifetime of NAND flash memory can be extended if the optimal read reference voltage that corresponds to the retention age of the data is used.

3. Retention Optimized Reading (ROR)

To optimize flash memory performance without compromising flash lifetime, we first breakdown and analyze the components of the flash memory read latency. A read operation typically makes use of the read-retry operation [3, 4, 5, 9, 28], which performs multiple data read attempts using different read reference voltages until the read succeeds (i.e., ECC successfully corrects all of the raw bit errors). A detailed analysis of the flash memory read latency can be found in Section 4.1 of our HPCA 2015 paper [10]. We summarize the following four observations from this analysis:

- The read latency of NAND flash memory can be reduced by minimizing the number of reads performed during read-retry.
- The number of reads can be reduced by using a closer-to-optimal starting read reference voltage in the read-retry process.
- The optimal read reference voltages of pages in the same block are close, while those of pages in *different* blocks are *not* always close.
- The optimal read reference voltage of pages in a block is upper-bounded by the optimal read reference voltage of the page in the block that was programmed *last*.

Based on these observations, we propose *Retention Optimized Reading* (ROR), which consists of two components: 1) an online pre-optimization algorithm that learns the *starting* read reference voltage for each block, and 2) an improved read-retry technique that uses the starting read reference voltage to reduce the search space of OPT (i.e., the optimal read reference voltage) for the block. Section 4.2 of our HPCA 2015 paper [10] provides a detailed description of the components of ROR. We briefly summarize the components below.

The first component, the online pre-optimization algorithm, is triggered both daily and after power-on for each block. This algorithm consists of the following four steps:

- *Step 1:* The flash controller first reads the highest-numbered page in a flash block (e.g., page 255 in a block that contains 256 pages), with any default read reference voltage $V_{default}$, and attempts to correct the errors in the raw data read from the page. We chose the highest-numbered page in the block because it is programmed last, and, thus, has the lowest retention age and the highest OPT value within the block. Hence, we use the OPT for the highest-numbered page as a tight *upper bound* of OPT for the block. Next, we record the number of raw bit errors as the current lowest error count (N_{ERR}), and the applied read reference voltage as $V_{ref} = V_{default}$. If we cannot find the error count (i.e., the error is uncorrectable), we record the maximum number of errors correctable by ECC as N_{ERR}.

- *Step 2:* The controller tries to read the page using a lower read reference voltage. Since we want to find the optimal read reference voltage for the highest-numbered page in the block, we approach it from the current starting read reference voltage step by step. Since OPT typically decreases over retention age, we first attempt to lower the read reference voltage. We decrease the read reference voltage to ($V_{ref} - \Delta V$) and read the highest-numbered page. If the number of corrected errors in the new data is less than or equal to the old N_{ERR}, we update N_{ERR} and V_{ref} with the new values. We repeat Step 2 until the number of corrected errors in the new data is greater than the previous value of N_{ERR}, or the lowest possible read reference voltage is reached.
- *Step 3:* The controller tries to read the page using a higher read reference voltage. Since the optimal threshold voltage might increase in rare cases, we also attempt to increase the read reference voltage. We increase the read reference voltage to ($V_{ref} + \Delta V$) and read the highest-numbered page in the block. Again, if the number of corrected errors in the new data is less than or equal to N_{ERR}, we update N_{ERR} and V_{ref} with the new values. We repeat Step 3 until the number of corrected errors in the new data is greater than the previous value of N_{ERR}, or the highest possible read reference voltage is reached.
- *Step 4:* Record the optimal read reference voltage. After Step 3, the most recently-used value of V_{ref} is the optimal read reference voltage for the highest-numbered page. Thus, we record this voltage as the *upper bound* of the optimal read reference voltages for the block.

The second component is an improved read-retry technique that takes advantage of the recorded starting read reference voltage. During a normal read operation, the flash controller first attempts to read the data with the recorded starting read reference voltage. Then, since the recorded starting read reference voltage is the upper bound of the OPTs within the block, we iteratively decrease the read reference voltage until the read operation succeeds. Note that the starting read reference voltages are accessed frequently (on each read operation) by the flash controller, so we store them in the SSD's DRAM buffer to allow fast access.

Our key evaluation results show that ROR achieves the same flash lifetime improvements as naive read-retry, which has a read latency that is 64% longer than a baseline that uses a fixed read reference voltage. Due to a reduction in raw bit error rate, ROR reduces the ECC decoding latency by 10.1% on average compared to the baseline, which is equivalent to a 2.4% reduction in overall flash read latency. Compared with the original read-retry technique, which we explain in detail in Section 4.1 of our HPCA 2015 paper [10], ROR reduces the read-retry operation count by 70.4%, and thus reduces the overall read latency by the same fraction. This reduction is due to two reasons: 1) ROR starts the read-retry process at a close-to-optimal starting read reference voltage that is estimated and recorded daily and upon power-on; and 2) ROR approaches OPT in a known, informed direction from this starting read reference voltage.

Section 4.4 of our HPCA 2015 paper [10] provides more results from our evaluation of ROR. In our HPCA 2015 paper, we show that the performance overhead of ROR, which is periodically triggered by an online pre-optimization algorithm, can be largely hidden by executing the algorithm only when the SSD is idle, or in the background at a lower priority. This is because, even considering the worst-case scenario, we obtain an estimated pre-optimization latency of 3, 15, and 23 seconds for flash memory with a 1-day, 7-day, and 30-day-equivalent retention age, respectively. Since the flash pages within a block is programmed at similar times, the optimal read reference voltages of these pages are close. So we store one byte per block for each starting read reference voltage learned for the ER-P1 OPT, the P1–P2 OPT, and the P2–P3 OPT. We also show that ROR requires only 768 KB of storage overhead, to store the entire read reference voltage table for an assumed 512 GB flash drive.

4. Retention Failure Recovery (RFR)

Even with ROR, the retention error rate will eventually exceed the ECC limit as retention age keeps increasing. At that point, some reads will have more raw errors than can be corrected by ECC, preventing the drive from returning the data to the user. Traditionally, this would be the point of *data loss* and thus the end of flash memory lifetime.

We show that retention failure is avoidable under various circumstances. In Section 5.1 of our HPCA 2015 paper [10], we show that high temperature can significantly increase the number of retention errors in a short period of time, which leads to unexpected data loss. For example, if the required refresh period of the flash memory is one week at room temperature, uncorrectable errors may start to accumulate after a mere 36 minutes. We also discuss why completely avoiding such retention failure is unrealistic. No previous technique can prevent data loss *after* retention failure happens.

We introduce *Retention Failure Recovery* (RFR), which *enables* us to recover data from a failed flash page *offline* after the number of errors in the page exceed the total number of errors that ECC can correct. Due to process variation, different flash cells on the same chip can have different charge leakage speeds. We describe a technique to classify fast- and slow-leaking cells in just a few days, which enables RFR to probabilistically *infer* the original value stored in each flash cell. Our evaluation, based on data from real NAND flash chips, shows that RFR can reduce raw bit error rate by 50%, and thus ECC can then be used to recover a majority of the data in pages with retention failures.

Figure 5 shows how the threshold voltage of a retention-prone cell (i.e., a *fast-leaking* cell, labeled P in the figure) decreases over time (i.e., the cell shifts to the left) due to retention leakage, while the threshold voltage of a retention-resistant cell (i.e., a *slow-leaking* cell, labeled R in the figure) does *not* change significantly over time. Retention Failure Recovery (RFR) uses this classification of retention-prone versus retention-resistant cells to correct the data from the failed page *without* the assistance of ECC. Without loss of generality, let us assume that we are studying susceptible cells near the intersection of two threshold voltage distributions X and Y, where Y contains higher voltages than X. Figure 5 highlights the region of cells considered susceptible by RFR using a box, labeled *Susceptible*. A susceptible cell within the box that is retention prone likely belongs to distribution Y, as a retention-prone cell shifts rapidly to a lower voltage (see the circled cell labeled P within the *susceptible* region in the figure). A retention-resistant cell in the same *susceptible* region likely belongs to distribution X (see the boxed cell labeled R within the *susceptible* region in the figure).

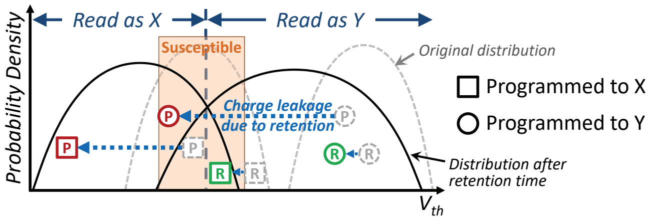


Figure 5: Some retention-prone (P) and retention-resistant (R) cells are incorrectly read after charge leakage due to retention time. RFR identifies and corrects the incorrectly read cells based on their leakage behavior. Reproduced from [3].

RFR identifies fast- vs. slow-leaking cells, and uses selective bit flipping to correct retention failures, thus reducing RBER. With reduced raw bit errors, the read data may be reconstructed by ECC with a higher probability. RFR consists of the following four offline steps, which are triggered when an uncorrectable error is found:

- *Step 1:* Identify data with a retention failure. Once the flash controller fails to read a flash page, a retention failure is identified on that page.
- *Step 2:* Identify susceptible cells using three read operations. We read the failed page using three read reference voltages: OPT (the optimal read reference voltage) minus some margin δ (Step 2.1), OPT (Step 2.2), and OPT plus δ (Step 2.3). The value of δ is large enough to include the entire *Susceptible* region shown in Figure 5. Figure 6a illustrates the identification of susceptible (i.e., risky) cells, which are denoted as type ①, type ②, type ③, and type ④ cells.
- *Step 3:* Identify fast- and slow-leaking cells. We compare the threshold voltage of susceptible cells before and after several days of retention to classify them as fast- and slow-leaking cells. Figures 6b and 6c illustrate how the cells shift differently after additional retention loss. Among the susceptible cells, type ① and type ② cells are slow-leaking cells, whereas type ③ and type ④ cells are fast-leaking cells.
- *Step 4:* Selectively flip bits based on the identification results from Step 3. Using the leakage speed information, we now know that type ② and type ③ cells are likely misread. Thus, we simply flip those cells to correct these likely errors.

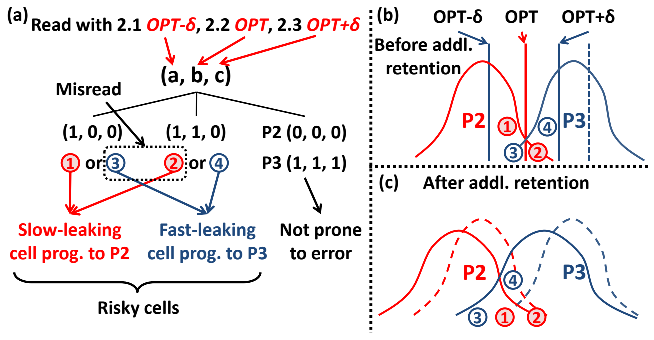


Figure 6: (a) Classification of risky (i.e., susceptible) cells to identify misread bits, (b) cells before additional retention loss, and (c) cells after additional retention loss. Reproduced from [10].

We evaluate RFR on data programmed to random values that has 28-day equivalent retention age. In Step 3, we introduce an additional 12 days' worth of equivalent retention age. Figure 7 shows the resulting raw bit error rate of RFR over a range of P/E cycles (compared to that of the baseline). This figure shows that RFR reduces the RBER by 50%, averaged across all evaluated wearout levels (P/E cycles). Thus, we expect the number of raw bit errors to be halved, increasing the chances that these errors are correctable by ECC.

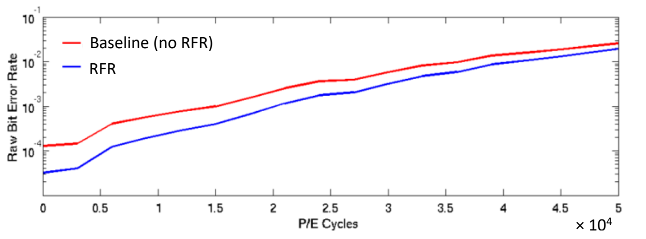


Figure 7: Effect of the RFR technique on raw bit error rate. Reproduced from [10].

5. Related Work

To our knowledge, our HPCA 2015 paper [10] is the first to 1) experimentally characterize and comprehensively analyze how the threshold voltage distribution changes *over different retention ages*, as well as the implication of these changes on the read reference voltage and lifetime, using real state-of-the-art 2Y-nm MLC NAND flash memory chips; and 2) proposes two novel techniques to mitigate the impact of retention age online and to recover from data loss by exploiting retention behavior. In this section, we briefly discuss various related works.

5.1. Works on NAND Flash Memory

NAND Flash Memory Retention Error Characterization. Multiple prior works characterize NAND flash data retention, but mainly in terms of RBER [8, 12, 13, 80]. These works show that 1) retention errors are the dominant errors in NAND flash memory, and 2) the retention error rate increases with the retention age and the P/E cycle. Papandreou et al. [91] characterize the retention effect on threshold voltage distributions under high temperature baking, and find that the distribution shifts to lower voltage over retention time, and so does the optimal read reference voltage. In contrast, our HPCA 2015 paper [10] characterizes data retention under room temperature, which is closer to how NAND flash memories are typically used [10]. Our recent work characterizes how data retention affects the threshold voltage distribution for TLC NAND flash memory [3,4,5], making similar findings as our HPCA 2015 paper [10].

NAND Flash Memory Error Characterization. Prior works study different types of NAND flash memory errors in MLC, planar NAND flash memory, including P/E cycling errors [9, 71, 80, 91, 93], programming errors [6, 71, 93], cell-to-cell program interference errors [9, 11, 14], retention errors [9,10,12,80,91], and read disturb errors [15,80,91]. These works characterize how raw bit error rate and threshold voltage distributions change with various types of noise. Our recent work characterizes the same types of errors in planar TLC NAND flash memory and has similar findings [3, 4, 5]. Thus, we believe that most of the findings on MLC NAND flash memory can be generalized to any types of planar NAND flash memory devices (e.g., SLC, MLC, TLC, or QLC). Recent works [77, 89, 101] have also studied SSD errors in the field, and have shown the system-level implications of these errors in large-scale data centers. Unlike our characterization, these in-the-field studies do *not* have access to the underlying NAND flash memory within the SSDs that they test, and, thus, are unable to show detailed data retention behavior.

3D NAND Flash Memory Error Characterization. Recently, manufacturers have begun to produce SSDs that contain *three-dimensional* (3D) NAND flash memory [36, 42, 78, 79, 92, 117]. In 3D NAND flash memory, *multiple layers* of flash cells are stacked vertically to increase the density and to improve the scalability of the memory [117]. In order to achieve this stacking, manufacturers have changed a number of underlying properties of the flash memory design. We refer readers to our prior work for a detailed comparison between 3D NAND flash memory and planar NAND flash memory [3, 4, 5]. Previous works [22, 82] compare the retention loss between 3D charge trap NAND flash memory and planar NAND flash memory through real device characterization, and find that 3D charge trap cells leak charge faster than planar NAND cells and thus experience the phenomenon of *early retention loss.* Our recent work [72] characterizes the impact of dwell time, i.e., the idle time between consecutive program cycles, and environmental temperature on the retention loss speed and program variation of 3D charge trap NAND flash memory, and proposes techniques to mitigate these issues to improve flash memory lifetime. Recent work [113] characterizes the latency and raw bit error rate of 3D NAND flash memory devices based on floating gate cells, and makes similar observations as those for planar NAND flash memory devices based on floating gate cells. Prior works have reported several differences between 3D NAND and planar NAND through circuit level measurements, including the fact that 3D NAND flash cells exhibit 1) smaller program variation at high P/E cycle [92], 2) smaller program interference [92], and 3) early retention loss [22, 22, 82]. The field (both academia and industry) is currently in much need of detailed rigorous experimental characterization and analysis of state-of-the-art 3D NAND flash memory devices.

Retention Error Mitigation Using Periodic Refresh. Prior works [12, 13, 69, 90] propose to use periodic refresh to mitigate retention errors. Cai et al. [12, 13] introduce 1) *remapping-based refresh*, which periodically reads data from each valid flash block, corrects any data errors, and *remaps* the data to a different physical location, 2) *in-place refresh*, which incrementally replenishes the lost charge of each page at its current location, and 3) *adaptive refresh*, which allows the controller to adaptively adjust the rate that the refresh mechanisms are invoked based on the wearout (i.e., the current P/E cycle count) of the NAND flash memory [12, 13]; or the temperature of the SSD [8, 10]. However, these techniques 1) require the system to be consistently powered on, and 2) are unaware of the fact that the optimal read reference voltage changes with different retention age. Note that these works always apply a *fixed* read reference voltage regardless of the retention age of the cell, which is suboptimal for reading flash blocks at *different* retention ages. In contrast, our ROR technique optimizes the read reference voltage of each flash block based on its retention age, leading to significant lifetime improvements. Several works [23, 70, 104] find that refresh operations consume a large number of P/E cycles, and propose techniques that exploit workload write-hotness to relax the guaranteed retention time of NAND flash memory without requiring refresh. For example, WARM [70] partitions write-hot and write-cold data using a lightweight mechanism designed for flash memory, and *eliminates* the need to refresh write-hot data, leading to significant lifetime improvements over existing periodic refresh mechanisms. Our techniques can be combined with such refresh elimination techniques for higher lifetime and performance.

Read Reference Voltage Optimization. A few works [11, 14, 91] propose optimizing the read reference voltage. Cai et al. [14] propose a technique to calculate the optimal read reference voltage from the mean and variance of the threshold voltage distributions, which are characterized by the read-retry technique [9]. The cost of such a technique is relatively high, as it requires periodically reading flash memory with all possible read reference voltages to discover the threshold voltage distributions. Papandreou et al. [91] propose to apply a per-block close-to-optimal read reference voltage by periodically sampling and averaging 6 OPTs within each block, learned by exhaustively trying all possible read reference voltages. In contrast, ROR can find the actual optimal read reference voltage at a much lower latency, thanks to the new findings and observations in our HPCA 2015 paper [10]. We show that ROR greatly outperforms naive read-retry. The latter is significantly simpler than the mechanism proposed in [91].

Recently, Luo et al. [71] propose to accurately predict the optimal read reference voltage using an online flash channel model for each chip learned online. Cai et al. [15] propose a new technique called V_{pass} tuning, which tunes the *pass-through voltage*, i.e., a high reference voltage applied to turn on unread cells in a block, to mitigate read disturb errors. Du et al. [27] propose to tune the optimal read reference voltages for ECC soft decoding to improve the ECC correction capability (i.e., the maximum number of errors that ECC can correct). Fukami et al. [28] propose to use read-retry to improve the reliability of the chip-off forensic analysis of NAND flash memory devices. Our proposals are complementary to all these techniques.

Error Recovery. To our knowledge, our HPCA 2015 paper [10] proposes the first mechanism that can recover data even *after* ECC is unable to successfully correct all of the errors due to retention loss. One of our works [15] builds on our HPCA 2015 paper and adapts the RFR mechanism to opportunistically recover from *read disturb errors* instead of retention errors. FlashDefibrillator (FD) [39] improves upon RFR to recover from data retention errors *online*. FD recovers data retention errors online by applying a sequence of diagnostic pulses that recharge the fast-leaking cells. This helps recover otherwise uncorrectable errors in two ways: (1) fast-leaking cells may be recharged back to the correct state, (2) fast-leaking cells recharge faster than slow-leaking cells, thus fast-leaking cells can be identified as the cells whose threshold voltages increase faster during the diagnostic pulses. These two more recent works [15,39] directly build upon our HPCA 2015 paper.

5.2. Data Retention Errors in DRAM

DRAM uses the charge within a capacitor to represent one bit of data. Much like the floating gate within NAND flash memory, charge leaks from the DRAM capacitor over time, leading to data retention issues. Unlike a NAND flash cell, where leakage typically leads to data loss after several days to years of retention time, leakage from a DRAM cell leads to data loss after a retention time on the order of *milliseconds* to *seconds* [67].

The retention time of a DRAM cell depends upon several factors [67], including (1) manufacturing process variation and (2) temperature. Manufacturing process variation affects the amount of current that leaks from each DRAM cell's capacitor and access transistor [67]. As a result, the retention time of the cells within a single DRAM chip vary significantly, resulting in *strong cells* that have high retention times and *weak cells* that have low retention times within each chip. The operating temperature affects the rate at which charge leaks from the capacitor. As the operating temperature increases, the retention time of a DRAM cell decreases exponentially [29, 67].

Due to the rapid charge leakage from DRAM cells, a DRAM controller periodically refreshes all DRAM cells in place [17,38,44,67,68,94,97] (similar to the periodic refresh techniques used in NAND flash memory, but at a much smaller time scale). DRAM standards require a DRAM cell to be refreshed once every 64 ms [38]. As the density of DRAM continues to increase over successive product generations (e.g., by 128x between 1999 and 2017 [16,18]), enabled by the scaling of DRAM to smaller manufacturing process technology nodes [73,84,85,87], the performance and energy overheads required to refresh an entire DRAM module have grown significantly [17, 68, 84, 85, 87]. It is expected that the refresh problem will get significantly worse and limit DRAM density scaling, as described in a recent work by Samsung and Intel [43] and by our group [68]. Prior analysis shows that when DRAM chip density reaches 64 Gbit, nearly 50% of the data throughput is lost due to the high amount of time spent on refreshing all of the rows in the chip, and nearly 50% of the DRAM chip power is spent on refresh operations [68]. Thus, data retention problems and refresh pose a clear challenge to DRAM scalability.

Various experimental studies of real DRAM chips (e.g., [32,44,45,50,62,67,68,94,97]) have studied the data retention time of DRAM cells in modern chips, and have shown that the vast majority of DRAM cells can retain data without loss for much longer than the 64 ms retention time specified by DRAM standards. A number of works take advantage of this variability in data retention time behavior across DRAM cells, by reducing the frequency at which the vast majority of DRAM rows within a module are refreshed (e.g., [2,37,44, 46,67,68,94,97,110]), or by reducing the interference caused by refresh requests on demand requests (e.g., [17,83,108]).

More findings on the nature of DRAM data retention and associated errors, as well as relevant experimental data from modern DRAM chips, can be found in our prior works [16,17, 32,44,45,46,47,62,67,68,84,94,97]. We also refer the readers to prior works on the design and operation of the underlying DRAM architecture [17,18,19,20,32,33,49,51,52,53,54,55,60, 61,62,63,64,67,68,94,102,103].

5.3. Errors in Emerging Nonvolatile Memory Technologies

DRAM operations are several orders of magnitude faster than SSD operations, but DRAM has two major disadvantages. First, DRAM offers orders of magnitude less storage density than NAND-flash-memory-based SSDs. Second, DRAM is volatile (i.e., the stored data is lost on a power outage). Emerging nonvolatile memories, such as *phase-change memory* (PCM) [57, 58, 59, 76, 98, 112, 115, 121], *spin-transfer torque magnetic RAM* (STT-RAM or STT-MRAM) [56, 88], *metal-oxide resistive RAM* (RRAM) [111], and *memristors* [26, 107], are expected to bridge the gap between DRAM and SSDs, providing DRAM-like access latency and energy, and at the same time SSD-like large capacity and nonvolatility (and hence SSD-like data persistence). These technologies are also expected to be used as part of *hybrid memory systems* (also called *heterogeneous memory systems*), where one part of the memory consists of DRAM modules and another part consists of modules of emerging technologies [21, 24, 25, 41, 65, 74, 75, 95, 98, 99, 100, 115, 116, 118, 119].

PCM-based devices are expected to have a limited lifetime, as PCM can only endure a certain number of writes [57, 98, 112], similar to the P/E cycling errors in NAND-flash-memory-based SSDs (though PCM's write endurance is higher than that of SSDs). PCM suffers from (1) *resistance drift* [35, 96, 112], where the resistance used to represent the value becomes higher over time (and eventually can introduce a bit error), similar to how charge leakage in NAND flash memory and DRAM lead to retention errors over time; and (2) *write disturb* [40], where the heat generated during the programming of one PCM cell dissipates into neighboring cells and can change the value that is stored within the neighboring cells. STT-RAM suffers from (1) *retention failures*, where the value stored for a single bit (as the magnetic orientation of the layer that stores the bit) can flip over time; and (2) *read disturb* (a conceptually different phenomenon from the read disturb in DRAM and flash memory), where reading a bit in STT-RAM can inadvertently induce a write to that same bit [88].

Due to the nascent nature of emerging nonvolatile memory technologies and the lack of availability of large-capacity devices built with them, extensive and dependable experimental studies have yet to be conducted on the reliability of real PCM, STT-RAM, RRAM, and memristor chips. However, we believe that error mechanisms conceptually or abstractly similar to those we discussed for flash memory and DRAM are likely to be prevalent in emerging technologies as well (as supported by some recent studies [1, 40, 48, 88, 105, 106, 120]), albeit with different underlying mechanisms and error rates. We expect that the ROR and RFR techniques we propose in our HPCA 2015 paper [10] can be easily adapted to NVM technologies.

6. Significance

Our HPCA 2015 paper [10] provides extensive characterization data and proposes novel mechanisms to mitigate retention errors in modern NAND flash memory and recover data when ECC fails. We believe that our characterization and mechanisms will have a significant impact on the community, as evidenced by multiple recent works directly building upon our HPCA 2015 paper [15, 39, 72].

6.1. Long-Term Impact

We believe our work will have long-term impact for the following three reasons. First, as NAND flash memory becomes denser in the future, data retention will become a bigger issue, and thus a better understanding of its implication and characteristics will be important to help maintain NAND flash reliability after scaling [3, 4, 5, 84]. Second, we propose an online technique that reduces flash read latency, and we give insights into the flash read-retry algorithm, thereby hopefully inspiring future works to further optimize flash read latency. Third, we propose an offline technique that leverages underlying flash characteristics to enable recovery from a retention failure even after the drive fails to correct it, thereby hopefully inspiring future works to look for more ways to prevent data loss.

Data Retention. Our work provides a comprehensive analysis of the retention loss effect on real NAND flash memory chips, which enhances the understanding of the retention loss effect in the research community. We hope that our analysis and solutions can inspire more works to handle data retention in better ways. As planar NAND flash memory becomes denser, each flash memory cell holds less charge and becomes more vulnerable to retention loss [8, 12]. Thus, in the future, we expect data retention to become a more important problem [3, 4, 5, 84], and expect that industry will be more open to adapt new solutions like our proposals, ROR and RFR. In fact, several flash-based SSDs currently use refresh as a solution to mitigate retention errors [31, 34, 114]. Our work shows that we can go significantly beyond refresh to tolerate the data retention problem in NAND flash memory.

Read Performance Optimization. The read performance advantage of flash memory over hard disk drives makes flash-based SSDs more appealing than hard disk drives. However, many existing solutions, such as read-retry [9, 28], trade off flash performance for reliability. Our HPCA 2015 paper [10] is the first to point out the read performance problem, and to provide a detailed analysis and new solution to this problem. We hope that our work can enhance the research community's understanding of flash read performance and bring more attention to flash read performance, which is critically important to overall system performance. Techniques that are developed in DRAM to reduce read latency [17, 18, 19, 20, 33, 51, 52, 53, 54, 60, 61, 62, 63, 64, 68, 94, 102, 103] can prompt inspiration for NAND flash memory.

Data Recovery. Prior to our work, after a retention failure happens, an uncorrectable error and resulting data corruption was considered to be unrecoverable from, resulting in data loss. To our knowledge, our HPCA 2015 paper [10] is the first to show that it is actually *possible* to recover this data using our RFR mechanism. As the reliability of NAND flash memory decreases, and the popularity of flash-based SSDs increases, SSD failures are expected to increase, creating a greater need for recovery techniques that can retrieve previously-unrecoverable data. In light of this, recent works [15, 39] have directly built upon RFR to provide additional data recovery mechanisms. We hope that our work draws more attention to flash memory data recovery, and inspires further solutions to this important problem.

6.2. New Research Directions

Our HPCA 2015 paper [10] presents characterization results for data retention in real NAND flash chips. By making such data and knowledge available, we believe that the flash memory and SSD research communities can have a better understanding of data retention, and can therefore develop better solutions to tackle the retention problem in the future. We hope that our work will continue to inspire future works in flash memory that can provide a comprehensive characterization and analysis of other NAND flash memory behavior using real chips, such as program/erase cycling and cell-to-cell program disturbance. We also hope that our ROR and RFR techniques bring more attention to both the flash read performance problem and data recovery problem, and that they will inspire researchers from both academia and industry to develop and adopt new solutions.

7. Conclusion

Our HPCA 2015 paper [10] comprehensively characterizes and analyzes how the threshold voltage distribution and the optimal read reference voltages of state-of-the-art 2Y-nm MLC NAND flash memory change over different retention ages. Based on these analyses, the paper proposes two new techniques. Retention Optimized Reading (ROR) improves reliability, lifetime, and performance of MLC NAND flash memory at modest storage cost by optimizing the read reference voltage of each flash memory block based on its retention age. We demonstrate significant benefits with ROR in terms of reduced RBER, extended flash lifetime, and reduction in flash read latency. Retention Failure Recovery (RFR) recovers data with uncorrectable errors by identifying and probabilistically correcting flash cells with retention errors. We demonstrate large raw bit error rate reductions with RFR. We hope that our comprehensive characterization of data retention in flash memory will enable better understanding of flash retention errors and motivate other new techniques to overcome these errors. We believe the importance of our two new techniques (ROR and RFR) will grow as NAND flash memory scales to smaller feature sizes and becomes even less reliable in the future.

Acknowledgments

We thank Nandita Vijaykumar and the anonymous reviewers for feedback. This work is partially supported by the Intel Science and Technology Center, CMU Data Storage Systems Center, and NSF grants 1212962 and 1320531.

References

[1] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, "Multilevel-Cell Phase-Change Memory: A Viable Technology," *JETCAS*, 2016.

[2] S. Baek, S. Cho, and R. Melhem, "Refresh Now and Then," *IEEE Trans. Computers*, Aug. 2014.

[3] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, Sep. 2017.

[4] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.

[5] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.

[6] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[7] Y. Cai, E. F. Haratsch, M. P. McCartney, and K. Mai, "FPGA-Based Solid-State Drive Prototyping Platform," in *FCCM*, 2011.

[8] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[9] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.

[10] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.

[11] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[12] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Flash Correct and Refresh: Retention Aware Management for Increased Lifetime," in *ICCD*, 2012.

[13] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," *Intel Technology Journal*, 2013.

[14] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor Cell Assisted Error Correction in MLC NAND Flash Memories," in *SIGMETRICS*, 2014.

[15] Y. Cai, Y. Luo, S. Ghose, E. F. Haratsch, K. Mai, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.

[16] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2017.

[17] K. K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes With Accesses," in *HPCA*, 2014.

[18] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[19] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[20] K. K. Chang, A. G. Yaglikci, A. Agrawal, N. Chatterjee, S. Ghose, A. Kashyap, H. Hassan, D. Lee, M. O'Connor, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[21] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in *MICRO*, 2012.

[22] B. Choi *et al.*, "Comprehensive Evaluation of Early Retention (Fast Charge Loss Within a Few Seconds) Characteristics in Tube-Type 3-D NAND Flash Memory," in *VLSIT*, 2016.

[23] W. Choi, M. Arjomand, M. Jung, and M. Kandemir, "Exploiting Data Longevity for Enhancing the Lifetime of Flash-based Storage Class Memory," in *SIGMETRICS*, 2017.

[24] C.-C. Chou, A. Jaleel, and M. K. Qureshi, "CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache," in *MICRO*, 2014.

[25] C.-C. Chou, A. Jaleel, and M. K. Qureshi, "BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches," in *ISCA*, 2015.

[26] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, 1971.

[27] Y. Du, Q. Li, L. Shi, D. Zou, H. Jin, and C. J. Xue, "Reducing LDPC Soft Sensing Latency by Lightweight Data Refresh for Flash Read Performance Improvement," in *DAC*, 2017.

[28] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, "Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices," *Digital Investigation*, 2017.

[29] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *IEEE Trans. Electron Devices*, Jun. 1998.

[30] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Labs Technical Journal*, 1950.

[31] P. Hansson, "When SSD Performance Goes Awry," http://www.techspot.com/article/997-samsung-ssd-read-performance-degradation/, 2015.
[32] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[33] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[34] G. Hippo, "Hippotizer V4," http://cdn.manula.com/user/8056/9036_9608_en_1479375721.pdf?v=20170201164158, 2017.
[35] D. Ielmini, A. L. Lacaita, and D. Mantegazza, "Recovery and Drift Dynamics of Resistance and Threshold Voltages in Phase-Change Memories," *TED*, 2007.
[36] J. Im *et al.*, "A 128Gb 3b/Cell V-NAND Flash Memory with 1Gb/s I/O Rate," in *ISSCC*, 2015.
[37] C. Isen and L. John, "ESKIMO — Energy Savings Using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM Subsystem," in *MICRO*, 2009.
[38] JEDEC Solid State Technology Assn., *DDR4 SDRAM Standard*, Publication JESD79-4A, 2013.
[39] J. Jeong, Y. Song, and J. Kim, "FlashDefibrillator: A Data Recovery Technique for Retention Failures in NAND Flash Memory," in *NVMSA*, 2015.
[40] L. Jiang, Y. Zhang, and J. Yang, "Mitigating Write Disturbance in Super-Dense Phase Change Memories," in *DSN*, 2014.
[41] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian, "CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms," in *HPCA*, 2010.
[42] D. Kang *et al.*, "7.1 256Gb 3b/cell V-NAND Flash Memory With 48 Stacked WL Layers," in *ISSCC*, 2016.
[43] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. Choi, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *Memory Forum*, 2014.
[44] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[45] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[46] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *IEEE Comput. Archit. Lett.*, 2016.
[47] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[48] W.-S. Khwa *et al.*, "A Resistance-Drift Compensation Scheme to Reduce MLC PCM Raw BER by Over 100x for Storage-Class Memory Applications," in *ISSCC*, 2016.
[49] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.
[50] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *IEEE Electron Device Lett.*, Aug. 2009.
[51] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[52] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[53] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[54] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[55] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.
[56] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.
[57] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[58] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," *Commun. ACM*, Jul. 2010.
[59] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, Feb. 2010.
[60] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[61] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *TACO*, 2016.
[62] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[63] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[64] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[65] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-Based Hybrid Memory Management," in *CLUSTER*, 2017.
[66] S. Lin and D. J. Costello, *Error Control Coding*. Prentice Hall, 2004.
[67] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[68] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[69] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing NAND Flash-Based SSDs via Retention Relaxation," in *FAST*, 2012.
[70] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime With Write-Hotness Aware Retention Management," in *MSST*, 2015.
[71] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.
[72] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.
[73] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," *IBM J. Research Develop.*, Mar. 2002.
[74] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[75] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management," *IEEE Comput. Archit. Lett.*, Feb. 2012.
[76] J. Meza, J. Li, and O. Mutlu, "Evaluating Row Buffer Locality in Future Non-Volatile Main Memories," Carnegie Mellon Univ., SAFARI Research Group, Tech. Rep. TR-SAFARI-2012-002, 2012.
[77] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A Large-Scale Study of Flash Memory Failures In The Field," in *SIGMETRICS*, 2015.
[78] R. Micheloni, Ed., *3D Flash Memories*. Dordrecht, Netherlands: Springer Netherlands, 2016.
[79] R. Micheloni, S. Aritome, and L. Crippa, "Array Architectures for 3-D NAND Flash Memories," *Proc. IEEE*, Sep. 2017.
[80] N. Mielke, T. Marquart, N.Wu, J.Kessenich, H. Belgal, E. Schares, and F. Triverdi, "Bit Error Rate in NAND Flash Memories," in *IRPS*, 2008.
[81] I. Min, "Enterprise NAND Flash Memory with 1x-nm Technology," in *FMS*, 2014.
[82] K. Mizoguchi, T. Takahashi, S. Aritome, and K. Takeuchi, "Data-Retention Characteristics Comparison of 2D and 3D TLC NAND Flash Memories," in *IMW*, 2017.
[83] J. Mukundan, H. Hunter, K.-H. Kim, J. Stuecheli, and J. F. Martínez, "Understanding and Mitigating Refresh Overheads in High-Density DDR4 DRAM Systems," in *ISCA*, 2013.
[84] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[85] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[86] O. Mutlu, "Error Analysis and Management for MLC NAND Flash Memory," in *FMS*, 2014.
[87] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[88] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[89] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD Failures in Datacenters: What? When? and Why?" in *SYSTOR*, 2016.
[90] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-Nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System Performance for Enterprise Applications," in *HPCA*, 2012.
[91] N. Papandreou, T. Parnell, H. Pozidis, T. Mittelholzer, E. Eleftheriou, C. Camp, T. Griffin, G. Tressler, and A. Walls, "Using adaptive read voltage thresholds to enhance the reliability of mlc nand flash memory systems," in *GLSVLSI*, 2014.
[92] K. Park *et al.*, "Three-Dimensional 128 Gb MLC Vertical NAND Flash Memory With 24-WL Stacked Layers and 50 MB/s High-Speed Programming," *J. Solid-State Circuits*, Jan. 2015.
[93] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis, "Modelling of the Threshold Voltage Distributions of Sub-20nm NAND Flash Memory," in *GLOBECOM*, 2014.
[94] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[95] S. Phadke and S. Narayanasamy, "MLP Aware Heterogeneous Memory System," in *DATE*, 2011.

[96] A. Pirovano, A. L. Lacaita, F. Pellizzer, S. A. Kostylev, A. Benvenuti, and R. Bez, "Low-Field Amorphous State Resistance and Threshold Voltage Drift in Chalcogenide Materials," *TED*, 2004.

[97] M. K. Qureshi, D. H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.

[98] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *ISCA*, 2009.

[99] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-Off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *MICRO*, 2012.

[100] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page Placement in Hybrid Memory Systems," in *ICS*, 2011.

[101] B. Schroeder, A. Merchant, and R. Lagisetty, "Reliability of NAND-based SSDs: What field studies tell us," *Proc. IEEE*, 2017.

[102] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[103] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[104] L. Shi, K. Wu, M. Zhao, C. J. Xue, D. Liu, and E. H.-M. Sha, "Retention Trimming for Lifetime Improvement of Flash Memory Storage Systems," *TCAD*, 2016.

[105] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy, "Challenges for High-Density 16Gb ReRAM with 27nm Technology," in *VLSIC*, 2015.

[106] S. Sills, S. Yasuda, J. Strand, A. Calderoni, K. Aratani, A. Johnson, and N. Ramaswamy, "A Copper ReRAM Cell for Storage Class Memory Applications," in *VLSIT*, 2014.

[107] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, 2008.

[108] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *MICRO*, 2010.

[109] S. Tanakamaru, C. Hung, A. Esumi, M. Ito, K. Li, and K. Takeuchi, "95%-Lower-BER 43%-Lower-Power Intelligent Solid-State Drive (SSD) With Asymmetric Coding and Stripe Pattern Elimination Algorithm," in *ISSCC*, 2011.

[110] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.

[111] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.

[112] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.

[113] Q. Xiong, F. Wu, Z. Lu, Y. Zhu, Y. Zhou, Y. Chu, C. Xie, and P. Huang, "Characterizing 3D Floating Gate NAND Flash," in *SIGMETRICS*, 2017.

[114] K. Yamada, "How to Fix Slow Speeds of Samsung TLC SSDs in Ultrabooks," http://www.makeuseof.com/tag/fix-slow-ultrabook-ssd/, 2015.

[115] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," *TACO*, 2014.

[116] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.

[117] J. H. Yoon, "3D NAND Technology: Implications to Enterprise Storage Applications," in *FMS*, 2015.

[118] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," in *MICRO*, 2017.

[119] W. Zhang and T. Li, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures," in *PACT*, 2009.

[120] Z. Zhang, W. Xiao, N. Park, and D. J. Lilja, "Memory Module-Level Testing and Error Behaviors for Phase Change Memory," in *ICCD*, 2012.

[121] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.

Read Disturb Errors in MLC NAND Flash Memory

Yu Cai[1] Yixin Luo[1] Saugata Ghose[1] Erich F. Haratsch[2] Ken Mai[1] Onur Mutlu[3,1]

[1]*Carnegie Mellon University* [2]*Seagate Technology* [3]*ETH Zürich*

This paper summarizes our work on experimentally characterizing, mitigating, and recovering read disturb errors in multi-level cell (MLC) NAND flash memory, which was published in DSN 2015 [16], and examines the work's significance and future potential. NAND flash memory reliability continues to degrade as the memory is scaled down and more bits are programmed per cell. A key contributor to this reduced reliability is read disturb, *where a read to one row of cells impacts the threshold voltages of* unread *flash cells in different rows of the same block. Such disturbances may shift the threshold voltages of these unread cells to different logical states than originally programmed, leading to read errors that hurt endurance.*

For the first time in open literature, this work experimentally characterizes read disturb errors on state-of-the-art 2Y-nm (i.e., 20-24 nm) MLC NAND flash memory chips. Our findings (1) correlate the magnitude of threshold voltage shifts with read operation counts, (2) demonstrate how program/erase cycle count and retention age affect the read-disturb-induced error rate, and (3) identify that lowering pass-through voltage levels reduces the impact of read disturb and extend flash lifetime. Particularly, we find that the probability of read disturb errors increases with both higher wear-out and higher pass-through voltage levels.

We leverage these findings to develop two new techniques. The first technique mitigates read disturb errors by dynamically tuning the pass-through voltage on a per-block basis. Using real workload traces, our evaluations show that this technique increases flash memory endurance by an average of 21%. The second technique recovers from previously-uncorrectable flash errors by identifying and probabilistically correcting cells susceptible to read disturb errors. Our evaluations show that this recovery technique reduces the raw bit error rate by 36%.

1. Introduction

NAND flash memory currently sees widespread usage as a storage device, having been incorporated into systems ranging from mobile devices and client computers to data center storage, as a result of its increasing capacity and decreasing cost per bit. The increasing capacity and lower cost are mainly driven by aggressive transistor scaling and *multi-level cell* (MLC) technology, where a single flash cell can store more than one bit of data. However, as NAND flash memory capacity increases, flash memory suffers from different types of circuit-level noise, which greatly impact its reliability. These include program/erase cycling noise [8, 9], cell-to-cell program interference noise [8, 11, 14], retention noise [8, 10, 12, 13, 49, 59], and read disturb noise [19, 26, 59, 87]. Among all of these types of noise, *read disturb* noise has largely been understudied in the past for MLC NAND flash, with no open-literature work available prior to our DSN 2015 paper [16] that characterizes and analyzes the read disturb phenomenon.

One reason for this prior neglect has been the heretofore low occurrence of read-disturb-induced errors in older flash technologies. In *single-level cell* (SLC) NAND flash, read disturb errors were only expected to appear after an average of one million reads to a single flash block [26, 54]. Even with the introduction of MLC NAND flash, first-generation MLC devices were expected to exhibit read disturb errors after 100,000 reads [29, 54]. As a result of manufacturing process technology scaling, some modern MLC NAND flash devices are now prone to read disturb errors after as few as 20,000 reads, with this number expected to drop even further with continued scaling [29, 54]. The exposure of these read disturb errors can be exacerbated by the uneven distribution of reads across flash blocks in contemporary workloads [65, 89], where certain flash blocks experience high temporal locality and can, therefore, more rapidly exceed the read count at which read disturb errors are induced. We refer the reader to our prior works for a more detailed background [4, 5, 6, 16].

Read disturb errors are an intrinsic result of the flash architecture. Inside each flash cell, data is stored as the *threshold voltage* of the cell, based on the logical value that the cell represents. As shown in Figure 1, during a read operation to the cell, a *read reference voltage* (i.e., V_a, V_b, or V_c) is applied to the transistor corresponding to this cell. If this read reference voltage is higher than the threshold voltage of the cell, the transistor is turned *on*. The region in which the threshold voltage of a flash cell falls represents the cell's current state, which can be ER (or erased), P1, P2, or P3. Each state decodes into a 2-bit value that is stored in the flash cell (e.g., 11, 10, 00, or 01). Note that the threshold voltage of all flash cells in a chip is bounded by an upper limit, V_{pass}, which is the *pass-through voltage*. More detailed explanations of how NAND flash memory cells work and the data retention errors in NAND flash memory can be found in our prior works [4, 5, 6, 10].

Within a flash block, the transistors of multiple cells, each from a different flash page, are tied together as a single *bitline*, which is connected to a single output wire. Only one cell is read at a time per bitline. In order to read one cell (i.e., to determine whether it is turned *on* or *off*), the transistors for the cells *not being read* must be kept *on* to allow the value from the cell being read to propagate to the output. This requires

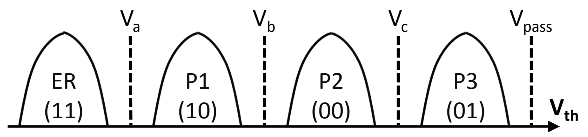


Figure 1: Threshold voltage distribution in 2-bit MLC NAND flash. Stored data values are represented as the tuple (LSB, MSB). Reproduced from [16].

the transistors to be powered with a *pass-through voltage*, which is a read reference voltage guaranteed to be higher than *any* stored threshold voltage (see Figure 1). Though these other cells are *not* being read, this high pass-through voltage induces electric tunneling that can shift the threshold voltages of these unread cells to higher values, thereby *disturbing the cell contents on a read operation to a neighboring page.* As we scale down the size of flash cells, the transistor oxide becomes thinner, which in turn increases this tunneling effect. With each read operation having an increased tunneling effect, it takes fewer read operations to neighboring pages for the unread flash cells to become disturbed (i.e., shifted to higher threshold voltages) and move into a different logical state.

In light of the increasing sensitivity of flash memory to read disturb errors, our goal is to (1) develop a thorough understanding of read disturb errors in state-of-the-art MLC NAND flash memories, by performing experimental characterization of such errors on existing commercial 2Y-nm (i.e., 20-24 nm) flash memory chips, and (2) develop mechanisms that can tolerate read disturb errors, making use of insights gained from our read disturb error characterization. The *key findings* from our quantitative characterization are:

- The effect of read disturb on threshold voltage distributions and raw bit error rates increases with both the number of reads to neighboring pages and the number of program/erase cycles on a block.
- Cells with lower threshold voltages are more susceptible to errors as a result of read disturb.
- As the pass-through voltage decreases, (1) the read disturb effect of each individual read operation becomes smaller, but (2) the read errors can increase due to reduced ability in allowing the read value to pass through the unread cells.
- If a page is recently written, a significant margin within the *ECC correction capability* (i.e., the total number of bit errors it can correct for a single read) is unused (i.e., the page can still tolerate more errors), which enables the page's pass-through voltage to be lowered safely).

We exploit these studies on the relation between the read disturb effect and the pass-through voltage (V_{pass}), to design two mechanisms that reduce the reliability impact of read disturb. First, we propose a low-cost dynamic mechanism called *V_{pass} Tuning*, which, for each block, finds the lowest pass-through voltage that retains data correctness. V_{pass} Tuning extends flash endurance by exploiting the finding that a lower V_{pass} reduces the read disturb error count. Our evaluations using real workload traces show that V_{pass} Tuning extends flash lifetime by 21%. Second, we propose *Read Disturb Recovery* (RDR), a mechanism that exploits the differences in the susceptibility of different cells to read disturb to extend the effective correction capability of error-correcting codes (ECC). RDR probabilistically identifies and corrects cells susceptible to read disturb errors. Our evaluations show that RDR reduces the raw bit error rate by 36%.

2. Characterizing Read Disturb in Real NAND Flash Memory Chips

We use an FPGA-based NAND flash testing platform in order to characterize read disturb on state-of-the-art flash chips [4,5,6,7]. We use the *read-retry* operation present within MLC NAND flash devices to accurately read the cell threshold voltage [4, 5, 6, 9, 10, 11, 12, 14, 15, 22, 69]. As threshold voltage values are proprietary information, we present our results using a *normalized threshold voltage*, where the nominal value of V_{pass} is equal to 512 in our normalized scale, and where 0 represents GND.

One limitation of using commercial flash devices is the inability to alter the V_{pass} value, as no such interface currently exists. We work around this by using the read-retry mechanism, which allows us to change the read reference voltage V_{ref} one wordline at a time. Since both V_{pass} and V_{ref} are applied to wordlines, we can mimic the effects of changing V_{pass} by instead changing V_{ref} and examining the impact on the wordline being read. We perform these experiments on one wordline per block, and repeat them over ten different blocks.

We present our major findings below. For a complete description of all of our observations, we refer the reader to our DSN 2015 paper [16].

2.1. Quantifying Read Disturb Perturbations

First, we quantify the amount by which read disturb shifts the threshold voltage, by measuring threshold voltage values for unread cells after 0, 250K, 500K, and 1 million read operations to other cells within the same flash block. Figure 2a shows the distribution of the threshold voltages for cells in a flash block after 0, 250K, 500K, and 1 million read operations. Figure 2b zooms in on this to illustrate the distribution for values in the ER state. We find that *the magnitude of the threshold voltage shift for a cell due to read disturb (1) increases with the number of read disturb operations, and (2) is higher if the cell has a lower threshold voltage.*

2.2. Effect of Read Disturb on Raw Bit Error Rate

Second, we aim to relate these threshold voltage shifts to the *raw bit error rate* (RBER), which refers to the probability of reading an incorrect state from a flash cell. We measure whether flash cells that are more worn out (i.e., cells that have been programmed and erased more times) are impacted differently due to read disturb. Figure 3 shows the RBER over an increasing number of read disturb operations for different

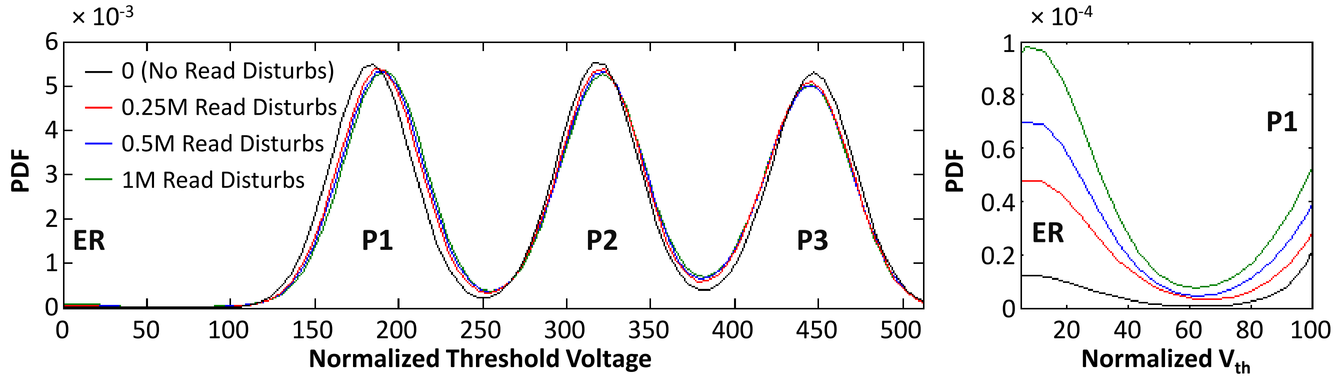
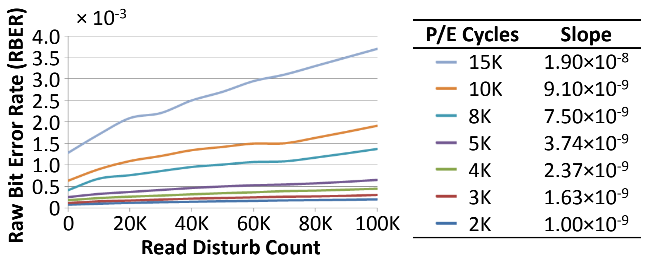


Figure 2: (a) Threshold voltage distribution of all programmed states before and after read disturb; (b) Threshold voltage distribution between erased state and P1 state. Reproduced from [16].



P/E Cycles	Slope
15K	1.90×10^{-8}
10K	9.10×10^{-9}
8K	7.50×10^{-9}
5K	3.74×10^{-9}
4K	2.37×10^{-9}
3K	1.63×10^{-9}
2K	1.00×10^{-9}

Figure 3: Raw bit error rate vs. read disturb count under different levels of program and erase (P/E) wear. Reproduced from [16].

amounts of P/E cycle wear (i.e., the amount of wearout in P/E cycles) on flash blocks. Each level shows a linear RBER increase as the read disturb count increases. We find that *(1) for a given amount of P/E cycle wear on a block, the raw bit error rate increases roughly linearly with the number of read disturb operations*, and that *(2) the effects of read disturb are greater for cells that have experienced a larger number of P/E cycles.*

2.3. Pass-Through Voltage Impact on Read Disturb

Third, we show that the cause of read disturb can be reduced by reducing (i.e., relaxing) the pass-through voltage using a circuit-level model of the flash cell, and verify this observation using real measurements. Figure 4 shows the measured change in RBER as a function of the number of read operations, for selected relaxations of V_{pass}. Note that the x-axis uses a log scale. *For a fixed number of reads, even a small decrease in the V_{pass} value can yield a significant decrease in RBER.* As an example, at 100K reads, lowering V_{pass} by 2% can reduce the RBER by as much as 50%. Conversely, *for a fixed RBER, a decrease in V_{pass} exponentially increases the number of tolerable read disturbs.* However, decreasing V_{pass} can prevent some cells' values from propagating correctly along the bitline on a read, as an unread flash cell transistor may be incorrectly turned off, thus generating new errors.

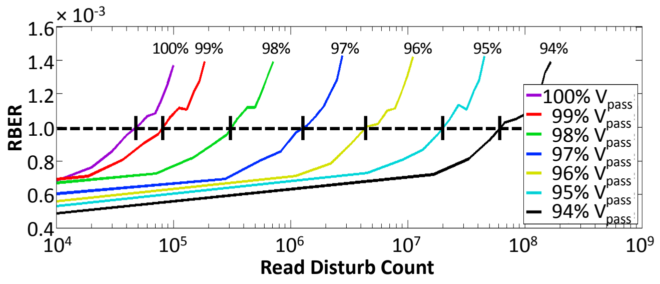


Figure 4: Raw bit error rate vs. read disturb count for different V_{pass} values, for flash memory under 8K program/erase cycles of wear. Reproduced from [16].

Unlike read disturb errors, these bitline propagation errors (or *read errors*) *do not alter the threshold voltage of the flash cell.*

2.4. Effect of Pass-Through Voltage on Raw Bit Error Rate

Fourth, setting V_{pass} to a value slightly lower than the maximum V_{th} leads to a trade-off. On the one hand, it can substantially reduce the effects of read disturb. On the other hand, it causes a small number of unread cells to incorrectly stay *off* instead of passing through a value, potentially leading to a *read error.* Therefore, if the number of read disturb errors can be dropped significantly by lowering V_{pass}, the small number of read errors introduced may be warranted. If too many read errors occur, we can always fall back to using the maximum threshold voltage for V_{pass} without consequence. Naturally, this trade-off depends on the magnitude of these error rate changes. We now explore the gains and costs, in terms of overall RBER, for relaxing V_{pass} *below* the maximum threshold voltage of a block.

To identify the extent to which relaxing V_{pass} affects the raw bit error rate, we experimentally sweep over V_{pass}, reading the data after a range of different retention ages, as shown in Figure 5. First, we observe that across all of our studied retention ages, *V_{pass} can be lowered to some degree without inducing any read errors.* For greater relaxations, though, the error rate increases as more unread cells are incorrectly turned *off* during read operations. We also note that, *for a given V_{pass} value, the additional read error rate is lower if the read is performed a longer time after the data is programmed into the flash (i.e., if the retention age is longer).* This is because of the retention loss effect, where cells slowly leak charge and thus have lower threshold voltage values over time. Naturally, as the threshold voltage of every cell decreases, a relaxed V_{pass} becomes more likely to correctly turn *on* the unread cells.

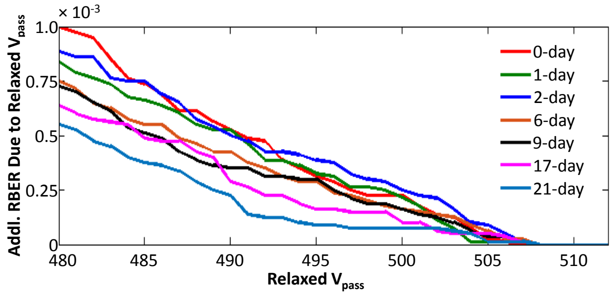


Figure 5: Additional raw bit error rate induced by relaxing V_{pass}, shown across a range of data retention ages. Reproduced from [16].

2.5. Error Correction with Reduced Pass-Through Voltage

Fifth, while we have shown, in Section 3.6 of our DSN 2015 paper [16], that V_{pass} can be lowered to some degree without introducing new raw bit errors, we would ideally like to further decrease V_{pass} to lower the read disturb impact more. This can enable flash devices to tolerate many more reads. The ECC used for NAND flash memory can tolerate an RBER of up to 10^{-3} [12, 13], which occurs only during worst-case conditions such as long retention time. Our goal is to identify how many additional raw bit errors the current level of ECC provisioning in flash chips can sustain. Figure 6 shows how the expected RBER changes over a 21-day period for our tested flash chip *without read disturb*, using a block with 8,000 P/E cycles of wear. An RBER margin (20% of the total ECC correction capability) is reserved to account for variations in the distribution of errors and other potential errors (e.g., program and erase errors). For each retention age, the maximum percentage of *safe* V_{pass} reduction (i.e., the lowest value of V_{pass} at which all read errors can still be corrected by ECC) is listed on the top of Figure 6. As we can see, by exploiting the previously-unused ECC correction capability, V_{pass} can be safely reduced by as much as 4% when the retention age is low (less than 4 days).

Our key insight from this study is that *a lowered V_{pass} can reduce the effects of read disturb, and that the read errors induced from lowering V_{pass} can be tolerated by the built-in error correction mechanism within modern flash controllers.* More results and more detailed analysis are in our DSN 2015 paper [16].

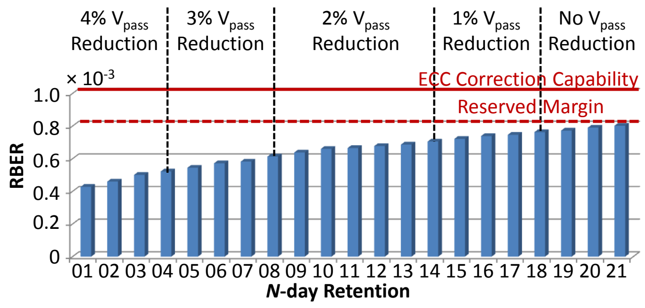


Figure 6: Overall raw bit error rate and tolerable V_{pass} reduction vs. retention age, for a flash block with 8K P/E cycles of wear. Reproduced from [16].

3. Mitigation: Pass-Through Voltage Tuning

To minimize the effect of read disturb, we propose a mechanism called V_{pass} *Tuning*, which *learns the minimum pass-through voltage for each block, such that all data within the block can be read correctly with ECC.* Figure 7 provides an exaggerated illustration of how the unused ECC capability changes over the retention period (i.e., the *refresh interval*). At the start of each retention period, there are no retention errors or read disturb errors, as the data has just been restored. In these cases, the large unused ECC capability allows us to design an *aggressive* read disturb mitigation mechanism, as we can safely *introduce correctable errors.* Thanks to read disturb mitigation, we can reduce the effect of each individual read disturb, thus lowering the total number of read disturb errors accumulated by the end of the refresh interval. This reduction in read disturb error count leads to lower *error count peaks* at the end of each refresh interval, as shown in Figure 7 by the distance between the solid black line and the dashed red line. Since flash lifetime is dictated by the number of data errors (i.e., when the total number of errors exceeds the ECC correction capability, the flash device has reached the end of its life), lowering the error count peaks extends lifetime by extending the time before these peaks exhaust the ECC correction capability.

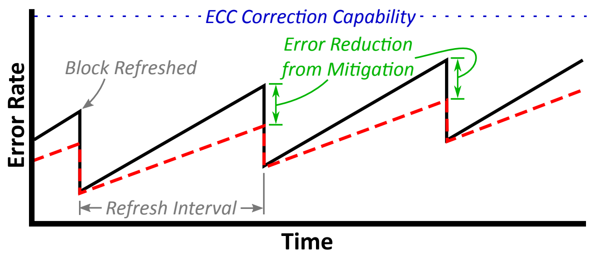


Figure 7: Exaggerated example of how read disturb mitigation reduces error rate peaks for each refresh interval. Solid black line is the unmitigated error rate, and dashed red line is the error rate after mitigation. (Note that the error rate does not include read errors introduced by reducing V_{pass}, as the unused error correction capability can tolerate errors caused by V_{pass} *Tuning*.) Reproduced from [16].

Our learning mechanism works online and is triggered on a daily basis. V_{pass} *Tuning* can be fully implemented within the *flash controller*, and has two components:

1. It first finds the size of the ECC margin M (i.e., the unused correction capability within ECC) that can be exploited to tolerate additional read errors for each block. In order to do this, our mechanism discovers the page with *approximately* the highest number of raw bit errors.
2. Once it knows the available margin M, our mechanism calibrates the pass-through voltage V_{pass} *on a per-block basis* to find the lowest value of V_{pass} that introduces no more than M additional raw errors.

The first component of our mechanism must first approximately discover the page with the highest error count, which we call the *predicted worst-case page*. After manufacturing, we statically find the predicted worst-case page by programming pseudo-randomly generated data to each page within the block, and then immediately reading the page to find the error count, as prior work on error analysis has done [8]. For each block, we record the page number of the page with the highest error count. Our mechanism obtains the error count, which we define as our *maximum estimated error* (*MEE*), by performing *a single read* to this page and reading the error count provided by ECC (once a day). We conservatively reserve 20% of the spare ECC correction capability in our calculations. Thus, if the maximum number of raw bit errors correctable by ECC is C, we calculate the available ECC margin for a block as $M = (1 - 0.2) \times C - MEE$.

The second component of our mechanism identifies the greatest V_{pass} reduction that introduces no more than M raw bit errors. The general V_{pass} *identification process* requires three steps:

Step 1: Aggressively reduce V_{pass} to $V_{pass} - \Delta$, where Δ is the smallest resolution by which V_{pass} can change.

Step 2: Apply the new V_{pass} to *all* wordlines in the block. Count the number of 0's read from the page (i.e., the number of bitlines incorrectly switched *off*) as N. If $N \leq M$ (recall that M is the extra available ECC correction margin), the read errors resulting from this V_{pass} value can be corrected by ECC, so we repeat Steps 1 and 2 to try to further reduce V_{pass}. If $N > M$, it means we have reduced V_{pass} too aggressively, so we proceed to Step 3 to roll back to an acceptable value of V_{pass}.

Step 3: Increase V_{pass} to $V_{pass} + \Delta$, and verify that the introduced read errors can be corrected by ECC (i.e., $N \leq M$). If this verification fails, we repeat Step 3 until the read errors are reduced to an acceptable range.

The implementation can be simplified greatly in practice, as the error rate changes are relatively slow over time. Over the course of the seven-day refresh interval, our mechanism must perform one of two actions each day:

Action 1: When a block is *not* refreshed, our mechanism checks once daily if V_{pass} should *increase*, to accommodate the slowly-increasing number of errors due to dynamic factors (e.g., retention errors, read disturb errors).

Action 2: When a block is refreshed, all retention and read disturb errors accumulated during the previous refresh interval are corrected. At this time, our mechanism checks how much V_{pass} can be *lowered* by.

Our mechanism repeats the V_{pass} identification process for each block that contains valid data to learn the *minimum pass-through voltage we can use*. It also repeats the entire V_{pass} learning process daily to adapt to threshold voltage changes due to retention loss [11, 14]. As such, the pass-through voltage of *all blocks* in a flash drive can be fine-tuned *continuously* to reduce read disturb and thus improve overall flash lifetime. Our DSN 2015 paper [16] describes this mechanism in more detail, and discusses a fallback mechanism for extreme cases where the additional errors accumulating between tunings exceed our 20% margin of unused error correction capability. For more detail, we refer the reader to Section 4 of our DSN 2015 paper [16].

Our mechanism can reduce V_{pass} by as much as 4%. Through a series of optimizations, described in more detail in Section 4 of our DSN 2015 paper [16], it only incurs an average daily performance overhead of 24.34 sec for a 512GB SSD, and uses only 128KB storage overhead to record per-block data.

We evaluate V_{pass} *Tuning* with I/O traces collected from a wide range of real workloads with different use cases [38, 43, 65, 83, 89]. Figure 8 shows how our mechanism can increase the endurance (measured as the number of program/erase cycles that take place before the NAND flash memory can no longer be used). We find that for a variety of our workloads, our V_{pass} tuning mechanism increases flash memory endurance by an average of 21.0%, thanks to its success in reducing the number of raw bit errors that occur due to read disturb.

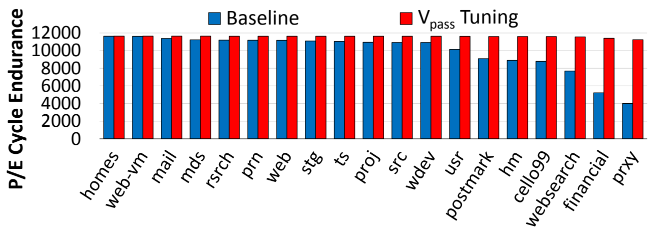


Figure 8: Endurance improvement with V_{pass} *Tuning*. Reproduced from [16].

4. Read Disturb Oriented Error Recovery

Even if we mitigate the impact of read disturb errors, a flash device will eventually exhaust its lifetime. At that point, some reads will have more raw errors to correct than can be corrected by ECC, preventing the drive from returning the correct data to the user. Traditionally, this is referred to as the point of *data loss*.

We propose to take advantage of our understanding of read disturb behavior, by designing a mechanism that can recover data *even after the device has exceeded its lifetime*.

This mechanism, which we call *Read Disturb Recovery* (RDR), (1) identifies flash cells that are *susceptible* to generating errors due to read disturb (i.e., *disturb-prone* cells), and (2) probabilistically corrects the data stored in these cells without the assistance of ECC. After these probabilistic corrections, the number of errors for a read will be brought back down, to a point at which ECC can successfully correct the remaining errors and return valid data to the user.

To understand why identifying disturb-prone cells can help with correcting errors, we study why read disturb errors occur to begin with. Figure 9a shows the state of four flash cells before read disturb happens. The two blue cells are both programmed with a two-bit value of 11, and the two red cells are programmed with a two-bit value of 00, with each two-bit value being assigned to a different range of threshold voltages (V_{th}). Between each assigned range is a margin. When read disturb occurs, the blue cells, which are *disturb-prone*, experience large V_{th} shifts upwards, while the blue cells, which are *disturb-resistant*, do not shift much, as shown in Figure 9b. Now that the distributions of these two-bit values overlap, a read error will occur for these four cells.

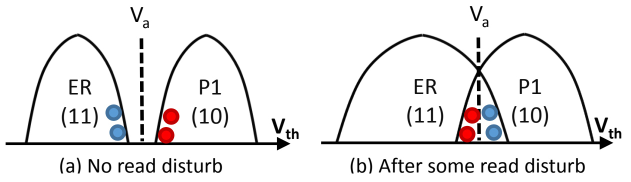


Figure 9: V_{th} distributions before and after read disturb. Reproduced from [16].

Identifying Susceptible Cells. In order to identify susceptible cells, RDR induces a significant number of additional read disturbs (e.g., 100K) within the flash cells that contain uncorrectable errors. We do this by characterizing the degree of the threshold voltage shift (ΔV_{th}) induced by the additional read disturbs, and comparing the shift to a delta threshold voltage (ΔV_{ref}) at the intersection of the two probability density functions. We classify cells with a higher threshold voltage change ($\Delta V_{th} > \Delta V_{ref}$) as *disturb-prone* cells. We classify cells with a lower or negative threshold voltage change ($\Delta V_{th} < \Delta V_{ref}$) as *disturb-resistant* cells. Section 5.2 of our DSN 2015 paper [16] provides more detailed results and analysis of disturb-prone and disturb-resistant cells.

Correcting Susceptible Cells. For flash cells with threshold voltages close to the boundary between two different data values, RDR predicts that the disturb-prone cells belong to the lower of the two voltage distributions (ER in Figure 9). Likewise, disturb-resistant cells near the boundary likely belong to the higher voltage distribution (P1 in Figure 9). This does *not* eliminate all errors, but decreases the raw bit errors in disturb-prone cells. RDR attempts to correct the remaining raw bit errors using ECC. Section 5.3 of our DSN 2015 paper [16] provides more detail on the RDR mechanism.

We evaluate how the overall RBER changes when we use RDR. Fig. 10 shows experimental results for error recovery in a flash block with 8,000 P/E cycles of wear. When RDR is applied, the *reduction in overall RBER grows with the read disturb count, from a few percent for low read disturb counts* up to 36% for 1 million read disturb operations. As data experiences a greater number of read disturb operations, the read disturb error count contributes to a significantly larger portion of the total error count, which our recovery mechanism targets and reduces. We therefore conclude that RDR can provide a large effective extension of the ECC correction capability.

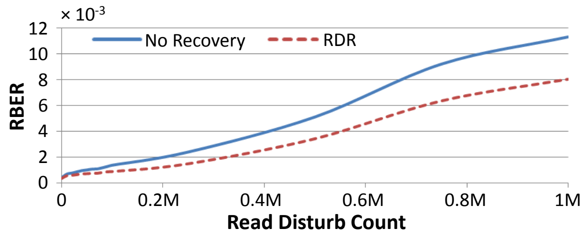


Figure 10: Raw bit error rate vs. number of read disturb operations, with and without RDR, for a flash block with 8,000 P/E cycles of wear. Reproduced from [16].

5. Related Work

We break down related work on NAND flash memory (Section 5.1) into six major categories: (1) read disturb error characterization, (2) NAND flash memory error characterization, (3) 3D NAND error characterization, (4) read disturb error mitigation, (5) voltage optimization, and (6) error recovery. We then introduce related work on read disturb errors in DRAM (Section 5.2) and emerging memory technologies (Section 5.3).

5.1. Related Works on NAND Flash Memory

Read Disturb Error Characterization. Prior to this work [16], the read disturb phenomenon for NAND flash memory has not been well explored in openly-available literature. Prior work [42] experimentally characterizes and proposes solutions for read disturb errors in DRAM. The mechanisms for disturbance and techniques to mitigate them are different between DRAM and NAND flash due to device-level differences [61]. Recent work has characterized concentrated read disturb effect and find that there are more read disturb errors on the direct neighbors to the page being repeatedly read [97]. Recent work has found that read disturb errors significantly reduce the reliability of unprogrammed and partially-programmed wordlines within a flash block, and can cause security vulnerabilities [15, 67]. These unprogrammed and partially programmed wordlines have lower threshold voltages (e.g., all cells in unprogrammed wordlines are in erased state), they are more sensitive to read disturb effect. When the wordlines are fully-programmed, NAND flash memory chip cannot correct any of these read disturb errors and thus program the misread flash cells into an incorrect state.

NAND Flash Memory Error Characterization. There are many past works from us and other research groups that analyze many different types of NAND flash memory errors in MLC, planar NAND flash memory, including P/E cycling errors [9, 52, 59, 68, 72], programming errors [15, 52, 72], cell-to-cell program interference errors [9, 11, 14], retention errors [9, 10, 12, 59, 68], and read disturb errors [16, 59, 68], and propose many different mitigation mechanisms. These works complement our DSN 2015 paper. A survey of these works (and many other related ones) can be found in our recent works [4, 5, 6]. These works characterize how raw bit error rate and threshold voltage change over various types of noise. Our recent work characterizes the same types of errors in TLC, planar NAND flash memory and has similar findings [4, 5, 6]. Thus, we believe that most of the findings on MLC NAND flash memory can be generalized to any types of planar NAND flash memory devices (e.g., SLC, MLC, TLC, or QLC). Recent work has also studied SSD errors in the field, and has shown the system-level implications of these errors to large-scale data centers [56, 66, 77].

3D NAND Error Characterization. Recently, manufacturers have begun to produce SSDs that contain *three-dimensional* (3D) NAND flash memory [33, 37, 57, 58, 70, 96]. In 3D NAND flash memory, *multiple layers* of flash cells are stacked vertically to increase the density and to improve the scalability of the memory [96]. In order to achieve this stacking, manufacturers have changed a number of underlying properties of the flash memory design. However, the internal organization of a flash block remains unchanged. Thus, read disturb errors are similar in 3D NAND flash memory. But the rate of read disturb errors are significantly reduced in today's 3D NAND because it currently uses a larger manufacturing process technology [23, 25]. We refer the reader to our prior work for a more detailed comparison between 3D NAND and planar NAND [4, 5, 6]. Recent work characterizes the latency and raw bit error rate of 3D NAND devices based on floating gate cells [94] and make similar observations as in planar NAND devices based on floating gate cells. Recent work has reported several differences between 3D NAND and planar NAND through circuit level measurements. These differences include 1) smaller program variation at high P/E cycle [70], 2) smaller program interference [70], 3) early retention loss [17, 17, 60]. We characterize the impact of dwell time, i.e., idle time between consecutive program cycles, and environment temperature on the retention loss speed and programming accuracy in 3D charge trap NAND flash cells [53]. The field (both academia and industry) is currently in much need of rigorous experimental characterization and analysis of 3D NAND flash memory devices.

Read Disturb Error Mitigation. Prior work proposes to mitigate read disturb errors by caching recently read data to avoid a read operation [85]. Prior work also proposes to mitigate read disturb errors using an idea similar to remapping-based refresh [12], known as *read reclaim*. The key idea of read reclaim is to remap the data in a block to a new flash block, if the block has experienced a high number of reads [21,29,30,40]. To bound the number of read disturb errors, some flash vendors specify a maximum number of tolerable reads for a flash block, at which point read reclaim rewrites the data to a new block (just as is done for remapping- based refresh).

Two mechanisms are currently being implemented within Yaffs (Yet Another Flash File System) to handle read disturb errors, though they are not yet available [54]. The first mechanism is similar to read reclaim [29], where a block is rewritten after a fixed number of page reads are performed to the block (e.g., 50,000 reads for an MLC chip). The second mechanism periodically inserts an additional read (e.g., a read every 256 block reads) to a page within the block, to check whether that page has experienced a read disturb error, in which case the page is copied to a new block.

Recent work proposes to remap read-hot pages to blocks configured as SLC, which are resistant to read disturb [48,100]. Ha et al. combine this read-hot page mapping technique with our V_{pass} Tuning technique and read reclaim [30] to further reduce read disturb errors. This shows that the techniques proposed by prior work are orthogonal to our read disturb mitigation techniques, and can be combined with our work for even greater protection.

Voltage Optimization. While the pass-through voltage optimization is specific to read disturb error mitigation, a few works that propose optimizing the read reference voltage have the same spirit [11, 14, 68]. Cai et al. propose a technique to calculate the optimal read reference voltage from the mean and variance of the threshold voltage distributions [14], which are characterized by the read-retry technique [9]. The cost of such a technique is relatively high, as it requires periodically reading flash memory with all possible read reference voltages to discover the threshold voltage distributions. Papandreou et al. propose to apply a per-block close-to-optimal read reference voltage by periodically sampling and averaging 6 OPTs within each block, learned by exhaustively trying all possible read reference voltages [68]. In contrast, ROR can find the actual optimal read reference voltage at a much lower latency, thanks to the new findings and observations in our DSN 2015 paper [10]. We already showed in our DSN 2015 paper that ROR greatly outperforms naive read-retry, which is significantly simpler than the mechanism proposed in [68].

Recently, Luo et al. propose to accurately predict the optimal read reference voltage using an online flash channel model for each chip learned online [52]. Cai et al. proposes a new technique called V_{pass} tuning, which tunes the *pass-through voltage*, i.e., a high reference voltage applied to turn on unread cells in a block, to mitigate read disturb errors [16]. Du et al. proposes to tune the optimal read reference voltages for ECC code soft decoding to improve ECC correction capability [20]. Fukami et al. proposes to use read-retry to improve the reliability of chip-off forensic analysis of NAND flash memory devices [22].

Error Recovery. To our knowledge, no prior work other than our DSN 2015 paper can recover the data from an uncorrectable error that is beyond the error correction capability of ECC caused by read disturb [16]. We have proposed a mechanism called RFR to opportunistically recover from uncorrectable data retention errors [4, 5, 6, 16]. RFR, similar to RDR proposed in this work, identifies *fast- and slow-leaking cells*, rather than disturb-prone and disturb-resistant cells, and probabilistically correct uncorrectable retention errors offline.

5.2. Read Disturb Errors in DRAM

Commodity DRAM chips that are sold and used in the field today exhibit read disturb errors [42], also called *RowHammer*-induced errors [61], which are *conceptually* similar to the read disturb errors found in NAND flash memory. Repeatedly accessing the same row in DRAM can cause bit flips in data stored in adjacent DRAM rows. In order to access data within DRAM, the row of cells corresponding to the requested address must be *activated* (i.e., opened for read and write operations). This row must be *precharged* (i.e., closed) when another row in the same DRAM bank needs to be activated. Through experimental studies on a large number of real DRAM chips, we show that when a DRAM row is activated and precharged repeatedly (i.e., *hammered*) enough times within a DRAM refresh interval, one or more bits in physically-adjacent DRAM rows can be flipped to the wrong value [42].

We tested 129 DRAM modules manufactured by three major manufacturers (A, B, and C) between 2008 and 2014, using an FPGA-based experimental DRAM testing infrastructure [31] (more detail on our experimental setup, along with a list of all modules and their characteristics, can be found in our original RowHammer paper [42]). Figure 11 shows the rate of RowHammer errors that we found, with the 129 modules that we tested categorized based on their manufacturing date. We find that 110 of our tested modules exhibit RowHammer errors, with the earliest such module dating back to 2010. In particular, we find that *all* of the modules manufactured in 2012–2013 that we tested are vulnerable to RowHammer. Like with many NAND flash memory error mechanisms, especially read disturb, RowHammer is a recent phenomenon that especially affects DRAM chips manufactured with more advanced manufacturing process technology generations.

Figure 12 shows the distribution of the number of rows (plotted in log scale on the y-axis) within a DRAM module that flip the number of bits along the x-axis, as measured for example DRAM modules from three different DRAM manufacturers [42]. We make two observations from the figure. First, the number of bits flipped when we hammer a row (known as the *aggressor row*) can vary significantly within a module. Second, each module has a different distribution

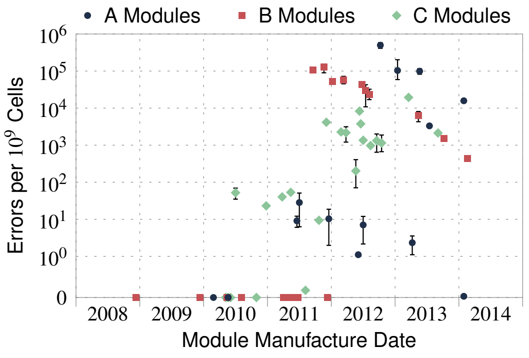


Figure 11: RowHammer error rate vs. manufacturing dates of 129 DRAM modules we tested. Reproduced from [42].

of the number of rows. Despite these differences, we find that this DRAM failure mode affects more than 80% of the DRAM chips we tested [42]. As indicated above, this read disturb error mechanism in DRAM is popularly called RowHammer [61].

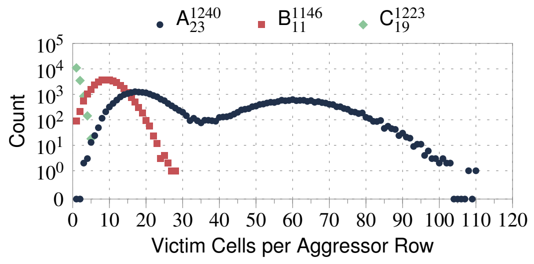


Figure 12: Number of victim cells (i.e., number of bit errors) when an aggressor row is repeatedly activated, for three representative DRAM modules from three major manufacturers. We label the modules in the format X_n^{yyww}, where X is the manufacturer (A, B, or C), $yyww$ is the manufacture year (yy) and week of the year (ww), and n is the number of the selected module. Reproduced from [42].

Various recent works show that RowHammer can be maliciously exploited by user-level software programs to (1) induce errors in existing DRAM modules [42, 61] and (2) launch attacks to compromise the security of various systems [2, 3, 27, 28, 34, 61, 74, 76, 78, 79, 90, 93]. For example, by exploiting the RowHammer read disturb mechanism, a user-level program can gain kernel-level privileges on real laptop systems [78, 79], take over a server vulnerable to RowHammer [28], take over a victim virtual machine running on the same system [2], and take over a mobile device [90]. Thus, the RowHammer read disturb mechanism is a prime (and perhaps the first) example of how a circuit-level failure mechanism in DRAM can cause a practical and widespread system security vulnerability.

Note that various solutions to RowHammer exist [41, 42, 61], but we do not discuss them in detail here. Our recent work [61] provides a comprehensive overview. A very promising proposal is to modify either the memory controller or the DRAM chip such that it probabilistically refreshes the physically-adjacent rows of a recently-activated row, with very low probability. This solution is called *Probabilistic Ad-*

jacent Row Activation (PARA) [42]. Our prior work shows that this low-cost, low-complexity solution, which does not require any storage overhead, greatly closes the RowHammer vulnerability [42].

The RowHammer effect in DRAM worsens as the manufacturing process scales down to smaller node sizes [42,61,62,63]. More findings on RowHammer, along with extensive experimental data from real DRAM devices, can be found in our prior works [41, 42, 61].

5.3. Errors in Emerging Memory Technologies

Emerging nonvolatile memories [55], such as *phase-change memory* (PCM) [45, 46, 47, 75, 92, 95, 99], *spin-transfer torque magnetic RAM* (STT-RAM or STT-MRAM) [44, 64], *metal-oxide resistive RAM* (RRAM) [91], and *memristors* [18, 84], are expected to bridge the gap between DRAM and NAND-flash-memory-based SSDs, providing DRAM-like access latency and energy, and at the same time SSD-like large capacity and nonvolatility (and hence SSD-like data persistence). While their underlying designs are different from DRAM and NAND flash memory, these emerging memory technologies have been shown to exhibit similar types of errors.

PCM-based devices are expected to have a limited lifetime, as PCM can only endure a certain number of writes [45,75,92], similar to the P/E cycling errors in SSDs (though PCM's write endurance is higher than that of SSDs). PCM suffers from (1) *resistance drift* [32, 73, 92], where the resistance used to represent the value becomes higher over time (and eventually can introduce a bit error), similar to how charge leakage in NAND flash memory and DRAM lead to retention errors over time; and (2) *write disturb* [35], where the heat generated during the programming of one PCM cell dissipates into neighboring cells and can change the value that is stored within the neighboring cells, similar in concept to cell-to-cell program interference in NAND flash memory.

STT-RAM suffers from (1) *retention failures*, where the value stored for a single bit (as the magnetic orientation of the layer that stores the bit) can flip over time [36,82,86]; and (2) *read disturb* (a conceptually different phenomenon from the read disturb in DRAM and flash memory), where reading a bit in STT-RAM can inadvertently induce a write to that same bit [64].

Due to the nascent nature of emerging nonvolatile memory technologies and the lack of availability of large-capacity devices built with them, extensive and dependable experimental studies have yet to be conducted on the reliability of real PCM, STT-RAM, RRAM, and memristor chips. However, we believe that error mechanisms conceptually or abstractly similar to those for flash memory and DRAM are likely to be prevalent in emerging technologies as well (as supported by some recent studies [1, 35, 39, 64, 80, 81, 98]), albeit with different underlying mechanisms and error rates.

6. Significance

Our DSN 2015 paper [16] is the first openly-available work to (1) characterize the impact of read disturb errors on commercially-available NAND flash memory devices, and (2) propose novel solutions to the read disturb errors that minimize them or recover them after error occurrence. We believe that our characterization results, analyses, and mechanisms can have a wide impact on future research on read disturb and NAND flash memory reliability.

6.1. Long-Term Impact

As flash devices continue to become more pervasive, there is renewed concern about the fewer number of writes that these flash devices can endure as they continue to scale [19, 29, 54]. This lower write endurance is a result of the larger number of errors introduced from manufacturing process technology scaling, and the use of multi-level cell technology. Today's planar NAND flash devices can endure only on the order of 100 program and erase cycles [71] without the assistance of aggressive error mitigation techniques such as data refresh [12, 50].

While there are several solutions for other types of NAND flash memory errors, read disturb has in the past been largely neglected because it has only become a significant problem at these smaller process technology nodes [29, 54]. Our work has the potential to change this relative lack of attention to read disturb for several reasons:

- We demonstrate on existing devices that read disturb is a significant problem today, and that it contributes a large number of errors that further reduce NAND flash memory endurance.
- We provide key insights as to *why* these errors occur, as well as why they will only worsen as technology scaling progresses.
- We show that it is possible to develop lightweight solutions that can alleviate the impact of read disturb.

Unfortunately, unless error mitigation techniques for read disturb are deployed in production NAND flash memory, read disturb will continue to negatively impact flash lifetime. While today's 3D NAND flash devices use larger process technologies that are less prone to read disturb effects [6, 51], future 3D NAND flash chips are expected to return to using smaller process technologies that remain susceptible to read disturb, as manufacturers continue to aggressively increase flash device densities [24, 88, 96]. With flash devices expected to remain a large component of the storage market for the foreseeable future, and with continued demand for higher flash densities, we expect that our work on read disturb can inspire manufacturers and researchers to adopt effective solutions to the read disturb problem.

The recovery mechanism that we propose, RDR, provides a new protective scheme for data storage that people have not considered before. Today, an increasingly larger volume of data is stored in data centers belonging to cloud service providers, who must provide a strong guarantee of data integrity for their end users. With flash storage continuing to expand in data centers [56, 66, 77], RDR (as well as other recovery solutions that RDR might inspire) can reduce the probability of unrecoverable data loss for high-density storage. In fact, the availability of a recovery mechanism like RDR can also influence more data centers to adopt flash memory for storage.

6.2. New Research Directions

In our DSN 2015 paper [16], we present a number of new quantitative results on the impact of read disturb errors on NAND flash reliability, as well as how several key factors affect the number of errors induced by read disturb, such as the pass-through voltage, the number of program/erase cycles, and the retention age. Such a detailed characterization was not openly available in the past. We believe that by releasing our characterization data, researchers in both academia and industry will be able to use the data to develop further mechanisms for read disturb recovery and mitigation. In addition, by exposing the importance of the read disturb problem in contemporary NAND flash devices, we expect that our work will draw more attention to the problem, and will inspire other researchers to further characterize and understand the read disturb phenomenon.

In fact, one of our recent works builds on our DSN 2015 paper and shows that read disturb errors can potentially cause security vulnerabilities in modern SSDs [15].

We also expect that RDR, our recovery approach, will inspire researchers to design other data recovery mechanisms for NAND flash memory that also leverage the intrinsic properties of flash devices. To our knowledge, our new data recovery mechanism is the first to do so, by discovering and exploiting the variation in read disturb shifts that arise from the underlying process variation within a flash chip.

7. Conclusion

We provide the first detailed experimental characterization of read disturb errors for 2Y-nm MLC NAND flash memory chips. We find that bit errors due to read disturb are much more likely to take place in cells with lower threshold voltages, as well as in cells with greater wear. We also find that reducing the pass-through voltage can effectively mitigate read disturb errors. Using these insights, we propose (1) a mitigation mechanism, called V_{pass} *Tuning*, which dynamically adjusts the pass-through voltage for each flash block online to minimize read disturb errors, and (2) an error recovery mechanism, called *Read Disturb Recovery*, which exploits the differences in susceptibility of different cells to read disturb, to probabilistically correct read disturb errors. We hope that our characterization and analysis of the read disturb phenomenon enables the development of other error mitigation and tolerance mechanisms, which will become increasingly necessary as continued flash memory scaling leads to greater susceptibility to read disturb. We also hope that our results will motivate NAND flash manufacturers to add pass-through voltage controls to next-generation chips, allowing flash controller designers to exploit our findings and design controllers that tolerate read disturb more effectively.

Acknowledgments

We thank the anonymous reviewers for their feedback. This work is partially supported by the Intel Science and Technology Center, the CMU Data Storage Systems Center, and NSF grants 0953246, 1065112, 1212962, and 1320531.

References

[1] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, “Multilevel-Cell Phase-Change Memory: A Viable Technology,” *JETCAS*, 2016.

[2] E. Bosman, K. Razavi, H. Bos, and C. Guiffrida, “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector,” in *SP*, 2016.

[3] W. Burleson, O. Mutlu, and M. Tiwari, “Who is the Major Threat to Tomorrow’s Security? You, the Hardware Designer,” in *DAC*, 2016.

[4] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives,” *Proc. IEEE*, Sep. 2017.

[5] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives,” arXiv:1706.08642 [cs.AR], 2017.

[6] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery,” arXiv:1711.11427 [cs.AR], 2017.

[7] Y. Cai, E. F. Haratsch, M. P. McCartney, and K. Mai, “FPGA-Based Solid-State Drive Prototyping Platform,” in *FCCM*, 2011.

[8] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis,” in *DATE*, 2012.

[9] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Threshold Voltage Distribution in NAND Flash Memory: Characterization, Analysis, and Modeling,” in *DATE*, 2013.

[10] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, “Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery,” in *HPCA*, 2015.

[11] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, “Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation,” in *ICCD*, 2013.

[12] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, “Flash Correct and Refresh: Retention Aware Management for Increased Lifetime,” in *ICCD*, 2012.

[13] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, “Error Analysis and Retention-Aware Error Management for NAND Flash Memory,” *Intel Technology Journal (ITJ)*, 2013.

[14] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, “Neighbor Cell Assisted Error Correction in MLC NAND Flash Memories,” in *SIGMETRICS*, 2014.

[15] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, “Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques,” in *HPCA*, 2017.

[16] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery,” in *DSN*, 2015.

[17] B. Choi *et al.*, “Comprehensive Evaluation of Early Retention (Fast Charge Loss Within a Few Seconds) Characteristics in Tube-Type 3-D NAND Flash Memory,” in *VLSIT*, 2016.

[18] L. Chua, “Memristor—The Missing Circuit Element,” *TCT*, 1971.

[19] J. Cooke, “The Inconvenient Truths of NAND Flash Memory,” *Flash Memory Summit*, 2007.

[20] Y. Du, Q. Li, L. Shi, D. Zou, H. Jin, and C. J. Xue, “Reducing LDPC Soft Sensing Latency by Lightweight Data Refresh for Flash Read Performance Improvement,” in *DAC*, 2017.

[21] H. H. Frost, C. J. Camp, T. J. Fisher, J. A. Fuxa, and L. W. Shelton, “Efficient Reduction of Read Disturb Errors in NAND Flash Memory,” U.S. Patent 7,818,525, 2010.

[22] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” *Digital Investigation*, vol. 20, pp. S1–S11, 2017.

[23] T. G. Gary Tressler and P. Breen, “Read Disturb Characterization for Next-Generation Flash Systems ,” in *Flash Memory Summit*, 2015.

[24] A. Goda and K. Parat, "Scaling Directions for 2D and 3D NAND Cells," in *IEDM*, 2012.
[25] A. Grossi, C. Zambelli, and P. Olivo, "Reliability of 3D NAND Flash Memories," in *3D Flash Memories*. Springer, 2016, pp. 29–62.
[26] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing Flash Memory: Anomalies, Observations, and Applications," in *MICRO*, 2009.
[27] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," arXiv:1710.00551 [cs.CR], 2017.
[28] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.
[29] K. Ha, J. Jeong, and J. Kim, "A Read-Disturb Management Technique for High-Density NAND Flash Memory," in *APSys*, 2013.
[30] K. Ha, J. Jeong, and J. Kim, "An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory," *TCAD*, vol. 35, no. 7, pp. 1079–1091, 2016.
[31] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[32] D. Ielmini, A. L. Lacaita, and D. Mantegazza, "Recovery and Drift Dynamics of Resistance and Threshold Voltages in Phase-Change Memories," *TED*, 2007.
[33] J. Im *et al.*, "A 128Gb 3b/Cell V-NAND Flash Memory with 1Gb/s I/O Rate," in *ISSCC*, 2015.
[34] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SysTEX*, 2017.
[35] L. Jiang, Y. Zhang, and J. Yang, "Mitigating Write Disturbance in Super-Dense Phase Change Memories," in *DSN*, 2014.
[36] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in *DAC*, 2012.
[37] D. Kang *et al.*, "7.1 256Gb 3b/cell V-NAND Flash Memory With 48 Stacked WL Layers," in *ISSCC*, 2016.
[38] J. Katcher, "Postmark: A New File System Benchmark," Network Appliance, Tech. Rep. TR3022, 1997.
[39] W.-S. Khwa *et al.*, "A Resistance-Drift Compensation Scheme to Reduce MLC PCM Raw BER by Over 100x for Storage-Class Memory Applications," in *ISSCC*, 2016.
[40] N. Kim and J.-H. Jang, "Nonvolatile Memory Device, Method of Operating Nonvolatile Memory Device and Memory System Including Nonvolatile Memory Device," U.S. Patent 8,203,881, 2012.
[41] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon Univ., 2015.
[42] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[43] R. Koller and R. Rangaswami, "I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance," *TOS*, 2010.
[44] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.
[45] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[46] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, 2010.
[47] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, 2010.
[48] C.-Y. Liu, Y.-M. Chang, and Y.-H. Chang, "Read Leveling for Flash Storage Systems," in *SYSTOR*, 2015.
[49] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing NAND Flash-Based SSDs via Retention Relaxation," in *FAST*, 2012.
[50] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime With Write-Hotness Aware Retention Management," in *MSST*, 2015.
[51] Y. Luo, "Architectural Techniques for Improving NAND Flash Memory Reliability," Ph.D. dissertation, Carnegie Mellon Univ., 2018.
[52] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.
[53] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.
[54] C. Manning, "Yaffs NAND Flash Failure Mitigation," http://www.yaffs.net/sites/yaffs.net/files/YaffsNandFailureMitigation.pdf, 2012.
[55] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[56] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A Large-Scale Study of Flash Memory Failures In The Field," in *SIGMETRICS*, 2015.
[57] R. Micheloni, Ed., *3D Flash Memories*. Dordrecht, Netherlands: Springer Netherlands, 2016.
[58] R. Micheloni, S. Aritome, and L. Crippa, "Array Architectures for 3-D NAND Flash Memories," *Proc. IEEE*, Sep. 2017.
[59] N. Mielke, T. Marquart, N.Wu, J.Kessenich, H. Belgal, E. Schares, and F. Triverdi, "Bit Error Rate in NAND Flash Memories," in *IRPS*, 2008.
[60] K. Mizoguchi, T. Takahashi, S. Aritome, and K. Takeuchi, "Data-Retention Characteristics Comparison of 2D and 3D TLC NAND Flash Memories," in *IMW*, 2017.
[61] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[62] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[63] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[64] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[65] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-Loading: Practical Power Management for Enterprise Storage," *TOS*, 2008.
[66] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD Failures in Datacenters: What? When? and Why?" in *SYSTOR*, 2016.
[67] N. Papandreou, T. Parnell, T. Mittelholzer, H. Pozidis, T. Griffin, G. Tressler, T. Fisher, and C. Camp, "Effect of Read Disturb on Incomplete Blocks in MLC NAND Flash Arrays," in *IMW*, 2016.
[68] N. Papandreou, T. Parnell, H. Pozidis, T. Mittelholzer, E. Eleftheriou, C. Camp, T. Griffin, G. Tressler, and A. Walls, "Using Adaptive Read Voltage Thresholds to Enhance the Reliability of MLC NAND Flash Memory Systems," in *GLSVLSI*, 2014.
[69] K.-T. Park *et al.*, "A 7MB/s 64Gb 3-Bit/Cell DDR NAND Flash Memory in 20nm-Node Technology," in *ISSCC*, 2011.
[70] K. Park *et al.*, "Three-Dimensional 128 Gb MLC Vertical NAND Flash Memory With 24-WL Stacked Layers and 50 MB/s High-Speed Programming," *J. Solid-State Circuits*, Jan. 2015.
[71] T. Parnell and R. Pletka, "NAND Flash Basics & Error Characteristics – Why Do We Need Smart Controllers?" in *Flash Memory Summit*, 2017.
[72] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis, "Modelling of the Threshold Voltage Distributions of Sub-20nm NAND Flash Memory," in *GLOBECOM*, 2014.
[73] A. Pirovano, A. L. Lacaita, F. Pellizzer, S. A. Kostylev, A. Benvenuti, and R. Bez, "Low-Field Amorphous State Resistance and Threshold Voltage Drift in Chalcogenide Materials," *TED*, 2004.
[74] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking Deterministic Signature Schemes Using Fault Attacks," Cryptology ePrint Archive, Report 2017/1014, 2017.
[75] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *ISCA*, 2009.
[76] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Guiffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.
[77] B. Schroeder, A. Merchant, and R. Lagisetty, "Reliability of NAND-Based SSDs: What Field Studies Tell Us," *Proc. IEEE*, 2017.
[78] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," Google Project Zero Blog, 2015.
[79] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," in *BlackHat*, 2015.
[80] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy, "Challenges for High-Density 16Gb ReRAM with 27nm Technology," in *VLSIC*, 2015.
[81] S. Sills, S. Yasuda, J. Strand, A. Calderoni, K. Aratani, A. Johnson, and N. Ramaswamy, "A Copper ReRAM Cell for Storage Class Memory Applications," in *VLSIT*, 2014.
[82] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches," in *HPCA*, 2011.
[83] Storage Network Industry Assn., "IOTTA Repository: Cello 1999." http://iotta.snia.org/traces/21
[84] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, 2008.
[85] T. Sugahara and T. Furuichi, "Memory Controller for Suppressing Read Disturb When Data Is Repeatedly Read Out," US Patent No. 8725952. 2014.
[86] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme," in *MICRO*, 2011.
[87] K. Takeuchi, S. Satoh, T. Tanaka, K. Imamiya, and K. Sakui, "A Negative Vth Cell Architecture for Highly Scalable, Excellently Noise-Immune, and Highly Reliable NAND Flash Memories," *JSSC*, 1999.
[88] Toshiba, "Toshiba Develops World's First 256Gb, 48-Layer BiCS FLASH," http://toshiba.semicon-storage.com/us/company/taec/news/2015/08/memory-20150803-1.html, 2015.
[89] Univ. of Massachusetts, "Storage: UMass Trace Repository."
[90] V. van der Veen, Y. Fratanonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Guiffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.
[91] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.

[92] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.

[93] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.

[94] Q. Xiong, F. Wu, Z. Lu, Y. Zhu, Y. Zhou, Y. Chu, C. Xie, and P. Huang, "Characterizing 3D Floating Gate NAND Flash," in *SIGMETRICS*, 2017.

[95] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," *TACO*, 2014.

[96] J. H. Yoon, "3D NAND Technology: Implications to Enterprise Storage Applications," in *Flash Memory Summit*, 2015.

[97] C. Zambelli, P. Olivo, L. Crippa, A. Marelli, and R. Micheloni, "Uniform and Concentrated Read Disturb Effects in Mid-1X TLC NAND Flash Memories for Enterprise Solid State Drives," in *IRPS*, 2017.

[98] Z. Zhang, W. Xiao, N. Park, and D. J. Lilja, "Memory Module-Level Testing and Error Behaviors for Phase Change Memory," in *ICCD*, 2012.

[99] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.

[100] Y. Zhu, F. Wu, Q. Xiong, Z. Lu, and C. Xie, "ALARM: A Location-Aware Redistribution Method to Improve 3D FG NAND Flash Reliability," in *NAS*, 2017.

Characterizing, Exploiting, and Mitigating Vulnerabilities in MLC NAND Flash Memory Programming

Yu Cai[1] Saugata Ghose[2] Yixin Luo[1,2]
Ken Mai[2] Onur Mutlu[3,2] Erich F. Haratsch[1]

[1]*Seagate Technology* [2]*Carnegie Mellon University* [3]*ETH Zürich*

This paper summarizes our work on experimentally analyzing, exploiting, and addressing vulnerabilities in multi-level cell NAND flash memory programming, which was published in the industrial session of HPCA 2017 [9], and examines the work's significance and future potential. Modern NAND flash memory chips use multi-level cells *(MLC), which store two bits of data in each cell, to improve chip density. As MLC NAND flash memory scaled down to smaller manufacturing process technologies, manufacturers adopted a* two-step programming method *to improve reliability. In two-step programming, the two bits of a multi-level cell are programmed using two separate steps, in order to minimize the amount of cell-to-cell program interference induced on neighboring flash cells.*

In this work, we demonstrate that two-step programming exposes new reliability and security vulnerabilities in state-of-the-art MLC NAND flash memory. We experimentally characterize contemporary 1X-nm (i.e., 15–19nm) flash memory chips, and find that a partially-programmed *flash cell (i.e., a cell where the second programming step has not yet been performed) is much more vulnerable to cell-to-cell interference and read disturb than a fully-programmed cell. We show that it is possible to exploit these vulnerabilities on solid-state drives (SSDs) to alter the partially-programmed data, causing (potentially malicious) data corruption. Based on our observations, we propose several new mechanisms that eliminate or mitigate these vulnerabilities in partially-programmed cells, and at the same time increase flash memory lifetime by 16%.*

1. Introduction

Solid-state drives (SSDs), which consist of NAND flash memory chips, are widely used for storage today due to significant decreases in the per-bit cost of NAND flash memory, which, in turn, have driven great increases in SSD capacity. These improvements have been enabled by both aggressive process technology scaling and the development of *multi-level cell* (MLC) technology. NAND flash memory stores data by changing the threshold voltage of each flash cell, where a cell consists of a *floating-gate transistor* [44, 74, 81]. In single-level cell (SLC) flash memory, the threshold voltage range could represent only a single bit of data. A multi-level cell uses the same threshold voltage range to represent *two* bits of data within a single cell (i.e., the range is split up into four windows, known as *states*, where each state represents one of the data values 00, 01, 10, or 11), thereby doubling storage capacity [11, 20, 37, 63, 92, 114]. In a NAND flash memory chip, a row of cells is connected together by a common *wordline*, which typically spans 32K–64K cells. Each wordline contains two *pages* of data, where a page is the granularity at which the data is read and written (i.e., programmed). The most significant bits (MSBs) of all cells on the same wordline are combined to form an *MSB page*, and the least significant bits (LSBs) of all cells on the wordline are combined to form an *LSB page* [13].

To precisely control the threshold voltage of a flash cell, the flash memory device uses *incremental step pulse programming* (ISPP) [20, 37, 63, 114]. ISPP applies multiple short pulses of a high programming voltage to each cell in the wordline being programmed, with each pulse increasing the threshold voltage of the cell by some small amount. SLC and older MLC devices programmed the threshold voltage in *one shot*, issuing all of the pulses back-to-back to program *both* bits of data at the same time. However, as flash memory scales down to smaller technology nodes, the distance between neighboring flash cells decreases, which in turn increases the *program interference* that occurs due to cell-to-cell coupling. This program interference causes errors to be introduced into neighboring cells during programming [13, 16, 29, 66, 68, 92]. To reduce this interference by half [13], manufacturers have been using *two-step programming* for MLC NAND flash memory since the 40nm technology node [92]. A large fraction of SSDs on the market today use sub-40nm MLC NAND flash memory.

Two-step programming stores each bit within an MLC flash memory cell using two *separate, partial programming* steps, as shown in Figure 1. An unprogrammed cell starts in the erased (ER) state. The first programming step programs the LSB page: for each flash cell within the page, the cell is *partially programmed* depending on the LSB being written to the cell. If the LSB of the cell should be 0, the cell is programmed into a temporary program state (TP); otherwise, it remains in the ER state. The maximum voltage of a partially-programmed cell is approximately half of the maximum possible threshold voltage of a fully-programmed flash cell. In its second step, two-step programming programs the MSB page: it reads the LSB value into a buffer inside the flash chip (called the *internal LSB buffer*) to determine the partially-programmed state of the cell's threshold voltage, and then partially programs the cell again, depending on whether the MSB of the cell is a 0 or a 1. The second programming step moves the threshold voltage from the partially-programmed state to the desired final state (i.e., ER, P1, P2, or P3). By breaking MLC programming into two separate steps, manufacturers *halve* the program interference of each programming operation [13, 68]. The SSD controller employs *shadow program sequencing* [6, 7, 8, 13, 25, 91], which interleaves the partial programming steps of a cell with the partial programming steps of neighboring cells to ensure that a *fully-programmed* cell experiences interference only from a single neighboring partial programming step.[1]

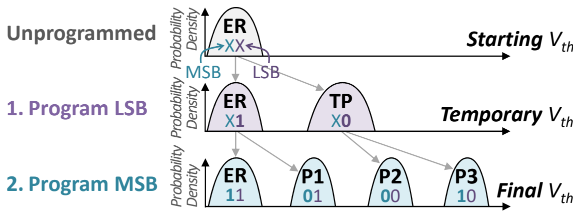


Figure 1: Starting (after erase), temporary (after LSB programming), and final (after MSB programming) states for two-step programming. Reproduced from [9].

2. Error Sources in Two-Step Programming

In our HPCA 2017 paper [9], we demonstrate that two-step programming introduces *new possibilities* for flash memory errors that can corrupt some of the data stored within flash cells without accessing them, and that these errors can be exploited to design malicious attacks. As there is a delay between programming the LSB and the MSB of a single cell due to the interleaved writes to neighboring cells, raw bit errors can be introduced into the already-programmed LSB page *before* the MSB page is programmed. These errors can cause a cell to be programmed to an incorrect state in the second programming step. During the second step, both the MSB and LSB of each cell are required to determine the final target threshold voltage of the cell. As shown in Figure 2, the data to be programmed into the MSB is loaded from the SSD controller to the internal MSB buffer (① in the figure). Concurrently, the LSB data is loaded into the internal LSB buffer from the flash memory wordline (②). By buffering the LSB data inside the flash chip and not in the SSD controller, flash manufacturers avoid data transfer between the chip and the controller during the second programming step, thereby reducing the step's latency. Unfortunately, this means that the errors loaded from the internal LSB buffer *cannot* be corrected as they would otherwise be during a read operation, because the error correction (ECC) engine resides only *inside the controller* (③), and not inside the flash chip. As a result, the final cell voltage can be *incorrectly* set during MSB programming, *permanently corrupting* the LSB data.

[1]We refer the reader to our prior works [6, 7, 8, 9, 11, 12, 14, 14, 15, 16, 17, 18, 72] for a detailed background on NAND flash memory. Our recent survey paper [6, 7, 8] provides an extensive survey of the state-of-the-art in NAND flash memory.

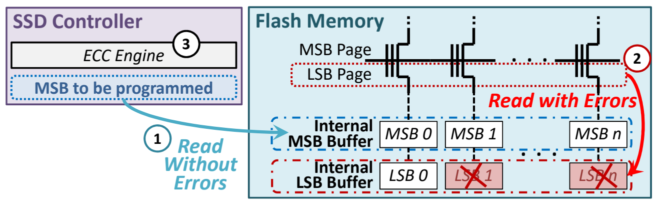


Figure 2: In the second step of two-step programming, LSB data does not go to the controller, and is not corrected when read into the internal LSB buffer, resulting in program errors. Reproduced from [9].

We briefly discuss two sources of errors that can corrupt LSB data, and characterize their impact on *real* state-of-the-art 1X-nm (i.e., 15-19nm) MLC NAND flash chips. We perform our characterization using an FPGA-based flash testing platform [10, 11] that allows us to issue commands directly to raw NAND flash memory chips. In order to determine the threshold voltage stored within each cell, we use the *read-retry* mechanism built into modern SSD controllers [13, 17, 108, 130]. Throughout this work, we present *normalized* voltage values, as actual voltage values are proprietary information to flash manufacturers. Our complete characterization results can be found in our HPCA 2017 paper [9].

2.1. Cell-to-Cell Program Interference

The first error source, *cell-to-cell program interference*, introduces errors into a flash cell when neighboring cells are programmed, as a result of parasitic capacitance coupling [6, 7, 8, 13, 16, 28, 29, 32, 68]. While two-step programming reduces program interference for fully-programmed cells, we find that interference during two-step programming is a significant error source for *partially-programmed cells*. As an example, we look at a flash block in the commonly-used all-bit-line (ABL) flash architecture [13, 19, 20], which is shown in Figure 3. After the LSB page on Wordline 1 (Page 1 in Figure 3) is programmed, the next two pages that are programmed (Pages 2 and 3) reside on directly-adjacent wordlines. Therefore, before the MSB page on Wordline 1 (Page 4) is programmed, the LSB page (Page 1) could be *susceptible* to program interference when Pages 2 and 3 are programmed.

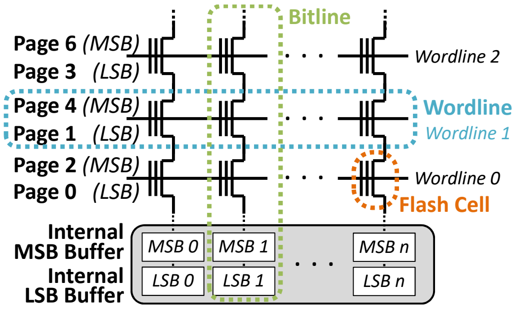


Figure 3: Internal architecture of a block of all-bit-line (ABL) flash memory. Reproduced from [9].

Figure 4 shows the measured raw bit error rate for Page 1 in real NAND flash memory devices after four different times, normalized to the error rate just after Page 1 is programmed:

A. Just after Page 1 is programmed (no interference),
B. Page 2 is programmed with pseudo-random data,
C. Pages 2 and 3 are programmed with pseudo-random data,
D. Pages 2 and 3 are programmed with a data pattern that induces the *worst-case* program interference.

We observe that the amount of interference is especially high when Pages 2 and 3 in Figure 3 are written with the worst-case data pattern, after which the raw bit error rate of Page 1 is *4.9x the rate before interference*. Note that the worst-case data pattern that we write to Pages 2 and 3 *requires no knowledge of the data stored within Page 1* [9].

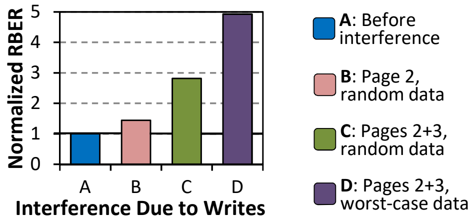


Figure 4: Normalized raw bit error rate of partially-programmed Page 1, before and after cell-to-cell program interference. Adapted from [9].

2.2. Read Disturb

The second error source, *read disturb*, disrupts the contents of a flash cell when another cell is read [6, 7, 8, 18, 28, 32, 35, 77, 90, 115]. NAND flash memory cells are organized into multiple *flash blocks* (two-dimensional cell arrays), where each block contains a set of *bitlines* that connect multiple flash cells in series. To accurately read the value from one cell, the SSD controller applies a *pass-through voltage* to turn on the *unread* cells on the bitline, which allows the value to propagate through the bitline. Unfortunately, this pass-through voltage induces a *weak programming effect* on an unread cell: it slightly increases the cell threshold voltage [6, 7, 8, 18]. As more neighboring cells within a block are read, an unread cell's threshold voltage can increase enough to change the data value stored in the cell [6, 7, 8, 18, 35, 90]. In two-step programming, a partially-programmed cell is more likely to have a lower threshold voltage than a fully-programmed cell, and the weak programming effect is stronger on cells with a lower threshold voltage. Measuring errors in real NAND flash memory devices, we find that the raw bit error rates for an LSB page in a partially-programmed or unprogrammed wordline is *an order of magnitude greater* than the rate for an LSB page in a fully-programmed wordline. However, existing read disturb management solutions are designed to protect fully-programmed cells [18, 31, 35, 36, 52, 105], and offer little mitigation for partially-programmed cells.

3. Exploiting Two-Step Programming Errors

Two major issues arise from the program interference and read disturb vulnerabilities of partially-programmed and unprogrammed cells. First, the vulnerabilities induce a large number of errors on these cells, exhausting the SSD's error correction capacity and limiting the SSD lifetime. Second, the vulnerabilities can potentially allow (malicious) applications to aggressively corrupt and change data belonging to other programs and further hurt the SSD lifetime. We present two example sketches of potential exploits in our HPCA 2017 paper [9], which we briefly summarize here.

3.1. Sketch of Program Interference Based Exploit

In this exploit, a malicious application can induce a significant amount of *program interference* onto a flash page that belongs to another, benign victim application, corrupting the page and shortening the SSD lifetime. Recall from Section 2.1 that writing the worst-case data pattern can induce 4.9x the number of errors into a neighboring page (with respect to an interference-free page). The goal of this exploit is for a malicious application to write this worst-case data pattern in a way that ensures that the page that is disrupted belongs to the victim application, and that the page that is disrupted experiences the greatest amount of program interference possible. Figure 5 illustrates the contents of the pages within neighboring 8KB *wordlines* (rows of flash cells within a block). The SSD controller uses *shadow program sequencing* to interleave partial programming steps to pages in ascending order of the page numbers shown on the left side of the figure. A malicious application can write a small 16KB file with all 1s to prepare for the attack (① in the figure), and then waits for the victim application to write its data to Wordline *n* (②). Once the victim writes its data, the malicious application then writes all 0s to a second 16KB file (③a and ③b). This induces the largest possible change in voltage on the victim data, and can be used to flip bits within the data. In our HPCA 2017 paper [9], we discuss how a malicious application can (1) work around SSD scrambling and (2) monitor victim application data writes.

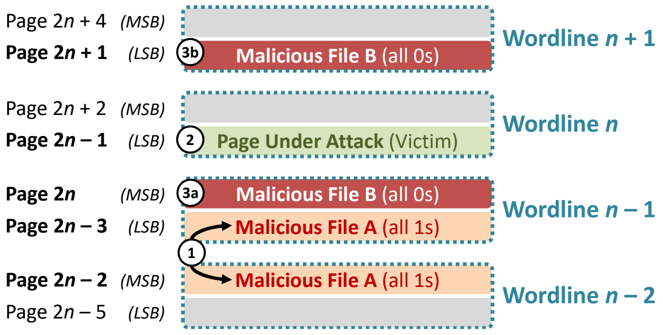


Figure 5: Layout of data within a flash block during a program interference based exploit. Reproduced from [9].

3.2. Sketch of Read Disturb Based Exploit

In this exploit, a malicious application can induce a significant amount of *read disturb* onto *several* flash pages that belong to other, benign victim applications. Recall from Section 2.2 that the error rate after read disturb for an LSB page in a partially-programmed wordline is an order of magnitude greater than the error rate for an LSB page in a fully-programmed wordline. The goal of this exploit is for a malicious application to quickly perform a large number of read operations in a very short amount of time, to induce read disturb errors that corrupt both pages already written to partially-programmed wordlines and pages that *have yet to be written*. The malicious application writes an 8KB file, with arbitrary data, to the SSD. Immediately after the file is written, the malicious application repeatedly forces the file system to send a new read request to the SSD. Each request induces read disturb on the other wordlines within the flash block, causing the cell threshold voltages of these wordlines to increase. After the malicious application finishes performing the repeated read requests, a victim application writes data to a file. As the SSD is unaware that an attack took place, it does not detect that the data cannot be written correctly due to the increased cell threshold voltages. As a result, bit flips can occur in the victim application's data. Unlike the program interference exploit, which attacks a single page, the read disturb exploit can corrupt multiple pages with a single attack, and the corruption can affect pages written at a much later time than the attack if the host write rate is low.

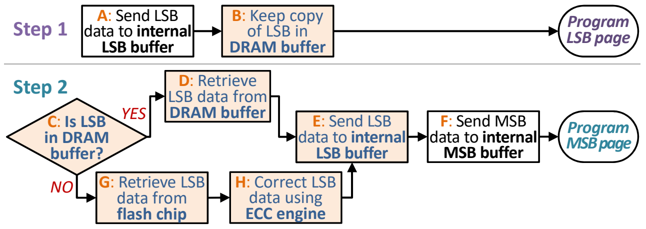
4. Protection and Mitigation Mechanisms

We propose three mechanisms to eliminate or mitigate the program interference and read disturb vulnerabilities of partially-programmed and unprogrammed cells due to two-step programming. Table 1 summarizes the cost and benefits of each mechanism. We briefly discuss our three mechanisms here, and provide more detail on them in our HPCA 2017 paper [9].

Table 1: Summary of our proposed protection mechanisms. Reproduced from [9].

Mechanism	Protects Against	Overhead	Error Rate Reduction
Buffering LSB Data in the Controller	interference read disturb	2MB storage 1.3–15.7% latency	100%
Adaptive LSB Read Reference Voltage	interference read disturb	64B storage 0.0% latency	21–33%
Multiple Pass-Through Voltages	read disturb	0B storage 0.0% latency	72%

Our first mechanism buffers LSB data in the SSD controller, eliminating the need to read the LSB page from flash memory at the beginning of the second programming step, thereby *completely eliminating the vulnerabilities.* It maintains a copy of all partially-programmed LSB data within DRAM buffers that exist in the SSD near the controller. Doing so ensures that the LSB data is read without any errors from the DRAM buffer, where it is free from the vulnerabilities (instead of from the flash memory, where it incurs errors that are not corrected), in the second programming step. Figure 6 shows a flowchart of our modified two-step programming algorithm. This solution increases the programming latency of the flash memory by 4.9% in the common case, due to the long latency of sending the LSB data from the controller to the internal LSB buffer inside flash memory.



Step 1
A: Send LSB data to internal LSB buffer
B: Keep copy of LSB in DRAM buffer
Program LSB page
Step 2
C: Is LSB in DRAM buffer?
YES
NO
D: Retrieve LSB data from DRAM buffer
E: Send LSB data to internal LSB buffer
F: Send MSB data to internal MSB buffer
Program MSB page
G: Retrieve LSB data from flash chip
H: Correct LSB data using ECC engine

Figure 6: Modified two-step programming, using a DRAM buffer for LSB data (modifications shown in shaded boxes). Reproduced from [9].

The two other mechanisms that we develop largely mitigate (but do not *fully* eliminate) the probability of two-step programming errors at much lower latency impact. Our second mechanism adapts the LSB read operation to account for threshold voltage changes induced by program interference and read disturb. It adaptively learns an *optimized* read reference voltage for LSB data, lowering the probability of an LSB read error. Our third mechanism greatly reduces the errors induced during read disturb, by customizing the pass-through voltage for unprogrammed and partially-programmed flash cells. State-of-the-art SSDs apply a single pass-through voltage (V_{pass}) to all of the unread cells, as shown in Figure 7a. This leaves a large gap between the pass-through voltage and the threshold voltage of a partially-programmed or unprogrammed cell, which greatly increases the impact of read disturb [9, 18]. To minimize this gap, and, thus, the impact of read disturb, we propose to use *three* pass-through voltages, as shown in Figure 7b: V_{pass}^{erase} for unprogrammed cells, $V_{pass}^{partial}$ for partially-programmed cells, and the same pass-through voltage as before (V_{pass}) for fully-programmed cells. This mechanism decreases the number of errors induced by read operations to neighboring cells by 72%, which translates to a 16% increase in NAND flash memory lifetime (see Section 6.3 of our HPCA 2017 paper [9] for more detail).

We conclude that, by eliminating or reducing the probability of introducing errors during two-step programming, our solutions completely close or greatly reduce the exposure to reliability and security vulnerabilities.

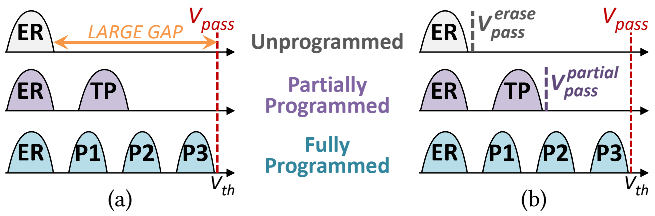


Figure 7: (a) Applying single V_{pass} to all unread wordlines; (b) Our multiple pass-through voltage mechanism, where different voltages are applied based on the the wordline's programming status, to minimize the effects of read disturb. Reproduced from [9].

5. Related Work

To our knowledge, our HPCA 2017 paper [9] is the first to (1) experimentally characterize both program interference and read disturb errors that occur due to the two-step programming method commonly used in MLC NAND flash memory; (2) reveal new reliability and security vulnerabilities exposed by two-step programming in flash memory; and (3) develop novel solutions to reduce these vulnerabilities. We briefly describe related works in the areas of DRAM and NAND flash memory. We note that a thorough survey of error mechanisms in NAND flash memory is provided in our recent works [6, 7, 8].

5.1. Read Disturb Errors in DRAM

Commodity DRAM chips that are sold and used in the field today exhibit read disturb errors [55], also called *RowHammer*-induced errors [82], which are *conceptually* similar to the read disturb errors found in NAND flash memory (see Section 2.2). Repeatedly accessing the same row in DRAM can cause bit flips in data stored in adjacent DRAM rows. In order to access data within DRAM, the row of cells corresponding to the requested address must be *activated* (i.e., opened for read and write operations). This row must be *precharged* (i.e., closed) when another row in the same DRAM bank needs to be activated. Through experimental studies on a large number of real DRAM chips, we show that when a DRAM row is activated and precharged repeatedly (i.e., *hammered*) enough times within a DRAM refresh interval, one or more bits in physically-adjacent DRAM rows can be flipped to the wrong value [55].

In our original RowHammer paper [55], we tested 129 DRAM modules manufactured by three major manufacturers (A, B, and C) between 2008 and 2014, using an FPGA-based experimental DRAM testing infrastructure [38] (more detail on our experimental setup, along with a list of all modules and their characteristics, can be found in our original RowHammer paper [55]). Figure 8 shows the rate of RowHammer errors that we found, with the 129 modules that we tested categorized based on their manufacturing date. We find that 110 of our tested modules exhibit RowHammer errors, with the earliest such module dating back to 2010. In particular, we find that *all* of the modules manufactured in 2012–2013 that we tested are vulnerable to RowHammer. Like with many NAND flash memory error mechanisms, especially read disturb, RowHammer is a recent phenomenon that especially affects DRAM chips manufactured with more advanced manufacturing process technology generations [82]. The phenomenon is due to reliability problems caused by DRAM technology scaling [82, 83, 84, 85].

Figure 9 shows the distribution of the number of rows (plotted in log scale on the y-axis) within a DRAM module that flip the number of bits shown along the x-axis, as measured for example DRAM modules from three different DRAM manufacturers [55]. We make two observations from the figure.

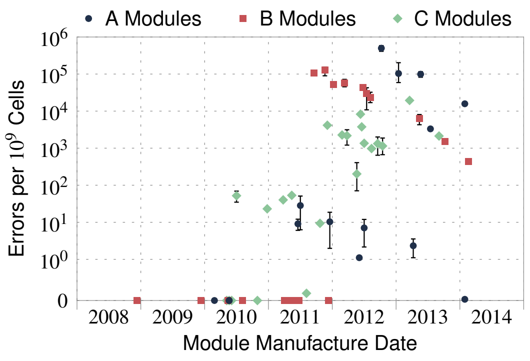


Figure 8: RowHammer error rate vs. manufacturing dates of 129 DRAM modules we tested. Reproduced from [55].

First, the number of bits flipped when we hammer a row (known as the *aggressor row*) can vary significantly within a module. Second, each module has a different distribution of the number of rows. Despite these differences, we find that this DRAM failure mechanism affects more than 80% of the DRAM chips we tested [55]. As indicated above, this read disturb error mechanism in DRAM is popularly called RowHammer [82].

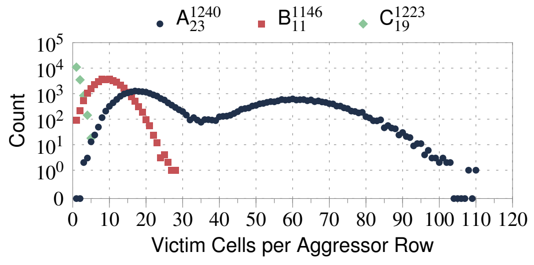


Figure 9: Number of victim cells (i.e., number of bit errors) when an aggressor row is repeatedly activated, for three representative DRAM modules from three major manufacturers. We label the modules in the format X_n^{yyww}, where X is the manufacturer (A, B, or C), *yyww* is the manufacture year (*yy*) and week of the year (*ww*), and n is the number of the selected module. Reproduced from [55].

Various recent works show that RowHammer can be maliciously exploited by user-level software programs to (1) induce errors in existing DRAM modules [55, 82] and (2) launch attacks to compromise the security of various systems [3, 4, 33, 34, 82, 101, 106, 107, 117, 123]. For example, by exploiting the RowHammer read disturb mechanism, a user-level program can gain kernel-level privileges on real laptop systems [106, 107], take over a server vulnerable to RowHammer [34], take over a victim virtual machine running on the same system [3], and take over a mobile device [117]. Thus, the RowHammer read disturb mechanism is a prime (and perhaps the first) example of how a circuit-level failure mechanism in DRAM can cause a practical and widespread system security vulnerability.

Note that various solutions to RowHammer exist [53, 55, 82], but we do not discuss them in detail here. Our recent work [82] provides a comprehensive overview. A very promising proposal is to modify either the memory controller or the DRAM chip such that it probabilistically refreshes the physically-adjacent rows of a recently-activated row, with very low probability. This solution is called *Probabilistic Adjacent Row Activation* (PARA) [55]. Our prior work shows that this low-cost, low-complexity solution, which does not require any storage overhead, greatly closes the RowHammer vulnerability [55].

The RowHammer effect in DRAM worsens as the manufacturing process scales down to smaller node sizes [55, 82]. More findings on RowHammer, along with extensive experimental data from real DRAM devices, can be found in our prior works [53, 55, 82].

5.2. Cell-to-Cell Interference Errors in DRAM

Like NAND flash memory cells, DRAM cells are susceptible to cell-to-cell interference. In DRAM, one important way in which cell-to-cell interference exhibits itself is the data-dependent retention behavior, where the retention time of a DRAM cell is dependent on the values written to *nearby* DRAM cells [46, 47, 48, 49, 70, 82, 97]. This phenomenon is called *data pattern dependence* (DPD) [70]. Data pattern dependence in DRAM is similar to the data-dependent nature of program interference that exists in NAND flash memory (see Section 2.1). Within DRAM, data pattern dependence occurs as a result of parasitic capacitance coupling (between DRAM cells). Due to this coupling, the amount of charge stored in one cell's capacitor can inadvertently affect the amount of charge stored in an adjacent cell's capacitor [46, 47, 48, 49, 70, 82, 97]. As DRAM cells become smaller with technology scaling, cell-to-cell interference worsens because parasitic capacitance coupling between cells increases [46, 70]. More findings on cell-to-cell interference and the data-dependent nature of cell retention times in DRAM, along with experimental data obtained from modern DRAM chips, can be found in our prior works [46, 47, 48, 49, 70, 82, 97].

5.3. Errors in Emerging Memory Technologies

Emerging nonvolatile memories, such as *phase-change memory* (PCM) [60, 61, 62, 100, 122, 125, 129], *spin-transfer torque magnetic RAM* (STT-RAM or STT-MRAM) [57, 86], *metal-oxide resistive RAM* (RRAM) [121], and *memristors* [26, 113], are expected to bridge the gap between DRAM and NAND-flash-memory-based SSDs, providing DRAM-like access latency and energy, and at the same time SSD-like large capacity and nonvolatility (and hence SSD-like data persistence). While their underlying designs are different from DRAM and NAND flash memory, these emerging memory technologies have been shown to exhibit similar types of errors. PCM-based devices are expected to have a limited lifetime, as PCM can only sustain a limited number of writes [60, 100, 122], similar to the P/E cycling errors in SSDs (though PCM's write endurance is higher than that of SSDs [60]). PCM suffers from (1) *resistance drift* [41, 98, 122], where the resistance used to represent the value becomes higher over time (and eventually can introduce a bit error), similar to how charge leakage in NAND flash memory and DRAM lead to retention errors over time; and (2) *write disturb* [43], where the heat generated during the programming of one PCM cell dissipates into neighboring cells and can change the value that is stored within the neighboring cells, similar in concept to cell-to-cell program interference in NAND flash memory. STT-RAM suffers from (1) *retention failures*, where the value stored for a single bit (as the magnetic orientation of the layer that stores the bit) can flip over time; and (2) *read disturb* (a conceptually different phenomenon from the read disturb in DRAM and flash memory), where reading a bit in STT-RAM can inadvertently induce a write to that *same* bit [86].

Due to the nascent nature of emerging nonvolatile memory technologies and the lack of availability of large-capacity devices built with them, extensive and dependable experimental studies have yet to be conducted on the reliability of real PCM, STT-RAM, RRAM, and memristor chips. However, we believe that error mechanisms conceptually or abstractly similar to those for flash memory and DRAM are likely to be prevalent in emerging technologies as well (as supported by some recent studies [2, 43, 50, 86, 109, 110, 128]), albeit with different underlying mechanisms and error rates.

5.4. Other Related Works

Memory Error Characterization and Understanding. Prior works study various types of NAND flash memory errors derived from circuit-level noise, such as data retention noise [6, 7, 8, 11, 12, 14, 15, 73, 77, 79], read disturb noise [6, 7, 8, 18, 77, 90], cell-to-cell program interference noise [11, 13, 15, 16], and P/E cycling noise [6, 7, 8, 11, 15, 17, 72, 77, 96]. Other prior works examine the aggregate effect of these errors on large sets of SSDs that are deployed in the production data centers of Facebook [75], Google [103], and Microsoft [87]. None of these works characterize how program interference and read disturb significantly increase errors within the unprogrammed or partially-programmed cells of an open block due to the vulnerabilities in two-step programming, nor do they develop mechanisms that exploit or mitigate such errors.

A concurrent work by Papandreou et al. [89] characterizes the impact of read disturb on partially-programmed and unprogrammed cells in state-of-the-art MLC NAND flash memory. The authors come to similar conclusions as we do about the impact of read disturb. However, unlike our work, they do not (1) characterize the impact of cell-to-cell program interference on partially-programmed cells, (2) propose exploits that can take advantage of the vulnerabilities in partially-programmed cells, or (3) propose mechanisms that mitigate or eliminate the vulnerabilities.

Similar to the characterization studies performed for NAND flash memory, DRAM latency, reliability, and variation have been experimentally characterized at both a small scale (e.g., hundreds of chips) [21, 22, 23, 38, 46, 47, 48, 49, 51, 53, 55, 64, 65, 67, 70, 97, 99] and a large scale (e.g., tens of thousands of chips) [40, 76, 104, 111, 112].

Program Interference Error Mitigation Mechanisms. Prior works [13, 16] model the behavior of program interference, and propose mechanisms that estimate the optimal read reference voltage once interference has occurred. These works minimize program interference errors only for *fully-programmed* wordlines, by modeling the change in the threshold voltage distribution as a result of the interference. These models are fitted to the distributions of wordlines after *both* the LSB and MSB pages are programmed, and are unable to determine and mitigate the shift that occurs for wordlines that are *partially programmed*. In contrast, we propose mechanisms that specifically address the program interference resulting from two-step programming, and reduce the number of errors induced on LSB pages in *both* partially-programmed and unprogrammed wordlines.

Read Disturb Error Mitigation Mechanisms. One patent [31] proposes a mechanism that uses counters to monitor the total number of reads to each block. Once a block's counter exceeds a threshold, the mechanism remaps and rewrites all of the valid pages within the block to remove the accumulated read disturb errors [31]. Another patent [105] proposes to monitor the MSB page error rate to ensure that it does not exceed the ECC error correction capability, to avoid data loss. Both of these mechanisms monitor pages *only* from *fully-programmed wordlines*. Unfortunately, as we observed, LSB pages in partially-programmed and unprogrammed wordlines are twice as susceptible to read disturb as pages in fully-programmed wordlines (see Section 2.2). If only the MSB page error rate is monitored, read disturb may be detected too late to correct some of the LSB pages.

Our earlier work [18] dynamically changes the pass-through voltage for each block to reduce the impact of read disturb. As a single voltage is applied to the whole block, this mechanism does *not* help significantly with the LSB pages in partially-programmed and unprogrammed wordlines. In contrast, our read disturb mitigation technique (see Section 4) specifically targets these LSB pages by applying multiple different pass-through voltages in an open block, optimized to the different programmed states of each wordline, to reduce read disturb errors.

Other prior works [35, 36, 52] propose to use *read reclaim* to mitigate read disturb errors. The key idea of read reclaim is to remap the data in a block to a new flash block, if the block has experienced a high number of reads [35, 36, 52]. Read reclaim is similar to the remapping-based refresh mechanism [14, 15, 71, 80, 88] employed by many modern SSDs to mitigate data retention errors [6, 7, 8]. Read reclaim can remap the contents of a wordline only after the wordline is fully programmed, and does *not* mitigate the impact of read disturb on partially-programmed or unprogrammed wordlines.

Using Flash Memory for Security Applications. Some prior works studied how flash memory can be used to enhance the security of applications. One work [119] uses flash memory as a secure channel to hide information, such as a secure key. Other works [118, 124] use flash memory to generate random numbers and digital fingerprints. None of these works study vulnerabilities that exist within the flash memory.

Based on our HPCA 2017 paper [9], recent work [58] demonstrates how an attack can be performed on a real SSD using our program interference based exploit (see Section 3.1). The authors use our exploit to perform a file system level attack on a Linux machine, using the attack to gain root privileges.

Two-Step vs. One-Shot Programming. One-shot programming shifts flash cells directly from the erased state to their final target state in a single step. For smaller transistors with less distance between neighboring flash cells, such as those in sub-40nm planar (i.e., 2D) NAND flash memory, two-step programming has replaced one-shot programming to alleviate the coupling capacitance resulting from cell-to-cell program interference [92]. 3D NAND flash memory currently uses one-shot programming [94, 95, 127], as 3D NAND flash memory chips use larger process technology nodes (i.e., 30–50 nm) [102, 126] and employ charge trap transistors [30, 42, 45, 56, 93, 116, 120] for flash cells, as opposed to the floating-gate transistors used in planar NAND flash memory. However, once the number of 3D-stacked layers reaches its upper limit [59, 69], 3D NAND flash memory is expected to scale to smaller transistors [126], and we expect that the increased program interference will again require partial programming (just as it happened for planar NAND flash memory in the past [54, 92]). More detail on 3D NAND flash memory is provided in a recent survey article [8].

6. Long-Term Impact

As we discuss in Section 5, our HPCA 2017 paper [9] makes several novel contributions on characterizing, exploiting, and mitigating vulnerabilities in the two-step programming algorithm used in state-of-the-art MLC NAND flash memory. We believe that these contributions are likely to have a significant impact on academic research and industry.

6.1. Exposing the Existence of Errors

NAND flash manufacturers use two-step programming widely in their contemporary MLC NAND flash devices. Prior to our HPCA 2017 paper [9] and concurrent work by Papandreou et al. [89], there was no publicly-available knowledge about how two-step programming introduced new error sources that did *not* exist in the prior one-shot programming approach. Using real off-the-shelf contemporary NAND flash memory chips, our HPCA 2017 paper exposes the fact that fundamental limitations of the two-step programming method introduce program errors that reduce the lifetime of SSDs available on the market today.

Through a rigorous characterization, our HPCA 2017 paper [9] analyzes two major sources of these errors, program interference and read disturb, demonstrating how they can corrupt data stored in a partially-programmed flash cell. While prior works have addressed both program interference (e.g. [13, 29, 68, 92]) and read disturb (e.g., [18, 31, 35, 105]) errors in the past, we find that none of these existing solutions are able to protect the vulnerable partially-programmed pages produced during two-step programming. We expect that by exposing these errors and the unique vulnerabilities of partially-programmed cells, our work will (1) provide NAND flash memory manufacturers and the academic community with significant insight into the problem; (2) foster the development of new solutions that can reduce or eliminate this vulnerability; and (3) inspire others to search for other reliability and security vulnerabilities that exist in NAND flash memory.

6.2. Security Implications for Flash Memory

Our HPCA 2017 paper [9] proposes two sketches of new potential security exploits based on errors arising from two-step programming. Malicious applications can be developed to use these (or other similar) exploits to corrupt data belonging to other applications. For example, our paper has already enabled the development and demonstration of a file system based attack by IBM security researchers [58]. In that work, the researchers built upon our program interference based exploits to show how to use the file system to acquire root privileges on a real machine. The work confirms that our exploit sketches are likely viable on a real system, and that the threat of maliciously exploiting vulnerabilities in two-step programming is real (and needs to be addressed).

As was the case for RowHammer attacks in DRAM (see Section 5.1), our findings have already generated significant interest and concern in the broader technology community (e.g., [5, 24, 27, 39]). The reason behind the broader impact of our work is that many existing drives in the field today can be attacked. After IBM researchers demonstrated the ability to perform such attacks on a real system [58], there has been further interest in NAND flash memory attacks (e.g., [1, 78]).

We hope and expect that other researchers will take our cue and begin to investigate how other reliability issues in NAND flash memory can be exploited by applications to perform malicious attacks. We believe that this is a new area of research that will grow in importance as SSDs and flash memory become even more widely used.

6.3. Eliminating Program Error Attacks

Our HPCA 2017 paper [9] proposes three solutions that either eliminate or mitigate vulnerabilities to program interference and read disturb during two-step programming. We intentionally design all three of our solutions to be low overhead and easily implementable in commercial SSDs. One of our three solutions completely eliminates the vulnerabilities, albeit with a small increase in flash programming latency. We expect our work to have a direct impact on the NAND flash memory industry, as manufacturers will likely incorporate solutions such as the ones we propose to mitigate or eliminate these vulnerabilities in their new SSDs. We also expect manufacturers and researchers to explore new mechanisms, inspired by our work and by our solutions, that can eliminate these or other vulnerabilities and exploits due to NAND flash memory reliability errors.

7. Conclusion

Our HPCA 2017 paper [9] shows that the two-step programming mechanism commonly employed in modern MLC NAND flash memory chips opens up new vulnerabilities to errors, based on an experimental characterization of modern 1X-nm MLC NAND flash chips. We show that the root cause of these vulnerabilities is the fact that when a partially-programmed cell is set to an intermediate threshold voltage, it is much more susceptible to both cell-to-cell program interference and read disturb. We demonstrate that (1) these vulnerabilities lead to errors that reduce the overall reliability of flash memory, and (2) attackers can potentially exploit these vulnerabilities to maliciously corrupt data belonging to other programs. Based on our experimental observations and the resulting understanding, we propose three new mechanisms that can remove or mitigate these vulnerabilities, by eliminating or reducing the errors introduced as a result of the two-step programming method. Our experimental evaluation shows that our new mechanisms are effective: they can either eliminate the vulnerabilities with modest/low latency overhead, or drastically reduce the vulnerabilities and reduce errors with negligible latency or storage overhead. We hope that the vulnerabilities we analyzed and exposed in this work, along with the experimental data we provided, open up new avenues for mitigation as well as for exposure of other potential vulnerabilities due to internal flash memory operation.

Acknowledgments

We thank the anonymous reviewers for their feedback on our HPCA 2017 paper [9]. This work is partially supported by the Intel Science and Technology Center, CMU Data Storage Systems Center, and NSF grants 1212962 and 1320531.

References

[1] J. W. Aldershoff, "IBM Researchers: Rowhammer-Like Attack on Flash Memory Can Provide Root Privileges to Attacker," Myce, 2017.

[2] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, "Multilevel-Cell Phase-Change Memory: A Viable Technology," *JETCAS*, 2016.

[3] E. Bosman, K. Razavi, H. Bos, and C. Guiffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *SP*, 2016.

[4] W. Burleson, O. Mutlu, and M. Tiwari, "Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer," in *DAC*, 2016.

[5] G. Burton, "Rowhammer-Style NAND Flash Attack Can Corrupt SSD Data," The Inquirer, 2017.

[6] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, 2017.

[7] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives," arXiv:1706.08642 [cs.AR], 2017.

[8] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," arXiv:1711.11427 [cs.AR], 2017.

[9] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[10] Y. Cai, E. F. Haratsch, M. P. McCartney, and K. Mai, "FPGA-Based Solid-State Drive Prototyping Platform," in *FCCM*, 2011.

[11] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[12] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.

[13] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[14] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Flash Correct and Refresh: Retention Aware Management for Increased Lifetime," in *ICCD*, 2012.

[15] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," *Intel Technology Journal*, 2013.

[16] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor Cell Assisted Error Correction in MLC NAND Flash Memories," in *SIGMETRICS*, 2014.

[17] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.

[18] Y. Cai, Y. Luo, S. Ghose, E. F. Haratsch, K. Mai, and O. Mutlu, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.

[19] R. Cernea *et al.*, "A 34MB/s-Program-Throughput 16Gb MLC NAND with All-Bitline Architecture in 56nm," in *ISSCC*, 2008.

[20] R.-A. Cernea *et al.*, "A 34 MB/s MLC Write Throughput 16 Gb NAND with All Bit Line Architecture on 56 nm Technology," *JSSC*, 2009.

[21] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon Univ., 2017.

[22] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[23] K. K. Chang, A. G. Yaglikci, A. Agrawal, N. Chatterjee, S. Ghose, A. Kashyap, H. Hassan, D. Lee, M. O'Connor, and O. Mutlu, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[24] R. Chirgwin, "Rowhammer RAM Attack Adapted to Hit Flash Storage," The Register, 2017.

[25] K. Choi, "NAND Flash Memory," Samsung Electronics Co., Ltd., 2010.

[26] L. Chua, "Memristor—The Missing Circuit Element," *TCT*, 1971.

[27] C. Cimpanu, "SSD Drives Vulnerable to Attacks That Corrupt User Data," Bleeping Computer, 2017.

[28] J. Cooke, "The Inconvenient Truths of NAND Flash Memory," in *Flash Memory Summit*, 2007.

[29] G. Dong, S. Li, and T. Zhang, "Using Data Postcompensation and Prediction to Tolerate Cell-to-Cell Interference in MLC NAND Flash Memory," *TCAS I*, 2010.

[30] B. Eitan, "Non-Volatile Semiconductor Memory Cell Utilizing Asymmetrical Charge Trapping," U.S. Patent No. 5,768,192, 1998.

[31] H. H. Frost, C. J. Camp, T. J. Fisher, J. A. Fuxa, and L. W. Shelton, "Efficient Reduction of Read Disturb Errors in NAND Flash Memory," U.S. Patent No. 7,818,525, 2010.

[32] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing Flash Memory: Anomalies, Observations, and Applications," in *MICRO*, 2009.

[33] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," arXiv:1710.00551 [cs.CR], 2017.

[34] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.

[35] K. Ha, J. Jeong, and J. Kim, "A Read-Disturb Management Technique for High-Density NAND Flash Memory," in *APSys*, 2013.

[36] K. Ha, J. Jeong, and J. Kim, "An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory," *TCAD*, 2016.

[37] T. Hara, K. Fukunda, K. Kanazawa, and N. Shibata, "A 146 mm^2 8 Gb NAND Flash Memory with 70 nm CMOS Technology," in *ISSCC*, 2005.

[38] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.

[39] J. Hruska, "SSDs Vulnerable to Deliberate, Low-Level Data Corruption Attacks," ExtremeTech, 2017.

[40] A. Hwang, I. Stefanovici, and B. Schroeder, "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," in *ASPLOS*, 2012.

[41] D. Ielmini, A. L. Lacaita, and D. Mantegazza, "Recovery and Drift Dynamics of Resistance and Threshold Voltages in Phase-Change Memories," *TED*, 2007.

[42] J. Jang *et al.*, "Vertical Cell Array Using TCAT (Terabit Cell Array Transistor) Technology for Ultra High Density NAND Flash Memory," in *VLSIT*, 2009.

[43] L. Jiang, Y. Zhang, and J. Yang, "Mitigating Write Disturbance in Super-Dense Phase Change Memories," in *DSN*, 2014.

[44] D. Kahng and S. M. Sze, "A Floating Gate and Its Application to Memory Devices," *Bell System Technical Journal*, 1967.

[45] R. Katsumata *et al.*, "Pipe-Shaped BiCS Flash Memory with 16 Stacked Layers and Multi-Level-Cell Operation for Ultra High Density Storage Devices," in *VLSIT*, 2009.

[46] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[47] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.

[48] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *CAL*, 2016.

[49] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.

[50] W.-S. Khwa *et al.*, "A Resistance-Drift Compensation Scheme to Reduce MLC PCM Raw BER by Over 100x for Storage-Class Memory Applications," in *ISSCC*, 2016.

[51] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.

[52] N. Kim and J.-H. Jang, "Nonvolatile Memory Device, Method of Operating Nonvolatile Memory Device and Memory System Including Nonvolatile Memory Device," U.S. Patent No. 8,203,881, 2012.

[53] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon Univ., 2015.

[54] Y. S. Kim, D. J. Lee, C. K. Lee, H. K. Choi, S. S. Kim, J. H. Song, D. H. Song, J.-H. Choi, K.-D. Suh, and C. Chung, "New Scaling Limitation of the Floating Gate Cell in NAND Flash Memory," in *IRPS*, 2010.

[55] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[56] Y. Komori, M. Kido, M. Kito, R. Katsumata, Y. Fukuzumi, H. Tanaka, Y. Nagata, M. Ishiduki, H. Aochi, and A. Nitayama, "Disturbless Flash Memory Due to High Boost Efficiency on BiCS Structure and Optimal Memory Film Stack for Ultra High Density Storage Device," in *IEDM*, 2008.

[57] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.

[58] A. Kurmus, N. Ioannou, M. Neigschwandter, N. Papandreou, and T. Parnell, "From Random Block Corruption to Privilege Escalation: A Filesystem Attack Vector for Rowhammer-Like Attacks," in *WOOT*, 2017.

[59] M. LaPedus, "How to Make 3D NAND," Semiconductor Engineering, 2016.

[60] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.

[61] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, 2010.

[62] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, 2010.

[63] C. Lee *et al.*, "A 32-Gb MLC NAND Flash Memory with Vth Endurance Enhancing Schemes in 32 nm CMOS," *JSSC*, 2011.

[64] D. Lee, "Reducing DRAM Energy at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon Univ., 2016.

[65] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.

[66] D.-H. Lee and W. Sung, "Least Squares Based Cell-to-Cell Interference Cancelation Technique for Multi-Level Cell NAND Flash Memory," in *ICASSP*, 2012.

[67] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.

[68] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of Floating-Gate Interference on NAND Flash Memory Cell Operation," *EDL*, 2002.

[69] S.-Y. Lee, "Limitations of 3D NAND Scaling," EE Times, 2017.

[70] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention

Time Profiling Mechanisms," in *ISCA*, 2013.
[71] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime With Write-Hotness Aware Retention Management," in *MSST*, 2015.
[72] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.
[73] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.
[74] F. Masuoka, M. Momodomi, Y. Iwata, and R. Shirota, "New Ultra High Density EPROM and Flash EEPROM With NAND Structure Cell," in *IEDM*, 1987.
[75] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A Large-Scale Study of Flash Memory Errors in the Field," in *SIGMETRICS*, 2015.
[76] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.
[77] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit Error Rate in NAND Flash Memories," in *IRPS*, 2008.
[78] M. Mimoso, "Rowhammer Attacks Come to MLC NAND Flash Memory," Threatpost, 2017.
[79] K. Mizoguchi, T. Takahashi, S. Aritome, and K. Takeuchi, "Data-Retention Characteristics Comparison of 2D and 3D TLC NAND Flash Memories," in *IMW*, 2017.
[80] V. Mohan, S. Sankar, and S. Gurumurthi, "reFresh SSDs: Enabling High Endurance, Low Cost Flash in Datacenters," Univ. of Virginia, Tech. Rep. CS-2012-05, 2012.
[81] M. Momodomi, F. Masuoka, R. Shirota, Y. Itoh, K. Ohuchi, and R. Kirisawa, "Electrically Erasable Programmable Read-Only Memory With NAND Cell Structure," U.S. Patent No. 4,959,812, 1988.
[82] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.
[83] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[84] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *MEMCON*, 2013.
[85] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[86] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STT-RAM Scaling and Retention Failure," *Intel Technology Journal*, 2013.
[87] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD Failures in Datacenters: What? When? and Why?" in *SYSTOR*, 2016.
[88] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-Nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System Performance for Enterprise Applications," in *HPCA*, 2012.
[89] N. Papandreou, T. Parnell, T. Mittelholzer, H. Pozidis, T. Griffin, G. Tressler, T. Fisher, and C. Camp, "Effect of Read Disturb on Incomplete Blocks in MLC NAND Flash Arrays," in *IMW*, 2016.
[90] N. Papandreou, T. Parnell, H. Pozidis, T. Mittelholzer, E. Eleftheriou, C. Camp, T. Griffin, G. Tressler, and A. Walls, "Using Adaptive Read Voltage Thresholds to Enhance the Reliability of MLC NAND Flash Memory Systems," in *GLSVLSI*, 2014.
[91] J. Park, J. Jeong, S. Lee, Y. Song, and J. Kim, "Improving Performance and Lifetime of NAND Storage Systems Using Relaxed Program Sequence," in *DAC*, 2016.
[92] K.-T. Park, M. Kang, D. Kim, S.-W. Hwang, B. Y. Choi, Y.-T. Lee, C. Kim, and K. Kim, "A Zeroing Cell-to-Cell Interference Page Architecture with Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories," *JSSC*, 2008.
[93] K. Park *et al.*, "Three-Dimensional 128 Gb MLC Vertical NAND Flash Memory With 24-WL Stacked Layers and 50 MB/s High-Speed Programming," *J. Solid-State Circuits*, Jan. 2015.
[94] T. Parnell, "NAND Flash Basics & Error Characteristics: Why Do We Need Smart Controllers?" in *Flash Memory Summit*, 2016.
[95] T. Parnell and R. Pletka, "NAND Flash Basics & Error Characteristics," in *Flash Memory Summit*, 2017.
[96] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis, "Modelling of the Threshold Voltage Distributions of Sub-20nm NAND Flash Memory," in *GLOBECOM*, 2014.
[97] M. Patel, J. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[98] A. Pirovano, A. L. Lacaita, F. Pellizzer, S. A. Kostylev, A. Benvenuti, and R. Bez, "Low-Field Amorphous State Resistance and Threshold Voltage Drift in Chalcogenide Materials," *TED*, 2004.
[99] M. Qureshi, D. H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[100] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *ISCA*, 2009.
[101] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Guiffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.
[102] Samsung Electronics Co., Ltd., "Samsung V-NAND Technology," http://www.samsung.com/us/business/oem-solutions/pdfs/V-NAND_technology_WP.pdf. 2014.
[103] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash Reliability in Production: The Expected and the Unexpected," in *FAST*, 2016.
[104] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009.
[105] A. Schushan, "Refreshing of Memory Blocks Using Adaptive Read Disturb Threshold," U.S. Patent Appl. No. 20140173239, 2014.
[106] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," Google Project Zero Blog, 2015.
[107] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," in *BlackHat*, 2015.
[108] H. Shim, S.-S. Lee, and B. Kim, "Highly Reliable 26nm 64Gb MLC E2NAND (Embedded-ECC & Enhanced-Efficiency) Flash Memory with MSP (Memory Signal Processing) Controller," in *VLSIT*, 2011.
[109] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy, "Challenges for High-Density 16Gb ReRAM with 27nm Technology," in *VLSIC*, 2015.
[110] S. Sills, S. Yasuda, J. Strand, A. Calderoni, K. Aratani, A. Johnson, and N. Ramaswamy, "A Copper ReRAM Cell for Storage Class Memory Applications," in *VLSIT*, 2014.
[111] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," in *SC*, 2013.
[112] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory Errors in Modern Systems: The Good, The Bad, and the Ugly," in *ASPLOS*, 2015.
[113] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The Missing Memristor Found," *Nature*, 2008.
[114] K.-D. Suh, B.-H. Suh, Y.-H. Lim, and J.-K. Kim, "A 3.3V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme," *JSSC*, 1995.
[115] K. Takeuchi, S. Satoh, T. Tanaka, K. Imamiya, and K. Sakui, "A Negative Vth Cell Architecture for Highly Scalable, Excellently Noise-Immune, and Highly Reliable NAND Flash Memories," *JSSC*, 1999.
[116] H. Tanaka *et al.*, "Bit Cost Scalable Technology with Punch and Plug Process for Ultra High Density Flash Memory," in *VLSIT*, 2007.
[117] V. van der Veen, Y. Fratanonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Guiffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.
[118] Y. Wang, W.-K. Yu, S. Wu, G. Malysa, and G. E. Suh, "Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints," in *SP*, 2012.
[119] Y. Wang, W.-K. Yu, S. Q. Xu, E. Kan, and G. E. Suh, "Hiding Information in Flash Memory," in *SP*, 2013.
[120] H. A. R. Wegener, A. J. Lincoln, H. C. Pao, M. R. O'Connell, R. E. Oleksiak, and H. Lawrence, "The Variable Threshold Transistor, A New Electrically-Alterable, Non-Destructive Read-Only Storage Device," in *IEDM*, 1967.
[121] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.
[122] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proc. IEEE*, 2010.
[123] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.
[124] S. Q. Xu, W.-K. Yu, G. E. Suh, and E. Kan, "Understanding Sources of Variations in Flash Memory for Physical Unclonable Functions," in *IMW*, 2014.
[125] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," *TACO*, 2014.
[126] J. H. Yoon, "3D NAND Technology – Implications to Enterprise Storage Applications," in *Flash Memory Summit*, 2015.
[127] J. H. Yoon, R. Godse, G. Tressler, and H. Hunter, "3D-NAND Scaling and 3D-SCM — Implications to Enterprise Storage," in *Flash Memory Summit*, 2017.
[128] Z. Zhang, W. Xiao, N. Park, and D. J. Lilja, "Memory Module-Level Testing and Error Behaviors for Phase Change Memory," in *ICCD*, 2012.
[129] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.
[130] L. Zuolo, C. Zambelli, R. Micheloni, and M. Indaco, "SSDExplorer: A Virtual Platform for Performance/Reliability-Oriented Fine-Grained Design Space Exploration of Solid State Drives," *TCAD*, 2015.

Welcome to IPSI BgD Conferences and Journals!

http://tir.ipsitransactions.org

http://www.ipsitransactions.org

CIP – Katalogizacija u publikaciji
Narodna biblioteka Srbije, Beograd
ISSN 1820 – 4503
The IPSI BGD Transactions
on Internet Research
COBISS.SR - ID 119127052

ISSN 1820-4503
9 771820 450009