An Experimental Analysis of RowHammer in HBM2 DRAM Chips

Ataberk Olgun[1] Majd Osseiran[1,2] A. Giray Yağlıkçı[1] Yahya Can Tuğrul[1]
Haocong Luo[1] Steve Rhyner[1] Behzad Salami[1] Juan Gomez Luna[1] Onur Mutlu[1]
[1]*SAFARI Research Group, ETH Zürich* [2]*American University of Beirut*

RowHammer (RH) is a significant and worsening security, safety, and reliability issue of modern DRAM chips that can be exploited to break memory isolation. Therefore, it is important to understand real DRAM chips' RH characteristics. Unfortunately, no prior work extensively studies the RH vulnerability of modern 3D-stacked high-bandwidth memory (HBM) chips, which are commonly used in modern GPUs.

In this work, we experimentally characterize the RH vulnerability of a real HBM2 DRAM chip. We show that 1) different 3D-stacked channels of HBM2 memory exhibit significantly different levels of RH vulnerability (up to 79 % difference in bit error rate), 2) the DRAM rows at the end of a DRAM bank (rows with the highest addresses) exhibit significantly fewer RH bitflips than other rows, and 3) a modern HBM2 DRAM chip implements undisclosed RH defenses that are triggered by periodic refresh operations. We describe the implications of our observations on future RH attacks and defenses and discuss future work for understanding RH in 3D-stacked memories.

1. Introduction

Modern DRAM chips suffer from the RowHammer (RH) phenomenon [1–5] where repeatedly opening (i.e., activating) and closing a DRAM row (i.e., aggressor row) induces bitflips in physically nearby rows (i.e., victim rows), breaking memory isolation [1, 2, 4–27]. Numerous studies experimentally demonstrate that a malicious attacker can reliably induce RH bitflips in a targeted manner to compromise system integrity, confidentiality, and availability [1,2,15,16,25,27–85]. RH worsens in newer DRAM chips with smaller technology nodes, where RH bitflips 1) occur with fewer row activations, e.g., more than 10× reduction in less than a decade [4], and 2) manifest in more DRAM cells, compared to older DRAM chips [2, 4, 5, 14, 17, 25–27].

To meet the high bandwidth requirements of modern data-intensive applications (e.g., GPU workloads [86–89]), DRAM designers develop High Bandwidth Memory (HBM) [90] DRAM chips, which contain multiple layers of 3D-stacked DRAM dies, using cutting-edge technology nodes. It is important to understand RH in HBM DRAM chips that have new architectural characteristics (e.g., multiple layers of DRAM dies and area- and energy-intensive through-silicon vias) which might potentially affect the RH vulnerability in various ways. Such understanding can help identify potential RH-induced security, safety, and reliability issues in HBM-based systems and allow system designers to develop effective and efficient defense mechanisms. Unfortunately, *no* prior work studies the RH vulnerability of modern HBM DRAM chips.

Our goal in this work is to experimentally analyze how vulnerable HBM DRAM chips are to RH. To this end, we provide the first detailed experimental characterization of the RH vulnerability in a modern HBM2 DRAM chip. We provide two main analyses in our study. First, we analyze the spatial variation in RH vulnerability 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 (i.e., Bit Error Rate, *BER*) and the minimum aggressor row activation count necessary to cause an RH bitflip (i.e., HC_{first}). §4 shows detailed results on how RH vulnerability varies across HBM2 channels, pseudo channels, banks, and rows. Second, we investigate undisclosed in-DRAM RH mitigations, triggered by periodic refreshes (e.g., TRR [25, 64, 91]) in HBM2 (§5).[1]

We summarize the three major observations from our experimental analyses. First, different 3D-stacked channels of the HBM2 chip exhibit significantly different levels of RH vulnerability in *BER* (up to 79%) and HC_{first} (up to 20%). Second, DRAM rows near the end of a DRAM bank (the last 832 out of 16K rows) exhibit substantially smaller *BER* than other DRAM rows. Third, the tested HBM2 DRAM chip implements an undisclosed in-DRAM RH defense mechanism (§5). Our experimental analyses also show that the RH vulnerability of a cell depends on i) the cell's physical location within a DRAM bank and ii) data stored in the neighboring cells, similar to prior works' findings in DDR3/4 DRAM chips [1, 4, 14].

We make the following contributions:

- We present the first detailed experimental characterization of the RowHammer (RH) vulnerability in a modern HBM2 DRAM chip and show that it is susceptible to RH bitflips.
- We show that the RH vulnerability significantly varies across HBM2 DRAM channels and rows within each channel.
- We show that an HBM2 DRAM chip implements an undisclosed in-DRAM RH mitigation mechanism, which resembles the one found in recent DDR4 DRAM chips manufactured by a major DRAM manufacturer [64].

2. Background

2.1. HBM2 Organization & Operation

Fig. 1 presents the organization of an HBM2 DRAM chip [90, 92] when used in an FPGA-based system. The FPGA (❶) has a memory controller communicating with one or multiple stacks of HBM using the HBM2 interface via a silicon interposer. An HBM2 stack (❷) contains multiple DRAM dies that are stacked on top of the buffer die and connected using through-silicon vias (TSVs). Each HBM2 die has one or more (shown two) independent HBM2 channels. An HBM2 channel (❸) consists of two pseudo channels (❹), each of which with multiple

[1]The HBM2 standard [90] specifies a mode called Target Row Refresh (TRR). To enable TRR Mode, the memory controller issues a well-defined series of commands. Different from this mode, we investigate whether an *undisclosed* TRR mechanism is implemented in the tested chip.

(e.g., 16) DRAM banks. A DRAM bank (❺) consists of hundreds of subarrays [93–95]. A DRAM subarray (❻) has many DRAM cells that are laid in a two-dimensional array of rows and columns, and a row buffer. DRAM cells are internally accessed at DRAM row granularity by asserting a wire called wordline. Each DRAM cell in a row is connected to the row buffer (i.e., sense amplifiers) via another wire called bitline.

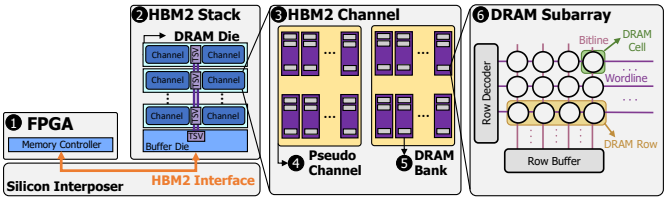

Figure 1: HBM2 DRAM System Organization

Operation. The memory controller issues an activate (ACT) command targeting a DRAM row to access a DRAM cell. The row decoder asserts the row's wordline, copying the data in the row to the row buffer. To access a different cell in another DRAM row, the memory controller issues a precharge (*PRE*) command, which deasserts the wordline.

Periodic refresh. A DRAM cell stores data as charge in its capacitor. Because the capacitor loses charge over time, it must be periodically refreshed to prevent data corruption. To refresh DRAM cells, the memory controller periodically issues refresh (*REF*) commands (e.g., every 3.9 µs) such that each cell is refreshed once at a fixed refresh period (e.g., 32 ms).

3. Experimental Infrastructure

We experimentally study an HBM2 chip using a modified version of the DRAM Bender testing infrastructure [96–99]. This infrastructure allows us to precisely control the HBM2 command timings at the granularity of 1.66 ns (i.e., the HBM2 interface clock speed is 600 MHz). Our HBM2 chip has i) a stack density of 4 GiB, ii) 8 channels, iii) 2 pseudo channels, iv) 16 banks, v) 16384 rows, and vi) 32 columns.

Testing setup. Figure 2 shows our testing setup. We conduct our experiments using a Bittware XUPVVH HBM2 FPGA board [100] (1). We use the heating pad (2) and the cooling fan (3) to change the ambient temperature of the HBM2 chip. The Arduino MEGA [101] 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. The temperature controller controls the heating pad and the cooling fan using a closed-loop PID controller. A host machine executes the test programs described in §3.1 on the FPGA board using the PCIe connection (5).

3.1. Testing Methodology

Disabling Sources of Interference. We identify four sources that can interfere with our characterization results: 1) periodic refresh [90], 2) on-die RH defense mechanisms (e.g., TRR [25, 64, 91]), 3) data retention failures [102–105], and 4) ECC [106–109]. First, we do *not* issue periodic refresh commands in our experiments. Second, disabling periodic refresh disables all known on-die RH defense mechanisms [4, 14, 17, 64, 110]. Third, we ensure that our experiments

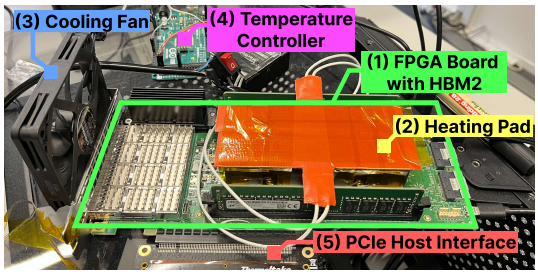

Figure 2: HBM2 DRAM testing infrastructure

finish within 27 ms, which is significantly smaller than the standard refresh period of 32 ms where manufacturers guarantee no data retention errors will occur [90]. Fourth, we disable ECC by setting the corresponding HBM2 mode register bit to zero [90].

RH Access Pattern. We use double-sided RH [1, 4, 14, 37], whereby we alternate activations to each of the aggressor rows neighboring a victim row. To find which rows are physically adjacent, we reverse-engineer the logical (memory-controller-visible) to physical (in-DRAM) row address mapping [1, 14, 32, 50, 59, 102, 104, 108, 111–118], following the methodology described in prior work [14].

RH Test Parameters. We define one hammer as a pair of activations to the two aggressor rows. We measure two metrics in our tests: *BER* and HC_{first}, as defined in §1. We use 256K hammers (i.e., 512K activations) in our *BER* experiments and up to 256K hammers in our HC_{first} experiments. We repeat both experiments for each of the four data patterns shown in Table 1. We define the worst-case data pattern (*WCDP*) as the data pattern that causes the smallest HC_{first} for a given row. When multiple data patterns cause the smallest HC_{first}, we select *WCDP* as the data pattern that causes the largest *BER* at a hammer count of 256K. To maintain a reasonable experiment time, we study the effects of RH on the first, middle, and last 3K rows in a bank in every channel and repeat all experiments five times. The HBM2 chip's temperature is kept at 85°C, the maximum operating temperature at the nominal refresh rate, in all of our experiments.

Table 1: Data patterns used in our RH tests

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

4. Spatial Variation Analysis

We provide the first spatial variation analysis of RH across HBM2 channels, pseudo channels, banks, and rows.

Fig. 3 shows a box-and-whiskers plot depicting the distribution of *BER* (y-axis) across different DRAM rows for a given data pattern (x-axis) in a channel (color-coded).[2] In addition to the four data patterns, we choose and plot the *BER* for the *WCDP* of each row.

We make four major observations from Fig. 3. First, RH bitflips occur in *every* tested DRAM row across *all* HBM channels. Second, *BER* varies across channels. A subset of channels

[2]The lower- and upper-bounds of the box are the first and the third quartiles, marking the medians of the first and second half of the ordered set of data points, respectively. Whiskers show the minimum and maximum values. The circle marker in each box shows the distribution's mean.

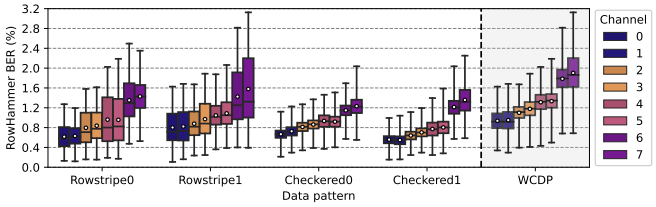

Figure 3: The fraction of DRAM cells that experience a bitflip in a DRAM row (*BER*) across different DRAM rows, channels, and data patterns. Error bars show the range of *BER* across rows.

(channels 6 and 7) exhibit significantly higher *BER* than other channels. For example, channel 7 (with the highest *BER*) has 2.03× higher *BER* than channel 0 (with the lowest *BER*) for *WCDP*. Third, 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 the figure. We hypothesize that groups of channels are spread across *different* HBM2 DRAM dies. The variation in *BER* across the channel groups could be due to manufacturing process variation (similar to that in DDR3/4 DRAM chips [1, 4, 6, 7, 14]). Fourth, *BER* changes with data pattern, e.g., the maximum *BER* in channel 7 is 3.13% and 2.04% for data patterns Rowstripe1 and Checkered0, respectively.

Fig. 4 shows the distribution of HC_{first} (y-axis) across different DRAM rows for a given data pattern (x-axis) in a channel (color-coded) using a box-and-whiskers plot.[2] In addition to the four data patterns, we choose and plot the HC_{first} for the *WCDP* of each row.

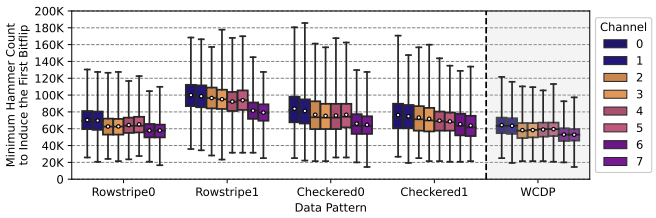

Figure 4: The minimum aggressor row activation count necessary to cause a RowHammer bitflip (HC_{first}) across different DRAM rows, channels, and data patterns.

We make three major observations from Fig. 4. First, HC_{first} is as low as 14531 across all tested channels and data patterns. Second, different channels exhibit different HC_{first} distributions. For example, channels 7 and 6 contain more rows with smaller HC_{first} values than other channels. Because these channels also exhibit a higher number of RH bitflips than other channels, we hypothesize that these channels belong in the die with the worst RH vulnerability across all dies. Third, the HC_{first} distribution in a channel depends on the data pattern used. For example, the mean HC_{first} for Rowstripe0 and Rowstripe1 in channel 0 are 57925 and 79179, respectively. We conclude that testing with different data patterns is necessary to assess the RH vulnerability of HBM2 DRAM chips as no data pattern achieves the smallest HC_{first} or *BER* (Fig. 3).

Fig. 5 shows the BER (y-axis) for tested DRAM rows (x-axis) when we use the per-row *WCDP* to initialize the rows. Different channels are color-coded and different subplots show the three regions (first, middle, and last 3K rows) we test.

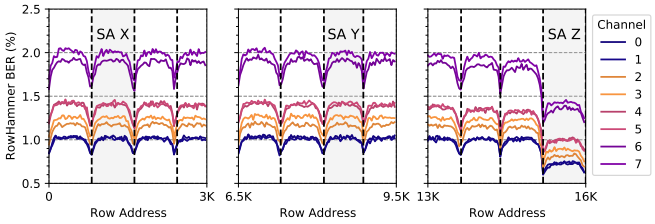

Figure 5: The fraction of DRAM cells that experience a bitflip in a DRAM row (*BER*) for different rows across a bank in different channels. Higlighted regions show individual DRAM subarrays.

We make two major observations from Fig. 5. First, the *BER* periodically increases and decreases across DRAM rows such that it is higher in the middle of a *subarray* and lower towards either end of the subarray.[3] We hypothesize that this pattern results from the physical organization of the DRAM banks. For example, the RH vulnerability of a row could increase with the row's distance from the row buffer. Second, significantly fewer bitflips occur in the last subarray of the bank (e.g., the last 832 rows in SA Z) compared to the rest of the bank. We hypothesize that this is due to the DRAM bank's physical placement. For example, assuming that proximity to the shared I/O circuitry on the DRAM die affects the RH vulnerability of a subarray, the last subarray might be placed near this shared I/O circuitry [119]. We leave further analysis on subarrays to future work.

We examine RH vulnerability differences across banks by comparing the RH *BER* distribution over 300 rows (the first, the middle, and the last 100 rows) in each of the 256 banks (across 8 channels and two pseudo channels per channel). Fig. 6 shows each bank in a scatter plot based on its *BER* distribution: 1) the coefficient of variation (CV)[4] on the x-axis and 2) the mean on the y-axis. A marker's color and shape indicate the bank's channel and pseudo channel, respectively.

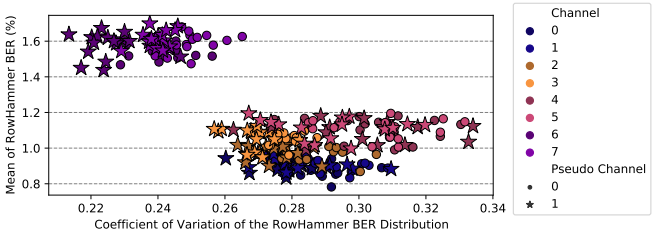

Figure 6: *BER* variation across banks. Each bank is represented by the average BER (y-axis) and the coefficient of variation in BER (x-axis) across the rows within the bank.

We make two major observations. First, there is variation in *BER* across banks and pseudo channels. For example, there is up to 0.23% difference in mean *BER* across banks in channel 7. Second, *BER* variation across banks is dominated by variation across channels. Banks in different channels tend to have a larger *BER* difference than banks in the same channel (Fig. 3). We conclude that testing different channels is more important than testing different banks or pseudo channels to assess the RH vulnerability of a given HBM2 DRAM chip.

[3] We reverse engineer subarray boundaries by performing single-sided RH [1, 4] 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. 5) or 768 (SA Y in Fig. 5) DRAM rows.

[4] CV is a distribution's standard deviation, normalized to its mean [120].

Summary. We summarize the two key takeaways of our spatial variation study. First, different 3D-stacked channels of the HBM2 DRAM chip exhibit significantly different levels of RH vulnerability in terms of *BER* (up to 79 %) and HC_{first} (up to 20 %). Second, DRAM rows near the end of a DRAM bank exhibit substantially smaller *BER* than other DRAM rows on average across all HBM2 DRAM channels. Our key takeaways have two implications for RH attacks and defenses. First, an RH attack can use the most-RH-vulnerable HBM2 channel to reduce the time it spends on i) *preparing* for an attack, by finding exploitable RH bitflips faster (i.e., by accelerating memory templating), and ii) *performing* the attack, by benefiting from a small HC_{first} value. Second, an RH defense mechanism can adapt itself to the heterogeneous distribution of the RH vulnerability across channels and subarrays, which may allow the defense mechanism to more efficiently prevent RH bitflips.

5. Uncovering in-DRAM RH Mitigations

To prevent RH bitflips, DRAM manufacturers equip their chips with a mitigation mechanism broadly referred to as Target Row Refresh (TRR) [25, 64, 91]. Proprietary versions of TRR (e.g., in DDR4) operate transparently from the perspective of the memory controller. At a high level, TRR identifies potential aggressor rows and preventively refreshes their victim rows upon receiving a *REF* command. We demonstrate that the tested chip implements a proprietary TRR (similar to the ones used in DDR4 [64, 121]) in addition to the TRR mode documented in the HBM2 standard [90].

Methodology. We use the U-TRR methodology [64, 121] 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.

One iteration of our experiment consists of six steps. First, we profile a row (R) to find its retention time (T), after which retention errors accumulate in row R unless row R is refreshed. Second, we activate and precharge row R once (i.e., refresh it) and wait for T/2. Third, we activate and precharge row R+1. We hypothesize that if the HBM2 chip implements an undisclosed TRR mechanism, it will sample the activation of R+1 and refresh the neighboring rows (i.e., row R) once a periodic *REF* command triggers TRR. Fourth, to trigger the TRR mechanism, we issue a periodic *REF* command. Fifth, we wait for another T/2 such that if TRR does *not* refresh row R, it will exhibit retention errors. Sixth, we check row R for bitflips to see if *any* TRR refresh occurred (i.e., if there are *no* bitflips). We perform 100 iterations of this experiment to check if TRR refreshes the victim row R.

Results. We observe that the profiled row (R) is refreshed *once* every *17* iterations. Therefore, we conclude that i) the tested HBM2 DRAM chip implements a proprietary, undisclosed TRR mechanism, and ii) this TRR mechanism performs a victim row refresh once every 17 periodic *REF* commands, resembling a TRR mechanism that U-TRR [64, 121] uncovers in DRAM chips from *Vendor C*. We intend to uncover more details of the proprietary TRR mechanism as part of future work.

6. Future Work

This work presents the results of our RH characterization study on a real HBM2 DRAM chip. We plan to present more insights into how RH behaves in real HBM2 DRAM chips by strengthening our characterization study at least in the following three directions.

1) Testing more HBM chips. We intend to repeat our experiments on a larger number of HBM2 chips to improve the statistical significance of our observations.

2) More characterization for each HBM chip. We plan to understand RH's sensitivities to multiple other factors. We will investigate how RH varies: 1) across different stacks in HBM2 chips, 2) based on the time an aggressor row remains active [14], 3) based on a richer set of data patterns used in initializing victim and aggressor rows, and 4) across different HBM2 voltage and temperature levels. We also intend to study the effect of RowPress [110].

3) Investigating cross-channel interference. HBM2 chips stack DRAM dies such that certain HBM channels are placed on top of each other. We aim to investigate if frequently accessing one or more *aggressor channels* can induce bitflips or worsen the reliability characteristics of other *victim channels*.

7. Related Work

We present the first experimental characterization of the RowHammer (RH) vulnerability in a modern HBM2 DRAM chip. Prior works [1, 2, 4–27] analyze new aspects of the RH vulnerability by testing real DDR3/4 DRAM chips. Other prior works [122–124] characterize real HBM chips to understand their i) soft error resiliency [124], ii) performance and reliability characteristics under reduced voltage [122], and iii) susceptibility to retention failures at different temperatures [123]. No prior work investigates the RowHammer vulnerability in a real HBM chip.

8. Conclusion

We present the results of our detailed characterization study of the RowHammer (RH) vulnerability in a modern HBM2 chip. We show that the RH vulnerability is heterogeneously distributed across various components in the HBM2 chip, which we believe has important implications for future RH attacks and defenses. We discover that the HBM2 chip implements a proprietary RH mitigation mechanism and explain how the mitigation mechanism works. We hope and expect that our findings will lead to a deeper understanding of and new solutions to the RH vulnerability in HBM-based systems.

Acknowledgments

We thank the anonymous reviewers of DSN 2023 Disrupt for feedback. We thank the SAFARI Research Group members for valuable feedback and the stimulating scientific and intellectual environment they provide. We acknowledge the generous gift funding provided by our industrial partners (especially Google, Huawei, Intel, Microsoft, VMware), which has been instrumental in enabling the decade-long research we have been conducting on read disturbance in DRAM. 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] Onur Mutlu. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In *DATE*, 2017.

[3] Onur Mutlu and Jeremie Kim. RowHammer: A Retrospective. *IEEE TCAD Special Issue on Top Picks in Hardware and Embedded Security*, 2019.

[4] Jeremie S. Kim, Minesh Patel, Abdullah Giray Yağlıkcı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques. In *ISCA*, 2020.

[5] Onur Mutlu, Ataberk Olgun, and A. Giray Yaglikci. Fundamentally Understanding and Solving RowHammer. In *ASP-DAC*, 2023.

[6] Kyungbae Park, Donghyuk Yun, and Sanghyeon Baeg. Statistical Distributions of Row-hammering Induced Failures in DDR3 Components. *Microelectronics Reliability*, 2016.

[7] Kyungbae Park, Chulseung Lim, Donghyuk Yun, and Sanghyeon Baeg. Experiments and Root Cause Analysis for Active-precharge Hammering Fault in DDR3 SDRAM under 3× nm Technology. *Microelectronics Reliability*, 2016.

[8] Chulseung Lim, Kyungbae Park, and Sanghyeon Baeg. Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM. *TNS*, 2017.

[9] Seong-Wan Ryu, Kyungkyu Min, Jungho Shin, Heimi Kwon, Donghoon Nam, Taekyung Oh, Tae-Su Jang, Minsoo Yoo, Yongtaik Kim, and Sungjoo Hong. Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing. In *IEDM*, 2017.

[10] Chulseung Lim, Kyungbae Park, Geunyong Bak, Donghyuk Yun, Myungsang Park, Sanghyeon Baeg, Shi-Jie Wen, and Richard Wong. Study of Proton Radiation Effect to Row Hammer Fault in DDR4 SDRAMs. *Microelectronics Reliability*, 2018.

[11] Donghyuk Yun, Myungsang Park, Chulseung Lim, and Sanghyeon Baeg. Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs. In *IRPS*, 2018.

[12] Thomas Yang and Xi-Wei Lin. Trap-Assisted DRAM Row Hammer Effect. *EDL*, 2019.

[13] Andrew J. Walker, Sungkwon Lee, and Dafna Beery. On DRAM RowHammer and the Physics on Insecurity. *IEEE TED*, 2021.

[14] Lois Orosa, A Giray Yağlıkcı, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S. Kim, and Onur Mutlu. A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. In *MICRO*, 2021.

[15] Lois Orosa, Ulrich Rührmair, A Giray Yaglikci, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. SpyHammer: Using RowHammer to Remotely Spy on Temperature. arXiv:2210.04084, 2022.

[16] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D Keromytis, Yossi Oren, and Yuval Yarom. HammerScope: Observing DRAM Power Consumption Using Rowhammer. In *CCS*, 2022.

[17] A. Giray Yağlıkcı, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In *DSN*, 2022.

[18] Mohammad Nasim Imtiaz Khan and Swaroop Ghosh. Analysis of Row Hammer Attack on STTRAM. In *ICCD*, 2018.

[19] S. Agarwal, H. Dixit, D. Datta, M. Tran, D. Houssameddine, D. Shum, and F. Benistant. Rowhammer for Spin Torque based Memory: Problem or not? In *INTERMAG*, 2018.

[20] Haitong Li, Hong-Yu Chen, Zhe Chen, Bing Chen, Rui Liu, Gang Qiu, Peng Huang, Feifei Zhang, Zizhen Jiang, Bin Gao, Lifeng Liu, Xiaoyan Liu, Shimeng Yu, H.-S. Philip Wong, and Jinfeng Kang. Write Disturb Analyses on Half-Selected Cells of Cross-Point RRAM Arrays. In *IRPS*, 2014.

[21] Kai Ni, Xueqing Li, Jeffrey A. Smith, Matthew Jerry, and Suman Datta. Write Disturb in Ferroelectric FETs and Its Implication for 1T-FeFET AND Memory Arrays. *IEEE EDL*, 2018.

[22] Paul R. Genssler, Victor M. van Santen, Jörg Henkel, and Hussam Amrouch. On the Reliability of FeFET On-Chip Memory. *TC*, 2022.

[23] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. WhistleBlower: A System-level Empirical Study on RowHammer. *IEEE Transactions on Computers*, 2023.

[24] Sanghyeon Baeg, Donghyuk Yun, Myungsun Chun, and Shi-Jie Wen. Estimation of the Trap Energy Characteristics of Row Hammer-Affected Cells in Gamma-Irradiated DDR4 DRAM. *IEEE Transactions on Nuclear Science*, 2022.

[25] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *S&P*, 2020.

[26] Onur Mutlu. RowHammer. `https://people.inf.ethz.ch/omutlu/pub/onur-Rowhammer-TopPicksinHardwareEmbeddedSecurity-November-8-2018.pdf`, 2018. Top Picks in Hardware and Embedded Security.

[27] Onur Mutlu. RowHammer and Beyond. In *COSADE*, 2019.

[28] Apostolos P Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks. *Electronics*, 2017.

[29] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking Deterministic Signature Schemes using Fault Attacks. In *EuroS&P*, 2018.

[30] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks Over the Network and Defenses. In *USENIX ATC*, 2018.

[31] Sebastien Carre, Matthieu Desjardins, Adrien Facon, and Sylvain Guilley. OpenSSL Bellcore's Protection Helps Fault Attack. In *DSD*, 2018.

[32] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks. In *IVSW*, 2018.

[33] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In *ASHES*, 2018.

[34] Sarani Bhattacharya and Debdeep Mukhopadhyay. Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug. In *Fault Tolerant Architectures for Cryptography and Hardware Security*. 2018.

[35] Mark Seaborn and Thomas 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.

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

[37] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. *Black Hat*, 2015.

[38] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS*, 2016.

[39] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript. In *DIMVA*, 2016.

[40] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security*, 2016.

[41] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security*, 2016.

[42] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security*, 2016.

[43] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector. In *S&P*, 2016.

[44] Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious Case of RowHammer: Flipping Secret Exponent Bits using Timing Analysis. In *CHES*, 2016.

[45] Wayne Burleson, Onur Mutlu, and Mohit Tiwari. Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer. In *DAC*, 2016.

[46] Rui Qiao et al. A New Approach for RowHammer Attacks. In *HOST*, 2016.

[47] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory. In *USENIX Security*, 2017.

[48] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SysTEX*, 2017.

[49] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks. In *HOST*, 2017.

[50] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In *RAID*, 2018.

[51] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In *S&P*, 2018.

[52] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults Through Network Requests. arXiv:1805.04956, 2018.

[53] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In *DIMVA*, 2018.

[54] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In *S&P*, 2018.

[55] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *S&P*, 2019.

[56] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In *ASIACCS*, 2019.

[57] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *USENIX Security*, 2019.

[58] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In *S&P*, 2020.

[59] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *S&P*, 2020.

[60] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. JackHammer: Efficient Rowhammer on Heterogeneous FPGA–CPU Platforms. arXiv:1912.11523, 2020.

[61] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. PTHammer: Cross-User-Kernel-Boundary Rowhammer Through Implicit Accesses. In *MICRO*, 2020.

[62] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the Intelli-

gence of Deep Neural Networks Through Targeted Chain of Bit Flips. In *USENIX Security*, 2020.
[63] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript. In *USENIX Security*, 2021.
[64] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications. In *MICRO*, 2021.
[65] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable Rowhammering in the Frequency Domain. In *S&P*, 2022.
[66] M Caner Tol, Saad Islam, Berk Sunar, and Ziming Zhang. Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer. arXiv:2110.07683, 2022.
[67] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In *USENIX Security*, 2022.
[68] Zhi Zhang, Wei He, Yueqiang Cheng, Wenhao Wang, Yansong Gao, Dongxi Liu, Kang Li, Surya Nepal, Anmin Fu, and Yi Zou. Implicit Hammer: Cross-Privilege-Boundary Rowhammer through Implicit Accesses. *IEEE TDSC*, 2022.
[69] Liang Liu, Yanan Guo, Yueqiang Cheng, Youtao Zhang, and Jun Yang. Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack. *IEEE TC*, 2022.
[70] Mengxin Zheng, Qian Lou, and Lei Jiang. TrojViT: Trojan Insertion in Vision Transformers. arXiv:2208.13049, 2022.
[71] Michael Fahr Jr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, et al. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. *CCS*, 2022.
[72] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *S&P*, 2022.
[73] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *S&P*, 2022.
[74] Hakan Aydin and Ahmet Sertbaş. Cyber Security in Industrial Control Systems (ICS): A Survey of RowHammer Vulnerability. *Applied Computer Science*, 2022.
[75] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. Jolt: Recovering TLS Signing Keys via Rowhammer Faults. *Cryptology ePrint Archive*, 2022.
[76] Jianxin Wang, Hongke Xu, Chaoen Xiao, Lei Zhang, and Yuzheng Zheng. Research and Implementation of Rowhammer Attack Method based on Domestic NeoKylin Operating System. In *ICFTIC*, 2022.
[77] Sam Lefforge. Reverse Engineering Post-Quantum Cryptography Schemes to Find Rowhammer Exploits. Bachelor's Thesis, University of Arkansas, 2023.
[78] Michael Jacob Fahr. The Effects of Side-Channel Attacks on Post-Quantum Cryptography: Influencing FrodoKEM Key Generation Using the Rowhammer Exploit. 2022.
[79] Anandpreet Kaur, Pravin Srivastav, and Bibhas Ghoshal. Work-in-Progress: DRAMaUT: DRAM Address Mapping Unveiling Tool for ARM Devices. In *CASES*, 2022.
[80] Kunbei Cai, Zhenkai Zhang, and Fan Yao. On the Feasibility of Training-time Trojan Attacks through Hardware-based Faults in Memory. In *HOST*, 2022.
[81] Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Yu Sun, Zhenyu Guan, Qianhong Wu, and Jianwei Liu. CyberRadar: A PUF-based Detecting and Mapping Framework for Physical Devices. arXiv:2201.07597, 2022.
[82] Arman Roohi and Shaahin Angizi. Efficient Targeted Bit-Flip Attack Against the Local Binary Pattern Network. In *HOST*, 2022.
[83] Felix Staudigl, Hazem Al Indari, Daniel Schön, Dominik Sisejkovic, Farhad Merchant, Jan Moritz Joseph, Vikas Rana, Stephan Menzel, and Rainer Leupers. NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories. In *DATE*, 2022.
[84] Li-Hsing Yang, Shin-Shan Huang, Tsai-Ling Cheng, Yi-Ching Kuo, and Jian-Jhih Kuo. Socially-Aware Collaborative Defense System against Bit-Flip Attack in Social Internet of Things and Its Online Assignment Optimization. In *ICCCN*, 2022.
[85] Saad Islam, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. Signature Correction Attack on Dilithium Signature Scheme. In *Euro S&P*, 2022.
[86] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *ISPASS*, 2009.
[87] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IISWC*, 2009.
[88] Juan Gómez-Luna, Izzat El Hajj, Li-Wen Chang, Victor García-Floreszx, Simon Garcia De Gonzalo, Thomas B Jablin, Antonio J Pena, and Wen-mei Hwu. Chai: Collaborative Heterogeneous Applications for Integrated-Architectures. In *ISPASS*, 2017.
[89] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. Demystifying Complex Workload–DRAM Interactions: An Experimental Study. In *SIGMETRICS*, 2019.
[90] JEDEC. *JESD235D: High Bandwidth Memory DRAM (HBM1, HBM2)*, 2021.
[91] Micron. DDR4 SDRAM Datasheet. In *Micron*, page 380, 2016.
[92] James Michael O'Connor. *Energy Efficient High Bandwidth DRAM for Throughput Processors*. PhD thesis, UT Austin, 2021.
[93] Y. Kim et al. A case for subarray-level parallelism (SALP) in DRAM. In *ISCA*, 2012.
[94] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd Mowry. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*, 2013.
[95] Kevin K Chang, Donghyuk Lee, Zeshan Chishti, Alaa R Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. Improving DRAM Performance by Parallelizing Refreshes with Accesses. In *HPCA*, 2014.
[96] Ataberk Olgun, Hasan Hassan, A. Giray Yağlıkcı, Yahya Can Tuğrul, Lois Orosa, Haocong Luo, Minesh Patel, Ergin Oğuz, and Onur Mutlu. DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips. arXiv:2211.05838, 2022.
[97] SAFARI Research Group. DRAM Bender — GitHub Repository. `https://github.com/CMU-SAFARI/DRAM-Bender`, 2022.
[98] Hasan Hassan, Nandita Vijaykumar, Samira Khan, Saugata Ghose, Kevin Chang, Gennady Pekhimenko, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies. In *HPCA*, 2017.
[99] SAFARI Research Group. SoftMC — GitHub Repository. `https://github.com/CMU-SAFARI/softmc`, 2017.
[100] Bittware XUPVVH FPGA Board. `https://www.bittware.com/fpga/xup-vvh/`.
[101] Arduino MEGA Documentation. `https://docs.arduino.cc/hardware/mega-2560/`.
[102] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, Onur 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.
[103] Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa R Alameldeen, Chris Wilkerson, and Onur Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.
[104] Samira Khan, Donghyuk Lee, and Onur Mutlu. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016.
[105] M.K. Qureshi, Dae-Hyun Kim, S. Khan, P.J. Nair, and O. Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.
[106] Minesh Patel, Jeremie S Kim, and Onur Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. In *ISCA*, 2017.
[107] Minesh Patel, Jeremie S. Kim, Hasan Hassan, and Onur Mutlu. Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices. In *DSN*, 2019.
[108] Minesh Patel, Jeremie Kim, Taha Shahroodi, Hasan Hassan, and Onur Mutlu. Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics. In *MICRO*, 2020.
[109] Minesh Patel, Geraldo Francisco de Oliveira Jr., and Onur Mutlu. HARP: Practically and Effectively Identifying Uncorrectable Errors in Main Memory Chips That Use On-Die ECC. In *MICRO*, 2021.
[110] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkcı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In *ISCA*, 2023.
[111] Robert T Smith, James D Chlipala, JOHN FM Bindels, Roy G Nelson, Frederick H Fischer, and Thomas F Mantz. Laser Programmable Redundancy and Yield Improvement in a 64K DRAM. *JSSC*, 1981.
[112] Masashi Horiguchi. Redundancy Techniques for High-Density DRAMs. In *ISIS*, 1997.
[113] B. Keeth and R.J. Baker. *DRAM Circuit Design: A Tutorial*. Wiley, 2001.
[114] Kiyoo Itoh. *VLSI Memory Chip Design*. Springer, 2001.
[115] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses. In *MICRO*, 2015.
[116] Samira Khan, Chris Wilkerson, Zhe Wang, Alaa R Alameldeen, Donghyuk Lee, and Onur Mutlu. Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content. In *MICRO*, 2017.
[117] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu. Design-induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. *POMACS*, 2017.
[118] A. Giray Yağlıkcı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizibarzoki, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoullos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *HPCA*, 2021.
[119] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *IMW*, 2017.
[120] Everitt Biran. The Cambridge Dictionary of Statistics. *Cambridge University Press*, 1998.
[121] U-TRR. `https://github.com/CMU-SAFARI/u-trr`, 2021.
[122] Seyed Saber Nabavi Larimi, Behzad Salami, Osman S. Unsal, Adrián Cristal Kestelman, Hamid Sarbazi-Azad, and Onur Mutlu. Understanding Power Consumption and Reliability of High-Bandwidth Memory with Voltage Underscaling. In *DATE*, 2021.
[123] Junhyeong Kwon, Shi-Jie Wen, Rita Fung, and Sanghyeon Baeg. Temperature Estimation of HBM2 Channels with Tail Distribution of Retention Errors in FPGA-HBM2 Platform. *Electronics*, 2023.
[124] Michael B. Sullivan, Nirmal Saxena, Mike O'Connor, Donghyuk Lee, Paul Racunas, Saurabh Hukerikar, Timothy Tsai, Siva Kumar Sastry Hari, and Stephen W. Keckler. Characterizing And Mitigating Soft Errors in GPU DRAM. In *MICRO*, 2021.