Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques

Jeremie S. Kim[§†] Minesh Patel[§] A. Giray Yağlıkçı[§]
Hasan Hassan[§] Roknoddin Azizi[§] Lois Orosa[§] Onur Mutlu[§†]

[§]*ETH Zürich* [†]*Carnegie Mellon University*

RowHammer is a circuit-level DRAM vulnerability, first rigorously analyzed and introduced in 2014, where repeatedly accessing data in a DRAM row can cause bit flips in nearby rows. The RowHammer vulnerability has since garnered significant interest in both computer architecture and computer security research communities because it stems from physical circuit-level interference effects that worsen with continued DRAM density scaling. As DRAM manufacturers primarily depend on density scaling to increase DRAM capacity, future DRAM chips will likely be more vulnerable to RowHammer than those of the past. Many RowHammer mitigation mechanisms have been proposed by both industry and academia, but it is unclear whether these mechanisms will remain viable solutions for future devices, as their overheads increase with DRAM's vulnerability to RowHammer.

In order to shed more light on how RowHammer affects modern and future devices at the circuit-level, we first present an experimental characterization of RowHammer on 1580 DRAM chips (408× DDR3, 652× DDR4, and 520× LPDDR4) from 300 DRAM modules (60× DDR3, 110× DDR4, and 130× LPDDR4) with RowHammer protection mechanisms disabled, spanning multiple different technology nodes from across each of the three major DRAM manufacturers. Our studies definitively show that newer DRAM chips are more vulnerable to RowHammer: as device feature size reduces, the number of activations needed to induce a RowHammer bit flip also reduces, to as few as 9.6k (4.8k to two rows each) in the most vulnerable chip we tested.

We evaluate five state-of-the-art RowHammer mitigation mechanisms using cycle-accurate simulation in the context of real data taken from our chips to study how the mitigation mechanisms scale with chip vulnerability. We find that existing mechanisms either are not scalable or suffer from prohibitively large performance overheads in projected future devices given our observed trends of RowHammer vulnerability. Thus, it is critical to research more effective solutions to RowHammer.

1. Introduction

DRAM is the dominant main memory technology of nearly all modern computing systems due to its superior cost-per-capacity. As such, DRAM critically affects overall system performance and reliability. Continuing to increase DRAM capacity requires increasing the density of DRAM cells by reducing (i.e., scaling) the technology node size (e.g., feature size) of DRAM, but this scaling negatively impacts DRAM reliability. In particular, RowHammer [63] is an important circuit-level interference phenomenon, closely related to technology scaling, where repeatedly activating a DRAM row disturbs the values in adjacent rows. RowHammer can result in system-visible bit flips in DRAM regions that are *physically nearby* rapidly accessed (i.e., hammered) DRAM rows. RowHammer empowers an attacker who has access to DRAM address X with the ability to modify data in a different location Y such that X and Y are *physically*, but not necessarily *logically*, co-located. In particular, X and Y must be located in different DRAM rows that are in close proximity to one another. Because DRAM is widely used throughout modern computing systems, many systems are potentially vulnerable to RowHammer attacks, as shown by recent works (e.g., [21, 26, 34, 35, 50, 70, 82, 90, 100, 102, 108, 119, 122, 123, 130]).

RowHammer is a serious challenge for system designers because it exploits fundamental DRAM circuit behavior that cannot be easily changed. This means that RowHammer is a potential threat across all DRAM generations and designs. Kim et al. [63] show that RowHammer appears to be an effect of continued DRAM technology scaling [63, 89, 91, 92], which means that as manufacturers increase DRAM storage density, their chips are potentially more susceptible to RowHammer. This increase in RowHammer vulnerability is often quantified for a given DRAM chip by measuring the number of times a single row must be activated (i.e., single-sided RowHammer) to induce the first bit flip. Recently, Yang et al. [133] have corroborated this hypothesis, identifying a precise circuit-level charge leakage mechanism that may be responsible for RowHammer. This leakage mechanism affects nearby circuit components, which implies that as manufacturers continue to employ aggressive technology scaling for generational storage density improvements [42, 53, 86, 126], circuit components that are more tightly packed will likely increase a chip's vulnerability to RowHammer.

To mitigate the impact of the RowHammer problem, numerous works propose *mitigation mechanisms* that seek to prevent RowHammer bit flips from affecting the system. These include mechanisms to make RowHammer conditions impossible or very difficult to attain (e.g., increasing the default DRAM refresh rate by more than 7x [63], or probabilistically activating adjacent rows with a carefully selected probability [63]) and mechanisms that explicitly detect RowHammer conditions and intervene (e.g., access counter-based approaches [57, 63, 77, 113, 114]). However, *all* of these solutions [2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 15, 24, 25, 27, 30, 33, 38, 41, 44, 57, 62, 63, 67, 77, 78, 79, 116, 123, 127, 128, 129, 134] merely treat the symptoms of a RowHammer attack (i.e., prevent RowHammer conditions) without solving the core circuit vulnerability.

To better understand the problem in order to pursue more comprehensive solutions, prior works study the RowHammer failure mechanism both experimentally [63, 96, 97] and in simulation [133]. Unfortunately, there has been no work since the original RowHammer paper [63] that provides a rigorous characterization-based study to demonstrate how chips' vulnerabilities to RowHammer (i.e., the minimum number of activations required to induce the first RowHammer bit flip) scale across different DRAM technology generations. While many works [5, 64, 90, 91, 92] speculate that modern chips are more vulnerable, there is no rigorous experimental study that demonstrates exactly how the minimum activation count to induce the first RowHammer bit flip and other RowHammer characteristics behave in modern DRAM chips. Such an experimental study would enable us to predict future chips' vulnerability to RowHammer and estimate whether existing RowHammer mitigation mechanisms can effectively prevent RowHammer bit flips in modern and future chips.

Our goal in this work is to experimentally demonstrate how vulnerable modern DRAM chips are to RowHammer at the circuit-level and to study how this vulnerability will scale going forward. To this end, we provide a rigorous experimental characterization of 1580 DRAM chips (408× DDR3, 652× DDR4, and 520× LPDDR4) from 300 modern DRAM modules (60× DDR3, 110× DDR4, and 130× LPDDR4) from across all

three major DRAM manufacturers, spanning across multiple different technology node generations for each manufacturer. To study the RowHammer vulnerability at the *circuit* level instead of at the *system* level, we disable all accessible RowHammer mitigation mechanisms.[1] We observe that the worst-case circuit-level RowHammer conditions for a victim row are when we repeatedly access *both* physically-adjacent aggressor rows as rapidly as possible (i.e., *double-sided RowHammer*). To account for double-sided RowHammer in our study, we define *Hammer Count* (*HC*) as the number of times *each* physically-adjacent row is activated and HC_{first} as the minimum *HC* required to cause the first RowHammer bit flip in the DRAM chip. For each DRAM type (i.e., DDR3, DDR4, LPDDR4), we have chips from at least two different technology nodes, and for each of the LPDDR4 chips, we know the exact process technology node: 1x or 1y. This enables us to study and demonstrate the effects of RowHammer on two distinct independent variables: DRAM type and DRAM technology node.

Our experiments study the effects of manipulating two key testing parameters at a fixed ambient temperature on both aggregate and individual DRAM cell failure characteristics: (1) hammer count (*HC*), and (2) data pattern written to DRAM.

Our experimental results definitively show that newer DRAM chips manufactured with smaller technology nodes are increasingly vulnerable to RowHammer bit flips. For example, we find that HC_{first} across all chips of a given DRAM type reduces greatly from older chips to newer chips (e.g., $69.2k$ to $22.4k$ in DDR3, $87k$ to $10k$ in DDR4, and $16.8k$ to $4.8k$ in LPDDR4 chips).[2]

Using the data from our experimental studies, we perform cycle-accurate simulation to evaluate the performance overheads of five state-of-the-art RowHammer mitigation mechanisms [63, 77, 116, 134] and compare them to an ideal refresh-based RowHammer mitigation mechanism that selectively refreshes a row only just before it is about to experience a RowHammer bit flip. We show that, while the state-of-the-art mechanisms are reasonably effective at mitigating RowHammer in today's DRAM chips (e.g., 8% average performance loss in our workloads when PARA [63] is used in a DRAM chip with an HC_{first} value of $4.8k$), they exhibit prohibitively large performance overheads for projected degrees of RowHammer vulnerability (i.e., lower HC_{first} values) in future DRAM chips (e.g., the most-scalable existing RowHammer mitigation mechanism causes 80% performance loss when HC_{first} is 128). This means that the state-of-the-art RowHammer mitigation mechanisms are not scalable in the face of worsening RowHammer vulnerability, and DRAM-based computing systems will either require stronger failure mitigation mechanisms or circuit-level modifications that address the root cause of the RowHammer vulnerability.

Our simulation results of an ideal refresh-based mitigation mechanism, which selectively refreshes only those rows that are about to experience a RowHammer bit flip, demonstrates significant opportunity for developing a refresh-based RowHammer mitigation mechanism with low performance overhead that scales reasonably to low HC_{first} values. However, we observe that even this ideal mechanism significantly impacts overall system performance at very low HC_{first} values, indicating the potential need for a better approach to solving RowHammer in the future. We discuss directions for future research in this area in Section 6.3.1.

We make the following contributions in this work:

- We provide the first rigorous RowHammer failure characterization study of a broad range of real modern DRAM chips across different DRAM types, technology node generations, and manufacturers. We experimentally study 1580 DRAM chips (408× DDR3, 652× DDR4, and 520× LPDDR4) from 300 DRAM modules (60× DDR3, 110× DDR4, and 130× LPDDR4) and present our RowHammer characterization results for both aggregate RowHammer failure rates and the behavior of individual cells while sweeping the hammer count (*HC*) and stored data pattern.
- Via our rigorous characterization studies, we definitively demonstrate that the RowHammer vulnerability significantly worsens (i.e., the number of hammers required to induce a RowHammer bit flip, HC_{first}, greatly reduces) in newer DRAM chips (e.g., HC_{first} reduces from $69.2k$ to $22.4k$ in DDR3, $87k$ to $10k$ in DDR4, and $16.8k$ to $4.8k$ in LPDDR4 chips across multiple technology node generations).
- We demonstrate, based on our rigorous evaluation of five state-of-the-art RowHammer mitigation mechanisms, that even though existing RowHammer mitigation mechanisms are reasonably effective at mitigating RowHammer in today's DRAM chips (e.g., 8% average performance loss on our workloads when HC_{first} is 4.8k), they will cause significant overhead in future DRAM chips with even lower HC_{first} values (e.g., 80% average performance loss with the most scalable mechanism when HC_{first} is 128).
- We evaluate an ideal refresh-based mitigation mechanism that selectively refreshes a row only just before it is about to experience a RowHammer bit flip, and find that in chips with high vulnerability to RowHammer, there is still significant opportunity for developing a refresh-based RowHammer mitigation mechanism with low performance overhead that scales to low HC_{first} values. We conclude that it is critical to research more effective solutions to RowHammer, and we provide promising directions for future research.

2. DRAM Background

In this section, we describe the necessary background on DRAM organization and operation to explain the RowHammer vulnerability and its implications for real systems. For further detail, we refer the reader to prior studies on DRAM [16, 17, 18, 19, 22, 28, 29, 39, 40, 55, 58, 63, 65, 66, 71, 72, 73, 74, 75, 76, 109, 110, 111, 112, 135].

2.1. DRAM Organization

A typical computing system includes multiple DRAM *channels*, where each channel has a separate I/O bus and operates independently of the other channels in the system. As Figure 1 (left) illustrates, a memory controller can interface with multiple DRAM *ranks* by time-multiplexing the channel's I/O bus between the ranks. Because the I/O bus is shared, the memory controller serializes accesses to different ranks in the same channel. A DRAM rank comprises multiple DRAM *chips* that operate in lockstep. The combined data pins from all chips form the DRAM data bus.

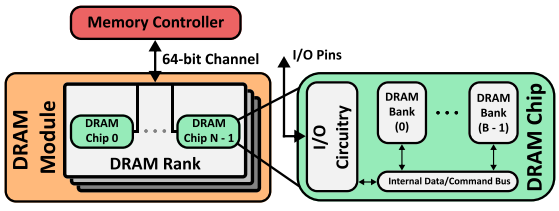

Figure 1: A typical DRAM-based system.

DRAM Chip Organization. A modern DRAM chip contains billions of cells, each of which stores a single bit of data.

[1] We cannot disable on-die ECC in our LPDDR4 chips [53, 68, 69, 88, 98].

[2] While we do not definitively know the exact technology nodes used in our DDR3/DDR4 chips, we group our chips into two sets (i.e., new and old) based on their manufacturing dates, datasheet publication dates, purchase dates, and distinctive RowHammer characterization results. We compare our results against those from our LPDDR4 chips whose exact technology nodes we know and observe the same trend of higher RowHammer vulnerability with newer chips (that likely use smaller DRAM process technology nodes).

Within a chip, cells are organized hierarchically to provide high density and performance. As Figure 1 (right) shows, a DRAM chip is composed of multiple (e.g., 8–16 [45, 46]) DRAM *banks*. All banks within a chip share the internal data and command bus.

Figure 2 (left) shows the internal organization of a DRAM bank. A bank comprises many (e.g., 128) subarrays [18, 65]. Each subarray contains a two-dimensional array of DRAM cells arranged in *rows* and *columns*. When accessing DRAM, the memory controller first provides the address of the row to be accessed. Then, the row decoder, which is also hierarchically organized into global and local components, opens the row by driving the corresponding *wordline*. DRAM cells that are connected to the same wordline are collectively referred to as a DRAM *row*. To read and manipulate a cell's contents, a wire (i.e., *bitline*) connects a column of cells to a *sense amplifier*. The collection of sense amplifiers in a subarray is referred to as the *local row buffer*. The local row buffers of the subarrays in a bank are connected to a per-bank *global row buffer*, which interfaces with the internal command and data bus of the DRAM chip.

As Figure 2 (right) shows, a DRAM cell consists of an *access transistor* and a *capacitor*. The wordline is connected to the gate of the access transistor that, when enabled, connects the cell capacitor to the bitline. A DRAM cell stores a single bit of data based on the charge level of the cell capacitor (e.g., a charged capacitor represents a logical value of "1" and a discharged capacitor a value of "0", or vice versa). Unfortunately, charge leaks from the storage capacitor over time due to various charge leakage paths in the circuit components. To ensure that the cell does not leak enough charge to cause a bit flip, a DRAM cell needs to be periodically refreshed [83, 84].

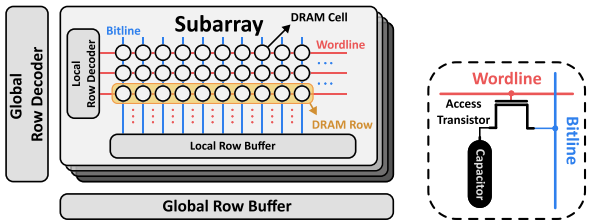

Figure 2: DRAM bank and cell.

2.2. DRAM Operation

A memory controller issues a sequence of DRAM commands to access data in a DRAM chip. First, the memory controller issues an *activate (ACT)* command to *open* a row that corresponds to the memory address to be accessed. Opening (i.e., activating) a DRAM row causes the data in the target DRAM row to be copied to its corresponding local row buffer. Second, the memory controller issues either a *READ* or a *WRITE* command to the DRAM to read out or update the target data within the row buffer, typically 32 or 64 bytes split across all chips in the rank. The memory controller can issue multiple consecutive *READ* and *WRITE* commands to an open row. While a row is open, its cells remain connected to the sense amplifiers in the local row buffer, so changes to the data stored in the row buffer propagate to the DRAM cells. When accesses to the open row are complete, the memory controller issues a *precharge (PRE)* command to *close* the open row and prepare the bank to activate a different row.

DRAM Refresh. DRAM cell capacitors lose their charge over time [83,84,107], potentially resulting in bit flips. A cell's *retention time* refers to the duration for which its capacitor maintains the correct value. Cells throughout a DRAM chip have different retention times, ranging from milliseconds to hours [36, 37, 40, 53, 54, 56, 61, 71, 80, 83, 84, 99, 101, 124]. To prevent data loss, the memory controller issues regular *refresh (REF)* commands that ensure every DRAM cell is refreshed at fixed intervals (typically every 32 or 64 ms according to DRAM specifications [46, 47, 48]).

2.3. RowHammer: DRAM Disturbance Errors

Modern DRAM devices suffer from *disturbance errors* that occur when a high rate of accesses to a single DRAM row unintentionally flip the values of cells in nearby rows. This phenomenon is known as *RowHammer* [63]. It inherently stems from electromagnetic interference between nearby cells. RowHammer is exacerbated by reduction in process technology node size because adjacent DRAM cells become both smaller and closer to each other. Therefore, as DRAM manufacturers continue to increase DRAM storage density, a chip's vulnerability to RowHammer bit flips increases [63, 90, 92].

RowHammer exposes a system-level security vulnerability that has been studied by many prior works both from the attack and defense perspectives. Prior works demonstrate that RowHammer can be used to mount system-level attacks for privilege escalation (e.g., [21, 26, 34, 35, 50, 82, 100, 102, 108, 119, 122, 130]), leaking confidential data (e.g., [70]), and denial of service (e.g., [34, 82]). These works effectively demonstrate that a system must provide protection against RowHammer to ensure robust (i.e., reliable and secure) execution.

Prior works propose defenses against RowHammer attacks both at the hardware (e.g., [4, 5, 6, 7, 8, 24, 27, 30, 33, 38, 52, 57, 63, 77, 104, 113, 116, 134]) and software (e.g., [2, 3, 10, 11, 12, 15, 25, 41, 44, 62, 63, 67, 78, 79, 123, 127, 128, 129]) levels. DRAM manufacturers themselves employ in-DRAM RowHammer prevention mechanisms such as *Target Row Refresh (TRR)* [46], which internally performs proprietary operations to reduce the vulnerability of a DRAM chip against potential RowHammer attacks, although these solutions have been recently shown to be vulnerable [26]. Memory controller and system manufacturers have also included defenses such as increasing the refresh rate [2, 3, 78] and Hardware RHP [43, 95, 121, 125]. For a detailed survey of the RowHammer problem, its underlying causes, characteristics, exploits building on it, and mitigation techniques, we refer the reader to [92].

3. Motivation and Goal

Despite the considerable research effort expended towards understanding and mitigating RowHammer, scientific literature still lacks rigorous experimental data on how the RowHammer vulnerability is changing with the advancement of DRAM designs and process technologies. In general, important practical concerns are difficult to address with existing data in literature. For example:

- How vulnerable to RowHammer are future DRAM chips expected to be at the circuit level?
- How well would RowHammer mitigation mechanisms prevent or mitigate RowHammer in future devices?
- What types of RowHammer solutions would cope best with increased circuit-level vulnerability due to continued technology node scaling?

While existing experimental characterization studies [63, 96, 97] take important steps towards building an overall understanding of the RowHammer vulnerability, they are too scarce and collectively do not provide a holistic view of RowHammer evolution into the modern day. To help overcome this lack of understanding, we need a unifying study of the RowHammer vulnerability of a broad range of DRAM chips spanning the time since the original RowHammer paper was published in 2014 [63].

To this end, **our goal** in this paper is to evaluate and understand how the RowHammer vulnerability of real DRAM chips at the circuit level changes across different chip types, manufacturers, and process technology node generations. Doing so enables us to predict how the RowHammer vulnerability in DRAM chips will scale as the industry continues to increase

storage density and reduce technology node size for future chip designs. To achieve this goal, we perform a rigorous experimental characterization study of DRAM chips from three different DRAM types (i.e., DDR3, DDR4, and LPDDR4), three major DRAM manufacturers, and at least two different process technology nodes from each DRAM type. We show how different chips from different DRAM types and technology nodes (abbreviated as "type-node" configurations) have varying levels of vulnerability to RowHammer. We compare the chips' vulnerabilities against each other and project how they will likely scale when reducing the technology node size even further (Section 5). Finally, we study how effective existing RowHammer mitigation mechanisms will be, based on our observed and projected experimental data on the RowHammer vulnerability (Section 6).

4. Experimental Methodology

We describe our methodology for characterizing DRAM chips for RowHammer.

4.1. Testing Infrastructure

In order to characterize the effects of RowHammer across a broad range of modern DRAM chips, we experimentally study DDR3, DDR4, and LPDDR4 DRAM chips across a wide range of testing conditions. To achieve this, we use two different testing infrastructures: (1) the SoftMC framework [40, 106] capable of testing DDR3 and DDR4 DRAM modules in a temperature-controlled chamber and (2) an in-house temperature-controlled testing chamber capable of testing LPDDR4 DRAM chips.

SoftMC. Figure 3 shows our SoftMC setup for testing DDR4 chips. In this setup, we use an FPGA board with a Xilinx Virtex UltraScale 95 FPGA [132], two DDR4 SODIMM slots, and a PCIe interface. To open up space around the DDR4 chips for temperature control, we use a vertical DDR4 SODIMM riser board to plug a DDR4 module into the FPGA board. We heat the DDR4 chips to a target temperature using silicone rubber heaters pressed to both sides of the DDR4 module. We control the temperature using a thermocouple, which we place between the rubber heaters and the DDR4 chips, and a temperature controller. To enable fast data transfer between the FPGA and a host machine, we connect the FPGA to the host machine using PCIe via a 30 cm PCIe extender. We use the host machine to program the SoftMC hardware and collect the test results. Our SoftMC setup for testing DDR3 chips is similar but uses a Xilinx ML605 FPGA board [131]. Both infrastructures provide fine-grained control over the types and timings of DRAM commands sent to the chips under test and provide precise temperature control at typical operating conditions.

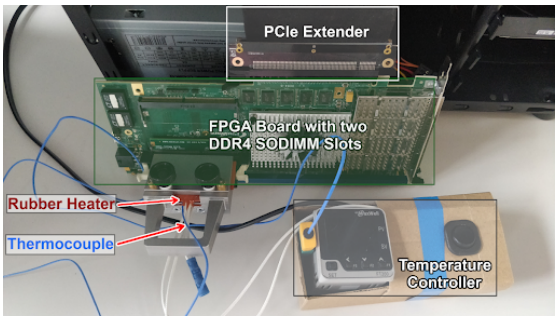

Figure 3: Our SoftMC infrastructure [40, 106] for testing DDR4 DRAM chips.

LPDDR4 Infrastructure. Our LPDDR4 DRAM testing infrastructure uses industry-developed in-house testing hardware for package-on-package LPDDR4 chips. The LPDDR4 testing infrastructure is further equipped with cooling and heating capabilities that also provide us with precise temperature control at typical operating conditions.

4.2. Characterized DRAM Chips

Table 1 summarizes the DRAM chips that we test using both infrastructures. We have chips from all of the three major DRAM manufacturers spanning DDR3, DDR4, and two known technology nodes of LPDDR4. We refer to the DRAM type (e.g., LPDDR4) and technology node of a DRAM chip as a *DRAM type-node configuration* (e.g., LPDDR4-1x). For DRAM chips whose technology node we do not exactly know, we identify their node as *old* or *new*.

Table 1: Summary of DRAM chips tested.

DRAM type-node	Number of Chips (Modules) Tested			
	Mfr. A	*Mfr. B*	*Mfr. C*	*Total*
DDR3-old	56 (10)	88 (11)	28 (7)	**172 (28)**
DDR3-new	80 (10)	52 (9)	104 (13)	**236 (32)**
DDR4-old	112 (16)	24 (3)	128 (18)	**264 (37)**
DDR4-new	264 (43)	16 (2)	108 (28)	**388 (73)**
LPDDR4-1x	12 (3)	180 (45)	N/A	**192 (48)**
LPDDR4-1y	184 (46)	N/A	144 (36)	**328 (82)**

DDR3 and DDR4. Among our tested DDR3 modules, we identify two distinct batches of chips based on their manufacturing date, datasheet publication date, purchase date, and RowHammer characteristics. We categorize DDR3 devices with a manufacturing date earlier than 2014 as DDR3-old chips, and devices with a manufacturing date including and after 2014 as DDR3-new chips. Using the same set of properties, we identify two distinct batches of devices among the DDR4 devices. We categorize DDR4 devices with a manufacturing date before 2018 or a datasheet publication date of 2015 as DDR4-old chips and devices with a manufacturing date including and after 2018 or a datasheet publication date of 2016 or 2017 as DDR4-new chips. Based on our observations on RowHammer characteristics from these chips, we expect that DDR3-old/DDR4-old chips are manufactured at an older date with an older process technology compared to DDR3-new/DDR4-new chips, respectively. This enables us to directly study the effects of shrinking process technology node sizes in DDR3 and DDR4 DRAM chips.

LPDDR4. For our LPDDR4 chips, we have two known distinct generations manufactured with different technology node sizes, 1x-nm and 1y-nm, where 1y-nm is smaller than 1x-nm. Unfortunately, we are missing data from some generations of DRAM from specific manufacturers (i.e., LPDDR4-1x from manufacturer C and LPDDR4-1y from manufacturer B) since we did not have access to chips of these manufacturer-technology node combinations due to confidentiality issues. Note that while we know the external technology node values for the chips we characterize (e.g., 1x-nm, 1y-nm), these values are *not* standardized across different DRAM manufacturers and the actual values are confidential. This means that a 1x chip from one manufacturer is not necessarily manufactured with the same process technology node as a 1x chip from another manufacturer. However, since we do know relative process node sizes of chips from the *same* manufacturer, we can directly observe how technology node size affects RowHammer on LPDDR4 DRAM chips.

4.3. Effectively Characterizing RowHammer

In order to characterize RowHammer effects on our DRAM chips at the circuit-level, we want to test our chips at the worst-case RowHammer conditions. We identify two conditions that our tests must satisfy to effectively characterize RowHammer at the circuit level: our testing routines must both: 1) run without interference (e.g., without DRAM refresh or RowHammer mitigation mechanisms) and 2) systematically test each DRAM row's vulnerability to RowHammer

by issuing the *worst-case sequence of DRAM accesses* for that particular row.

Disabling Sources of Interference. To directly observe RowHammer effects at the circuit level, we want to minimize the external factors that may limit 1) the effectiveness of our tests or 2) our ability to effectively characterize/observe circuit-level effects of RowHammer on our DRAM chips. First, we want to ensure that we have control over how our RowHammer tests behave without disturbing the desired access pattern in any way. Therefore, during the core loop of each RowHammer test (i.e., when activations are issued at a high rate to induce RowHammer bit flips), we disable all DRAM self-regulation events such as refresh and calibration, using control registers in the memory controller. This guarantees consistent testing without confounding factors due to intermittent events (e.g., to avoid the possibility that a victim row is refreshed during a RowHammer test routine such that we observe fewer RowHammer bit flips). Second, we want to directly observe the circuit-level bit flips such that we can make conclusions about DRAM's vulnerability to RowHammer at the circuit technology level rather than the system level. To this end, to the best of our knowledge, we disable all DRAM-level (e.g., TRR [26, 46, 48]) and system-level RowHammer mitigation mechanisms (e.g., pTRR [1]) along with all forms of rank-level error-correction codes (ECC), which could obscure RowHammer bit flips. Unfortunately, all of our LPDDR4-1x and LPDDR4-1y chips use on-die ECC [53, 68, 69, 88, 98] (i.e., an error correcting mechanism that corrects single-bit failures entirely within the DRAM chip [98]), which we cannot disable. Third, we ensure that the core loop of our RowHammer test runs for less than 32 ms (i.e., the lowest refresh interval specified by manufacturers to prevent DRAM data retention failures across our tested chips [45, 46, 48, 54, 83, 99]) so that we do not conflate retention failures with RowHammer bit flips.

Worst-case RowHammer Access Sequence. We leverage *three* key observations from prior work [3, 20, 34, 63, 130] in order to craft a worst-case RowHammer test pattern. First, a repeatedly accessed row (i.e., *aggressor row*) has the greatest impact on its immediate physically-adjacent rows (i.e., repeatedly accessing physical row N will cause the highest number of RowHammer bit flips in physical rows $N+1$ and $N-1$). Second, a *double-sided hammer* targeting physical victim row N (i.e., repeatedly accessing physical rows $N-1$ and $N+1$) causes the *highest* number of RowHammer bit flips in row N compared to any other access pattern. Third, increasing the rate of DRAM activations (i.e., issuing the same number of activations within shorter time periods) results in an increasing number of RowHammer bit flips. This rate of activations is limited by the DRAM timing parameter t_{RC} (i.e., the time between two successive activations) which depends on the DRAM clock frequency and the DRAM type: DDR3 (52.5ns) [45], DDR4 (50ns) [46], LPDDR4 (60ns) [48]. Using these observations, we test each row's worst-case vulnerability to RowHammer by repeatedly accessing the two directly physically-adjacent rows as fast as possible.

To enable the quick identification of physical rows $N-1$ and $N+1$ for a given row N, we reverse-engineer the *undocumented* and *confidential* logical-to-physical DRAM-internal row address remapping. To do this, we exploit RowHammer's key observation that repeatedly accessing an arbitrary row causes the two directly physically-adjacent rows to contain the *highest* number of RowHammer bit flips [63]. By repeating this analysis across rows throughout the DRAM chip, we can deduce the address mappings for each type of chip that we test. We can then use this mapping information to quickly test RowHammer effects at worst-case conditions. We note that for our LPDDR4-1x chips from Manufacturer B, when we repeatedly access a single row within two consecutive rows such that the first row is an even row (e.g., rows 2 and 3) in the logical row address space as seen by the memory controller, we observe 1) no RowHammer bit flips in either of the two consecutive rows and 2) a near equivalent number of RowHammer bit flips in each of the four immediately adjacent rows: the two previous consecutive rows (e.g., rows 0 and 1) and the two subsequent consecutive rows (e.g., rows 4 and 5). This indicates a row address remapping that is internal to the DRAM chip such that every pair of consecutive rows share the same internal wordline. To account for this DRAM-internal row address remapping, we test each row N in LPDDR4-1x chips from manufacturer B by repeatedly accessing physical rows $N-2$ and $N+2$.

Additional Testing Parameters. To investigate RowHammer characteristics, we explore two testing parameters at a stable ambient temperature of $50°C$:

1. **Hammer count (*HC*).** We test the effects of changing the number of times we access (i.e., activate) a victim row's physically-adjacent rows (i.e., aggressor rows). We count each pair of activations to the two neighboring rows as one *hammer* (e.g., one activation each to rows $N-1$ and $N+1$ counts as one hammer). We sweep the hammer count from 2k to 150k (i.e., 4k to 300k activations) across our chips so that the hammer test runs for less than 32ms.
2. **Data pattern (*DP*).** We test several commonly-used DRAM data patterns where every byte is written with the same data: Solid0 (SO0: 0x00), Solid1 (SO1: 0xFF), Colstripe0 (CO0: 0x55), Colstripe1 (CO1: 0xAA) [54, 83, 99]. In addition, we test data patterns where each byte in every other row, including the row being hammered, is written with the same data, Checkered0 (CH0: 0x55) or Rowstripe0 (RS0: 0x00), and all other rows are written with the inverse data, Checkered1 (CH1: 0xAA) or Rowstripe1 (RS1: 0xFF), respectively.

RowHammer Testing Routine. Algorithm 1 presents the general testing methodology we use to characterize RowHammer on DRAM chips. For different data patterns (*DP*) (line 2) and hammer counts (*HC*) (line 8), the test individually targets each row in DRAM (line 4) as a victim row (line 5). For each victim row, we identify the two physically-adjacent rows (*aggressor_row*1 and *aggressor_row*2) as aggressor rows (lines 6 and 7). Before beginning the core loop of our RowHammer test (Lines 11-13), two things happen: 1) the memory controller disables DRAM refresh (line 9) to ensure no interruptions in the core loop of our test due to refresh operations, and 2) we refresh the victim row (line 10) so that we begin inducing RowHammer bit flips on a fully-charged row, which ensures that bit flips we observe are not due to retention time violations. The core loop of our RowHammer test (Lines 11-13) induces RowHammer bit flips in the victim row by first activating *aggressor_row*1 then *aggressor_row*2,

Algorithm 1: DRAM RowHammer Characterization

```
1  DRAM_RowHammer_Characterization():
2      foreach DP in [Data Patterns]:
3          write DP into all cells in DRAM
4          foreach row in DRAM:
5              set victim_row to row
6              set aggressor_row1 to victim_row − 1
7              set aggressor_row2 to victim_row + 1
8              foreach HC in [HC sweep]:
9                  Disable DRAM refresh
10                 Refresh victim_row
11                 for n = 1 → HC: // core test loop
12                     activate aggressor_row1
13                     activate aggressor_row2
14                 Enable DRAM refresh
15                 Record RowHammer bit flips to storage
16                 Restore bit flips to original values
```

HC times. After the core loop of our RowHammer test, we re-enable DRAM refresh (line 14) to prevent retention failures and record the observed bit flips to secondary storage (line 15) for analysis (presented in Section 5). Finally, we prepare to test the next *HC* value in the sweep by restoring the observed bit flips to their original values (Line 16) depending on the data pattern (*DP*) being tested.

Fairly Comparing Data Across Infrastructures. Our carefully-crafted RowHammer test routine allows us to compare our test results between the two different testing infrastructures. This is because, as we described earlier, we 1) reverse engineer the row address mappings of each DRAM configuration such that we effectively test double-sided RowHammer on every single row, 2) issue activations as fast as possible for each chip, such that the activation rates are similar across infrastructures, and 3) disable all sources of interference in our RowHammer tests.

5. RowHammer Characterization

In this section, we present our comprehensive characterization of RowHammer on the 1580 DRAM chips we test.[3]

5.1. RowHammer Vulnerability

We first examine which of the chips that we test are susceptible to RowHammer. Across all of our chips, we sweep the hammer count (*HC*) between 2K and 150K (i.e., 4k and 300k activates for our double-sided RowHammer test) and observe whether we can induce any RowHammer bit flips at all in each chip. We find that we can induce RowHammer bit flips in all chips except many DDR3 chips. Table 2 shows the fraction of DDR3 chips in which we *can* induce RowHammer bit flips (i.e., *RowHammerable* chips).

Table 2: Fraction of DDR3 DRAM chips vulnerable to RowHammer when $HC < 150k$.

DRAM type-node	RowHammerable chips *Mfr. A*	*Mfr. B*	*Mfr. C*
DDR3-old	24/88	0/88	0/28
DDR3-new	8/72	44/52	96/104

Observation 1. *Newer DRAM chips appear to be more vulnerable to RowHammer based on the increasing fraction of RowHammerable chips from DDR3-old to DDR3-new DRAM chips of manufacturers B and C.*

We find that the fraction of manufacturer A's chips that are RowHammerable decreases from DDR3-old to DDR3-new chips, but we also note that the number of RowHammer bit flips that we observe across each of manufacturer A's chips is very low (< 20 on average across RowHammerable chips) compared to the number of bit flips found in manufacturer B and C's DDR3-new chips ($87k$ on average across RowHammerable chips) when $HC = 150K$. Since DDR3-old chips of all manufacturers and DDR3-new chips of manufacturer A have very few to no bit flips, we refrain from analyzing and presenting their characteristics in many plots in Section 5.

5.2. Data Pattern Dependence

To study data pattern effects on observable RowHammer bit flips, we test our chips using Algorithm 1 with *hammer_count* (*HC*) = $150k$ at $50^{\circ}C$, sweeping the 1) *victim_row* and 2) *data_pattern* (as described in Section 4.3).[4]

We first examine the set of all RowHammer bit flips that we observe when testing with different data patterns for a given *HC*. For each data pattern, we run our RowHammer test routine ten times. We then aggregate all unique RowHammer bit flips per data pattern. We combine all unique RowHammer bit flips found by all data patterns and iterations into a full set of observable bit flips. Using the combined data, we calculate the fraction of the full set of observable bit flips that each data pattern identifies (i.e., the data pattern's *coverage*). Figure 4 plots the coverage (y-axis) per individual data pattern (shared x-axis) for a single representative DRAM chip from each DRAM type-node configuration that we test. Each row of subplots shows the coverages for chips of the same manufacturer (indicated on the right y-axis), and the columns show the coverages for chips of the same DRAM type-node configuration (e.g., DDR3-new).

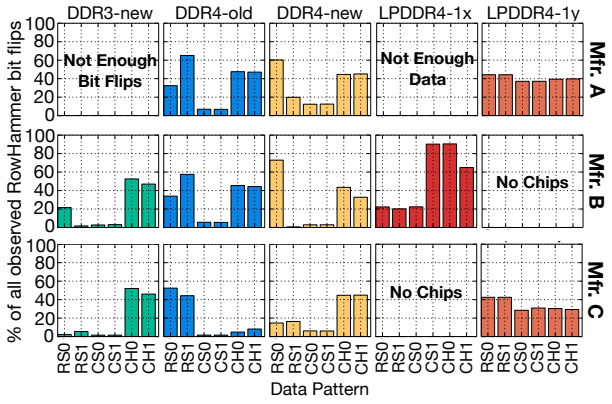

Figure 4: RowHammer bit flip coverage of different data patterns (described in Section 4.3) for a single representative DRAM chip of each type-node configuration.

Observation 2. *Testing with different data patterns is essential for comprehensively identifying RowHammer bit flips because no individual data pattern achieves full coverage alone.*

Observation 3. *The worst-case data pattern (shown in Table 3) is consistent across chips of the same manufacturer and DRAM type-node configuration.*[5]

Table 3: Worst-case data pattern for each DRAM type-node configuration at 50°C split into different manufacturers.

DRAM type-node	Worst Case Data Pattern at 50°C *Mfr. A*	*Mfr. B*	*Mfr. C*
DDR3-new	N/A	Checkered0	Checkered0
DDR4-old	RowStripe1	RowStripe1	RowStripe0
DDR4-new	RowStripe0	RowStripe0	Checkered1
LPDDR4-1x	Checkered1	Checkered0	N/A
LPDDR4-1y	RowStripe1	N/A	RowStripe1

We believe that different data patterns induce the most RowHammer bit flips in different chips because DRAM manufacturers apply a variety of proprietary techniques for DRAM cell layouts to maximize the cell density for different DRAM type-node configurations. For the remainder of this paper, we characterize each chip using *only* its worst-case data pattern.[6]

5.3. Hammer Count (*HC*) Effects

We next study the effects of increasing the hammer count on the number of observed RowHammer bit flips across our

[3] We list our full set of chips in Appendix A of our extended technical report [59] due to space constraints in this paper.

[4] Note that for a given data pattern (*DP*), the same data is always written to *victim_row*. For example, when testing Rowstripe0, every byte in *victim_row* is always written with 0x00 and every byte in the two physically-adjacent rows are written with 0xFF.

[5] We do not consider the true/anti cell pattern of a chip [26, 63, 83] and agnostically program the data pattern accordingly into the DRAM array. More RowHammer bit flips can be induced by considering the true/anti-cell pattern of each chip and devising corresponding data patterns to exploit this knowledge [26].

[6] We use the worst-case data pattern to 1) minimize the extensive testing time, 2) induce many RowHammer bit flips, and 3) experiment at worst-case conditions. A diligent attacker would also try to find the worst-case data pattern to maximize the probability of a successful RowHammer attack.

chips. Figure 5 plots the effects of increasing the number of hammers on the RowHammer bit flip rate[7] for our tested DRAM chips of various DRAM type-node configurations across the three major DRAM manufacturers. For all chips, we hammer each row, sweeping *HC* between 10,000 and 150,000. For each *HC* value, we plot the average rate of observed RowHammer bit flips across all chips of a DRAM type-node configuration.

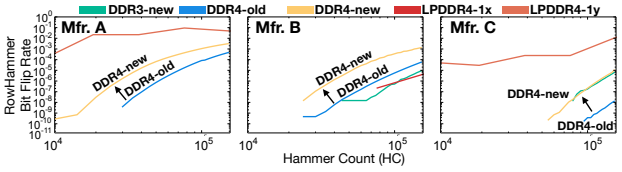

Figure 5: Hammer count (*HC*) vs. RowHammer bit flip rate across DRAM type-node configurations.

Observation 4. *The log of the number of RowHammer bit flips has a linear relationship with the log of HC.*[8]

We observe this relationship between *HC* and RowHammer bit flip rate because more accesses to a single row results in more cell-to-cell interference, and therefore more charge is lost in victim cells of nearby rows.

We examine the effects of DRAM technology node on the RowHammer bit flip rate in Figure 5. We observe that the bit flip rate curve shifts *upward* and *leftward* when going from DDR4-old to DDR4-new chips, indicating respectively, 1) a higher rate of bit flips for the same *HC* value and 2) occurrence of bit flips at lower *HC* values, as technology node size reduces from DDR4-old to DDR4-new.

Observation 5. *Newer DDR4 DRAM technology nodes show a clear trend of increasing RowHammer bit flip rates: the* same *HC value causes an increased average RowHammer bit flip rate from DDR4-old to DDR4-new DRAM chips of all DRAM manufacturers.*

We believe that due to increased density of DRAM chips from older to newer technology node generations, cell-to-cell interference increases and results in DRAM chips that are more vulnerable to RowHammer bit flips.

5.4. RowHammer Spatial Effects

We next experimentally study the spatial distribution of RowHammer bit flips across our tested chips. In order to normalize the RowHammer effects that we observe across our tested chips, we first take each DRAM chip and use a hammer count specific to that chip to result in a RowHammer bit flip rate of 10^{-6}.[9] For each chip, we analyze the spatial distribution of bit flips throughout the chip. Figure 6 plots the fraction of RowHammer bit flips that occur in a given row offset from the *victim_row* out of all observed RowHammer bit flips. Each column of subplots shows the distributions for chips of different manufacturers and each row of subplots shows the distribution for a different DRAM type-node configuration. The error bars show the standard deviation of the distribution across our tested chips. Note that the repeatedly-accessed rows (i.e., *aggressor rows*) are at $x = 1$ and $x = -1$ for all plots except in LPDDR4-1x chips from manufacturer B, where they are at $x = -2$ and $x = 2$ (due to the internal address remapping that occurs in these chips as we describe in Section 4.3). Because an access to a row essentially refreshes the data in the row, repeatedly accessing aggressor rows during the core loop of the RowHammer test prevents any bit flips from happening in the aggressor rows. Therefore, there are no RowHammer bit flips in the aggressor rows across each DRAM chip in our plots (i.e., $y = 0$ for $x = [-2, -1, 2, 3]$ for LPDDR4-1x chips from manufacturer B and for $x = 1$ and $x = -1$ for all other chips).

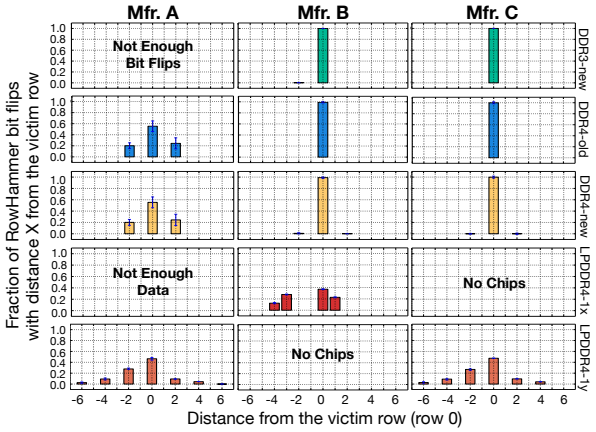

Figure 6: Distribution of RowHammer bit flips across row offsets from the victim row.

We make three observations from Figure 6. First, we observe a general trend across DRAM type-node configurations of a given DRAM manufacturer where newer DRAM technology nodes have an increasing number of rows that are susceptible to RowHammer bit flips that are *farther* from the victim row. For example, in LPDDR4-1y chips, we observe RowHammer bit flips in as far as 6 rows from the victim row (i.e., $x = -6$), whereas in DDR3 and DDR4 chips, RowHammer bit flips only occur in as far as 2 rows from the victim row (i.e., $x = -2$). We believe that this effect could be due to 1) an increase in DRAM cell density, which leads to cell-to-cell interference extending farther than a single row, with RowHammer bit flips occurring in rows increasingly farther away from the aggressor rows (e.g., 5 rows away) for higher-density chips, and 2) more shared structures internal to the DRAM chip, which causes farther (and multiple) rows to be affected by circuit-level interference.

Observation 6. *For a given DRAM manufacturer, chips of newer DRAM technology nodes can exhibit RowHammer bit flips 1) in more rows and 2) farther away from the victim row.*

Second, we observe that rows containing RowHammer bit flips that are farther from the victim row have fewer RowHammer bit flips than rows closer to the victim row. Non-victim rows adjacent to the aggressor rows ($x = 2$ and $x = -2$) contain RowHammer bit flips, and these bit flips demonstrate the effectiveness of a single-sided RowHammer attack as only one of their adjacent rows are repeatedly accessed. As discussed earlier (Section 4.3), the single-sided RowHammer attack is not as effective as the double-sided RowHammer attack, and therefore we find fewer bit flips in these rows. In rows farther away from the victim row, we attribute the diminishing number of RowHammer bit flips to the diminishing effects of cell-to-cell interference with distance.

Observation 7. *The number of RowHammer bit flips that occur in a given row decreases as the distance from the victim row increases.*

Third, we observe that only even-numbered offsets from the victim row contain RowHammer bit flips in all chips except LPDDR4-1x chips from Manufacturer B. However, the rows containing RowHammer bit flips in Manufacturer B's LPDDR4-1x chips would be even-numbered offsets if we translate all rows to physical rows based on our observation in Section 4.3 (i.e., divide each row number by 2 and round down). While we are uncertain why we observe RowHammer bit flips only in physical even-numbered offsets from the

[7] We define the RowHammer bit flip rate as the number of observed RowHammer bit flips to the total number of bits in the tested DRAM rows.

[8] Our observation is consistent with prior work [97].

[9] We choose a RowHammer bit flip rate of 10^{-6} since we are able to observe this bit flip rate in most chips that we characterize with $HC < 150k$.

victim row, we believe that it may be due to the internal circuitry layout of DRAM rows.

We next study the spatial distribution of RowHammer-vulnerable DRAM cells in a DRAM array using the same set of RowHammer bit flips. Figure 7 shows the distribution of 64-bit words containing x RowHammer bit flips across our tested DRAM chips. We find the proportion of 64-bit words containing x RowHammer bit flips out of all 64-bit words in each chip containing any RowHammer bit flip and plot the distribution as a box-and-whisker plot for each x value.

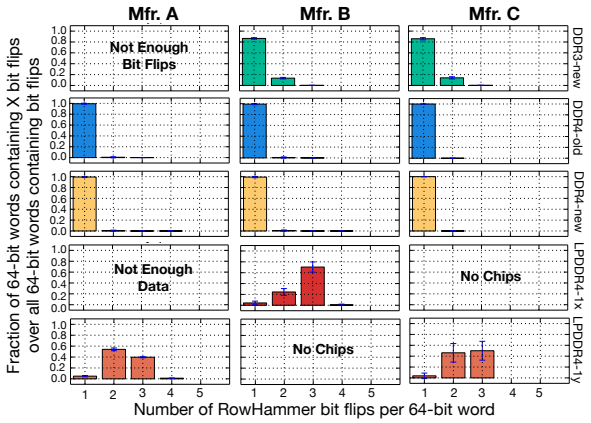

Figure 7: Distribution of the number of RowHammer bit flips per 64-bit word for each DRAM type-node configuration.

Observation 8. *At a RowHammer bit flip rate of* 10^{-6}*, a single 64-bit value can contain up to four RowHammer bit flips.*

Because ECC [53, 88, 94, 98] is typically implemented for DRAM at a 64-bit granularity (e.g., a single-error correcting code would only protect a 64-bit word if it contains at most one error), observation 8 indicates that even at a relatively low bit flip rate of 10^{-6}, a DRAM chip can only be protected from RowHammer bit flips with a strong ECC code (e.g., 4-bit error correcting code), which has high hardware overhead.

Observation 9. *The distribution of RowHammer bit flip density per word changes significantly in LPDDR4 chips compared to other DRAM types.*

We find DDR3 and DDR4 chips across all manufacturers to exhibit an exponential decay curve for increasing RowHammer bit flip densities with most words containing only one RowHammer bit flip. However, LPDDR4 chips across all manufacturers exhibit a much smaller fraction of words containing a single RowHammer bit flip and significantly larger fractions of words containing two and three RowHammer bit flips compared to DDR3 and DDR4 chips. We believe this change in the bit flip density distribution is due to the on-die ECC that manufacturers have included in LPDDR4 chips [53, 88, 94, 98], which is a 128-bit single-error correcting code that corrects and hides *most* single-bit failures within a 128-bit ECC word using redundant bits (i.e., *parity-check bits*) that are hidden from the system.

With the failure rates at which we test, many ECC words contain several bit flips. This exceeds the ECC's correction strength and causes the ECC logic to behave in an undefined way. The ECC logic may 1) correct one of the bit flips, 2) do nothing, or 3) introduce an *additional* bit flip by corrupting an error-free data bit [98, 117]. On-die ECC makes single-bit errors rare because 1) any true single-bit error is immediately corrected and 2) a multi-bit error can *only* be reduced to a single-bit error when there are no more than two bit flips within the data bits *and* the ECC logic's undefined action happens to change the bit flip count to exactly one. In contrast, there are many more scenarios that yield two or three bit-flips within the data bits, and a detailed experimental analysis of how on-die ECC affects DRAM failure rates in LPDDR4 DRAM chips can be found in [98].

5.5. First RowHammer Bit Flips

We next study the vulnerability of each chip to RowHammer. One critical component of vulnerability to the double-sided RowHammer attack [63] is identifying the weakest cell, i.e., the DRAM cell that fails with the fewest number of accesses to physically-adjacent rows. In order to perform this study, we sweep *HC* at a fine granularity and record the *HC* that results in the first RowHammer bit flip in the chip (HC_{first}). Figure 8 plots the distribution of HC_{first} across all tested chips as box-and-whisker plots.[10] The subplots contain the distributions of each tested DRAM type-node configuration for the different DRAM manufacturers. The x-axis organizes the distributions by DRAM type-node configuration in order of age (older on the left to younger on the right). We further subdivide the subplots for chips of the same DRAM type (e.g., DDR3, DDR4, LPDDR4) with vertical lines. Chips of the same DRAM type are colored with the same color for easier visual comparison across DRAM manufacturers.

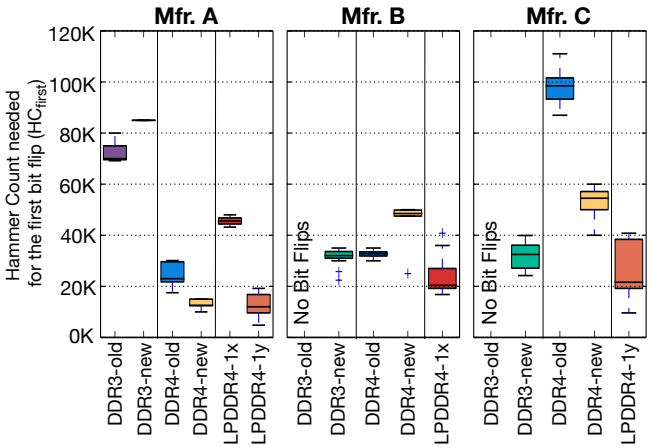

Figure 8: Number of hammers required to cause the first RowHammer bit flip (HC_{first}) per chip across DRAM type-node configurations.

Observation 10. *Newer chips from a given DRAM manufacturer appear to be more vulnerable to RowHammer bit flips. This is demonstrated by the clear reduction in* HC_{first} *values from old to new DRAM generations (e.g., LPDDR4-1x to LPDDR4-1y in manufacturer A, or DDR4-old to DDR4-new in manufacturers A and C).*

We believe this observation is due to DRAM technology process scaling wherein both 1) DRAM cell capacitance reduces and 2) DRAM cell density increases as technology node size reduces. Both factors together lead to more interference between cells and likely faster charge leakage from the DRAM cell's smaller capacitors, leading to a higher vulnerability to RowHammer. We find two exceptions to this trend (i.e., a general increase in HC_{first} from DDR3-old to DDR3-new chips of manufacturer A and from DDR4-old to DDR4-new chips of manufacturer B), but we believe these potential anomalies may be due to our inability to identify explicit manufacturing dates and correctly categorize these particular chips.

[10] A box-and-whiskers plot emphasizes the important metrics of a dataset's distribution. The box is lower-bounded by the first quartile (i.e., the median of the first half of the ordered set of data points) and upper-bounded by the third quartile (i.e., the median of the second half of the ordered set of data points). The median falls within the box. The *inter-quartile range* (IQR) is the distance between the first and third quartiles (i.e., box size). Whiskers extend an additional $1.5 \times IQR$ on either sides of the box. We indicate outliers, or data points outside of the range of the whiskers, with pluses.

Observation 11. *In LPDDR4-1y chips from manufacturer A, there are chips whose weakest cells fail after only 4800 hammers.*

This observation has serious implications for the future as DRAM technology node sizes will continue to reduce and HC_{first} will only get smaller. We discuss these implications further in Section 6. Table 4 shows the lowest observed HC_{first} value for any chip within a DRAM type-node configuration (i.e., the minimum values of each distribution in Figure 8).

Table 4: Lowest HC_{first} values ($\times$1000) across all chips of each DRAM type-node configuration.

DRAM type-node	HC_{first} (Hammers until first bit flip) $\times$1000		
	Mfr. A	*Mfr. B*	*Mfr. C*
DDR3-old	69.2	157	155
DDR3-new	85	22.4	24
DDR4-old	17.5	30	87
DDR4-new	10	25	40
LPDDR4-1x	43.2	16.8	N/A
LPDDR4-1y	4.8	N/A	9.6

Effects of ECC. The use of error correcting codes (ECC) to improve the reliability of a DRAM chip is common practice, with most system-level [9, 21, 31, 60] or on-die [53, 68, 69, 88, 98] ECC mechanisms providing single error correction capabilities at the granularity of 64- or 128-bit words. We examine 64-bit ECCs since, for the same correction capability (e.g., single-error correcting), they are stronger than 128-bit ECCs. In order to determine the efficacy with which ECC can mitigate RowHammer effects on real DRAM chips, we carefully study three metrics across each of our chips: 1) the lowest *HC* required to cause the first RowHammer bit flip (i.e., HC_{first}) for a given chip (shown in Figure 8), 2) the lowest *HC* required to cause at least two RowHammer bit flips (i.e., HC_{second}) within any 64-bit word, and 3) the lowest *HC* required to cause at least three RowHammer bit flips (i.e., HC_{third}) within any 64-bit word. These quantities tell us, for ECCs of varying strengths (e.g., single-error correction code, double-error correction code), at which *HC* values the ECC can 1) mitigate RowHammer bit flips and 2) no longer reliably mitigate RowHammer bit flips for that particular chip.

Figure 9 plots as a bar graph the *HC* (left y-axis) required to find the first 64-bit word containing one, two, and three RowHammer bit flips (x-axis) across each DRAM type-node configuration. The error bars represent the standard deviation of *HC* values across all chips tested. On the same figure, we also plot with red boxplots, the increase in *HC* (right y-axis) between the *HC*s required to find the first 64-bit word containing one and two RowHammer bit flips, and two and three RowHammer bit flips. These multipliers indicate how HC_{first} would change in a chip if the chip uses single-error correcting ECC or moves from a single-error correcting to a double-error correcting ECC. Note that we 1) leave two plots (i.e., Mfr. A DDR3-new and Mfr. C DDR4-old) empty since we are unable to induce enough RowHammer bit flips to find 64-bit words containing more than one bit flip in the chips and 2) do not include data from our LPDDR4 chips because they already include on-die ECC [53, 68, 69, 88, 98], which obfuscates errors potentially exposed to any other ECC mechanisms [98].

Observation 12. *A single-error correcting code can significantly improve HC_{first} by up to 2.78$\times$ in DDR4-old and DDR4-new DRAM chips, and 1.65$\times$ in DDR3-new DRAM chips.*

Observation 13. *Moving from a double-error correcting code to a triple-error correcting code has diminishing returns in DDR4-old and DDR4-new DRAM chips (as indicated by the reduction in the HC multiplier) compared to when moving from a single-error correcting code to a double-error correcting code. However, using a triple-error correcting code in DDR3-new DRAM chips continues to further improve the HC_{first} and thus reduce the DRAM chips' vulnerability to RowHammer.*

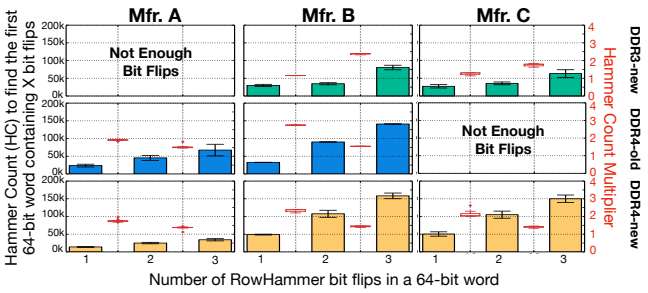

Figure 9: Hammer Count (left y-axis) required to find the first 64-bit word containing one, two, and three RowHammer bit flips. Hammer Count Multiplier (right y-axis) quantifies the *HC* difference between every two points on the x-axis (as a multiplication factor of the left point to the right point).

5.6. Single-Cell RowHammer Bit Flip Probability

We examine how the failure probability of a single RowHammer bit flip changes as *HC* increases. We sweep *HC* between 25k to 150k with a step size of 5k and hammer each DRAM row over 20 iterations. For each *HC* value, we identify each cell's bit flip probability (i.e., the number of times we observe a RowHammer bit flip in that cell out of all 20 iterations). We then observe how each cell's bit flip probability changes as *HC* increases. We expect that by exacerbating the RowHammer conditions (e.g., increasing the hammer count), the exacerbated circuit-level interference effects should result in an increasing RowHammer bit flip probability for each individual cell. Out of the full set of bits that we observe *any* RowHammer bit flips in, Table 5 lists the percentage of cells that have a strictly monotonically increasing bit flip probability as we increase *HC*.

Table 5: Percentage of cells with monotonically increasing RowHammer bit flip probabilities as *HC* increases.

DRAM type-node	Cells with monotonically increasing RowHammer bit flip probabilities (%)		
	Mfr. A	*Mfr. B*	*Mfr. C*
DDR3-new	97.6 $\pm$ 0.2	100	100
DDR4-old	98.4 $\pm$ 0.1	100	100
DDR4-new	99.6 $\pm$ 0.1	100	100
LPDDR4-1x	50.3 $\pm$ 1.2	52.4 $\pm$ 1.4	N/A
LPDDR4-1y	47.0 $\pm$ 0.8	N/A	54.3 $\pm$ 5.7

Observation 14. *For DDR3 and DDR4 chips, an overwhelming majority (i.e., more than 97%) of the cells tested have monotonically increasing RowHammer bit flip probabilities for DDR3 and DDR4 chips.*

This observation indicates that exacerbating the RowHammer conditions by increasing *HC* increases the probability that a DRAM cell experiences a RowHammer bit flip. However, we find that the proportion of cells with monotonically increasing RowHammer bit flip probabilities as *HC* increases is around only 50% in the LPDDR4 chips that we test. We believe that this decrease is due to the addition of on-die ECC in LPDDR4 chips, which can obscure the probability of observing a RowHammer bit flip from the system's perspective in two ways. First, a RowHammer bit flip at bit X can no longer be observable from the system's perspective if another RowHammer bit flip at bit Y occurs within the same ECC word as a result of increasing *HC*, and the error correction logic corrects the RowHammer bit flip at bit X. Second, the system may temporarily observe a bit flip at bit X at a specific *HC* if the set of real RowHammer bit flips within an ECC word results in a miscorrection at bit X. Since this bit flip is a result of the ECC logic misbehaving rather than circuit-level interference, we do not observe the expected trends for these transient miscorrected bits.

6. Implications for Future Systems

Our characterization results have major implications for continued DRAM technology scaling since DRAM's increased vulnerability to RowHammer means that systems employing future DRAM devices will likely need to handle significantly elevated failure rates. While prior works propose a wide variety of RowHammer failure mitigation techniques (described in Sections 6.1 and 7), these mechanisms will need to manage increasing failure rates going forward and will likely suffer from high overhead (as we show in Section 6.2).

While DRAM and system designers currently implement several RowHammer mitigation mechanisms (e.g., *pseudo Target Row Refresh* (pTRR) [51], *Target Row Refresh* (TRR) [81])[11], the designers make a number of unknown implementation choices in these RowHammer mitigation mechanisms that are not discussed in public documentation. Therefore, we cannot fairly evaluate how their performance overheads scale as DRAM chips become more vulnerable to RowHammer. Instead, we evaluate five state-of-the-art academic proposals for RowHammer mitigation mechanisms [63, 77, 116, 134] as well as an ideal refresh-based mitigation mechanism.

We evaluate each RowHammer mitigation mechanism in terms of two major challenges that they will face going forward as they will need to support DRAM chips more vulnerable to RowHammer: design scalability and system performance overhead. We first qualitatively explain and discuss the five state-of-the-art mitigation mechanisms and how they can potentially scale to support DRAM chips that are more vulnerable to RowHammer. We then quantitatively evaluate their performance overheads in simulation as HC_{first} decreases. In order to show the opportunity for reducing performance overhead in RowHammer mitigation, we also implement and study an *ideal refresh-based mechanism* that prevents RowHammer by refreshing a DRAM row *only* immediately before it is about to experience a bit flip.

6.1. RowHammer Mitigation Mechanisms

There is a large body of work (e.g., [3, 10, 11, 15, 27, 34, 44, 62, 67, 79, 123, 128, 129]) that proposes software-based RowHammer mitigation mechanisms. Unfortunately, many of these works have critical weaknesses (e.g., inability to track all DRAM activations) that make them vulnerable to carefully-crafted RowHammer attacks, as demonstrated in some followup works (e.g., [34]). Therefore, we focus on evaluating six mechanisms (i.e., five state-of-the-art hardware proposals and one *ideal* refresh-based mitigation mechanism), which address a strong threat model that assumes an attacker can cause row activations with precise memory location and timing information. We briefly explain each mitigation mechanism and how its design scales for DRAM chips with increased vulnerability to RowHammer (i.e., lower HC_{first} values).

Increased Refresh Rate [63]. The original RowHammer study [63] describes increasing the overall DRAM refresh rate such that it is impossible to issue enough activations within one refresh window (i.e., the time between two consecutive refresh commands to a single DRAM row) to any single DRAM row to induce a RowHammer bit flip. The study notes that this is an undesirable mitigation mechanism due to its associated performance and energy overheads. In order to reliably mitigate RowHammer bit flips with this mechanism, we scale the refresh rate such that the refresh window (i.e., t_{REFW}; the time interval between consecutive refresh commands to a single row) equals the number of hammers until the first RowHammer bit flip (i.e., HC_{first}) multiplied by the activation latency t_{RC}. Due to the large number of rows that must be refreshed within a refresh window, this mechanism inherently does not scale to HC_{first} values below 32k.

PARA [63]. Every time a row is opened and closed, PARA (Probabilistic Adjacent Row Activation) refreshes one or more of the row's adjacent rows with a low probability p. Due to PARA's simple approach, it is possible to easily tune p when PARA must protect a DRAM chip with a lower HC_{first} value. In our evaluation of PARA, we scale p for different values of HC_{first} such that the bit error rate (BER) does not exceed 1e-15 per hour of continuous hammering.[12]

ProHIT [116]. ProHIT maintains a history of DRAM activations in a set of tables to identify any row that may be activated HC_{first} times. ProHIT manages the tables probabilistically to minimize the overhead of tracking frequently-activated DRAM rows. ProHIT [116] uses a pair of tables labeled "Hot" and "Cold" to track the victim rows. When a row is activated, ProHIT checks whether each adjacent row is already in either of the tables. If a *row* is not in either table, it is inserted into the cold table with a probability p_i. If the table is full, the least recently inserted entry in the cold table is then evicted with a probability $(1 - p_e) + p_e/(\#cold\_entries)$ and the other entries are evicted with a probability $p_e/(\#cold\_entries)$. If the row already exists in the cold table, the row is promoted to the highest-priority entry in the hot table with a probability $(1 - p_t) + p_t/(\#hot\_entries)$ and to other entries with a probability $p_t/(\#hot\_entries)$. If the row already exists in the hot table, the entry is upgraded to a higher priority position. During each refresh command, ProHIT simultaneously refreshes the row at the top entry of the hot table, since this row has likely experienced the most number of activations, and then removes the entry from the table.

For ProHIT [116] to effectively mitigate RowHammer with decreasing HC_{first} values, the size of the tables and the probabilities for managing the tables (e.g., p_i, p_e, p_t) must be adjusted. Even though Son et al. show a low-cost mitigation mechanism for a specific HC_{first} value (i.e., 2000), they do *not* provide models for appropriately setting these values for arbitrary HC_{first} values and how to do so is not intuitive. Therefore, we evaluate ProHIT only when $HC_{\text{first}} = 2000$.

MRLoc [134]. MRLoc refreshes a victim row using a probability that is dynamically adjusted based on each row's access history. This way, according to memory access locality, the rows that have been recorded as a victim more recently have a higher chance of being refreshed. MRLoc uses a queue to store victim row addresses on each activation. Depending on the time between two insertions of a given victim row into the queue, MRLoc adjusts the probability with which it issues a refresh to the victim row that is present in the queue.

MRLoc's parameters (the queue size and the parameters used to calculate the probability of refresh) are tuned for $HC_{\text{first}} = 2000$. You et al. [134] choose the values for these parameters empirically, and there is no concrete discussion on how to adjust these parameters as HC_{first} changes. Therefore we evaluate MRLoc for only $HC_{\text{first}} = 2000$.

As such, even though we quantitatively evaluate both ProHIT [116] and MRLoc [134] for completeness and they may seem to have good overhead results at one data point, we are unable to demonstrate how their overheads scale as DRAM chips become more vulnerable to RowHammer.

TWiCe [77]. TWiCe tracks the number of times a victim row's aggressor rows are activated using a table of counters and refreshes a victim row when its count is above a threshold such that RowHammer bit flips cannot occur. TWiCe uses two counters per entry: 1) a lifetime counter, which tracks the length of time the entry has been in the table, and 2) an *activation counter*, which tracks the number of times an

[11] Frigo et al. [26] recently demonstrated that these mechanisms do *not* prevent *all* RowHammer bit flips from being exposed to the system, and an attacker can still take over a system even with these mechanisms in place.

[12] We adopt this BER from typical consumer memory reliability targets [13, 14, 49, 85, 87, 99].

aggressor row is activated. The key idea is that TWiCe can use these two counters to determine the rate at which a row is being hammered and can quickly prune entries that have a low rate of being hammered. TWiCe also minimizes its table size based on the observation that the number of rows that can be activated enough times to induce RowHammer failures within a refresh window is bound by the DRAM chip's vulnerability to RowHammer.

When a row is activated, TWiCe checks whether its adjacent rows are already in the table. If so, the activation count for each row is incremented. Otherwise, new entries are allocated in the table for each row. Whenever a row's activation count surpasses a threshold t_{RH} defined as $HC_{\text{first}}/4$, TWiCe refreshes the row. TWiCe also defines a pruning stage that 1) increments each lifetime counter, 2) checks each row's hammer rate based on both counters, and 3) prunes entries that have a lifetime hammer rate lower than a *pruning threshold*, which is defined as t_{RH} divided by the number of refresh operations per refresh window (i.e., $t_{RH}/(t_{REFW}/t_{REFI})$). TWiCe performs pruning operations during refresh commands so that the latency of a pruning operation is hidden behind the DRAM refresh commands.

If t_{RH} is lower than the number of refresh intervals in a refresh window (i.e., 8192), a couple of complications arise in the design. TWiCe either 1) cannot prune its table, resulting in a very large table size since every row that is accessed at least once will remain in the table until the end of the refresh window or 2) requires floating point operations in order to calculate thresholds for pruning, which would significantly increase the latency of the pruning stage. Either way, the pruning stage latency would increase significantly since a larger table also requires more time to check each entry, and the latency may no longer be hidden by the refresh command.

As a consequence, TWiCe does *not* support t_{RH} values lower than the number of refresh intervals in a refresh window ($\sim 8k$ in several DRAM standards, e.g., DDR3, DDR4, LPDDR4). This means that in its current form, we *cannot* fairly evaluate TWiCe for HC_{first} values below $32k$, as $t_{RH} = HC_{\text{first}}/4$. However, we do evaluate an ideal version of TWiCe (i.e., *TWiCe-ideal*) for HC_{first} values below $32k$ assuming that TWiCe-ideal solves *both* issues of the large table size and the high-latency pruning stage at lower HC_{first} values.

Ideal Refresh-based Mitigation Mechanism. We implement an ideal refresh-based mitigation mechanism that tracks all activations to every row in DRAM and issues a refresh command to a row only right before it can potentially experience a RowHammer bit flip (i.e., when a physically-adjacent row has been activated HC_{first} times).

6.2. Evaluation of Viable Mitigation Mechanisms

We first describe our methodology for evaluating the five state-of-the-art RowHammer mitigation mechanisms (i.e., increased refresh rate [63], PARA [63], ProHIT [116], MRLoc [134], TWiCe [77]) and the ideal refresh-based mitigation mechanism.

6.2.1. Evaluation Methodology. We use Ramulator [66, 105], a cycle-accurate DRAM simulator with a simple core model and a system configuration as listed in Table 6, to implement and evaluate the RowHammer mitigation mechanisms. To demonstrate how the performance overhead of each mechanism would scale to future devices, we implement, to the best of our ability, parameterizable methods for scaling the mitigation mechanisms to DRAM chips with varying degrees of vulnerability to RowHammer (as described in Section 6.1).

Workloads. We evaluate 48 8-core workload mixes drawn randomly from the full SPEC CPU2006 benchmark suite [118] to demonstrate the effects of the RowHammer mitigation mechanisms on systems during typical use (and *not* when a RowHammer attack is being mounted). The set of workloads exhibit a wide range of memory intensities. The workloads' MPKI values (i.e., last-level cache misses per kilo-instruction) range from 10 to 740. This wide range enables us to study the effects of RowHammer mitigation on workloads with widely varying degrees of memory intensity. We note that there could be other workloads with which mitigation mechanisms exhibit higher performance overheads, but we did not try to maximize the overhead experienced by workloads by biasing the workload construction in any way. We simulate each workload until each core executes at least 200 million instructions. For all configurations, we initially warm up the caches by fast-forwarding 100 million instructions.

Table 6: System configuration for simulations.

Parameter	Configuration
Processor	4GHz, 8-core, 4-wide issue, 128-entry instr. window
Last-level Cache	64-Byte cache line, 8-way set-associative, 16MB
Memory Controller	64 read/write request queue, FR-FCFS [103, 136]
Main Memory	DDR4, 1-channel, 1-rank, 4-bank groups, 4-banks per bank group, 16k rows per bank

Metrics. Because state-of-the-art RowHammer mitigation mechanisms rely on additional DRAM refresh operations to prevent RowHammer, we use two different metrics to evaluate their impact on system performance. First, we measure *DRAM bandwidth overhead*, which quantifies the fraction of the total system DRAM bandwidth consumption coming from the RowHammer mitigation mechanism. Second, we measure overall workload performance using the *weighted speedup* metric [23, 115], which effectively measures job throughput for multi-core workloads [23]. We normalize the weighted speedup to its baseline value, which we denote as 100%, and find that when using RowHammer mitigation mechanisms, most values appear below the baseline. Therefore, for clarity, we refer to normalized weighted speedup as *normalized system performance* in our evaluations.

6.2.2. Evaluation of Mitigation Mechanisms. Figure 10 shows the results of our evaluation of the RowHammer mitigation mechanisms (as described in Section 6.1) for chips of varying degrees of RowHammer vulnerability (i.e., $200k \geq HC_{\text{first}} \geq 64$) for our two metrics: 1) DRAM bandwidth overhead in Figure 10a and 2) normalized system performance in Figure 10b. Each data point shows the average value across 48 workloads with minimum and maximum values drawn as error bars.

For each DRAM type-node configuration that we characterize, we plot the minimum HC_{first} value found across chips within the configuration (from Table 4) as a vertical line to show how each RowHammer mitigation mechanism would impact the overall system when using a DRAM chip of a particular configuration. Above the figures (sharing the x-axis with Figure 10), we draw horizontal lines representing the ranges of HC_{first} values that we observe for every tested DRAM chip per DRAM type-node configuration across manufacturers. We color the ranges according to DRAM type-node configuration colors in the figure, and indicate the average value with a gray point. Note that these lines directly correspond to the box-and-whisker plot ranges in Figure 8.

We make *five* key observations from this figure. First, DRAM bandwidth overhead is highly correlated with normalized system performance, as DRAM bandwidth consumption is the main source of system interference caused by RowHammer mitigation mechanisms. We note that several points (i.e., ProHIT, MRLoc, and TWiCe and Ideal evaluated at higher HC_{first} values) are not visible in Figure 10a since we are plotting an inverted log graph and these points are very close to zero. Second, in the latest DRAM chips (i.e., the LPDDR4-1y chips), only PARA, ProHIT, and MRLoc are viable options for mitigating RowHammer bit flips with reasonable average normalized system performance: 92%, 100%,

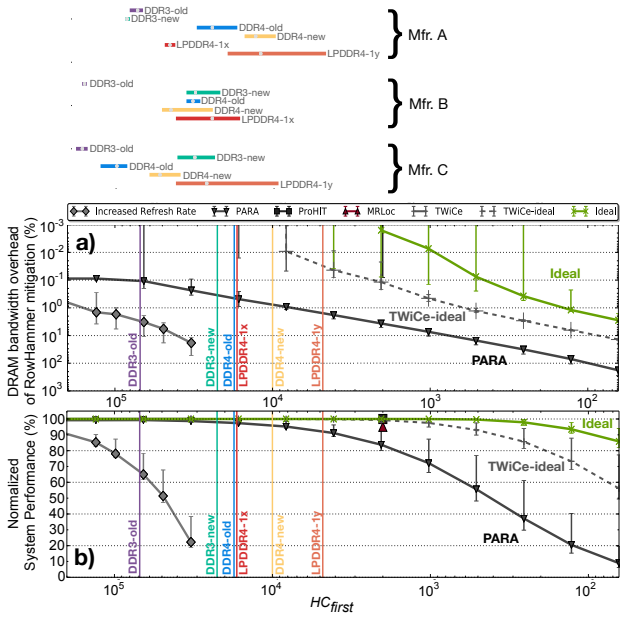

Figure 10: Effect of RowHammer mitigation mechanisms on a) DRAM bandwidth overhead (note the inverted log-scale y-axis) and b) system performance, as DRAM chips become more vulnerable to RowHammer (from left to right).

and 100%, respectively. Increased Refresh Rate and TWiCe do not scale to such degrees of RowHammer vulnerability (i.e., $HC_{\text{first}} = 4.8k$), as discussed in Section 6.1. Third, only PARA's design scales to low HC_{first} values that we may see in future DRAM chips, but has very low average normalized system performance (e.g., 72% when $HC_{\text{first}} = 1024$; 47% when $HC_{\text{first}} = 256$; 20% when $HC_{\text{first}} = 128$). While TWiCe-ideal has higher normalized system performance over PARA (e.g., 98% when $HC_{\text{first}} = 1024$; 86% when $HC_{\text{first}} = 256$; 73% when $HC_{\text{first}} = 128$), there are significant practical limitations in enabling TWiCe-ideal for such low HC_{first} values (discussed in Section 6.1). Fourth, ProHIT and MRLoc both exhibit high normalized system performance at their single data point (i.e., 95% and 100%, respectively when $HC_{\text{first}} = 2000$), but these works do not provide models for scaling their mechanisms to lower HC_{first} values and how to do so is not intuitive (as described in Section 6.1). Fifth, the ideal refresh-based mitigation mechanism is *significantly* and increasingly better than any existing mechanism as HC_{first} reduces below 1024. This indicates that there is still significant opportunity for developing a refresh-based RowHammer mitigation mechanism with low performance overhead that scales to low HC_{first} values. However, the ideal mechanism affects system performance at very low HC_{first} values (e.g., 99.96% when $HC_{\text{first}} = 1024$; 97.91% when $HC_{\text{first}} = 256$; 93.53% when $HC_{\text{first}} = 128$), indicating the potential need for a better approach to solving RowHammer in future ultra-dense DRAM chips.

We conclude that while existing mitigation mechanisms may exhibit reasonably small performance overheads for mitigating RowHammer bit flips in modern DRAM chips, their overheads do *not* scale well in future DRAM chips that will likely exhibit higher vulnerability to RowHammer. Thus, we need new mechanisms and approaches to RowHammer mitigation that will scale to DRAM chips that are highly vulnerable to RowHammer bit flips.

6.3. RowHammer Mitigation Going Forward

DRAM manufacturers continue to adopt smaller technology nodes to improve DRAM storage density and are forecasted to reach 1z and 1a technology nodes within the next couple of years [120]. Unfortunately, our findings show that future DRAM chips will likely be increasingly vulnerable to RowHammer. This means that, to maintain market competitiveness without suffering factory yield loss, manufacturers will need to develop effective RowHammer mitigations for coping with increasingly vulnerable DRAM chips.

6.3.1. Future Directions in RowHammer Mitigation. RowHammer mitigation mechanisms have been proposed across the computing stack ranging from circuit-level mechanisms built into the DRAM chip itself to system-level mechanisms that are agnostic to the particular DRAM chip that the system uses. Of these solutions, our evaluations in Section 6.1 show that, while the ideal refresh-based RowHammer mitigation mechanism, which inserts the minimum possible number of additional refreshes to prevent RowHammer bit flips, scales reasonably well to very low HC_{first} values (e.g., only 6% performance loss when HC_{first} is 128), existing RowHammer mitigation mechanisms either *cannot* scale or cause severe system performance penalties when they scale.

To develop a scalable and low-overhead mechanism that can prevent RowHammer bit flips in DRAM chips with a high degree of RowHammer vulnerability (i.e., with a low HC_{first} value), we believe it is essential to explore all possible avenues for RowHammer mitigation. Going forward, we identify two promising research directions that can potentially lead to new RowHammer solutions that can reach or exceed the scalability of the ideal refresh-based mitigation mechanism: (1) DRAM-system cooperation and (2) profile-guided mechanisms. The remainder of this section briefly discusses our vision for each of these directions.

DRAM-System Cooperation. Considering either DRAM-based or system-level mechanisms alone ignores the potential benefits of addressing the RowHammer vulnerability from both perspectives together. While the root causes of RowHammer bit flips lie within DRAM, their negative effects are observed at the system-level. Prior work [89, 93] stresses the importance of tackling these challenges at all levels of the stack, and we believe that a holistic solution can achieve a high degree of protection at relatively low cost compared to solutions contained within either domain alone.

Profile-Guided Mechanisms. The ability to accurately profile for RowHammer-susceptible DRAM cells or memory regions can provide a powerful substrate for building targeted RowHammer solutions that efficiently mitigate RowHammer bit flips at low cost. Knowing (or effectively predicting) the locations of bit flips before they occur in practice could lead to a large reduction in RowHammer mitigation overhead, providing new information that no known RowHammer mitigation mechanism exploits today. For example, within the scope of known RowHammer mitigation solutions, increasing the refresh rate can be made far cheaper by only increasing the refresh rate for known-vulnerable DRAM rows. Similarly, ECC or DRAM access counters can be used only for known-vulnerable cells, and even a software-based mechanism can be adapted to target only known-vulnerable rows (e.g., by disabling them or remapping them to reliable memory).

Unfortunately, there exists no such effective RowHammer error profiling methodology today. Our characterization in this work essentially follows the naïve approach of individually testing each row by attempting to induce the worst-case testing conditions (e.g., *HC*, data pattern, ambient temperature etc.). However, this approach is extremely time consuming due to having to test each row individually (potentially multiple times with various testing conditions). Even for a relatively small DRAM module of 8GB with 8KB rows, hammering each row only once for only one refresh window of 64ms requires over 17 hours of continuous testing, which means that the naïve approach to profiling is infeasible for a general mechanism that may be used in a production environment or for online operation. We believe that developing a fast and effective RowHammer profiling mechanism is a key

research challenge, and we hope that future work will use the observations made in this study and other RowHammer characterization studies to find a solution.

7. Related Work

Although many works propose RowHammer attacks and mitigation mechanisms, only three works [63, 96, 97] provide detailed failure-characterization studies that examine how RowHammer failures manifest in real DRAM chips. However, none of these studies show how the number of activations to induce RowHammer bit flips is changing across modern DRAM types and generations, and the original RowHammer study [63] is already six years old and limited to DDR3 DRAM chips only. This section highlights the most closely related prior works that study the RowHammer vulnerability of older generation chips or examine other aspects of RowHammer.

Real Chip Studies. Three key studies (i.e., the pioneering RowHammer study [63] and two subsequent studies [96, 97]) perform extensive experimental RowHammer failure characterization using older DDR3 devices. However, these studies are restricted to only DDR3 devices and do not provide a scaling study of hammer counts across DRAM types and generations. In contrast, our work provides the first rigorous experimental study showing how RowHammer characteristics scale across different DRAM generations and how DRAM chips designed with newer technology nodes are increasingly vulnerable to RowHammer. Our work complements and furthers the analyses provided in prior studies.

Simulation Studies. Yang et al. [133] use device-level simulations to explore the root cause of the RowHammer vulnerability. While their analysis identifies a likely explanation for the failure mechanism responsible for RowHammer, they do not present experimental data taken from real devices to support their conclusions.

RowHammer Mitigation Mechanisms. Many prior works [2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 15, 24, 25, 30, 33, 38, 41, 44, 57, 62, 63, 67, 77, 78, 79, 116, 123, 127, 128, 129, 134] propose RowHammer mitigation techniques. Additionally, several patents for RowHammer prevention mechanisms have been filed [4, 5, 6, 7, 8, 32]. However, these works do not analyze how their solutions will scale to future DRAM generations and do not provide detailed failure characterization data from modern DRAM devices. Similar and other related works on RowHammer can be found in a recent retrospective [92].

8. Conclusion

We provide the first rigorous experimental RowHammer failure characterization study that demonstrates how the RowHammer vulnerability of modern DDR3, DDR4, and LPDDR4 DRAM chips scales across DRAM generations and technology nodes. Using experimental data from 1580 real DRAM chips produced by the three major DRAM manufacturers, we show that modern DRAM chips that use smaller process technology node sizes are significantly more vulnerable to RowHammer than older chips. Using simulation, we show that existing RowHammer mitigation mechanisms 1) suffer from prohibitively large performance overheads at projected future hammer counts and 2) are still far from an *ideal* selective-refresh-based RowHammer mitigation mechanism. Based on our study, we motivate the need for a scalable and low-overhead solution to RowHammer and provide two promising research directions to this end. We hope that the results of our study will inspire and aid future work to develop efficient solutions for the RowHammer bit flip rates we are likely to see in DRAM chips in the near future.

Acknowledgments

We thank the anonymous reviewers of ISCA 2020 for feedback and the SAFARI group members for feedback and the stimulating intellectual environment they provide. We also thank our industrial partners for their generous donations.

References

[1] B. Aichinger, "DDR Memory Errors Caused by Row Hammer," in *HPEC*, 2015.
[2] Apple Inc., "About the Security Content of Mac EFI Security Update 2015-001," https://support.apple.com/en-us/HT204934, 2015.
[3] Z. B. Aweke *et al.*, "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks," in *ASPLOS*, 2016.
[4] K. Bains *et al.*, "Row Hammer Refresh Command," 2015, US Patent 9,117,544.
[5] K. S. Bains *et al.*, "Row Hammer Monitoring Based on Stored Row Hammer Threshold Value," 2015, US Patent 9,032,141.
[6] K. S. Bains *et al.*, "Distributed Row Hammer Tracking," 2016, US Patent 9,299,400.
[7] K. S. Bains *et al.*, "Row Hammer Refresh Command," US Patent 9,236,110, 2016.
[8] K. S. Bains *et al.*, "Method, Apparatus and System for Providing a Memory Refresh," 2015, US Patent 9,030,903.
[9] R. Balasubramonian, "Innovations in the Memory System," *Synthesis Lectures on Computer Architecture*, 2019.
[10] C. Bock *et al.*, "RIP-RH: Preventing Rowhammer-Based Inter-Process Attacks," in *ASIA-CCS*, 2019.
[11] F. Brasser *et al.*, "Can't Touch This: Practical and Generic Software-only Defenses Against RowHammer Attacks," *USENIX Security*, 2017.
[12] L. Bu *et al.*, "SRASA: A Generalized Theoretical Framework for Security and Reliability Analysis in Computing Systems," *HaSS*, 2018.
[13] Y. Cai *et al.*, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.
[14] Y. Cai *et al.*, "Flash Memory SSD Errors, Mitigation, and Recovery," in *Proc. IEEE*, 2017.
[15] A. Chakraborty *et al.*, "Deep Learning Based Diagnostics for Rowhammer Protection of DRAM Chips," in *ATS*, 2019.
[16] K. K. Chang, "Understanding and Improving Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.
[17] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[18] K. K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[19] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[20] L. Cojocar *et al.*, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," *SP*, 2020.
[21] L. Cojocar *et al.*, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against RowHammer Attacks," in *SP*, 2019.
[22] R. H. Dennard, "Field-Effect Transistor Memory," 1968, US Patent 3,387,286.
[23] S. Eyerman *et al.*, "System-level Performance Metrics for Multiprogram Workloads," in *IEEE Micro*, 2008.
[24] D. E. Fisch *et al.*, "DRAM Adjacent Row Disturb Mitigation," 2017, US Patent 9,812,185.
[25] T. Fridley *et al.*, "Mitigations Available for the DRAM Row Hammer Vulnerability," http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability, 2015.
[26] P. Frigo *et al.*, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *SP*, 2020.
[27] M. Ghasempour *et al.*, "ARMOR: A Run-Time Memory Hot-Row Detector," 2015.
[28] S. Ghose *et al.*, "Demystifying Complex Workload-DRAM Interactions: An Experimental Study," *SIGMETRICS*, 2019.
[29] S. Ghose *et al.*, "What Your DRAM Power Models are not Telling You: Lessons from a Detailed Experimental Study," *SIGMETRICS*, 2018.
[30] H. Gomez *et al.*, "DRAM Row-Hammer Attack Reduction using Dummy Cells," in *NORCAS*, 2016.
[31] S.-L. Gong *et al.*, "Duo: Exposing on-chip Redundancy to Rank-level ECC for High Reliability," in *HPCA*, 2018.
[32] Z. Greenfield *et al.*, "Row Hammer Condition Monitoring," 2015, US Patent 8,938,573.
[33] Z. Greenfield *et al.*, "Method, Apparatus and System for Determining A Count of Accesses to A Row of Memory," US Patent App. 13/626,479, 2014.
[34] D. Gruss *et al.*, "Another Flip in the Wall of RowHammer Defenses," in *SP*, 2018.
[35] D. Gruss *et al.*, "Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript," in *CoRR*, 2016.
[36] J. A. Halderman *et al.*, "Lest we Remember: Cold-Boot Attacks on Encryption Keys," *USENIX Security*, 2008.
[37] T. Hamamoto *et al.*, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," in *ED*, 1998.
[38] H. Hassan *et al.*, "CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," in *ISCA*, 2019.
[39] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[40] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[41] Hewlett-Packard Enterprise, "HP Moonshot Component Pack Version 2015.05.0," http://h17007.www1.hp.com/us/en/enterprise/servers/products/moonshot/component-pack/index.aspx, 2015.
[42] S. Hong, "Memory Technology Trend and Future Challenges," in *IEDM*, 2010.
[43] Intel Corporation, "CannonLake Intel Firmware Support Package (FSP) Integration Guide," https://usermanual.wiki/Pdf/CannonLakeFSPIntegrationGuide.58784693.pdf, 2017.
[44] G. Irazoqui *et al.*, "MASCAT: Stopping Microarchitectural Attacks Before Execution," *IACR Cryptology ePrint Archive*, 2016.
[45] JEDEC, "Double Data Rate 3 (DDR3) SDRAM Specification," 2012.
[46] JEDEC, "Double Data Rate 4 (DDR4) SDRAM Standard," 2012.

[47] JEDEC, "Low Power Double Data Rate 3 (LPDDR3) SDRAM Specification," 2012.
[48] JEDEC, "Low Power Double Data Rate 4 (LPDDR4) SDRAM Specification," 2014.
[49] JEDEC Solid State Technology Association, "Failure Mechanisms and Models for Semiconductor Devices," *JEDEC Publication JEP122G*, 2011.
[50] S. Ji *et al.*, "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks," in *ASIACCS*, 2019.
[51] M. Kaczmarski, "Thoughts on Intel Xeon e5–2600 v2 Product Family Performance Optimisation–Component Selection Guidelines," 2014.
[52] I. Kang *et al.*, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, 2020.
[53] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[54] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[55] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[56] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[57] D.-H. Kim *et al.*, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *IEEE CAL*, 2014.
[58] J. S. Kim *et al.*, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.
[59] J. S. Kim *et al.*, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *arXiv*, 2020.
[60] J. Kim *et al.*, "Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory," in *HPCA*, 2015.
[61] K. Kim *et al.*, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," in *EDL*, 2009.
[62] M. Kim *et al.*, "An Effective DRAM Address Remapping for Mitigating Rowhammer Errors," *TC*, 2019.
[63] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[64] Y. Kim *et al.*, "Rowhammer: Reliability Analysis and Security Implications," *arXiv*, 2016.
[65] Y. Kim *et al.*, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[66] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," in *CAL*, 2016.
[67] R. K. Konoth *et al.*, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *OSDI*, 2018.
[68] N. Kwak *et al.*, "A 4.8 Gb/s/pin 2Gb LPDDR4 SDRAM with Sub-100μA Self-Refresh Current for IoT Applications," in *ISSCC*, 2017.
[69] H. Kwon *et al.*, "An Extremely Low-Standby-Power 3.733 Gb/s/pin 2Gb LPDDR4 SDRAM for Wearable Devices," in *ISSCC*, 2017.
[70] A. Kwong *et al.*, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *SP*, 2020.
[71] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[72] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.
[73] D. Lee *et al.*, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," in *TACO*, 2016.
[74] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[75] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[76] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[77] E. Lee *et al.*, "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters," in *ISCA*, 2019.
[78] Lenovo, "Row Hammer Privilege Escalation," https://support.lenovo.com/us/en/product_security/row_hammer, 2015.
[79] C. Li *et al.*, "Detecting Malicious Attacks Exploiting Hardware Vulnerabilities Using Performance Counters," in *COMPSAC*, 2019.
[80] Y. Li *et al.*, "DRAM Yield Analysis and Optimization by a Statistical Design Approach," in *CSI*, 2011.
[81] J. Lin *et al.*, "Handling Maximum Activation Count Limit and Target Row Refresh in DDR4 SDRAM," 2017, US Patent 9,589,606.
[82] M. Lipp *et al.*, "Nethammer: Inducing RowHammer Faults Through Network Requests," *arXiv*, 2018.
[83] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[84] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[85] Y. Luo *et al.*, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," in *JSAC*, 2016.
[86] J. A. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-access Memory (DRAM)," in *IBM JRD*, 2002.
[87] R. Micheloni *et al.*, "Apparatus and Method Based on LDPC Codes for Adjusting A Correctable Raw Bit Error Rate Limit in A Memory System," 2015, US Patent 9,092,353.
[88] Micron Technology inc., "ECC Brings Reliability and Power Efficiency to Mobile Devices," Micron Technology inc., Tech. Rep., 2017.
[89] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *IMW*, 2013.
[90] O. Mutlu, "The RowHammer Problem and Other Issues We may Face as Memory Becomes Denser," in *DATE*, 2017.
[91] O. Mutlu, "RowHammer and Beyond," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2019.
[92] O. Mutlu *et al.*, "RowHammer: A Retrospective," *TCAD*, 2019.
[93] O. Mutlu *et al.*, "Research Problems and Opportunities in Memory Systems," in *SUPERFRI*, 2014.
[94] T.-Y. Oh *et al.*, "A 3.2Gbps/pin 8Gbit 1.0V LPDDR4 SDRAM with Integrated ECC Engine for Sub-1V DRAM Core Operation," *JSSC*, vol. 50, 2015.
[95] Omron, "NY-series Industrial Box PC - Hardware User's Manual," https://assets.omron.eu/downloads/manual/en/v6/w553_ny-series_industrial_box_pc_users_manual_en.pdf, 2019.
[96] K. Park *et al.*, "Experiments and Root Cause Analysis for Active-Precharge Hammering Fault in DDR3 SDRAM under 3× nm Technology," *MR*, 2016.
[97] K. Park *et al.*, "Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components," *MR*, 2016.
[98] M. Patel *et al.*, "Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices," in *DSN*, 2019.
[99] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[100] R. Qiao *et al.*, "A New Approach for RowHammer Attacks," in *HOST*, 2016.
[101] M. K. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[102] K. Razavi *et al.*, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.
[103] S. Rixner *et al.*, "Memory Access Scheduling," in *ISCA*, 2000.
[104] S.-W. Ryu *et al.*, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing," in *IEDM*, 2017.
[105] SAFARI Research Group, "Ramulator Source Code," https://github.com/CMU-SAFARI/ramulator.
[106] SAFARI Research Group, "SoftMC Source Code," https://github.com/CMU-SAFARI/SoftMC.
[107] K. Saino *et al.*, "Impact of Gate-induced Drain Leakage Current on the Tail Distribution of DRAM Data Retention Time," in *IEDM*, 2000.
[108] M. Seaborn *et al.*, "Exploiting the DRAM RowHammer Bug to Gain Kernel Privileges," *Black Hat*, 2015.
[109] V. Seshadri, "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2016.
[110] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[111] V. Seshadri *et al.*, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[112] V. Seshadri *et al.*, "In-DRAM Bulk Bitwise Execution Engine," *Advances in Computers*, 2020.
[113] S. M. Seyedzadeh *et al.*, "Mitigating Wordline Crosstalk Using Adaptive Trees of Counters," in *ISCA*, 2018.
[114] S. M. Seyedzadeh *et al.*, "Counter-based Tree Structure for Row Hammering Mitigation in DRAM," *CAL*, 2017.
[115] A. Snavely *et al.*, "Symbiotic Jobscheduling for a Simultaneous Mutlithreading Processor," in *ASPLOS*, 2000.
[116] M. Son *et al.*, "Making DRAM Stronger Against Row Hammering," in *DAC*, 2017.
[117] Y. H. Son *et al.*, "CiDRA: A Cache-Inspired DRAM Resilience Architecture," in *HPCA*, 2015.
[118] Standard Performance Evaluation Corp., "SPEC CPU@2006," 2006, http://www.spec.org/cpu2006.
[119] A. Tatar *et al.*, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *RAID*, 2018.
[120] Tech Insights, "DRAM Technology/Products Roadmap," 2019.
[121] TQ-Systems, "TQMx80UC User's Manual," https://www.tq-group.com/filedownloads/files/products/embedded/manuals/x86/embedded-modul/COM-Express-Compact/TQMx80UC/TQMx80UC.UM.0102.pdf, 2020.
[122] V. Van Der Veen *et al.*, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.
[123] V. van der Veen *et al.*, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.
[124] R. K. Venkatesan *et al.*, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, 2006.
[125] VersaLogic Corporation, "Blackbird BIOS Reference Manual," https://www.versalogic.com/wp-content/themes/vsl-new/assets/pdf/manuals/MEPU44624562BRM.pdf, 2019.
[126] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.
[127] Y. Wang *et al.*, "Detect DRAM Disturbance Error by Using Disturbance Bin Counters," *CAL*, 2019.
[128] Y. Wang *et al.*, "Reinforce Memory Error Protection by Breaking DRAM Disturbance Correlation Within ECC Words," in *ICCD*, 2019.
[129] X.-C. Wu *et al.*, "Protecting Page Tables from RowHammer Attacks using Monotonic Pointers in DRAM True-Cells," in *ASPLOS*, 2019.
[130] Y. Xiao *et al.*, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.
[131] Xilinx, "ML605 Hardware User Guide," https://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf.
[132] Xilinx, "Virtex UltraScale FPGAs," https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html.
[133] T. Yang *et al.*, "Trap-Assisted DRAM Row Hammer Effect," *EDL*, 2019.
[134] J. M. You *et al.*, "MRLoc: Mitigating Row-Hammering Based on Memory Locality," in *DAC*, 2019.
[135] T. Zhang *et al.*, "Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-Grained Activation," in *ISCA*, 2014.
[136] W. K. Zuravleff *et al.*, "Controller for a Synchronous DRAM that Maximizes Throughput by Allowing Memory Requests and Commands to be Issued Out of Order," 1997, US Patent 5,630,096.