2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)

Read Disturbance in High Bandwidth Memory: A Detailed Experimental Study on HBM2 DRAM Chips

Ataberk Olgun[1] Majd Osseiran[1] A. Giray Yağlıkçı[1] Yahya Can Tuğrul[1]
Haocong Luo[1] Steve Rhyner[1] Behzad Salami[2] Juan Gomez Luna[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*BSC*

We experimentally demonstrate the effects of read disturbance (RowHammer and RowPress) and uncover the inner workings of undocumented read disturbance defense mechanisms in High Bandwidth Memory (HBM). Detailed characterization of six real HBM2 DRAM chips in two different FPGA boards shows that (1) the read disturbance vulnerability significantly varies between different HBM2 chips and between different components (e.g., 3D-stacked channels) inside a chip, (2) DRAM rows at the end and in the middle of a bank are more resilient to read disturbance, (3) fewer additional activations are sufficient to induce more read disturbance bitflips in a DRAM row if the row exhibits the first bitflip at a relatively high activation count, (4) a modern HBM2 chip implements undocumented read disturbance defenses that track potential aggressor rows based on how many times they are activated. We describe how our findings could be leveraged to develop more powerful read disturbance attacks and more efficient defense mechanisms. We open source all our code and data to facilitate future research at `https://github.com/CMU-SAFARI/HBM-Read-Disturbance`.

1. Introduction

Modern DRAM chips suffer from read disturbance issues [1–4] that can be exploited to break memory isolation, threatening the robustness (including safety, security, and reliability) of modern DRAM-based computing systems. RowHammer [1] and RowPress [4] are two prominent examples of read disturbance. Repeatedly opening/activating and closing a DRAM row (i.e., aggressor row) *many times* (e.g., thousands of times) induces *RowHammer bitflips* in physically nearby rows (i.e., victim rows). Keeping the aggressor row open for a long period of time (i.e., a large aggressor row on time, t_{AggON}) amplifies the effects of read disturbance and induces *RowPress bitflips*, at much lower aggressor row activation counts [4]. Numerous studies demonstrate that a malicious attacker can reliably cause read disturbance bitflips in a targeted manner to compromise system integrity, confidentiality, and availability [1, 5–67]. Read disturbance worsens in new DRAM chips with smaller technology nodes, where RowHammer bitflips 1) happen with fewer row activations, e.g., 10× reduction in less than a decade [68] and 2) appear in more DRAM cells, compared to old DRAM chips [3, 27, 35, 38, 68–71].

To meet the high-bandwidth requirements of modern data-intensive applications (e.g., GPU workloads [72, 73], machine learning training and inference models [74–76]), DRAM designers develop High Bandwidth Memory (HBM) [77] DRAM chips, which contain multiple layers of 3D-stacked DRAM dies, using cutting-edge technology nodes.[1] It is important to understand read disturbance in HBM DRAM chips because 1) they have new architectural characteristics (e.g., multiple layers of DRAM dies, area- and energy-intensive through-silicon vias), which might affect the chip's read disturbance vulnerability in currently-unknown ways, and 2) they are extensively used in critical system infrastructures of today (e.g., machine learning training and inference [74, 78–82]). Such understanding can help identify potential read-disturbance-induced security and reliability issues in HBM-based systems and allow for effective and efficient defense mechanisms.

Our goal is to experimentally analyze how vulnerable HBM DRAM chips are to read disturbance. To this end, we provide the first detailed experimental characterization of the RowHammer and the RowPress vulnerabilities in six modern HBM2 DRAM chips in two different FPGA boards. We provide four main analyses in our study. First, we analyze the spatial variation in RowHammer vulnerability (§ 4) based on the physical location of victim rows in terms of two metrics: the fraction of DRAM cells that experience a bitflip in a DRAM row (*BER*) and the minimum hammer count necessary to cause a RowHammer bitflip (HC_{first}). We use these two metrics to quantify the RowHammer susceptibility of a DRAM row. For example, a row with a small *BER* value is susceptible to only a small number of RowHammer-induced bitflips. As such, this row is more *RowHammer-resilient* than other DRAM rows with higher *BER* values. Second, we analyze the number of aggressor row activations (i.e., hammer count) necessary to induce the first 10 bitflips in a DRAM row (§ 5). We demonstrate how many additional hammers over HC_{first} are needed to induce each of the first 10 bitflips. Third, we test RowPress and RowHammer's sensitivities to the amount of time a row remains active, i.e., the aggressor row on time (t_{AggON}) (§ 6). To do so, we sweep t_{AggON} from the minimum standard value of 29.0 ns to an extreme 16.0 ms. Fourth, we investigate undocumented in-DRAM read disturbance defense mechanisms that are triggered by periodic refresh operations (e.g., Target Row Refresh [38, 44, 83], or TRR for short) in an HBM2 chip (§ 7).[2] We summarize the key observations from our four main analyses.

1) Spatial variation in RowHammer (§ 4). First, DRAM rows near the end and in the middle of a DRAM bank (last and middle 832 rows) exhibit substantially smaller *BER* than other DRAM rows. Second, RowHammer *BER* and HC_{first}

[1]We use "chip" to refer to an *HBM2 stack*. An HBM2 stack contains one or multiple DRAM layers. We refer to each such layer using "DRAM die".

[2]The HBM2 standard specifies a Target Row Refresh (TRR) Mode. To enable TRR Mode, the memory controller issues a well-defined series of commands. We investigate undocumented TRR mechanisms that function even when the DRAM chip is *not* in TRR mode.

2158-3927/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN58291.2024.00022

vary between DRAM chips. For example, the chip-level mean *BER* and minimum HC_{first} differ by up to 0.49 percentage points (pp) and 3556, respectively. Third, different 3D-stacked channels of an HBM2 chip exhibit significantly different levels of RowHammer vulnerability in *BER* (e.g., the channel with the highest mean *BER* has 1.99× the mean *BER* of the channel with the lowest mean *BER*) and HC_{first}. Fourth, the mean *BER* variation across channels in multiple HBM2 chips is larger than the mean *BER* variation across all HBM2 chips, for all tested data patterns (Table 1). For example, the channel with the highest mean *BER* across all its rows in Chip 4 has 0.88 pp higher *BER* than the channel with that of smallest and the chip with the highest *BER* across all its rows has 0.38 pp higher *BER* than the chip with that of smallest.

2) RowHammer's Sensitivity to Hammer Count (§ 5). We show that the hammer count to induce more than one or more additional RowHammer bitflips (up to 10) in a row can be very close to (1.15× larger than) or very far from (5.22× larger than) the hammer count to induce the *first* RowHammer bitflip depending on the DRAM row. We find that, in general, it takes fewer additional hammer counts (over HC_{first}) to induce up to 10 RowHammer bitflips in a DRAM row that has a large HC_{first} compared to a DRAM row that has a small HC_{first}.

3) RowHammer's and RowPress's Sensitivities to t_{AggON} (§ 6). We observe that as the time an aggressor row remains open (t_{AggON}) increases, DRAM cells become more vulnerable to read disturbance. For example, the HC_{first} of a row at a t_{AggON} of 35.1 µs (the maximum allowed time to keep a row open [77]) is 222.57× smaller than the HC_{first} of the row at a t_{AggON} of 29.0 ns, averaged across all tested DRAM rows. We observe that *only* one DRAM row activation is sufficient to induce RowPress bitflips at an extreme t_{AggON} of 16 ms.

4) In-DRAM RowHammer defenses (§ 7). We uncover that an HBM2 DRAM chip implements an in-DRAM RowHammer defense mechanism that is not disclosed in the HBM2 specification [77]. The undocumented target row refresh (TRR) mechanism identifies as aggressor rows i) the *first row* that gets activated after a TRR operation (a victim row refresh) and ii) the row whose activation count exceeds half the number of total row activations within a refresh interval. We experimentally demonstrate that an attacker could practically and reliably defeat this undocumented TRR mechanism in real HBM2 DRAM chips by leveraging our observations.

We highlight three of the six key implications of our observations for future read disturbance attacks and defenses (§ 8): 1) the maximum *BER* (247 bitflips in a row of 8192 bits) is likely sufficient for conducting practical attacks (e.g., privilege escalation) in modern HBM-based systems, 2) a read disturbance attack could find exploitable bitflips faster by targeting the most-read-disturbance-vulnerable HBM2 channels, and 3) read disturbance defense mechanisms could more efficiently prevent read-disturb bitflips by adapting to the heterogeneous distribution of the RowHammer and RowPress vulnerability across channels and subarrays.

We make the following contributions:

- We present the first detailed experimental characterization of read disturbance (RowHammer and RowPress) in six state-of-the-art HBM2 DRAM chips. We show that all of the tested HBM2 DRAM chips are susceptible to read disturbance bitflips.
- We show that the RowHammer and RowPress vulnerability in HBM2 varies significantly across chips and HBM2 components within each chip (e.g., 3D-stacked channels, pseudo channels, banks, and rows).
- We present the first analysis on hammer count to induce up to 10 bitflips in a DRAM row. We show that a DRAM row with a large hammer count to induce the first bitflip is likely to require fewer additional hammer counts to exhibit the next 9 bitflips.
- We characterize the RowPress vulnerability in six HBM2 chips. Keeping the aggressor row open for the maximum allowed time reduces the activation count to induce a bitflip by three orders of magnitude on average across all tested chips. We show that all chips exhibit bitflips when the aggressor row is activated *only once* and kept open for a very long time (16 ms).
- We uncover the inner workings of an undocumented in-HBM-chip read disturbance defense mechanism. We analyze this defense mechanism and craft a specialized RowHammer access pattern that bypasses the defense.
- We open-source all our infrastructure [84], test programs, and raw data to enable 1) reproduction and replication of our results, and 2) further research on read disturbance in HBM chips.

2. Background & Motivation

We describe the necessary background on HBM2 organization and operation, and motivate read disturbance and its implications for real HBM2 systems.

2.1. HBM2 Organization

Fig. 1 shows the organization of an HBM2 DRAM chip [85] used in an FPGA-based system. The memory controller communicates with multiple stacks of HBM using the HBM2 interface (❶). An HBM2 stack contains multiple DRAM dies stacked on top of the buffer die and connected using through-silicon vias (TSVs) (❷). Each HBM2 die comprises one or multiple HBM2 channels that can operate independently (❸). An HBM2 channel contains multiple pseudo channels and each pseudo channel has multiple banks (❹). A DRAM bank comprises multiple DRAM cells that are laid in a two-dimensional array of rows and columns (❺). DRAM cells are typically partitioned into multiple DRAM subarrays [86–88] (not shown in the figure) that each contain a row buffer. When enabled, a wordline connects a DRAM cell to its bitline, copying the data stored in the DRAM cell to the row buffer.

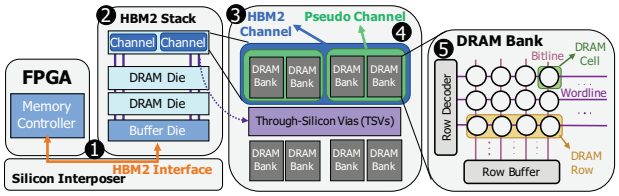

Figure 1: HBM2 DRAM system organization

2.2. HBM2 Operation

To access a DRAM chip, the memory controller needs to issue the following sequence of commands. First, the controller issues an activate (ACT) command targeting a DRAM row to access a DRAM cell. The row decoder enables the row's wordline, copying the data in the row to the row buffer. Second, to read from or write to a particular column in that row, a RD or WR command needs to be issued. Finally, when accesses to the open row are complete, the memory controller issues a precharge (*PRE*) command, which disables the enabled wordline so that the memory controller can later access a different cell in another DRAM row.

To maintain reliable DRAM operation, the memory controller must obey manufacturer-recommended, standard timing parameters. These timing parameters ensure that the DRAM circuitry has enough time to execute the operations dictated by DRAM commands. Two relevant timing parameters for our study are charge restoration latency (t_{RAS}) and refresh interval (t_{REFI}). First, t_{RAS}, the minimum time that a row should remain active before a *PRE* command is sent to the row's bank. t_{RAS} guarantees that DRAM sense amplifiers have enough time to restore charge in cells of the open row before the row is closed. Second, t_{REFI}, the periodic interval at which a refresh cycle is required. Since a DRAM cell stores data as charge in its capacitor and the capacitor naturally loses charge over time, the capacitor must be periodically refreshed to prevent data corruption. Consequently, the memory controller should issue a *REF* command on average every 3.9 µs [77] (t_{REFI}), such that every DRAM cell is refreshed once at a fixed refresh window (e.g., 32 ms). The memory controller may delay a *REF* command up to 35.1 µs ($9 * t_{REFI}$). However, any such large delay must be compensated by multiple smaller delays between successive *REF* commands following the large delay.

2.3. Motivation

Read-disturb phenomena (e.g., RowHammer [1] and RowPress [4]) break the fundamental building block of modern system security principles, i.e., *memory isolation*. This property allows read-disturb phenomena to be used in system-level attacks that compromise system integrity, confidentiality, and availability in various real computing systems, as many prior works have shown [1, 5–67]. Therefore, it is critical to understand the properties of the RowHammer and RowPress vulnerabilities to design defense mechanisms and protect modern DRAM chips against read-disturbance-based attacks. Unfortunately, no prior work extensively studies the RowHammer and RowPress vulnerabilities of modern HBM chips. To this end, our goal is to experimentally evaluate and understand the RowHammer and RowPress vulnerabilities in real HBM chips. To achieve this goal, we perform a rigorous experimental characterization study of read disturbance on six HBM2 chips in two different types of integrated circuit packages.

3. Experimental Infrastructure

We use a modified version of the DRAM Bender testing infrastructure [89, 90]. This infrastructure allows us to precisely control the HBM2 command timings at the granularity of 1.67 ns (i.e., the HBM2 interface clock speed is 600 MHz). All tested HBM2 chips have i) 8 channels, ii) 2 pseudo channels, iii) 16 banks, iv) 16384 rows, and v) 1 KiB rows.

Testing setup. Fig. 2 shows one of our six testing setups. We conduct experiments using one Bittware XUPVVH [91] (1) and five AMD Xilinx Alveo U50 FPGA boards [92] (not shown in Fig. 2). We use the heating pad (2) and the cooling fan (3) to increase and reduce the temperature of the HBM2 chip, respectively. The Arduino [93] temperature controller (4) communicates with i) the host machine to retrieve a target temperature and ii) the FPGA board to retrieve the HBM2 chip's temperature. A host machine executes the test programs described in § 3.1 on the FPGA board using the PCIe connection (5).

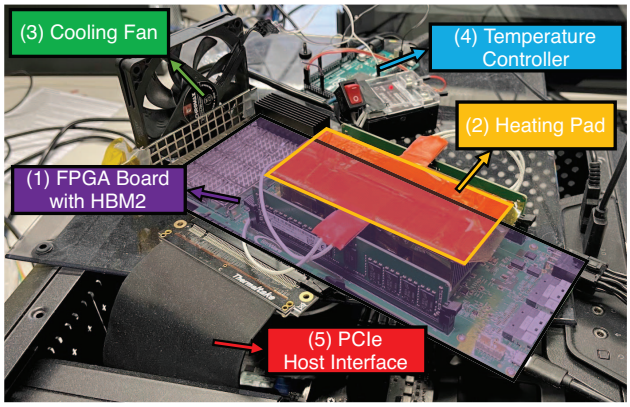

Figure 2: FPGA-based HBM2 DRAM tester.

3.1. Testing Methodology

Disabling Sources of Interference. We identify four sources that can interfere with our characterization results: 1) periodic refresh [77], 2) on-die read disturbance defense mechanisms (e.g., TRR [38, 44, 83]), 3) data retention failures [94, 95], and 4) ECC [77]. First, we do *not* issue periodic refresh commands in our experiments. Second, disabling periodic refresh disables all known on-die read disturbance defense mechanisms [44, 68–70]. Third, we ensure that our experiments finish within the 32 ms refresh interval where manufacturers guarantee no retention errors will occur [77]. Fourth, we disable ECC by setting the corresponding HBM2 mode register bit to zero [77].

RowHammer and RowPress Access Pattern. We use the double-sided read disturbance access pattern [1, 14, 68, 70], which alternately activates each aggressor row. We record the bitflips observed in the sandwiched victim row (i.e., the row between two aggressor rows).

Logical-to-Physical Row Mapping. DRAM manufacturers use mapping schemes to translate logical (memory-controller-visible) addresses to physical row addresses [1, 9, 28, 39, 70, 94, 96–105]. To identify aggressor rows that are physically adjacent to a victim row, we reverse-engineer the row mapping scheme following the methodology described in prior work [70].

RowHammer and RowPress Test Parameters. We configure our tests by tuning three parameters: 1) Hammer count: We define the *hammer count* of a double-sided read disturbance access pattern as the number of activations *each* aggressor row receives. Therefore, during a double-sided RowHammer or a RowPress test with a hammer count of 10, we activate each

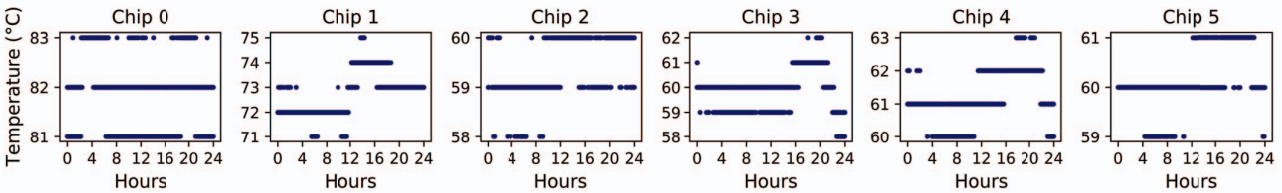

Figure 3: Tested HBM2 chips' temperature over time (24 hours). We draw temperature measurements taken every 5 seconds over a 24 hour time window.

of the two aggressor rows 10 times, resulting in a total of 20 row activations. 2) t_{AggON}: The time each aggressor row stays on with each activation during a RowHammer or a RowPress test. 3) Data pattern: We use the four data patterns (Table 1) that are widely used in memory reliability testing [106] and by prior work on DRAM characterization (e.g., [1, 4, 68, 70, 107]). For each DRAM row, we define *WCDP* as the data pattern that causes the smallest HC_{first}. When multiple data patterns cause the same HC_{first}, we select WCDP as the data pattern that causes the largest *BER* at hammer count = 256K.

Table 1: Data patterns used in our experiments

Row Addresses	*Rowstripe0*	*Rowstripe1*	*Checkered0*	*Checkered1*
Victim (V)	0x00	0xFF	0x55	0xAA
Aggressors (V ± 1)	0xFF	0x00	0xAA	0x55
V ± [2:8]	0x00	0xFF	0x55	0xAA

Read Disturbance Vulnerability Metrics. We measure the read disturbance vulnerability based on two metrics: 1) HC_{first} and 2) *BER* as defined in § 1.

Tested DRAM Components. To maintain a reasonable experiment time, the number of DRAM components (channels, pseudo channels, banks, and rows) we test varies depending on the experiment type. Table 2 summarizes the number of components tested for each experiment type.

Table 2: Tested DRAM components for each experiment type

Experiment Type	Rows (Per Bank)	Banks	Pseudo Channels	Channels
RowHammer *BER*	16384	1	1	8
RowHammer HC_{first}	3072	3	2	8
RowPress *BER*	384	1	1	3
RowPress HC_{first}	384	1	1	3

Experiment Repetitions. We repeat every experiment five times. We report 1) the average value across five repetitions for *BER* experiments and 2) the minimum value across five repetitions for HC_{first} experiments.

HBM2 Chip Labeling. We label the XUPVVH board's HBM2 chip as Chip 0 and the five U50 boards' HBM2 chips as Chip 1, 2, 3, 4, and 5.

Estimated ages of the tested HBM2 chips. We do *not* know the precise model information details (including the manufacturing date) of the HBM2 chips we test, as the manufacturer does not disclose the specifications for the HBM2 chips in their products [108] and as our custom memory controller does not yet support accessing this information from the mode status registers of the HBM2 chip. We estimate the ages of the tested HBM2 chips based on the dates that we acquired them: Chip0 was 2 years and 9 months, Chip1 was 8 months, and Chip2-5 were 3 months old when we started our experiments.

Temperature Control. We use the temperature controller setup shown in Fig. 2 for Chip 0 and set the target temperature for this chip to 82 °C. Fig. 3 shows how the temperature of each chip varies during 24 hours based on measurements taken every 5 seconds. Even though we do not have the same temperature controller setups for the six chips, we observe that their temperature is stable.[3]

4. Spatial Variation in RowHammer

We provide the first detailed spatial variation analysis of RowHammer across channels, pseudo channels, banks, and rows in six HBM2 chips.

4.1. RowHammer Across Chips

Fig. 4 shows the distribution of *BER* (y-axis) across all tested DRAM rows for each tested data pattern (x-axis) in all tested chips (color-coded). A higher *BER* indicates worse RowHammer vulnerability as more DRAM cells in a row exhibit RowHammer bitflips.

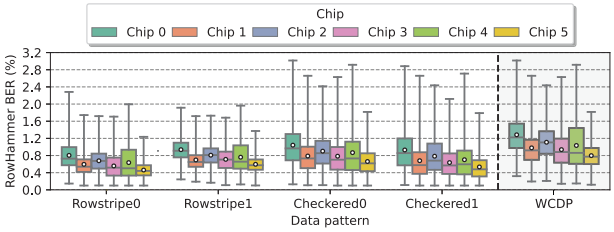

Figure 4: *BER* across different HBM2 chips. Error bars show the range of *BER* across all tested rows in each chip.

Obsv. 1. *There are RowHammer bitflips in all tested DRAM rows in all chips. RowHammer BER varies across chips.*

For example, the *BER* for a DRAM row in Chip 0 can reach up to 3.02% (i.e., 3.02% of all cells in a DRAM row exhibit RowHammer bitflips) and the mean *BER* across all DRAM rows in Chip 0 is 1.04%. In Chip 5, the highest *BER* for a DRAM row is 1.82% and the mean *BER* across all DRAM rows is 0.66%. We observe the most significant difference in mean *BER* across all rows in a chip between Chip 0 (1.28%) and Chip 5 (0.80%) as 0.49 percentage points (pp) for the worst case data pattern (WCDP, see § 3.1).

Obsv. 2. *Data pattern affects RowHammer BER.*

As an example, the Checkered0 and Checkered1 data patterns result in substantially higher mean *BER* across rows for every DRAM chip compared to the Rowstripe0 and Rowstripe1 data patterns. The mean *BER* across all tested DRAM rows (across all chips) is 0.76% and 0.67% for Checkered0/1 and Rowstripe0/1 data patterns, respectively.

Fig. 5 shows the distribution of HC_{first} (y-axis) across all tested DRAM rows for each tested data pattern (x-axis). A lower HC_{first} indicates worse RowHammer vulnerability as

[3] We retrieve the temperature for these five chips from an in-HBM2-chip temperature sensor using the IEEE 1500 test port [77].

DRAM cells exhibit RowHammer bitflips with fewer aggressor row activations (i.e., it takes a shorter time to induce the first RowHammer bitflip in a row).

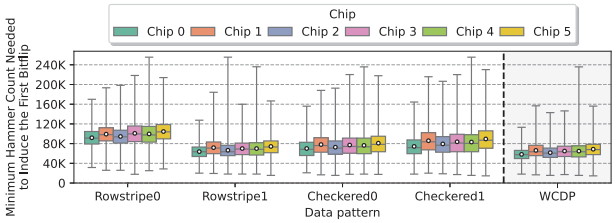

Figure 5: HC_{first} across different chips. Error bars show the range of HC_{first} across all tested rows in each chip.

Obsv. 3. *It takes* only *14531 aggressor row activations to induce a RowHammer bitflip.*

The most RowHammer vulnerable DRAM row across all tested rows has an HC_{first} of 14531. Causing this bitflip in a row in Chip 5 takes us 1.3 milliseconds.

Obsv. 4. *HC_{first} varies across chips and there are rows in every chip that exhibit relatively small HC_{first} values.*

There is variation in HC_{first} distributions between different chips for the same data pattern. For example, the mean HC_{first} value for Chip 5 is 10.59% higher than the mean HC_{first} value for Chip 2 using the Rowstripe0 data pattern. The other tested chips (Chips 0-4) display similar minimum HC_{first} values. It takes 18087, 16611, 15500, 17164, and 15500 aggressor row activations to induce the first RowHammer bitflip in Chips 0-4, respectively.

Takeaway 1. HBM2 chips exhibit different levels of RowHammer vulnerability in terms of mean *BER* (up to 0.49 percentage points difference) and minimum HC_{first} (up to 3556 difference).

4.2. RowHammer Across Channels

Fig. 6 shows the distribution of *BER* (y-axis) across different DRAM rows for a given data pattern (x-axis) in a channel (color-coded).

Obsv. 5. *Each tested DRAM row across all tested HBM2 channels exhibits RowHammer bitflips. The worst-case data pattern (WCDP) BER across all tested rows in all chips can reach 3.02% (i.e., 247 bitflips out of 8192 bits in a row).*

Obsv. 6. *BER varies across channels in a chip.*

For example, in Chip 0, channels CH0 and CH7 exhibit significantly higher *BER* than other channels. CH7 (where we observe the highest mean *BER*) has 1.99× the *BER* of CH3 (where we observe the lowest mean *BER*) for WCDP. We observe that channels can be classified into groups of two based on the number of bitflips they exhibit. We highlight these groups using different shades of the same color in Fig. 6. For example, channels CH3 and CH4 exhibit a similar *BER* distribution across rows in every tested HBM2 chip. We hypothesize that groups of channels are spread across different HBM2 DRAM dies. The difference in *BER* across the groups of channels could be due to process variation as a 3D-stacked chip is typically constructed by stacking DRAM dies that pass functionality and performance testing [109] (the difference in *BER* across groups

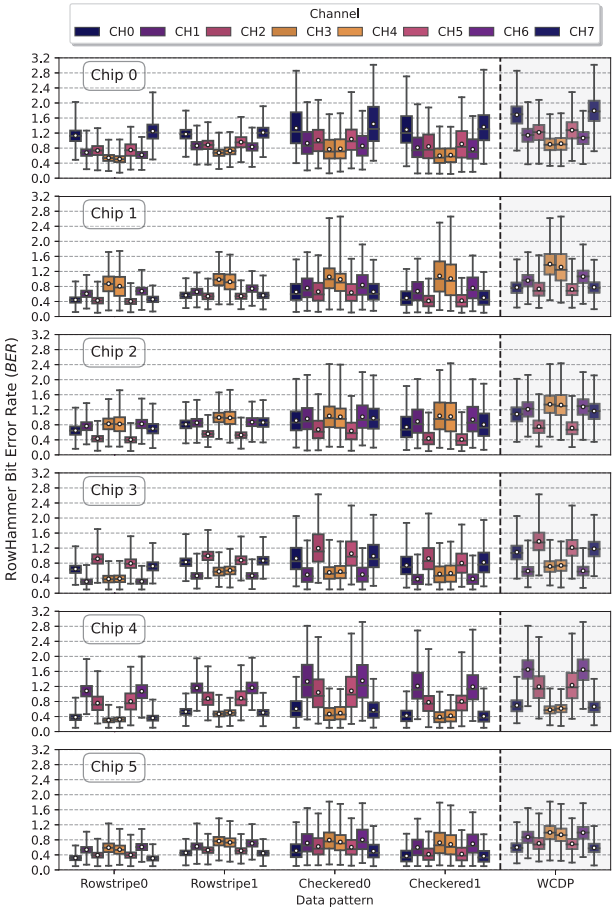

Figure 6: *BER* across different DRAM channels. Error bars show the range of *BER* across rows in each channel.

of two channels is similar to how different DDR3/4 chips exhibit different RowHammer characteristics [1, 68–70, 107, 110, 111]).

Obsv. 7. *The distribution of BER across channels changes from chip to chip.*

The most RowHammer vulnerable channel (i.e., a channel with the highest RowHammer *BER*) is *not* necessarily the same across every chip. For example, channels CH0 and CH7 have the highest average *BER* in Chip 0, whereas channels CH3 and CH4 have the highest average *BER* in Chip 1 for WCDP. We hypothesize that this difference could be due to the effects of process variation across HBM2 chips and dies inside chips.

Obsv. 8. *The mean BER across all rows in each channel has a wider distribution than the mean BER across all rows in each chip.*

For example, the difference between the maximum and the minimum mean *BER* across all rows in each channel in Chip 4 is 0.88 pp, whereas the difference between the maximum and the minimum mean *BER* across all rows in each chip is 0.38 pp (see Fig. 6), for the Checkered0 data pattern. This observation holds for all tested data patterns and chips except Chip 5. Chip 5 has a smaller difference between the maximum and the minimum mean *BER* across all rows in each channel than the difference across all rows in each chip. Manufacturing

process variation inherent to 3D die stacking process [109] could explain this observation. 3D die stacking could alter the RowHammer vulnerability characteristics (e.g., *BER*) of DRAM dies that make up a chip stack. We hypothesize that this alteration is made in a way that widens the *BER* distribution across 3D-stacked dies in an HBM2 chip.

Fig. 7 shows the distribution of HC_{first} (y-axis) across different DRAM rows for a given data pattern (x-axis) in a channel (color-coded).

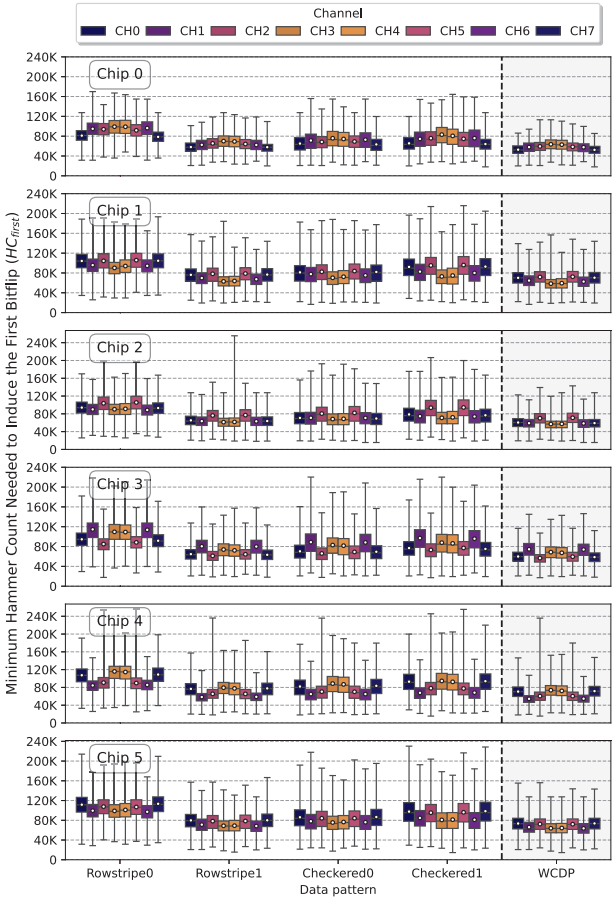

Figure 7: HC_{first} **across different DRAM channels.**

Obsv. 9. *Different channels exhibit different HC_{first} distributions. These distributions are affected by the data pattern.*

For example, channels CH3 and CH4 in chip 1 contain more rows with smaller HC_{first} values than other channels. Because these channels also exhibit more RowHammer bitflips than other channels in chip 1 (see Fig. 6), we hypothesize that these channels belong to the die with the worst RowHammer vulnerability across all dies. The median HC_{first} for Rowstripe0 and Rowstripe1 in channel CH0 in chip 1 are 103905 and 75990, respectively. Testing with different data patterns is necessary to assess the RowHammer vulnerability of an HBM2 DRAM chip, as no data pattern individually achieves the smallest HC_{first} or the highest *BER* (Fig. 6).

Takeaway 2. RowHammer *BER* and HC_{first} vary between different 3D-stacked channels in a chip and with the data patterns in aggressor and victim rows. The mean *BER* across all rows in each channel has a wider distribution than the mean *BER* across all rows in each chip.

Fig. 8 shows the *BER* over each DRAM row in a bank when we use the worst-case data pattern (WCDP) to initialize the rows. Each color-coded *BER* curve represents the *BER* for rows in three different channels. The shaded regions indicate variable-sized *subarray* boundaries.[4] For example, the green highlighted subarray (SA X) comprises 832, and the blue highlighted subarray (SA Y) comprises 768 DRAM rows.

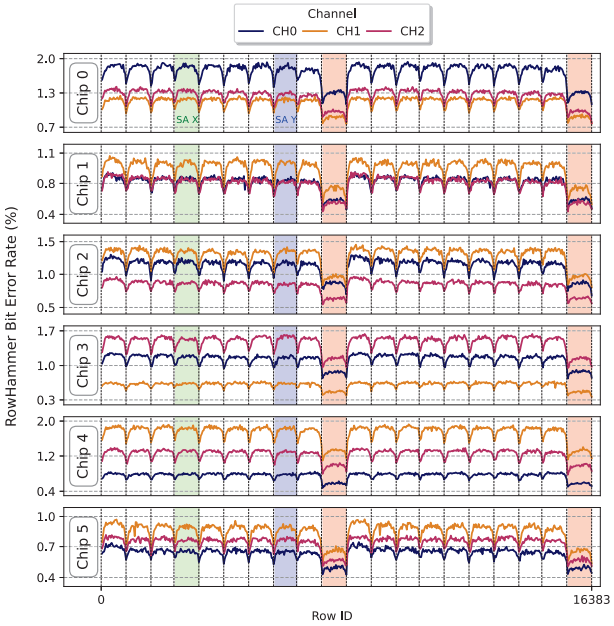

Figure 8: *BER* **for different rows across a bank in different channels. Dashed vertical lines indicate subarray boundaries.**

Obsv. 10. *BER periodically increases and decreases across DRAM rows.*

This observation is consistent in each tested chip. *BER* is higher in the middle of a *subarray* and lower towards either end. We hypothesize that the increasing and decreasing pattern results from the structure of the local DRAM array. For example, the RowHammer vulnerability of a row could increase with the row's distance from the row buffer.

Obsv. 11. *The last and the middle subarrays in a bank exhibit relatively low BER.*

The last and the middle subarrays in a bank (highlighted with red color in Fig. 8), which contain the last and the middle 832 DRAM rows, exhibit significantly lower BER than the other subarrays. We hypothesize that these subarrays exhibit smaller *BER* due to the micro-architectural characteristics of the DRAM bank. First, assuming that proximity to the shared I/O circuitry on the DRAM die affects the RowHammer vulnerability of a

[4] We reverse engineer subarray boundaries by performing single-sided RowHammer [1, 68] that induces bitflips in *only one* of the victim rows if the aggressor row is at the edge of a subarray. We find that a subarray contains either 832 (SA X in Fig. 8) or 768 (SA Y in Fig. 8) DRAM rows.

subarray, the last and the middle subarrays might be placed near this shared I/O circuitry [112]. Second, an edge subarray (a subarray at either edge of a bank) typically harbors different design characteristics than other subarrays [113, 114]. Edge subarrays (e.g., the last and the middle subarrays) in the tested chips could exhibit a design characteristic that results in their DRAM wordlines to be driven to a smaller wordline voltage (V_{PP}) than other subarrays' wordlines', which could explain the relatively low *BER* values in the middle and last subarrays [69]. Third, the tested data pattern may not be replicated in the same way in edge subarrays due to inaccessible "dummy bitlines" [113] and thus the data pattern in the last and middle subarrays could be inducing a smaller read disturbance effect.

Takeaway 3. A subset of HBM2 rows (the middle and the last 832 rows) are significantly more RowHammer resilient than the other rows in an HBM2 channel.

4.3. RowHammer Across Banks and Pseudo Channels

To investigate the variation in the RowHammer vulnerability across HBM2 banks and pseudo channels, we measure *BER* on 300 rows from 256 banks in chip 0.[5] Fig. 9 compares different banks' *BER* distributions across channels (color) and pseudo channels (marker style) in terms of the coefficient of variation (CV)[6] (x-axis) and mean BER (y-axis). We draw one marker for each bank. At a high level, a marker close to the y-axis (e.g., the leftmost markers) indicates that the variation in *BER* across rows in that bank is smaller. A marker close to the x-axis (e.g., the bottommost markers) suggests that the mean *BER* across rows in that bank is smaller.

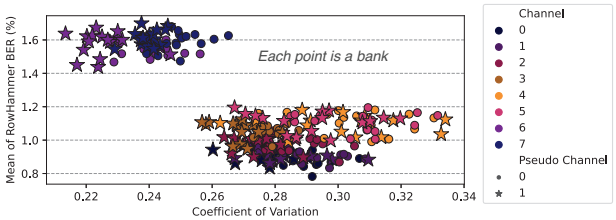

Figure 9: *BER* variation across banks. Each bank is represented by the average BER (y-axis) and the coefficient of variation in BER (x-axis) across all rows within the bank.

Obsv. 12. *RowHammer BER varies across banks and pseudo channels. The BER variation across banks is dominated by variation across channels.*

For example, there is up to 0.23 pp difference in mean *BER* across banks in channel 7. The markers follow a bimodal distribution (i.e., the markers are clustered around two points in the plot). The two clusters indicate that 1) a bank with a higher mean *BER* across its rows also has a smaller deviation (i.e., smaller coefficient of variation) from the average *BER* across its rows and 2) a bank with a smaller mean *BER* across its rows also has a larger deviation from the average *BER* across its rows. Banks in different channels tend to have a larger *BER* difference than banks in the same channel (Fig. 6), indicating that testing different channels is more important than testing different banks or pseudo channels in providing a comprehensive understanding of the RowHammer vulnerability in HBM2 DRAM chips.

Takeaway 4. RowHammer *BER* varies across pseudo channels and banks. This variation is less prominent than the *BER* variation between channels.

4.4. The Effect of Aging

A DRAM row's RowHammer vulnerability can change over time. A rigorous characterization study on many HBM2 chips over a large timespan is required to comprehensively demonstrate the effects of aging. Due to time and space limitations, we leave such a study for future work while presenting a preliminary analysis using four HBM2 chips (Chips 2-5, which have the same estimated age) as our best effort. We repeat our *BER* experiments for 3072 rows in 3 channels after 7 months of keeping the HBM2 chips powered on. Fig. 10 (left) shows the distribution of a row's *BER* after aging (New *BER*) over its *BER* before aging (Old *BER*) for the Checkered1 data pattern. Fig. 10 (right) shows the distribution of a row's *BER* before aging over its *BER* after aging. The left subplot shows only the distribution for the rows whose *BER*s increase after aging and the right subplot shows the distribution for the rest of the rows. The x-axis is marked with percentiles ranging from P1 to P99.

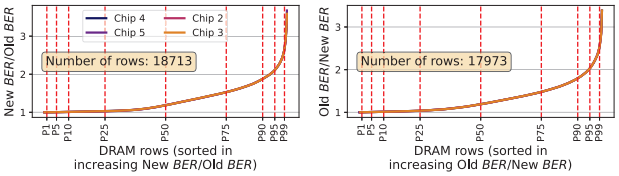

Figure 10: Distribution of the deviation of *BER* after aging from *BER* before aging across tested DRAM rows.

Obsv. 13. *The BER of a DRAM row changes after aging. A larger fraction of the tested DRAM rows have higher BER after aging.*

18713 and 17973 of the tested DRAM rows have a higher New *BER* and a lower New *BER*, respectively (we omit 178 outlier rows from the figure). We observe similar distributions for all tested HBM2 chips.

5. RowHammer's Sensitivity to Hammer Count

We analyze the number of aggressor row activations (hammer count) needed to induce up to 10 bitflips in a DRAM row. Our HC_{first} analysis (§ 4) already examined the hammer count to induce one (*the first*) bitflip in a row. We use the same naming convention used with HC_{first} to refer to these 9 new hammer counts that we determine in this analysis. For example, we call the hammer count to induce *the second* bitflip HC_{second} and *the tenth* bitflip HC_{tenth}. We report each of the 9 new hammer counts as a value normalized to HC_{first}. For example, if a row's HC_{first} is 10 and its HC_{second} normalized to HC_{first} is 2, the absolute HC_{second} of the row is 20. We record the hammer counts to induce up to 10 bitflips in 32 rows from each of the beginning, middle, and end of one bank in two channels (that exhibit the smallest HC_{first} across all channels) in every HBM2 chip. Fig. 11 plots the distribution of all 10 hammer counts

[5] First, middle, and last 100 rows in each of the 256 banks spread across eight channels and two pseudo channels.

[6] Coefficient of variation is the standard deviation of a distribution normalized to the mean.

(x-axis) normalized to HC_{first} (y-axis) for all tested DRAM rows (1152 such rows across all chips).

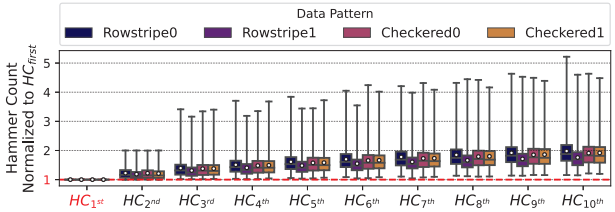

Figure 11: Distribution of hammer counts (y-axis) to induce up to 10 bitflips in a DRAM row (x-axis), normalized to the row's HC_{first}.

Obsv. 14. *Hammer count needed to induce up to 10 bitflips in a row significantly varies between rows.*

The hammer count to induce up to 10 bitflips in a row can be as small as 1.15× and as large as 5.22× the HC_{first} of the DRAM row. Fewer than 2× HC_{first} hammers are enough to induce 10 bitflips in a DRAM row on average across all tested DRAM rows. For example, an average DRAM row's HC_{second}, HC_{fourth}, HC_{eighth}, and HC_{tenth} are 1.19×, 1.41×, 1.66×, and 1.76× that of the row's HC_{first} for the Rowstripe1 data pattern.

Obsv. 15. *The hammer counts to induce up to 10 bitflips are moderately affected by data patterns.*

For example, the difference between the largest (Rowstripe0) and the smallest (Rowstripe1) mean normalized HC_{tenth} is 12.59%. The variation in normalized hammer count across data patterns resembles the variation in HC_{first} across data patterns (see Fig. 5).

Fig. 12 plots the additional hammer count (over HC_{first}) to induce the 10th bitflip (y-axis) for all tested DRAM rows whose HC_{first} values are depicted on the x-axis. We compute the additional hammer count for a row as the row's $HC_{tenth} - HC_{first}$. Fig. 12 shows one subplot for each of the 6 tested HBM2 chips.

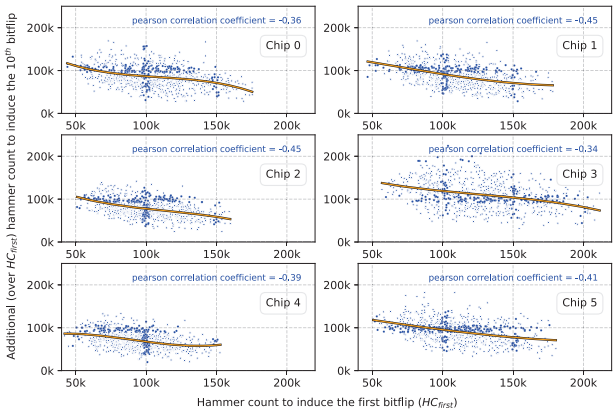

Figure 12: Additional (over HC_{first}, x-axis) hammer count needed to induce the tenth bitflip (y-axis) for each tested DRAM row in each tested chip (labeled Chip 0 to Chip 5 in the figure). We plot a polynomial curve fit (orange curve) for each distribution to highlight the decreasing additional hammer count trend with increasing HC_{first}.

Obsv. 16. *It takes fewer additional hammer counts (over HC_{first}) to induce the 10th RowHammer bitflip for a DRAM row with a large HC_{first} compared to a DRAM row with a small HC_{first}.*

We observe that increasing HC_{first} is correlated with decreasing *additional* hammer count to induce the 10th bitflip. We compute the Pearson correlation coefficient for each distribution to quantify the correlation. We conclude that increasing HC_{first} is *moderately* correlated with decreasing additional hammer count to induce the 10th bitflip, based on the weakest (−0.34) and the strongest (−0.45) Pearson correlation we observe across distributions for each chip (displayed on each subplot).

Takeaway 5. It can take fewer aggressor row activations to induce multiple (e.g., 10) bitflips in a DRAM row if it takes many activations to induce the first bitflip in the row.

Fig. 13 shows the distribution of the maximum change of HC_{first} from the minimum observed HC_{first} for a DRAM row over 50 experiment iterations using the Rowstripe0 data pattern. The x-axis shows all tested rows (768 rows from channel 0 in every tested HBM2 chip), sorted by increasing maximum change of HC_{first} and marked with percentiles ranging from P1 to P99.[7]

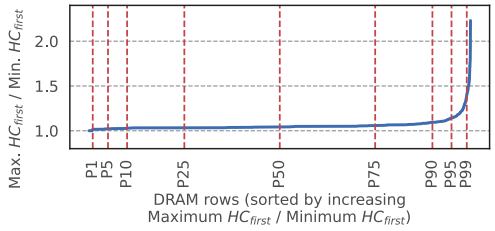

Figure 13: Distribution of the maximum observed HC_{first} over the minimum observed HC_{first} for a DRAM row across experiments, across all tested 4608 DRAM rows for the Rowstripe0 data pattern.

We make two key observations. First, the HC_{first} of a DRAM row can significantly change from the minimum observed HC_{first} across experiment iterations (i.e., over time). The greatest change that we observe for a row is 2.23× (the row's maximum observed HC_{first} is 2.23× of its minimum observed HC_{first}). Second, the majority of rows experience a small HC_{first} change. For example, 90% of all tested rows experience an HC_{first} change smaller than 1.09×.

6. RowHammer and RowPress's Sensitivities to Aggressor Row On Time

With aggressive technology node scaling, DRAM suffers from worsening read disturbance effects. One prominent example of read disturbance in DRAM is RowHammer, which we have already extensively characterized in our HBM2 chips. We also investigate the characteristics of another widespread read disturbance effect called RowPress, recently experimentally demonstrated in real DDR4 chips [4]. RowPress is the phenomenon that keeping an aggressor row open for a long

[7] To maintain a reasonable testing time, we investigate the maximum change of HC_{first} using one data pattern and 4608 DRAM rows. We leave the detailed analysis of the maximum change of HC_{first} that covers more data patterns, HBM2 components (e.g., channels, pseudo channels, banks, rows), and access patterns for future work.

period of time (i.e., a large aggressor row on time, t_{AggON}) induces bitflips in physically nearby DRAM rows with orders of magnitude smaller hammer counts compared to a traditional RowHammer access pattern which keeps the aggressor row open for a short period of time. We provide the first extensive analysis of RowPress in six real HBM2 chips.

Fig. 14 depicts how *BER* (y-axis) varies with increasing t_{AggON} (x-axis) across the first, middle, and last 128 rows in one DRAM bank for 8 channels when we use a hammer count of 150K (i.e., activate each aggressor row 150K times) and the Checkered0 data pattern. We plot the results for four relatively small (left subplots) and two relatively large (right subplots) t_{AggON} values. The minimum t_{AggON} is the t_{RAS} timing parameter [77, 85]. We choose two relatively large t_{AggON} values of interest as t_{REFI}, the average interval between two successive periodic refresh commands, and as $9 * t_{REFI}$, the maximum interval between two subsequent periodic refresh commands (i.e., the maximum time a row can remain open according to the HBM2 standard) [77].[8]

t_{AggON} in commodity systems. In general, the time a DRAM row is kept open depends on two factors: 1) t_{REFI} in the DRAM specification, and 2) the memory request scheduling algorithm implemented in the memory controller. For HBM2, the default t_{REFI} is $3.9\mu s$. The memory controller in a commodity CPU or a GPU may delay up to 8 periodic refresh commands and thereby keep a row open for as long as 9*t_{REFI} ($35.1\mu s$). The memory request scheduling algorithm is typically *not* disclosed by system manufacturers. However, it is reasonable to assume that a high-performance scheduling algorithm implemented in a commodity CPU or GPU's memory controller would keep a DRAM row open at least for as long as required to serve a regular memory access pattern that "streams" through a row (i.e., accesses every cache line in the row one by one) [115–118], to exploit row buffer locality and thus maximize memory throughput. We estimate the time it takes to stream through a row as 128.0*ns* (32*$tCCD_L$) based on the HBM2 timing parameter values depicted in [85].

Obsv. 17. *RowPress BER in each chip consistently increases with t_{AggON} in all tested channels.*

The average *BER* across every channel in every chip is 0.08%, 0.24%, 0.40%, 0.73%, 31.00%, and 50.35% at t_{AggON} of 29.0 ns, 58.0 ns, 87.0 ns, 116.0 ns, 3.9 µs, and 35.1 µs, respectively.

Obsv. 18. *Channels with high BER at low t_{AggON} values tend to have high BER also at high t_{AggON} values.*

For example, CH 1 in Chip 3 has the highest mean *BER* across all tested t_{AggON} values. We observe that all *BER* values converge to around 50% for the t_{AggON} of 35.1 µs across all tested channels. We hypothesize that this is due to the combined effect of 1) the data pattern that we use which initializes a victim DRAM row with alternating *logic-1* and *logic-0* (i.e., 10101010...) and 2) RowPress causing bitflips from *logic-1* to *logic-0* more frequently than from *logic-0* to *logic-1* (as demonstrated in [4] for a wide variety of DDR4 chips).

[8] t_{AggON} values above 116.0 ns combined with the 150*K* hammer count results in experiment times that are longer than the 32 ms refresh window (e.g., 150*K* hammers at $t_{AggON} = 35.1\mu s$ takes 10.53 seconds), where DRAM cells can exhibit *retention failures*. Because we want to analyze *only* the read disturbance bitflips, we remove bitflips that are caused by retention failures (0.109% of all bitflips we observe) from the set of bitflips we observe at every such t_{AggON}. We omit the details of our retention failure characterization methodology due to page limits.

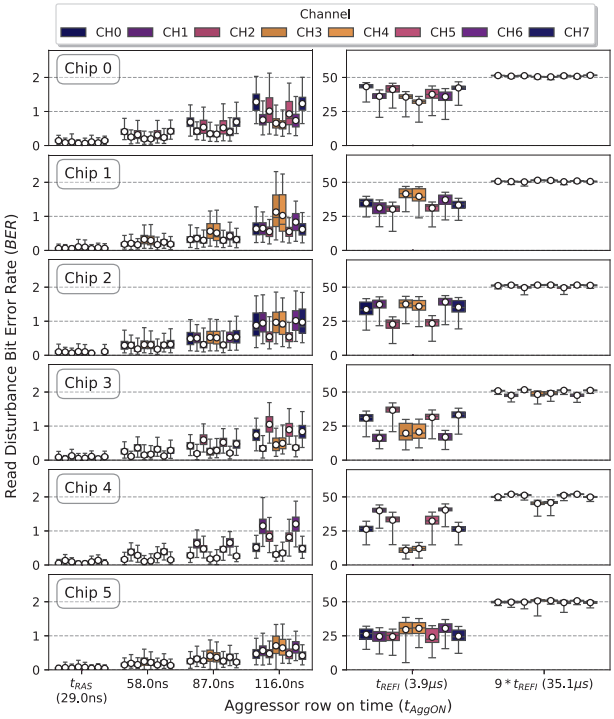

Figure 14: *BER* **with increasing** t_{AggON}**.**

Fig. 15 depicts how the HC_{first} values (y-axis) across 384 tested DRAM rows in a channel change with increasing t_{AggON} (x-axis) for 3 channels when we use the Checkered0 data pattern. From left to right, 1) the default t_{RAS} value (29.0 ns), 2) t_{REFI}, the average time interval between two successive periodic refresh commands, 3) $9 * t_{REFI}$, the maximum time a row can remain open according to the HBM2 specification, and 4) half the refresh window (t_{REFW}) [77] (such that each aggressor row can be activated once in a t_{REFW}). Fig. 15 shows one subplot for every tested HBM2 chip. The grey boxes indicate the number of rows we show in each subplot. We only show rows for which we observe the first read-disturb bitflip in a refresh window (under 32 ms) at every tested t_{AggON} value. We display the results for four t_{AggON} values on the x-axis.

Obsv. 19. *As the aggressor row remains open longer (i.e., as t_{AggON} increases), DRAM rows experience bitflips at smaller hammer counts.*

This observation is consistent across the three tested HBM2 channels. The average (minimum) HC_{first} values across all chips are 83689 (29183), 1519 (335), 376 (123), and 1 (1), for the four tested t_{AggON} values, respectively.

Takeaway 6. The read disturbance vulnerability of tested HBM2 chips worsens (i.e., *BER* increases and HC_{first} reduces) with increasing t_{AggON}.

Our observations are in line with prior works that investigate the effects of t_{AggON} in real DDR4 chips [4, 70].

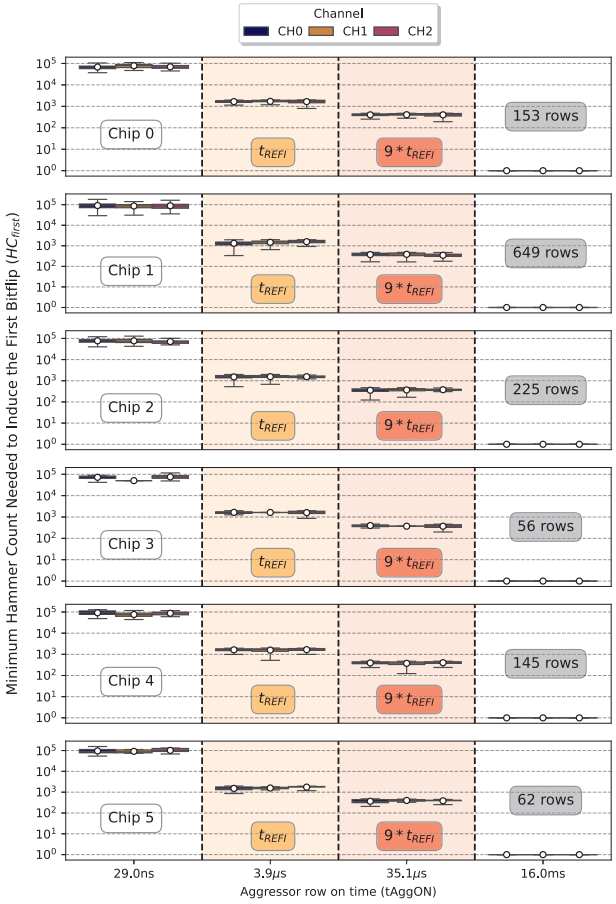

Figure 15: HC_{first} **with increasing** t_{AggON}**.**

7. In-DRAM RowHammer Defenses

To prevent RowHammer bitflips, DRAM manufacturers equip their chips with a mitigation mechanism broadly referred to as Target Row Refresh (TRR) [38, 44, 83]. Unfortunately, manufacturers do *not* disclose the operational principles or implementations of proprietary TRR versions (e.g., in DDR4). At a high level, TRR 1) identifies potential aggressor rows as the memory controller issues activate commands to the DRAM chip and 2) preventively refreshes their victim rows when the memory controller issues a periodic *REF* command [38, 44].

We demonstrate that a tested HBM2 chip (Chip 0) implements a form of proprietary TRR (similar to the ones used in DDR4).[2] We analyze the TRR mechanism and craft a specialized access pattern that bypasses the TRR mechanism and induces RowHammer bitflips.[9]

Methodology. We use U-TRR's [44] methodology to uncover the proprietary TRR mechanism. The key idea of this methodology is to use retention failures as a side channel to infer whether or *not* TRR refreshes a DRAM row. Our analysis consists of two steps. First, we identify multiple DRAM rows with similar retention times (e.g., two rows that can correctly retain data when they are *not* refreshed for the same amount of time) by profiling DRAM rows for their retention times. We test all of the DRAM rows in bank 0 for retention failures starting with a retention time of 64 ms with increments of 64 ms. We deem a row to have a retention time of T if any of the DRAM cells in the row exhibit a bitflip at a retention time of T. We find the smallest T that causes at least one retention bitflip for each tested row. Second, to understand if the TRR mechanism samples an aggressor row, we execute a four-step process: 1) We initialize DRAM rows that have a retention time of T (we call these rows *side-channel rows*) and wait for T/2 *without* refreshing these rows. 2) We activate each of the DRAM rows adjacent to the side-channel rows *once*. We hypothesize that the TRR mechanism samples an activation to an adjacent row as an aggressor row activation. 3) We issue a *REF* command to trigger the TRR mechanism. If the TRR mechanism samples any of the activated adjacent rows in step 2, we expect the TRR mechanism to refresh the side-channel rows. 4) We wait for T/2, read the data in the side-channel rows and check for any retention bitflips. The side-channel rows exhibit retention bitflips *only* if they are *not* refreshed by the TRR mechanism. We use this information to understand how the TRR mechanism works.

We repeat our experiment with various carefully crafted RowHammer access patterns to understand how the TRR mechanism tracks the aggressor rows.

Obsv. 20. *Every* 17th *REF command can perform a TRR victim row refresh (i.e., every* 17th *REF command is TRR-capable).*

Obsv. 21. *The TRR mechanism refreshes both of the adjacent rows of an aggressor row.*

If TRR identifies row R as an aggressor row, it refreshes rows R+1 and R-1. Obs. 20 & 21 resemble the TRR mechanism employed in real DDR4 chips from Vendor C in U-TRR [44].

Obsv. 22. *The first row that gets activated after a TRR-capable REF is* always *identified as an aggressor row by the TRR mechanism.*

Obsv. 23. *The TRR mechanism records the activation count of activated rows and uses this record to identify if a row is an aggressor row.*

Between two *REF* commands, if a row is activated more than half the total activation count, TRR identifies that row as an aggressor row. For example, if we issue 10 activations between two *REF* commands, the row that receives the first *ACT* command and the row that receives 5 *ACT* commands are identified by the TRR mechanism. We compare our findings on the HBM2 chip's TRR mechanism to U-TRR's findings [44] in DDR4 chips. U-TRR does *not* make similar observations to Observations 22 and 23. Thus, the HBM2 chip likely implements a previously-unknown type of TRR.

Takeaway 7. An HBM2 chip implements a proprietary TRR mechanism that tracks aggressor rows and proactively refreshes their victim rows.

Bypassing the proprietary TRR mechanism. Based on our observations, we craft a specialized access pattern that bypasses the TRR mechanism and causes RowHammer bitflips. We calculate the total activation count budget (i.e., the maximum number of *ACT* commands that the memory controller can issue)

[9] We note that the profiling required to perform system-level RowHammer attacks [1, 5–67] is *not* as extensive as what we do in this work and can be done using user-level programs, as shown in various works (e.g., [38, 43, 45]).

between two REF commands as $\lfloor (tREFI - tRFC)/tRC \rfloor = 78$ [77, 85] for the tested HBM2 chip and fully utilize the activation count budget in our access pattern. The key idea of this access pattern is to trick TRR into *not* identifying a *real* aggressor row by repeatedly accessing multiple *dummy* rows many times. The access pattern 1) activates dummy rows (we vary the number of dummy rows and the activation count of the dummy rows) and 2) performs double-sided RowHammer using two real aggressor rows. We create this access pattern such that the number of real aggressor activations does *not* exceed half of the total activation count budget. We repeat the access pattern $8205 * 2$ times (i.e., approximately for two $tREFW$, 64 ms) for each DRAM row in a bank so that we activate aggressor rows as many times as possible before each DRAM cell is refreshed.

We test all rows in a bank of the HBM2 chip using this access pattern while obeying manufacturer-recommended timing parameters (e.g., we issue a REF command every $tREFI$, $3.9\mu s$). Fig. 16 shows the distribution of BER for different numbers of dummy rows (x-axis) and aggressor activation counts (different boxes). Since we have a total activation count budget of 78 and we utilize the whole budget for aggressor row and dummy row activations, the number of dummy row activations varies between boxes displayed on the plot. For example, for 4 dummy rows and an aggressor activation count of 18 (leftmost box in Fig. 16), each dummy row is activated $\lfloor (78 - 18 * 2)/4 \rfloor = 10$ times.

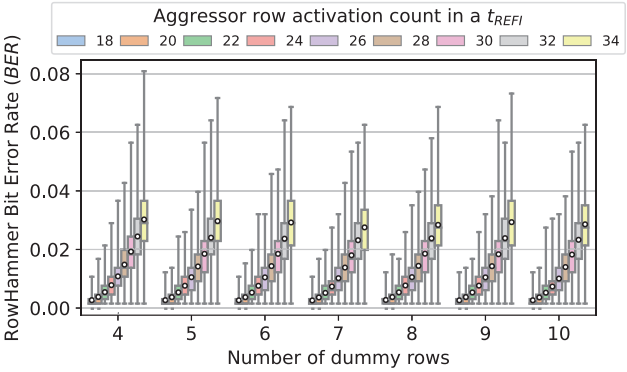

Figure 16: RowHammer *BER* distribution across all tested rows for the specialized access pattern with different number of dummy rows (x-axis) and different aggressor row activation counts (different colored boxes).

We make four key observations from Fig. 16. First, our specialized access patterns can induce RowHammer errors with a reasonably high BER. Second, the access pattern needs to activate at least 4 dummy rows to bypass the TRR mechanism (i.e., BER is 0 for x = 1, 2, and 3). Third, the number of dummy rows does *not* significantly affect the distribution of the bit error rate. For example, the mean bit error rate varies by 0.003 pp between the largest (4 dummy rows) and the smallest (7 dummy rows) value at an aggressor activation count of 34. Fourth, the number of bitflips per row increases as the aggressor activation count increases. For example, the mean bit error rate increases by 2.79×, 6.72×, and 10.28× as the aggressor activation count increases from 18 to 24, 30, and 34, respectively, when the number of dummy rows is 8.

Takeaway 8. A specialized access pattern that bypasses the undocumented TRR mechanism in HBM2 chips can induce many RowHammer bitflips.

8. Implications on Future Read Disturbance Attacks and Defenses

We describe and analyze the important implications of our observations on future read disturbance attacks and defenses.

8.1. Read Disturbance Attacks

Observation 5 and Takeaways 2, 6, and 7 have the following four implications for future read disturbance attacks on HBM2 chips. First, the maximum BER (247 bitflips in a row of 8192 bits) we observe across tested chips exceeds the correction capabilities of widely used error correcting codes (ECC), such as SECDED [119–122] which can detect two bitflips and correct one bitflip in a codeword (Observation 5).[10] 247 bitflips in a row are already sufficient to induce uncorrectable bitflips in multiple 64-bit words in the same DRAM row (by pigeonhole principle) and are likely to induce undetectable bitflips in at least one 64-bit word.

Analysis of the effectiveness of ECC. We investigate the distribution of word-level RowHammer bitflips (i.e., bitflips that occur in all non-overlapping consecutive 64 bits) in Chip 4. Fig. 17 plots the number of words that exhibit at least one bitflip (out of all 18M tested words) on the y-axis over 1, 2, ..., and more than 7 bitflips in a word depicted on the x-axis for different data patterns (different bars). The word counts across clusters of bars are non-overlapping; for example, the middle cluster depicts the number of words with exactly two bitflips.

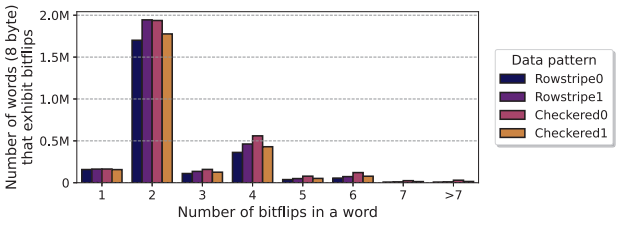

Figure 17: Number of words (y-axis) that exhibit 1, 2, ..., and more than 7 bitflips (x-axis) in Chip 4.

We make two observations. First, the number of unique words with more than two bitflips is large. We observe 974'935 words with more than two bitflips when we use the Checkered0 data pattern. These bitflips would *not* be detected by SECDED ECC. Second, most words with at least one bitflip actually have more than one bitflip. In other words, if RowHammer bitflips can be induced in a word, it is very likely that more than one bit location in the word will experience errors. Simple SECDED ECC *cannot* correct such bitflips.

We find that an HBM word can have 16 bitflips (not shown in the figure) in Chip 4. A (7,4) Hamming code [126] could

[10] Single symbol correcting codes (e.g., Chipkill) [123–125] are widely used in DDRx server systems. A prior work [121] proposes implementing a Chipkill-like scheme for HBM3 where 1) the HBM3 chip is carefully architected to ensure a hard fault in a DRAM component manifests as errors *only* within an *isolation boundary* (e.g., 16 consecutive bits), 2) a single symbol correcting code is used where each symbol corresponds to an isolation boundary. Such a scheme *alone* is likely *not* a good read disturbance countermeasure as read disturbance errors typically appear in multiple isolation boundaries.

correct such bitflips at very large DRAM storage overheads (75%, 3 parity bits for every 4 data bits). Thus, relying on ECC alone to prevent RowHammer bitflips in HBM2 is a very expensive solution. The bitflip distributions indicate that attackers could exploit RowHammer bitflips to escalate privilege and leak security-critical and secret data in HBM2 chips, similarly to real RowHammer attacks on DDR4-based computing systems. Even if an HBM2 chip is highly RowHammer resilient (i.e., has small mean *BER* across its rows), malicious parties could exploit RowHammer bitflips to practically increase the rate of correctable or detectable bitflips. This could reduce the lifetime of modern HBM-based systems (e.g., GPUs) by accelerating the rate of memory page retirements [127] beyond design-time estimates and exacerbate system maintenance costs.

Second, a RowHammer attack could target the most-RowHammer-vulnerable HBM2 channel to reduce the time it spends on i) *preparing* for an attack, by finding exploitable RowHammer bitflips faster (i.e., by accelerating memory templating), and ii) *performing* the attack, by benefitting from a small HC_{first} value (Takeaway 2). Third, an attacker could keep the aggressor row on for longer by executing specialized access patterns (as demonstrated by RowPress in a real DDR4-based system [4]) to benefit from the increased *BER* and reduced HC_{first} when an aggressor row is kept open for longer (Takeaway 6). Fourth, the RowHammer attack must uncover and take into account the functionality of the undocumented TRR mechanism in addition to the functionality of the documented TRR mode [77] to come up with an effective access pattern that bypasses all RowHammer defense mechanisms in an HBM2-based system (Takeaway 7). However, the attackers can also benefit from the undocumented TRR mechanism. Victim row refresh operations could be used as a near aggressor row activation, carrying over the read disturbance effects of the far aggressor row to the victim row in a HalfDouble access pattern [47].

8.2. Read Disturbance Defenses

Takeaways 2, 3, and 8 have the following two implications for future read disturbance defenses on HBM2 chips. First, a defense mechanism can adapt to the heterogeneous distribution of the RowHammer and RowPress vulnerabilities across channels and subarrays, which may allow the defense mechanism to more efficiently prevent read disturbance bitflips (Takeaways 2 and 3). Second, HBM2 memory controller designers likely need to implement other read disturbance defense mechanisms (e.g., [1,3,24,31,38,44,105,128–190]) in their designs because designers cannot rely on the undocumented TRR mechanism to mitigate read disturbance bitflips, as it is easily bypassed with a specialized RowHammer access pattern (Takeaway 8).

9. Related Work

We present the first detailed experimental characterization of the read disturbance vulnerability (RowHammer and RowPress) in modern HBM2 DRAM chips.

HBM2 RowHammer Characterization [113, 191]. A prior work [191] experimentally characterizes the RowHammer vulnerability in an HBM2 DRAM chip. Another work [113] studies the internal DRAM structure by analyzing RowHammer error characteristics of two HBM2 chips. Our work presents a detailed experimental characterization of both RowHammer and RowPress using six HBM2 chips. In addition to analyzing the spatial variation of read disturbance vulnerability, we analyze the hammer counts needed to induce the first 10 bitflips in a row. We uncover entirely the inner workings of the read disturbance defense mechanism in an HBM2 chip and demonstrate RowHammer access patterns that bypass this defense mechanism. Our new analyses and results lead to completely new observations (Observations 2, 4, 7, 8, 11, 13-23) and takeaways (Takeaway 1, 2, 5-8) that [191] and [113] do *not* contain.

(LP)DDR3/4 Read Disturbance Characterization [1, 4, 68–70,107,110,111,192–196]. These works experimentally demonstrate and analyze new aspects of the read disturbance vulnerability by testing real (LP)DDR3/4 DRAM chips. They do *not* experimentally analyze real HBM2 chips.

Besides demonstrating the interaction between the read disturbance vulnerability of an HBM2 chip with unique HBM characteristics for the first time, we make new observations (Observations 10, 11, 13, 14, 15, 16) that could also shed light on the read disturbance vulnerability behavior in (LP)DDRx chips. For example, our observation of the RowHammer bit error rate peaking towards the middle of a subarray (Observation 10) could be widespread across many different types of DRAM chips. § 8 highlights the importance of our new observations in the form of several key implications for future RowHammer attacks and defenses.

Other HBM2 Characterization [197–199]. These works characterize real HBM2 chips to understand their 1) soft error resilience characteristics by high-energy neutron beam testing [199], 2) performance and reliability characteristics under reduced supply voltage [197], and 3) data retention characteristics [198].

10. Conclusion

We present the results of our detailed characterization study on the read disturbance (RowHammer and RowPress) vulnerability in six modern HBM2 chips. Our study leads to 23 observations and 8 takeaways, which we believe have important implications for future read disturbance attacks and defenses. We uncover the inner workings of the proprietary read disturbance mitigation mechanism implemented in an HBM2 chip and develop a practical access pattern that bypasses it and induces read disturbance bitflips. We hope and expect that our findings will lead to a deeper understanding of and new solutions to the read disturbance vulnerabilities in HBM-based systems.

Acknowledgements

We thank the anonymous reviewers of HPCA 2024 and DSN 2024 for feedback and the SAFARI Research Group members for constructive feedback and the stimulating intellectual environment. We acknowledge the generous gift funding provided by our industrial partners (especially Google, Huawei, Intel, Microsoft), which has been instrumental in enabling the research we have been conducting on read disturbance in DRAM in particular and memory systems in general. This work was in part supported by the Google Security and Privacy Research Award and the Microsoft Swiss Joint Research Center.

References

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

[2] O. Mutlu and J. Kim, "RowHammer: A Retrospective," *IEEE TCAD Special Issue on Top Picks in Hardware and Embedded Security*, 2019.

[3] O. Mutlu, A. Olgun, and A. G. Yaglikci, "Fundamentally Understanding and Solving RowHammer," in *ASP-DAC*, 2023.

[4] H. Luo, A. Olgun, A. G. Yağlıkcı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *ISCA*, 2023.

[5] A. P. Fournaris, L. Pocero Fraile, and O. Koufopavlou, "Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks," *Electronics*, 2017.

[6] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking Deterministic Signature Schemes using Fault Attacks," in *EuroS&P*, 2018.

[7] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks Over the Network and Defenses," in *USENIX ATC*, 2018.

[8] S. Carre, M. Desjardins, A. Facon, and S. Guilley, "OpenSSL Bellcore's Protection Helps Fault Attack," in *DSD*, 2018.

[9] A. Barenghi, L. Breveglieri, N. Izzo, and G. Pelosi, "Software-only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks," in *IVSW*, 2018.

[10] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ASHES*, 2018.

[11] S. Bhattacharya and D. Mukhopadhyay, "Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug," in *Fault Tolerant Architectures for Cryptography and Hardware Security*, 2018.

[12] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.

[13] SAFARI Research Group, "RowHammer — GitHub Repository," https://github.com/CMU-SAFARI/rowhammer, 2014.

[14] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," *Black Hat*, 2015.

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

[16] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript," in *DIMVA*, 2016.

[17] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.

[18] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.

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

[20] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector," in *S&P*, 2016.

[21] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of RowHammer: Flipping Secret Exponent Bits using Timing Analysis," in *CHES*, 2016.

[22] W. Burleson, O. Mutlu, and M. Tiwari, "Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer," in *DAC*, 2016.

[23] R. Qiao *et al.*, "A New Approach for RowHammer Attacks," in *HOST*, 2016.

[24] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory," in *USENIX Security*, 2017.

[25] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SysTEX*, 2017.

[26] M. T. Aga, Z. B. Aweke, and T. Austin, "When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks," in *HOST*, 2017.

[27] O. Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," in *DATE*, 2017.

[28] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *RAID*, 2018.

[29] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *S&P*, 2018.

[30] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing Rowhammer Faults Through Network Requests," arXiv:1805.04956, 2018.

[31] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.

[32] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *S&P*, 2018.

[33] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.

[34] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks," in *ASIACCS*, 2019.

[35] O. Mutlu, "RowHammer and Beyond," in *COSADE*, 2019.

[36] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks," in *USENIX Security*, 2019.

[37] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *S&P*, 2020.

[38] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.

[39] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *S&P*, 2020.

[40] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "JackHammer: Efficient Rowhammer on Heterogeneous FPGA–CPU Platforms," arXiv:1912.11523, 2020.

[41] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "PTHammer: Cross-User-Kernel-Boundary Rowhammer Through Implicit Accesses," in *MICRO*, 2020.

[42] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the Intelligence of Deep Neural Networks Through Targeted Chain of Bit Flips," in *USENIX Security*, 2020.

[43] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript," in *USENIX Security*, 2021.

[44] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021.

[45] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *S&P*, 2022.

[46] M. C. Tol, S. Islam, B. Sunar, and Z. Zhang, "Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer," arXiv:2110.07683, 2022.

[47] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering From the Next Row Over," in *USENIX Security*, 2022.

[48] L. Orosa, U. Rührmair, A. G. Yaglikci, H. Luo, A. Olgun, P. Jattke, M. Patel, J. Kim, K. Razavi, and O. Mutlu, "SpyHammer: Using RowHammer to Remotely Spy on Temperature," arXiv:2210.04084, 2022.

[49] Z. Zhang, W. He, Y. Cheng, W. Wang, Y. Gao, D. Liu, K. Li, S. Nepal, A. Fu, and Y. Zou, "Implicit Hammer: Cross-Privilege-Boundary Rowhammer through Implicit Accesses," *IEEE TDSC*, 2022.

[50] L. Liu, Y. Guo, Y. Cheng, Y. Zhang, and J. Yang, "Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack," *IEEE TC*, 2022.

[51] Y. Cohen, K. S. Tharayil, A. Haenel, D. Genkin, A. D. Keromytis, Y. Oren, and Y. Yarom, "HammerScope: Observing DRAM Power Consumption Using Rowhammer," in *CCS*, 2022.

[52] M. Zheng, Q. Lou, and L. Jiang, "TrojViT: Trojan Insertion in Vision Transformers," arXiv:2208.13049, 2022.

[53] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer," *CCS*, 2022.

[54] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *S&P*, 2022.

[55] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories," in *S&P*, 2022.

[56] H. Aydin and A. Sertbaş, "Cyber Security in Industrial Control Systems (ICS): A Survey of RowHammer Vulnerability," *Applied Computer Science*, 2022.

[57] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, "Jolt: Recovering TLS Signing Keys via Rowhammer Faults," *Cryptology ePrint Archive*, 2022.

[58] J. Wang, H. Xu, C. Xiao, L. Zhang, and Y. Zheng, "Research and Implementation of Rowhammer Attack Method based on Domestic NeoKylin Operating System," in *ICFTIC*, 2022.

[59] S. Lefforge, "Reverse Engineering Post-Quantum Cryptography Schemes to Find Rowhammer Exploits," Bachelor's Thesis, University of Arkansas, 2023.

[60] M. J. Fahr, "The Effects of Side-Channel Attacks on Post-Quantum Cryptography: Influencing FrodoKEM Key Generation Using the Rowhammer Exploit," Master's thesis, University of Arkansas, 2022.

[61] A. Kaur, P. Srivastav, and B. Ghoshal, "Work-in-Progress: DRAM-MaUT: DRAM Address Mapping Unveiling Tool for ARM Devices," in *CASES*, 2022.

[62] K. Cai, Z. Zhang, and F. Yao, "On the Feasibility of Training-time Trojan Attacks through Hardware-based Faults in Memory," in *HOST*, 2022.

[63] D. Li, D. Liu, Y. Ren, Z. Wang, Y. Sun, Z. Guan, Q. Wu, and J. Liu, "CyberRadar: A PUF-based Detecting and Mapping Framework for Physical Devices," arXiv:2201.07597, 2022.

[64] A. Roohi and S. Angizi, "Efficient Targeted Bit-Flip Attack Against the Local Binary Pattern Network," in *HOST*, 2022.

[65] F. Staudigl, H. Al Indari, D. Schön, D. Sisejkovic, F. Merchant, J. M. Joseph, V. Rana, S. Menzel, and R. Leupers, "NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories," in *DATE*, 2022.

[66] L.-H. Yang, S.-S. Huang, T.-L. Cheng, Y.-C. Kuo, and J.-J. Kuo, "Socially-Aware Collaborative Defense System against Bit-Flip Attack in Social Internet of Things and Its Online Assignment Optimization," in *ICCCN*, 2022.

[67] S. Islam, K. Mus, R. Singh, P. Schaumont, and B. Sunar, "Signature Correction Attack on Dilithium Signature Scheme," in *Euro S&P*, 2022.

[68] J. S. Kim, M. Patel, A. G. Yağlıkcı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques," in *ISCA*, 2020.

[69] A. G. Yağlıkcı, H. Luo, G. F. De Oliviera, A. Olgun, M. Patel, J. Park, H. Hassan, J. S. Kim, L. Orosa, and O. Mutlu, "Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices," in *DSN*, 2022.

[70] L. Orosa, A. G. Yağlıkcı, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim,

and O. Mutlu, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," in *MICRO*, 2021.
[71] O. Mutlu, "RowHammer," https://people.inf.ethz.ch/omutlu/pub/onur-Rowhammer-TopPicksinHardwareEmbeddedSecurity-November-8-2018.pdf, 2018, Top Picks in Hardware and Embedded Security.
[72] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *ISPASS*, 2009.
[73] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.
[74] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *ISCA*, 2023.
[75] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *NIPS*, 2020.
[76] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*, 2019.
[77] JEDEC, *JESD235D: High Bandwidth Memory DRAM (HBM1, HBM2)*, 2021.
[78] NVIDIA, "NVIDIA A100 Tensor Core GPU Architecture," Whitepaper, 2020.
[79] NVIDIA, "NVIDIA H100 Tensor Core GPU Architecture," Whitepaper, 2022.
[80] N. P. Jouppi, D. H. Yoon, G. Kurian, S. Li, N. Patil, J. Laudon, C. Young, and D. Patterson, "A Domain-Specific Supercomputer for Training Deep Neural Networks," *Communications of the ACM*, 2020.
[81] AMD, "Introducing AMD CDNA Architecture," Whitepaper, 2020.
[82] AMD, "Introducing AMD CDNA™ 2 Architecture," Whitepaper, 2021.
[83] Micron, *16Gb: x4, x8, x16 DDR4 SDRAM Features - MT40A4G4, MT40A2G8, MT40A1G16*, 2018.
[84] SAFARI Research Group, "HBM Read Disturbance — GitHub Repository," https://github.com/CMU-SAFARI/HBM-Read-Disturbance, 2024.
[85] J. M. O'Connor, "Energy Efficient High Bandwidth DRAM for Throughput Processors," Ph.D. dissertation, UT Austin, 2021.
[86] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[87] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[88] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[89] A. Olgun, H. Hassan, A. G. Yağlıkcı, Y. C. Tuğrul, L. Orosa, H. Luo, M. Patel, E. Oğuz, and O. Mutlu, "DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips," *TCAD*, 2023.
[90] SAFARI Research Group, "DRAM Bender — GitHub Repository," https://github.com/CMU-SAFARI/DRAM-Bender, 2022.
[91] "Bittware XUPVVH FPGA Board," https://www.bittware.com/fpga/xup-vvh/.
[92] Xilinx Inc., "Xilinx Alveo U50 FPGA Board," https://www.xilinx.com/products/boards-and-kits/alveo/u50.html.
[93] "Arduino MEGA Documentation," https://docs.arduino.cc/hardware/mega-2560/.
[94] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, O. Mutlu, J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," in *ISCA*, 2013.
[95] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[96] R. T. Smith, J. D. Chlipala, J. F. Bindels, R. G. Nelson, F. H. Fischer, and T. F. Mantz, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," *JSSC*, 1981.
[97] M. Horiguchi, "Redundancy Techniques for High-Density DRAMs," in *ISIS*, 1997.
[98] B. Keeth and R. Baker, *DRAM Circuit Design: A Tutorial*. Wiley, 2001.
[99] K. Itoh, *VLSI Memory Chip Design*. Springer, 2001.
[100] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses," in *MICRO*, 2015.
[101] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[102] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[103] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," *POMACS*, 2017.
[104] M. Patel, J. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics," in *MICRO*, 2020.
[105] A. G. Yağlıkcı, M. Patel, J. S. Kim, R. Azizibarzoki, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoullos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021.
[106] A. van de Goor and I. Schanstra, "Address and Data Scrambling: Causes and Impact on Memory Tests," in *DELTA*, 2002.
[107] A. G. Yağlıkçı, G. F. Oliveira, Y. C. Tuğrul, I. E. Yuksel, A. Olgun, H. Luo, and O. Mutlu, "Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions," in *HPCA*, 2024.
[108] AMD Xilinx, "AMD Xilinx Support Website," https://support.xilinx.com/s/question/0D54U00007iVe9ESAS/where-can-we-find-the-dram-timing-parameters-for-the-hbm2-stacks-in-alveo-u50?language=en_US, 2024.
[109] A. Farmahini-Farahani, S. Gurumurthi, G. Loh, and M. Ignatowski, "Challenges of High-Capacity DRAM Stacks and Potential Directions," in *Workshop on Memory Centric High Performance Computing*, 2018.
[110] K. Park, D. Yun, and S. Baeg, "Statistical Distributions of Row-hammering Induced Failures in DDR3 Components," *Microelectronics Reliability*, 2016.
[111] K. Park, C. Lim, D. Yun, and S. Baeg, "Experiments and Root Cause Analysis for Active-precharge Hammering Fault in DDR3 SDRAM under 3× nm Technology," *Microelectronics Reliability*, 2016.
[112] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in *(IMW)*, 2017.
[113] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "X-ray: Discovering DRAM Internal Structure and Error Characteristics by Issuing Memory Commands," *IEEE CAL*, 2023.
[114] S. M. Kim, B. Song, and S.-O. Jung, "Imbalance-Tolerant Bit-Line Sense Amplifier for Dummy-Less Open Bit-Line Scheme in DRAM," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.
[115] A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of GPU Memory System for Multi-Application Execution," in *MEMSYS*, 2015.
[116] B. S. Nordquist and S. D. Lew, "Apparatus, System, and Method for Coalescing Parallel Memory Requests," 2009, U.S. Patent 7,492,368.
[117] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting Core Criticality for Enhanced GPU Performance," in *SIGMETRICS*, 2016.
[118] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, 2012.
[119] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers Inc., 2008.
[120] NVIDIA, "NVIDIA Tesla P100," https://www.nvidia.com/en-us/data-center/resources/pascal-architecture-whitepaper, 2016.
[121] S. Gurumurthi, K. Lee, M. Jang, V. Sridharan, A. Nygren, Y. Ryu, K. Sohn, T. Kim, and H. Chung, "HBM3 RAS: Enhancing Resilience at Scale," *IEEE CAL*, 2021.
[122] R. Krashinsky, O. Giroux, S. Jones, N. Stam, and S. Ramaswamy, "NVIDIA Ampere Architecture In-Depth," https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/, 2020.
[123] AMD, "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors," Developer's Guide, 2013.
[124] R. Yeleswarapu and A. K. Somani, "Addressing Multiple Bit/Symbol Errors in DRAM Subsystem," arXiv:1908.01806, 2020.
[125] C. Chen, "Symbol Error Correcting Codes for Memory Applications," in *FTCS*, 1996.
[126] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell system technical journal*, 1950.
[127] NVIDIA, "Dynamic Page Retirement," https://docs.nvidia.com/deploy/pdf/Dynamic_Page_Retirement.pdf, 2019.
[128] Apple Inc., "About the Security Content of Mac EFI Security Update 2015-001," https://support.apple.com/en-us/HT204934, 2015.
[129] 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.
[130] Lenovo, "Row Hammer Privilege Escalation," https://support.lenovo.com/us/en/product_security/row_hammer, 2015.
[131] Z. Greenfield and T. Levy, "Throttling Support for Row-Hammer Counters," 2016, U.S. Patent 9,251,885.
[132] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *IEEE CAL*, 2014.
[133] K. S. Bains and J. B. Halbert, "Distributed Row Hammer Tracking," US Patent: 9,299,400, 2016.
[134] K. Bains *et al.*, "Method, Apparatus and System for Providing a Memory Refresh," US Patent: 9,030,903, 2015.
[135] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks," in *ASPLOS*, 2016.
[136] K. Bains *et al.*, "Row Hammer Refresh Command," US Patents: 9,117,544 9,236,110 10,210,925, 2015.
[137] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *DAC*, 2017.
[138] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating Wordline Crosstalk Using Adaptive Trees of Counters," in *ISCA*, 2018.
[139] G. Irazoqui, T. Eisenbarth, and B. Sunar, "MASCAT: Stopping Microarchitectural Attacks Before Execution," *IACR Cryptology*, 2016.
[140] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-Hammering Based on Memory Locality," in *DAC*, 2019.
[141] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters," in *ISCA*, 2019.
[142] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong

yet Lightweight Row Hammer Protection," in *MICRO*, 2020.
[143] A. G. Yağlıkcı, J. S. Kim, F. Devaux, and O. Mutlu, "Security Analysis of the Silver Bullet Technique for RowHammer Prevention," arXiv:2106.07084, 2021.
[144] I. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, 2020.
[145] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking," in *ISCA*, 2022.
[146] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows," in *ASPLOS*, 2022.
[147] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *OSDI*, 2018.
[148] S. Vig, S. Bhattacharya, D. Mukhopadhyay, and S.-K. Lam, "Rapid Detection of Rowhammer Attacks Using Dynamic Skewed Hash Tree," in *HASP*, 2018.
[149] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022.
[150] G.-H. Lee, S. Na, I. Byun, D. Min, and J. Kim, "CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing," in *ISCA*, 2021.
[151] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *S&P*, 2022.
[152] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, S. Nepal, Y. Gao, K. Li, Z. Wang, and C. Wu, "SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh," in *USENIX ATC*, 2022.
[153] B. K. Joardar, T. K. Bletsch, and K. Chakrabarty, "Learning to Mitigate RowHammer Attacks," in *DATE*, 2022.
[154] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "CSI: Rowhammer–Cryptographic Security and Integrity against Rowhammer (to appear)," in *S&P*, 2023.
[155] A. G. Yağlikci, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips," in *MICRO*, 2022.
[156] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime," in *MICRO*, 2022.
[157] S. Enomoto, H. Kuzuno, and H. Yamada, "Efficient Protection Mechanism for CPU Cache Flush Instruction Based Attacks," *IEICE Transactions on Information and Systems*, 2022.
[158] E. Manzhosov, A. Hastings, M. Pancholi, R. Piersma, M. T. I. Ziad, and S. Sethumadhavan, "Revisiting Residue Codes for Modern Memories," in *MICRO*, 2022.
[159] S. M. Ajorpaz, D. Moghimi, J. N. Collins, G. Pokam, N. Abu-Ghazaleh, and D. Tullsen, "EVAX: Towards a Practical, Pro-active & Adaptive Architecture for High Performance & Security," in *MICRO*, 2022.
[160] A. Naseredini, M. Berger, M. Sammartino, and S. Xiong, "ALARM: Active LeArning of Rowhammer Mitigations," https://users.sussex.ac.uk/~mfb21/rh-draft.pdf, 2022.
[161] B. K. Joardar, T. K. Bletsch, and K. Chakrabarty, "Machine Learning-based Rowhammer Mitigation," *TCAD*, 2022.
[162] H. Hassan, A. Olgun, A. G. Yaglikci, H. Luo, and O. Mutlu, "A Case for Self-Managing DRAM Chips: Improving Performance, Efficiency, Reliability, and Security via Autonomous in-DRAM Maintenance Operations," arXiv:2207.13358, 2022.
[163] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi, and X. Koutsoukos, "Leveraging EM Side-Channel Information to Detect Rowhammer Attacks," in *S&P*, 2020.
[164] K. Loughlin, S. Saroiu, A. Wolman, and B. Kasikci, "Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations," in *HotOS*, 2021.
[165] F. Devaux and R. Ayrignac, "Method and Circuit for Protecting a DRAM Memory Device from the Row Hammer Effect," US Patent: 10,885,966, 2021.
[166] J.-W. Han, J. Kim, D. Beery, K. D. Bozdag, P. Cuevas, A. Levi, I. Tain, K. Tran, A. J. Walker, S. V. Palayam, A. Arreghini, A. Furnémont, and M. Meyyappan, "Surround Gate Transistor With Epitaxially Grown Si Pillar and Simulation Study on Soft Error and Rowhammer Tolerance for DRAM," *IEEE TED*, 2021.
[167] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection," in *HPCA*, 2022.
[168] S. Saroiu, A. Wolman, and L. Cojocar, "The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses," in *IRPS*, 2022.
[169] S. Saroiu and A. Wolman, "How to Configure Row-Sampling-Based Rowhammer Defenses," *DRAMSec*, 2022.
[170] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "MOESI-Prime: Preventing Coherence-Induced Hammering in Commodity Workloads," in *ISCA*, 2022.
[171] R. Zhou, S. Tabrizchi, A. Roohi, and S. Angizi, "LT-PIM: An LUT-Based Processing-in-DRAM Architecture With RowHammer Self-Tracking," *IEEE CAL*, 2022.
[172] S. Hong, D. Kim, J. Lee, R. Oh, C. Yoo, S. Hwang, and J. Lee, "DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm," arXiv:2302.03591, 2023.
[173] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "ProTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *S&P*, 2023.
[174] A. Di Dio, K. Koning, H. Bos, and C. Giuffrida, "Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks," in *NDSS*, 2023.
[175] S. Sharma, D. Sanyal, A. Mukhopadhyay, and R. H. Shaik, "A Review on Study of Defects of DRAM-RowHammer and Its Mitigation," *Journal For Basic Sciences*, 2022.
[176] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems," in *HPCA*, 2023.
[177] J. H. Park, S. Y. Kim, D. Y. Kim, G. Kim, J. W. Park, S. Yoo, Y.-W. Lee, and M. J. Lee, "RowHammer Reduction Using a Buried Insulator in a Buried Channel Array Transistor," *IEEE Transactions on Electron Devices*, 2022.
[178] M. Wi, J. Park, S. Ko, M. J. Kim, N. S. Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *HPCA*. IEEE, 2023.
[179] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang *et al.*, "A 1.1 V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement," in *ISSCC*. IEEE, 2023.
[180] C. Gude Ramarao, K. T. Kumar, G. Ujjinappa, and B. V. D. Naidu, "Defending SoCs with FPGAs from Rowhammer Attacks," *Material Science*, 2023.
[181] K. Guha and A. Chakrabarti, "Criticality based Reliability from Rowhammer Attacks in Multi-User-Multi-FPGA Platform," in *VLSID*. IEEE, 2022.
[182] L. France, F. Bruguier, M. Mushtaq, D. Novo, and P. Benoit, "Modeling Rowhammer in the gem5 Simulator," in *CHES 2022-Conference on Cryptographic Hardware and Embedded Systems*, 2022.
[183] L. France, F. Bruguier, D. Novo, M. Mushtaq, and P. Benoit, "Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations," in *18th CryptArchi Workshop*, 2022.
[184] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," in *DRAMSec*, 2021.
[185] K. Arıkan, A. Palumbo, L. Cassano, P. Reviriego, S. Pontarelli, G. Bianchi, O. Ergin, and M. Ottavi, "Processor security: Detecting microarchitectural attacks via count-min sketches," *VLSI*, 2022.
[186] C. Tomita, M. Takita, K. Fukushima, Y. Nakano, Y. Shiraishi, and M. Morii, "Extracting the secrets of openssl with rambleed," *Sensors*, 2022.
[187] A. Saxena, G. Saileshwar, J. Juffinger, A. Kogler, D. Gruss, and M. Qureshi, "PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks," in *DSN*, 2023.
[188] R. Zhou, S. Ahmed, A. S. Rakin, and S. Angizi, "DNN-Defender: An in-DRAM Deep Neural Network Defense Mechanism for Adversarial Weight Attack," arXiv:2305.08034, 2023.
[189] S. C. Woo, W. Elsasser, M. Hamburg, E. Linstadt, M. R. Miller, T. Song, and J. Tringali, "RAMPART: RowHammer Mitigation and Repair for Server Memory Systems," in *MEMSYS*, 2023.
[190] M. J. Kim, M. Wi, J. Park, S. Ko, J. Choi, H. Nam, N. S. Kim, J. H. Ahn, and E. Lee, "How to Kill the Second Bird with One ECC: The Pursuit of Row Hammer Resilient DRAM," in *MICRO*, 2023.
[191] A. Olgun, M. Osseiran, A. G. Yaglikci, Y. C. Tugrul, H. Luo, S. Rhyner, B. Salami, J. Gomez Luna, and O. Mutlu, "An Experimental Analysis of RowHammer in HBM2 DRAM Chips," in *DSN Disrupt*, 2023.
[192] C. Lim, K. Park, and S. Baeg, "Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM," *TNS*, 2017.
[193] S.-W. Ryu, K. Min, J. Shin, H. Kwon, D. Nam, T. Oh, T.-S. Jang, M. Yoo, Y. Kim, and S. Hong, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing," in *IEDM*, 2017.
[194] D. Yun, M. Park, C. Lim, and S. Baeg, "Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs," in *IRPS*, 2018.
[195] C. Lim, K. Park, G. Bak, D. Yun, M. Park, S. Baeg, S.-J. Wen, and R. Wong, "Study of Proton Radiation Effect to Row Hammer Fault in DDR4 SDRAMs," *Microelectronics Reliability*, 2018.
[196] Z. Lang, P. Jattke, M. Marazzi, and K. Razavi, "BLASTER: Characterizing the Blast Radius of Rowhammer," in *DRAMSec*, 2023.
[197] S. S. Nabavi Larimi, B. Salami, O. S. Unsal, A. C. Kestelman, H. Sarbazi-Azad, and O. Mutlu, "Understanding Power Consumption and Reliability of High-Bandwidth Memory with Voltage Underscaling," in *DATE*, 2021.
[198] J. Kwon, S.-J. Wen, R. Fung, and S. Baeg, "Temperature Estimation of HBM2 Channels with Tail Distribution of Retention Errors in FPGA-HBM2 Platform," *Electronics*, 2023.
[199] M. B. Sullivan, N. Saxena, M. O'Connor, D. Lee, P. Racunas, S. Hukerikar, T. Tsai, S. K. S. Hari, and S. W. Keckler, "Characterizing And Mitigating Soft Errors in GPU DRAM," in *MICRO*, 2021.