2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10070966

SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling

Minbok Wi*†, Jaehyun Park*†, Seoyoung Ko†, Michael Jaemin Kim†, Nam Sung Kim‡, Eojin Lee§, Jung Ho Ahn†
†Seoul National University, ‡University of Illinois at Urbana-Champaign, §Inha University
†{homakaka, wogus20002, seoyoungko, michael604, gajh}@snu.ac.kr, ‡nskim@illinois.edu, §ejlee@inha.ac.kr

***Abstract*—As Row Hammer (RH) attacks have been a critical threat to computer systems, numerous hardware-based (HW-based) RH mitigation strategies have been proposed. However, the advent of non-adjacent RH attacks and lower RH thresholds significantly increase the area and performance overhead of these prior solutions due to their conservative design characteristics.**

We propose a new in-DRAM RH protection solution named Shuffling Aggressor DRAM Rows (SHADOW). SHADOW dynamically randomizes DRAM row mapping information, preventing an attacker from targeting a specific victim row that may hold critical data. SHADOW is robust against non-adjacent RH attacks because it utilizes the in-DRAM row-shuffle technique. To realize the in-DRAM row-shuffle operation with low performance and energy overhead, we use a novel DRAM microarchitecture optimization technique. We also utilize the recently introduced JEDEC RFM interface to enable in-DRAM RH mitigation without any DRAM interface changes. By exploiting an additional DRAM row per subarray, SHADOW does not require costly SRAM- or CAM-based tracking structures other than intrinsic counters for the RFM interface. We demonstrate the strong probabilistic protection of SHADOW against RH attacks through adversarial pattern analysis and highlight the compelling performance, area, and energy overheads compared to those of state-of-the-art HW-based RH prevention solutions.

I. Introduction

The Row Hammer (RH) phenomenon is a DRAM reliability problem in which frequent activations on a specific (aggressor) row result in bit-flips in its nearby (victim) rows. It has been a critical threat to computer systems since its public demonstration in 2014 [45]. Recent changes in DRAM fabrication process scaling resulted in even worse vulnerabilities, i.e., RH occurring with even fewer activations [20], [43] (lower RH threshold, henceforth H_{cnt}), and RH victims being affected by non-adjacent RH attacks [43], [47].

The main reason why the RH phenomenon does not end at the reliability issue but has remained an infamous security vulnerability is that the attacker can trigger targeted bit-flips on a specific target row. When the specific row is sprayed carefully to hold some critical data, the attacker can pose various security threats, such as achieving privilege escalation, executing a cross-VM attack, and acquiring prohibited information [1], [7], [9], [13], [14], [19], [22], [23], [27], [31], [49], [65], [69], [83], [86], [87], [92], [95], [96], [99].

In attempts to prevent RH and related attacks, numerous hardware RH protection solutions [39], [41], [44], [45], [52], [55], [62], [72], [74], [76], [80], [80], [94], [98] have been proposed. However, most state-of-the-art works require expensive counter structures based on SRAM or content-addressable memory (CAM). If implemented on the memory controller (MC) [62], [67], [72], the result is a conservative design to support attached DRAM devices and a low RH threshold in what can be considered as a 'pessimistic' manner. DRAM-side implementations [44], [55] are limited by stringent area budgets. Moreover, the recent trend of lowering the RH threshold values [43] also exposes scalability challenges, as doing so often requires nearly exponential growth in the number of required counters.

The advent of non-adjacent RH-based attacks [43], [47] (henceforth blast-attacks) poses a new threat to the RH protection solutions. Although previously proposed solutions can be modified to protect against such attacks, such protection comes at the cost of a degraded effective H_{cnt} and higher performance and energy costs per mitigating action [44], [62].

To prevent RH robustly even under these unfavorable conditions, such as low RH thresholds and blast-attacks, we propose a holistic RH defense solution, dubbed SHADOW (**Sh**uffling **A**ggressor **D**RAM R**ow**s). SHADOW provides a strong RH defense capability against blast-attacks by breaking the spatial adjacency between attackers and victims through row shuffling. Specifically, SHADOW exploits the new refresh management (RFM) command supported by the DDR5 JEDEC standard and enhances the DRAM microarchitecture to randomize the physical address to DRAM address (PA-to-DA) mapping dynamically and then to transparently transfer the data of remapped rows via high-throughput in-DRAM row-copy operations [10], [75].

As such, SHADOW provides near-complete probabilistic RH protection at a low performance cost with row-shuffle and grants two major advantages over the other mitigating actions of target-row-refresh (TRR) [44], [52], [62], [76] or throttling [94]. First, SHADOW breaks the physical adjacency between aggressor and victim rows, preventing an attacker from targeting a particular victim row using a fixed PA. Second, it provides strong protection against blast-attacks because it protects multiple rows in the RH-affected range (henceforth blast radius) by moving the location of the aggressor row rather than those of the victim rows. The prior row-shuffle-based RH mitigation scheme, Randomized Row-Swap (RRS [72]), is another probabilistic RH-protection scheme exploiting row

*Both authors contributed equally to the paper.

shuffling. However, its row-swap operation incurs high performance overhead due to its long latency and memory-channel-blocking operations [53], [74].

SHADOW can be implemented in DRAM with negligible area, performance, and energy costs compared with other RH defense techniques, without modifying the memory interface. SHADOW stores all PA-to-DA mapping information in a spare-row-like [40], [81] DRAM remapping row instead of populating expensive SRAM or CAM in a DRAM [6] or an MC [72]. Based on the fact that each subarray holds its own row buffer, we also propose subarray-pairing, a novel DRAM microarchitecture optimization strategy that pairs every two subarrays. Each paired subarray holds the other subarray's remapping information to minimize the timing overhead incurred when reading the remapping row. SHADOW also improves its protection capability by employing a low-cost intra-subarray incremental refresh scheme. Finally, SHADOW uses the RFM command of the recent JEDEC standard [34]–[37], without the need for a change in the MC-to-DRAM interface.

Based on our analysis, SHADOW provides an RH-induced bit-flip probability of less than 1% per year per DDR5 rank, with a less than 3% performance degradation even on highly memory-intensive workloads at 4K H_{cnt}. We also demonstrate that SHADOW is less vulnerable to blast-attacks than other RFM-based RH defense techniques, with a negligible change in the protection capability and minimal performance degradation, even when the blast radius increases. SHADOW requires an additional 0.47% of area overhead of the DRAM chip size, with 0.6% DRAM capacity overhead for additional rows, including the remapping rows. Unlike most prior works, SHADOW's area overhead is fixed regardless of H_{cnt}, not requiring an additional SRAM- or CAM-based counter table besides the intrinsic counters for the RFM interface.

The key contributions of this paper are as follows:

- We propose SHADOW, a new in-DRAM RH-prevention solution using the JEDEC RFM interface, exploiting a row-shuffle operation that dynamically breaks the adjacency of an aggressor row and the corresponding victim rows.
- We implement row-shuffle inside DRAM subarrays by storing remapping information per subarray without the need of an additional SRAM or CAM structure, minimizing the overhead through microarchitecture optimization.
- We show that SHADOW has a negligible impact on performance and power consumption by conducting actual system experiments and architectural simulations.

II. Background

A. DRAM Organization

A DRAM-based main-memory system is hierarchically composed of channels, ranks, and banks connected to memory controllers (MCs), as shown in Figure 1. A bank is composed of multiple subarrays, each of which is a 2D array of rows and columns of DRAM cells indexed by wordlines and bitlines, respectively. One bitline sense amplifier (BLSA or local row buffer) is utilized per column of a subarray and is connected to the global IO (GIO) via the local IO (LIO). During every activation of a row, the global row decoder selects a subarray and the local row decoder selects a specific row.

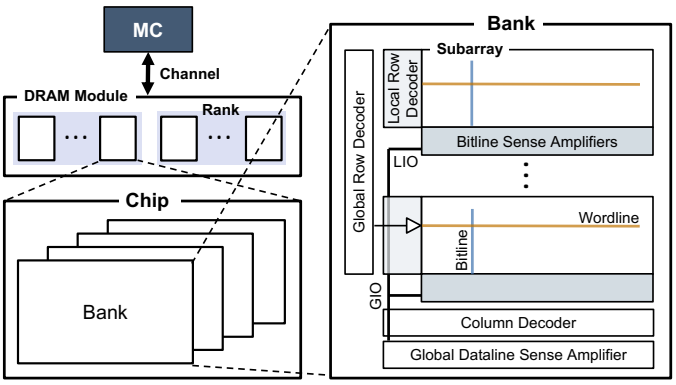

Fig. 1. DRAM-based main memory architecture

TABLE I
Terminology and Abbreviations

Symbol	Description
PA	Physical address. An MC sends PA to DRAM device.
DA	DRAM device address. Granted to each row and column.
H_{cnt}	Hammer Count. The minimum ACT count for incurring the bit-flip by hammering the row.
RAAIMT	A predefined RFM threshold. RFM command is issued when the per-bank ACT count reaches RAAIMT.
tRFM	The duration for processing RFM command.

Following the primary-secondary communication protocol of DDR memory interfaces, an MC issues DRAM commands to a DRAM device in a synchronous manner, with JEDEC-specified fixed timings [34], [35]. tRCD is the minimum time interval between an activate (ACT) and a read (RD) command. tRAS is the row restoration time, and tRP is the precharge (PRE) time. tAA is the time between the RD command and when the data return from a DRAM device. Two consecutive RD commands to the same bank should be separated by tCCD_L.

DRAM periodically refreshes each cell by means of an auto-refresh step because its cells leak charge over time. The MC sends a refresh command every refresh interval (tREFI), granting refresh cycle time (tRFC) slack to DRAM. The auto-refresh operations cause each cell to be refreshed once every tREFW. Recently, a refresh management (RFM) interface was introduced in DDR5, especially for RH-mitigation purposes [35]. Following this interface, a small per-bank ACT counter (called RAA count) exists at the MC. When the ACT count reaches a predefined threshold (RAAIMT), the MC issues an RFM command to the corresponding DRAM bank, provisioning a predefined time margin (tRFM). Table I summarizes the terminologies and abbreviations.

B. DRAM Address Mapping

The MC translates the received physical address (PA) from a host processor to a specific DRAM device address (DA). A memory request is sent to specific DRAM cells based on

the processor-specific way of dividing the PA into a tuple of channel, rank, bank, row, and column indices, allowing the processor to exploit parallelism [3], [28], [65]. When a PA arrives at a DRAM device, more complexity arises because DRAM manufacturers exploit additional spare rows and columns to remap faulty cells during the post-manufacturing process [33], [35] for better resilience [81]. Despite these complexities in PA-to-DA mapping, these issues are mostly *static* in nature. Prior studies have demonstrated that it is possible to reverse-engineer the PA tuple structures of the processors using the timing side channels or by means of physical probing [45], [50], [65], [83].

C. Row Hammer Attacks

Modern DRAM devices are vulnerable to Row Hammer (RH) attacks that flip bits in arbitrary DRAM cells. RH attacks exploit the electrical interference between two physically adjacent DRAM rows by repeatedly activating a specific (aggressor) row to cause bit-flips of its physically adjacent (victim) rows. RH is a critical threat to computer systems due to the threat of bit-flips on any row, leading to system integrity failures. The minimum ACT count to incur the RH phenomenon is referred to as the Hammer Count (H_{cnt}) [43].

An RH attack consists of a *memory templating* step and a *bit-flip triggering* step. First, memory templating gains control over the DRAM rows that are to be repeatedly accessed [16], [47], [65], [85], [86]. An attacker can infer the adjacency of DRAM rows by leveraging previously disclosed information [28] or by reverse-engineering static PA-to-DA mappings [65], allowing the attacker to accelerate the memory templating process.

In the second step, the attacker generates memory access requests repetitively to trigger RH bit-flips. Exploiting the information from the templating phase, memory massaging techniques [16], [47], [49], [69], [85], [86] are often used to allocate the desired data on a vulnerable victim row. Then, the attacker can generate sophisticated access patterns [16], [20], [26], [32], [47] that cause hammering on the victim row (e.g., double-sided attacks, many-sided attacks, and blast-attacks), sometimes even bypassing [16], [20] or abusing [47] any currently implemented RH protection scheme [26].

D. Threat Model

We assume the following threat model: 1) more than H_{cnt} activations on the aggressor row within the refresh interval (tREFW) cause bit-flips in the corresponding adjacent victim rows; 2) the aggressor row also affects non-adjacent rows within a short distance (blast-attack), where the effect is halved as the distance increases [43], [94]; 3) the specific subarray's aggressor row does not affect other subarrays' rows [93]; 4) the attacker can control the physical address of the system and is aware of the initial static mapping of the DRAM rows (i.e., physical adjacency information between rows) [26], [32], [47], [63], [65], [90]. Therefore, the attacker can perform a sophisticated RH attack by exploiting the effect of the blast radius.

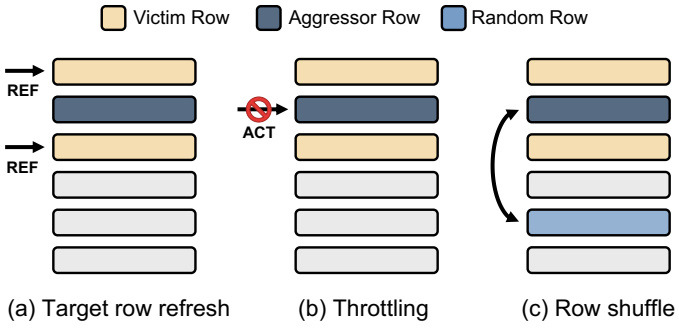

Fig. 2. Hardware-based Row Hammer mitigating actions

III. Prior HW-based RH Mitigation Schemes

This section describes prior HW-based RH mitigation schemes. We categorize them in terms of their mitigating action and the implementation locations, while focusing on robustness against blast-attacks and their counter structures.

A. Mitigating Actions and Robustness against Blast-attacks

Three major mitigating actions have been introduced as a means of RH protection for hardware-based RH protection schemes: an additional refresh of the victim row (TRR), throttling, and row-shuffle.

TRR (target row refresh) refreshes the target row to recharge DRAM cells, which prevents the RH bit-flips if executed at the proper time based on tracking mechanisms [44], [52], [62], [76] or at a high enough frequency based on random selection [45] (Figure 2(a)). TRR can be implemented using the recently introduced RFM command [44], [55] or by modifying the conventional MC to a DRAM interface [62].

Throttling scheme modifies the MC scheduler to identify the thread showing a suspicious memory access pattern. When the hazard is recognized, this method limits the ACT frequency of the corresponding thread or DRAM row [94] by adding delays (Figure 2(b)).

While RH protection schemes based on TRR or throttling possess unique strengths and weaknesses, they share a critical problem known as the *blast radius*. As PA-to-DA mapping is static, sophisticated blast-attacks can intentionally degrade protection capabilities. Therefore, prior works require modification to their vanilla versions by decreasing the target H_{cnt} [73], [94] and sometimes even requiring multiple mitigating actions for each event [44], [62], leading to exacerbated area, performance, and energy overhead.

Row-shuffle, a recently proposed RH mitigating action, can be a solution for such a drawback [25], [72]. It identifies or samples an aggressor row and swaps its location in the DA with another randomly chosen row in the DA [72] or an additional copy row [25] (Figure 2(c)). Multiple rounds of the row-shuffle operation can *dynamically* randomize the PA-to-DA mapping and the row adjacency. This randomization prevents an attack from targeting a specific victim row, minimizing the possibility that double-sided attacks or blast-attacks will be successful and thus boosting the protection capability. Moreover, memory

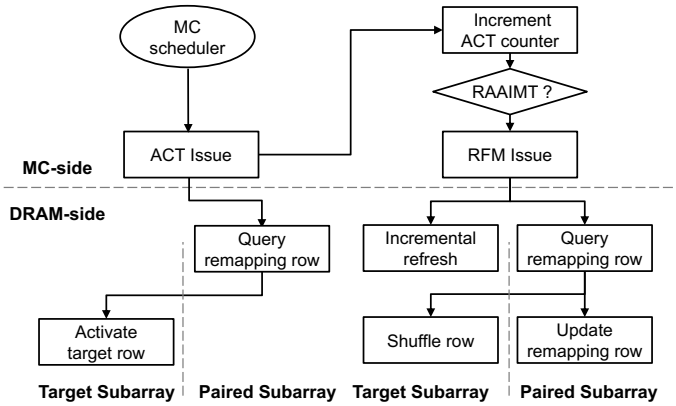

Fig. 3. Operational flow of SHADOW

templating as described in Section II-C cannot be undertaken successfully in such cases. Thus, well-known fatal attacks [20], [26], [32], [83] cannot be conducted easily as the attacker does not know which aggressor rows in the PA to hammer in order to induce RH bit-flips on a specific target victim row.

However, existing row-shuffle-based solutions incur serious performance overhead originating from their inefficient row-shuffle operations. For example, the row-shuffle of RRS [72] blocks the corresponding memory channel from processing normal memory requests for 4,000 nanoseconds or more. This memory channel blocking incurs severe increases in the memory access latency and bandwidth, which is unacceptable for latency-critical workloads [53].

B. Implementation Locations and Counter Structures

Most state-of-the-art RH protection schemes [44], [62], [67], [72], [94], [94] require some form of an SRAM- or CAM-based counter structure to track the number of ACTs, which grants them lower performance overhead at the cost of the area overhead. The MC-side implementation [62], [72], [94] allows one to take advantage of fast/dense transistors offered by a superior logic process and a large area budget. However, as one MC manages multiple DRAM devices, the total number of required counters must be set pessimistically. For example, RRS [72] requires a 43KB SRAM per bank, meaning more than 20MB SRAM per processor considering 16 DDR5 ranks, which comes close to the last-level cache size. Unlike the MC-side implementation, DRAM-side implementation [44], [55] has optimization potential if considering chip-specific characteristics. However, DRAM-side implementation is limited by the stringent area budget and requires duplicate counter structures for each DRAM device. In fact, most prior works encounter the scalability challenge [62], [67], [72], [94] of substantial chip area overhead of the counter structure under the recent trend of more DRAM channels, banks, lower H_{cnt} values, and a wider blast radius.

IV. SHADOW

We propose a new intra-subarray row-shuffle-based in-DRAM RH mitigation architecture dubbed SHADOW

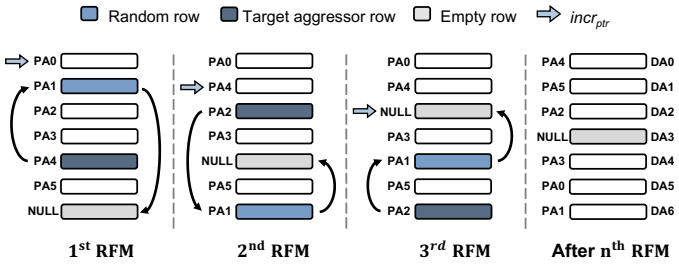

Fig. 4. The operations of SHADOW on RFM commands, including row-shuffle and incremental refresh. After multiple RFMs, the PA (physical address) to DA (DRAM device address) mapping of each row is randomized.

(**Sh**uffling **A**ggressor **D**RAM R**ow**s), which provides strong probabilistic protection against various RH attack patterns that exploit PA-to-DA mapping information but that does not require the maintenance of an additional expensive SRAM- or CAM-based ACT tracking table structure.

A. Overview

SHADOW mitigates RH by dynamically randomizing the spatial adjacency between aggressor and victim rows via an in-DRAM row-shuffle operation without the need for a costly additional counter structure or long latency memory channel blocking. First, to track the PA-to-DA mapping internally, SHADOW stores remapping information by populating an additional row, termed a remapping-row, inside each DRAM subarray. Also, SHADOW does not use a costly SRAM-based ACT tracking table to identify an aggressor row but simply selects random rows to shuffle. Second, the in-DRAM row-shuffle operation takes place upon every RFM command, leveraging the intra-subarray row-copy operation proposed in prior studies [10], [75] to perform a high-throughput row-shuffle *inside a subarray* without a data transfer over the MC. To realize the secure and low-overhead RH-protection scheme inside DRAM, we also leverage novel *incremental refresh* and *subarray-pairing strategies* as well as other previously suggested techniques. Their implementation details are discussed in Sections V and VI.

Figure 3 shows the operational flow of SHADOW. At every ACT command, SHADOW refers to the remapping-row to translate the PA to the DA. When the ACT count of a specific bank reaches RAAIMT, the MC issues an RFM command providing tRFM slack time for DRAM. Upon receiving the RFM command, SHADOW executes the in-DRAM row-shuffle operation targeting an aggressor and the incremental refresh strategy.

B. High Level Row-Shuffle Operations of SHADOW

Three different rows participate in a row-shuffle operation: an aggressor row (Row_{aggr}), a randomly selected row (Row_{rand}), and an empty row (Row_{empt}). Row_{aggr} is the main target of the row-shuffle, which is randomly selected among recent RAAIMT numbers of activated rows. Row_{rand} is an another randomly selected row in the same subarray of Row_{aggr}, which will be shuffled along with Row_{aggr}, providing

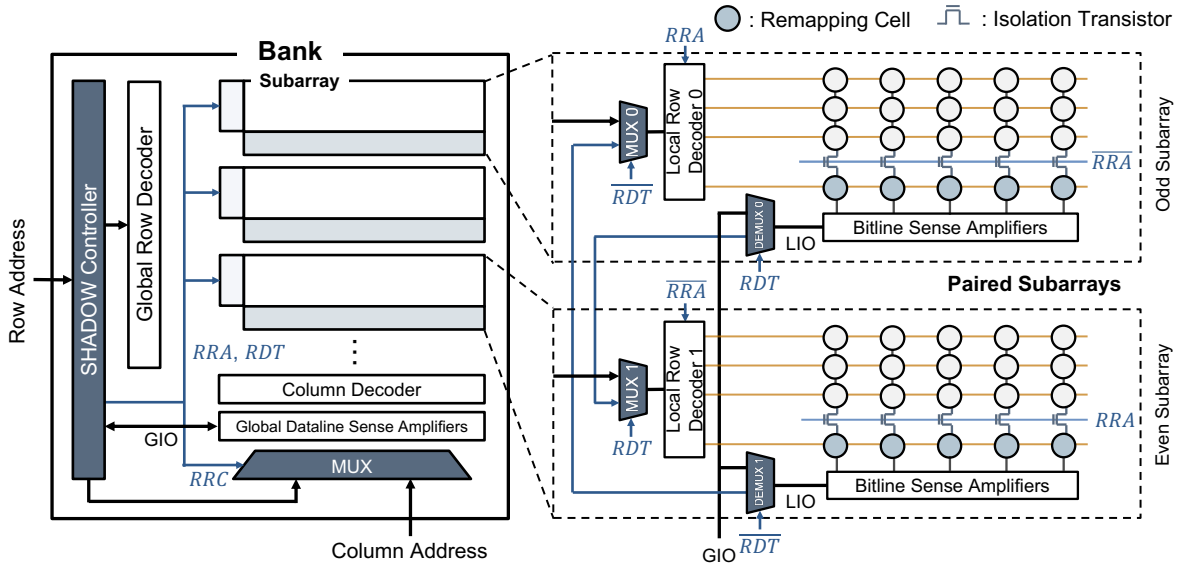

Fig. 5. Hardware architecture of SHADOW

randomness for SHADOW. Row_{empt} serves as a temporary buffer for an efficient row-shuffle with a dynamically changing DA location that is unreachable by the MC.

As depicted in Figure 4, two different row-copies are necessary for each row-shuffle operation at every RFM command. First, Row_{rand} is row-copied to Row_{empt}. Second, Row_{aggr} is copied to the previous DA location of Row_{rand}. Then, the previous DA location of Row_{aggr} now serves as Row_{empt}. After these movements, the remapping-row is updated accordingly to reflect the changed PA-to-DA location. This update preserves the functionality of SHADOW by redirecting the following memory accesses with the prior PAs to utilize the new DA locations. A series of row-shuffle operations leads to randomized PA-to-DA mapping, which changes dynamically at every RFM command.

C. Incremental Refresh

SHADOW also embodies a DA-based incremental refresh scheme to improve the protection capability by limiting the effective duration of an attack, especially on a frequently activated (possibly attacked) subarray. The incremental refresh executes an additional refresh on a row within the subarray where the row-shuffle is executed, in a round-robin fashion upon the DA at every RFM command. This technique shortens the effective refresh window for each row from tREFW (typically 64ms or 32ms) to less than a millisecond depending on the workload and SHADOW configuration. Therefore, this counterbalances the fact that the shuffling range of SHADOW is confined in a single subarray, which lowers the RH protection capability compared to a larger shuffling space, such as a whole bank [72]. The detailed impact on security is elaborated in Appendix XI. The performance overhead of the incremental refresh scheme is minimal because it only occurs once at every RFM command, and its latency is partially hidden via subarray-pairing (Section V-B).

D. Implications of Row-Shuffle in DRAM

When performing the row-shuffle operation, SHADOW relies on two distinct factors to achieve solid RH-protection capabilities. First, SHADOW thwarts the concentration of ACTs in a single row by distributing ACTs to multiple rows in the subarray where ACTs occur frequently through row-shuffling, similar to the wear leveling operation of flash memory [11], [59], [89]. Also, in a subarray with a high ACT frequency, the effective number of ACTs in each row is reset to zero within a short time due to the incremental refresh operation. Although the shuffled rows can be accessed again by accident, it is difficult for the effective number of ACTs in the row to reach H_{cnt} within that short duration before being reset. We analyze the RH-induced bit-flip probability against adversarial ACT patterns in Section VII-A.

V. Hardware Architecture of SHADOW

SHADOW holds the PA-to-DA remapping information of DRAM rows in an additional remapping-row, each specifically provisioned in every subarray. By confining the row-shuffling operation to a single subarray, SHADOW always tracks complete remapping information. We also propose subarray pairing, a microarchitectural optimization scheme that minimizes the timing overhead incurred when reading the remapping-row. Figure 5 describes the high-level architecture of SHADOW.

A. Remapping Row

SHADOW stores the PA-to-DA mapping information in an additional remapping-row of a DRAM subarray. The remapping-row stores the device addresses, and each is indexed by the physical address received from an MC. Based on the ordinary number of rows per subarray (512) and the size of a single row (1KB), only $513 \times 9\text{-bit} = 577\text{B}$ of storage is required per subarray as the row-shuffle of SHADOW is confined to a single subarray of 513 rows including Row_{empt}. We also store the incremental pointer of each subarray in

the remapping-row, occupying an additional 9 bits. Therefore, complete PA-to-DA mapping information can be comfortably held by a remapping-row 1KB in size. We implement the remapping-row to be inaccessible by the MC to prevent any contamination of the PA-to-DA mapping.

To minimize the latency of accessing the remapping-row, we place it closest to the row-buffer and adopt isolation transistors, which decouple a remapping-row from the rest of the ordinary rows (right of Figure 5). The ACT latency is primarily affected by the difference in capacitance between a bitline and a cell [51], [82] as a BLSA senses the change in the voltage level caused by the sharing of the cell charge with the bitline and amplifies it. An isolation transistor allows cells to share charges only with a short bitline whose capacitance is significantly lower than that of the original bitline when activating a remapping-row. Isolation transistors have already been used to share BLSAs across multiple bitlines in DRAM devices [21] and were exploited in previous studies as well [51], [54], [61], [91], one of which reported an area overhead of 0.8%, including the supporting circuitry [61]. For the open-bitline structure [30] in which a single subarray accesses two different row-buffers on the top and bottom, SHADOW can still preserve its access speed with two remapping-rows on both sides.

B. Subarray Pairing

To reduce the impact when accessing the remapping-row further, we propose a DRAM microarchitecture optimization technique, referred to here as *subarray pairing*. In SHADOW, every ACT accompanies an additional access to the remapping-row within the DRAM devices, increasing the ACT latency (tRCD). If the target row of an ACT and the corresponding remapping-row are located in the same subarray, the ACT for the target row must wait until the remapping-row is precharged. SHADOW pairs every two subarrays (even and odd subarrays) and places the remapping data of each subarray into the remapping-row of its paired subarray (see Figure 5). With a minor modification, we can concurrently activate a target row and a remapping-row that exist in different subarrays [46]. Then, a target-row ACT can be processed immediately after reading the remapping data without waiting for the PRE of the remapping-row in its pair. Therefore, the restore and PRE time of the remapping-row can be hidden by the ACT time of the target row.

Considering the open-bitline architecture in which two adjacent subarrays share a row buffer, SHADOW pairs two subarrays that sandwich another subarray between them (see Figure 5). Hereafter, the subarray containing the row to be activated by the MC or the row to be RH-mitigated is defined as SA_{target} (*target subarray*), and the subarray having the remapping row of the target subarray is defined as SA_{pair} (*paired subarray*).

SHADOW adds new signal paths to the subarrays to support subarray pairing. We add a remapping-row activation (RRA) signal to control the remapping-row ACT process that operates differently from other ACTs in the ordinary rows. The local row decoder receiving the RRA signal raises the wordline of the remapping-row, ignoring the address from the global row decoder. The inverse RRA signal turns off the isolation transistor to decouple the remapping-row from the long bitline. Moreover, to transfer the target row in the DA obtained from the remapping-row to SA_{pair}, we add new data paths between each subarray's LIO and the local row decoder of SA_{pair}. We put a DEMUX and a remapping data transfer (RDT) signal to select whether the LIO is connected to the newly added path (for remapping-row access) or the GIO (for ordinary access). An added MUX determines whether the DA is accepted from the remapping-row or whether the address is accepted from the global row decoder. The inverse RDT signal is used for SA_{pair} to avoid the simultaneous use of GIO.

C. SHADOW Controller

Each bank has a SHADOW controller that manages DRAM commands, addresses, and newly added signals according to the DRAM internal operation required for the target-row ACT and row-shuffling. First, this controller translates the row address in the PA of the ACT command transmitted from the MC to the corresponding column address in the remapping-row storing the target DA. This column address transfers to the column decoder through a MUX controlled by the remapping-row-column (RRC) signal. The SHADOW controller also manages random numbers for selecting Row_{aggr} and Row_{rand} for row-shuffling and has latches for temporarily storing their addresses during the row-shuffling operation. The RNG unit in each DRAM chip generates random numbers and securely buffers inside the SHADOW controller in advance to hide the random number generation latency. As default, cryptographically secure PRNG (CSPRNG) based on the PRINCE block cipher [8], [71], [77] is used. Further discussion, including the LFSR-based RNG option, is provided in Section VIII.

VI. Detailed Operation and Timing of SHADOW

In this section, we analyze the operational flows of SHADOW for normal DRAM access and row-shuffle operation. We also calculate their latencies with the corresponding timing parameters.

A. Operation and Timing on ACT

SHADOW affects the normal DRAM access latency because fetching the remapping-row for the PA-to-DA mapping information must precede the processing of every ACT command from an MC (see Figure 6(a)). It does not affect other DRAM commands, such as PRE, RD, or WR, because PRE does not require PA-to-DA mapping to precharge bitlines, and RD/WR targets data in the row buffer.

The required time interval between ACT and the following RD/WR command is defined as tRCD [35]. tRCD can be broken down into multiple steps: i) command and address (C/A) traversal, ii) global row decoding, iii) local row decoding, and iv) activated row sensing. SHADOW requires three additional steps between ii) and iii): remapping-row decoding, remapping-row sensing, and remapping information (the target

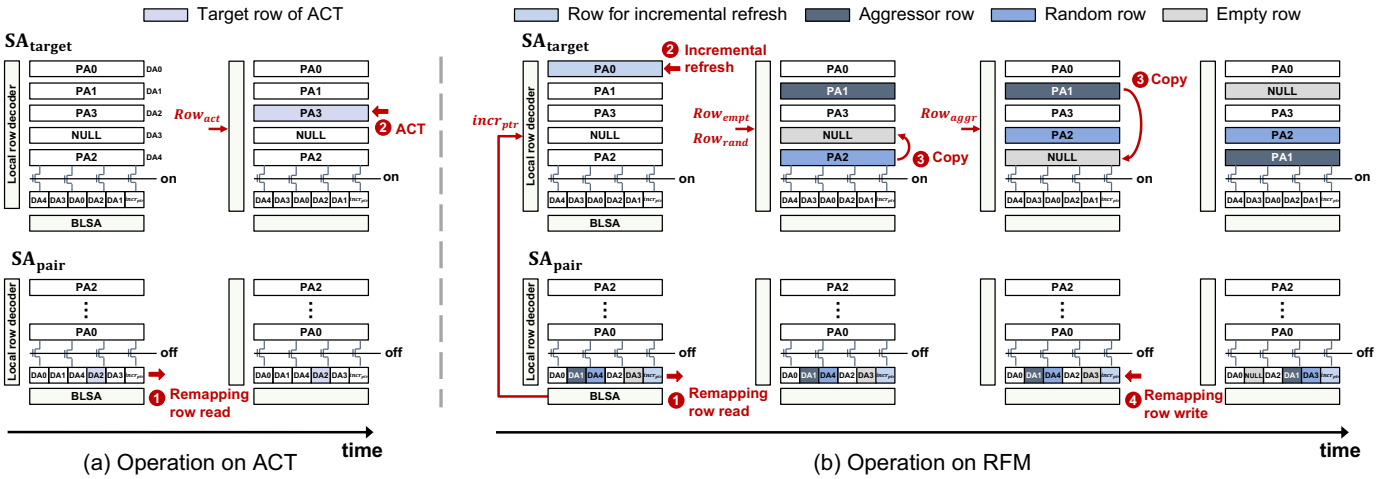

Fig. 6. Detailed operational flow of SHADOW for the target subarray (SA_{target}) and its paired subarray (SA_{pair}) upon ACT and RFM commands.

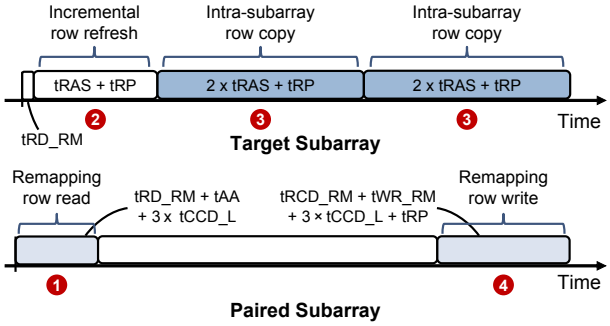

Fig. 7. SHADOW operation timing diagram for the row-shuffle target subarray (SA_{target}) and the corresponding paired subarray (SA_{pair}) in a the RFM command.

DA) traversal. We minimize the remapping-row decoding time using the RRA signal, as the location of the remapping-row is always fixed in a subarray. The remapping-row sensing time and the target DA traversal time are optimized using the isolation transistor and the newly added local data path between paired subarrays. We refer to this additional timing required to activate and read the remapping-row as tRD_RM, and an ACT command in SHADOW requires tRCD′ (a modified form of tRCD), which equals tRCD + tRD_RM.

B. Operation and Timing on RFM

We describe the detailed operation (Figure 6(b)) and timing (Figure 7) of the row-shuffle and incremental refresh operations of SHADOW, which are executed upon every RFM command.

① **Remapping-row read:** Before initiating any operation at RFM, the remapping-row of SA_{pair} must be read to acquire four types of DA information: $incr_{ptr}$, Row_{aggr}, Row_{rand}, and Row_{empt}. The latency of activating the remapping-row and reading the first value ($incr_{ptr}$ in DA) is tRD_RM. After reading and transmitting the first value to SA_{target}, three consecutive reads are executed with proper PA column addresses, reading Row_{aggr}, Row_{rand}, and Row_{empt} into the latches of the SHADOW controller (tAA+3×tCCD_L).

② **Incremental refresh:** After acquiring the $incr_{ptr}$ location, SA_{target} performs an incremental refresh by activating and precharging the pointed row. This process proceeds in parallel with the SA_{pair}, which is executing four additional reads. The incremental refresh ends with the additional (tRAS+tRP) time.

③ **Row-shuffle:** Following the incremental refresh, the row-shuffle operation in SA_{target} starts. It consists of two consecutive row copies. The DAs of Row_{aggr}, Row_{rand}, and Row_{empt} are already fetched by this time and stored in the SHADOW controller's latches during ①. The first row-copy step serves to sense the source row (Row_{rand}) to the row-buffer, which uses tRAS [75] to restore the row-buffer fully. After the source row is latched, the wordline of the destination row (Row_{empt}) is driven to write back the data in the row-buffer to the destination row, which again takes up to tRAS. After a single row-copy, bitlines are precharged (tRP), followed by the subsequent row-copy. Considering two row-copies, the overall execution time is (4×tRAS+2×tRP).

④ **Remapping-row write:** In SA_{pair}, the remapping-row must be updated. The DA values of $incr_{ptr}$, Row_{aggr}, Row_{rand}, and Row_{empt} are properly modified in the SHADOW controller and are written back to the remapping-row. Therefore, an additional internal activation for the remapping-row (tRCD_RM) is needed, followed by four consecutive writes (tWR_RM+3×tCCD_L) and an internal precharge (tRP) operation. tWR_RM refers to the write recovery time for the remapping-row. However, the latency of updating the remapping-row is *fully hidden* by the row-shuffle operation in SA_{target}.

In conclusion, the total execution time of the row-shuffle operation is (tRD_RM+tRAS+tRP+4×tRAS+2×tRP).

VII. Evaluation

A. Security Analysis

We analyze the protection capability of SHADOW by calculating the RH-induced bit-flip probability, which reveals the SHADOW configurations for near-complete protection. We stipulate that a probabilistic scheme achieves near-complete protection when the bit-flip probability of a DDR5-4800 rank with 32 banks is less than 1% per year, as in [44], [45], [62], [94]. The bit-flip probability is analyzed with regard

TABLE II
RH-INDUCED BIT-FLIP PROBABILITY OF SHADOW FOR A DRAM RANK WITHIN A YEAR. SHADOW'S BIT-FLIP PROBABILITY IS CALCULATED AS THE HIGHEST OF THE THREE ATTACK SCENARIOS. (SEE SECTION XI). SECURE CONFIGURATIONS FOR EACH H_{cnt} ARE MARKED IN BOLD.

RAAIMT	$H_{cnt} = 8K$	$H_{cnt} = 4K$	$H_{cnt} = 2K$
128	**2E-15**	4E-01	1
64	**2E-43**	**1E-14**	5E-01
32	**0**	**1E-43**	**9E-15**

to the bit-flip of *any victim row*, not a *specific victim row*. SHADOW prevents a bit-flip of a specific victim row more strongly than prior RH protection schemes that use static PA-to-DA mapping.

Defining adversarial patterns and attack scenarios: Upon our threat model, the attacker must rely on one of two different probabilities when forging an effective activation pattern against SHADOW. The former is the probability that the attacker guesses the new location of the shuffled row. The latter is the probability that there are multiple aggressors such that at least one of them does not become the target of a row-shuffle at every RFM until a bit-flip occurs.

Attack scenario I exploits the former, where the attacker activates a single row within an RFM interval (i.e., the period between two consecutive RFM commands) and alters the PA of the row to be activated in the same subarray when a new RFM interval begins. As any row in the same array can be affected by this type of attack, we calculate the bit-flip probability of this scenario while considering any bit-flip in the subarray. This approach is similar to the birthday-paradox attack pattern in [72]. We exclude the effect where the rows are refreshed due to ACT-PRE during a row-shuffle, thereby conservatively overestimating the possibility of a RH bit-flip.

Attack scenarios II and III take advantage of the latter probability, where the attacker activates multiple rows in the PA simultaneously. The attacker relies on the possibility that the activated aggressor will not become the row-shuffle target upon an RFM command. We again overestimated the RH bit-flip possibility by excluding cases in which the row-shuffle operation refreshes the victim row. The difference between scenarios II and III is that the former confines the aggressor rows in the PA within a single subarray and the latter activates aggressor rows in the PA across multiple subarrays.

SHADOW protection capability: Table II shows that SHADOW with an appropriate RAAIMT has sufficient protection capability in the form of a low bit-flip probability even when H_{cnt} is as low as 2K. The detailed mathematical formula for calculating the bit-flip probability considering the bank granularity and limited time interval (e.g., tREFW) for each attack scenario is provided in Section XI; the values in Table II are the expanded probabilities of the rank granularity and the one-year time period.

B. SPICE Circuit Simulation

We conducted a SPICE circuit simulation to measure the increased row activation time (tRCD′) and row-shuffle time.

TABLE III
TIMING VALUES OF SHADOW FROM THE SPICE CIRCUIT SIMULATION.

Definition	Abbreviation	Timing	Baseline	Ratio
Row activation in SHADOW	tRCD'	17.7ns	13.7ns	+29%
Row copy w/ precharge	-	73.9ns	-	-
Remapping-row sensing	tRCD_RM	2.3ns	13.7ns	-83%
Remapping-row write recovery	tWR_RM	9.0ns	11.8ns	-24%
Remapping-row read latency	tRD_RM	4.0ns	13.7ns	-71%

Methodology: We referenced the methodologies in the prior works [25], [54] and the open-source SPICE DRAM circuit models [54] to derive the sensing time of the remapping-row. We built a DRAM subarray using 55nm Rambus DRAM technology [68] with scaling to a 22nm process based on the ITRS roadmap [29]. We modeled the sense amplifier and isolation transistor using the PTM high-performance model and modeled access transistors using the PTM low-power model [66]. We also modeled the local wordline driver by referring to prior DRAM modeling work [88]. To calculate the timing for DA information reads from the remapping-row over to the paired subarray's local row decoder, we modeled the new data path using the wire resistance and capacitance data in [68], [84].

Simulation results: Table III summarizes the SPICE circuit simulation results. It shows that tRCD′, which contains the remapping-row ACT and RD time, is approximately 1.3× of the original tRCD. As mentioned in Section VI-A, tRCD′ is the time obtained by adding tRD_RM to the default tRCD. tRD_RM consists of the remapping-row decoding, remapping-row sensing, and remapping information traversal time. The decoding time in the local row decoder, which quickly turns on the wordline of the remapping-row with the RRA signal, is minimal at 0.33ns. The timing for remapping-row sensing (tRCD_RM) is less than 20% of the normal tRCD due to the isolation transistor. When the isolation transistor is turned off, the capacitance of the bitline for charge sharing with the remapping-row diminishes by more than 100×. Therefore, the sensing time of the remapping-row is 2.3ns, resulting in a total time for the remapping-row ACT of 2.63ns, whereas the baseline time is 13.7ns. Also, the remapping information (DA) traversal time is less than 10% of the baseline tAA. We conservatively set the DA traversal distance between two paired subarrays to half the bank size (i.e., half-height and half-width) by referring to Samsung's DDR4 [79]. In our SPICE simulation, the wire delay for DA traversal is less than 1ns. This result is reasonable considering that the data traversal time of normal RD in previous works is less than 40% of tAA and that the DA traversal distance is a quarter of the original distance [78], [82].

We also analyzed the timing for the row-shuffle precisely through a SPICE simulation. Inside the row-copy operation [75], we initially set the time required to copy the source row into the row-buffer as tRAS to restore the row-buffer fully, which is a conservative assumption. When SHADOW copies the value from the fully restored row-buffer to a destination row, the necessary time is 0.55×tRAS, based on our SPICE simulation considering the small capacitance of the destination

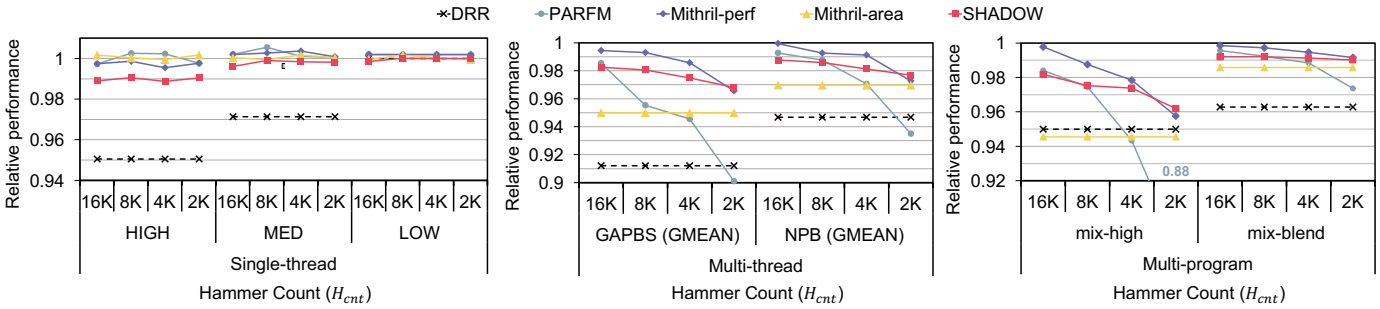

Fig. 8. The relative performance of SHADOW, PARFM, Mithril-perf, Mithril-area, and the double refresh rate (DRR) compared to the baseline (the system without RH prevention) on single-threaded SPEC CPU2017, multi-threaded GAPBS, and NPB applications, and multi-programmed SPEC CPU2017 on an actual system. HIGH, MED, and LOW indicate the average of the spec-high, spec-med, and spec-low applications, respectively. The performance represents the weighted speedup for multi-programmed workloads, while meaning the reciprocal of the execution time for single-threaded and multi-threaded workloads.

row compared to the bitline and row-buffer. Therefore, the total operation time of the row-shuffle described in Section VI-B is revised as (tRD_RM + tRAS + tRP + 3.1×tRAS + 2×tRP), being 178ns and 186ns in the DDR4-2666 and DDR5-4800 configurations, respectively.

C. Performance Overhead Analysis

Our performance overhead analysis is two-fold: i) an actual system environment experiment that emulates an RFM interface and ii) an experiment based on an architectural simulator. The former allows us to understand the overhead of SHADOW and other RFM-based schemes (PARFM and Mithril [44]) on an actual system, which may be more capable of hiding the overhead of RFM commands or the modified tRCD values. The latter enables comprehensive comparisons with additional state-of-the-art previous works (Mithril [44], BlockHammer [94], and RRS [72]).

Methodology: Table IV summarizes the actual system configuration used for the experiments. Because a processor or a DRAM device supporting RFM commands is not yet publicly available, we emulated the RFM commands by increasing the frequency of conventional refreshes (by reducing tREFI). We modified the DRAM timing parameters (tRCD and tREFI) to emulate SHADOW using a modified Intel Performance Counter Monitor (PCM) tool. For the architectural simulations, we use McSimA+ [2] in a configuration matching an actual system environment, except for a change to DDR5-4800.

We used SPEC CPU2017 [15], GAPBS [5], and NAS Parallel Benchmark suites (NPB) [4]. We categorized the applications in SPEC CPU2017 into three groups based on their memory access frequency: spec-high (bwaves, fotonik3d, lbm, mcf, wrf), spec-med (deepsjeng, gcc, xz), and spec-low (exchange2, imagick, leela). We constructed two multi-programmed workloads, mix-high consisting of 14 spec-high applications and mix-blend consisting of 14 applications uniformly chosen from spec-high, spec-med, and spec-low. Also, we used GAPBS traversing a Kronecker graph with 2^{26} vertices and NPB with class-C for multi-threaded workloads. For the simulation, we used mix-high and mix-blend workloads to populate the threads to 16 and 32 newly constructed mix-random workloads created by randomly selecting 16 SPEC CPU2017 applications. We used the inverse of the execution time and the weighted speedup metric [18] to compare the performance of single-/multi-threaded workloads and multi-programmed workloads, respectively.

TABLE IV
ACTUAL MACHINE SYSTEM CONFIGURATION

Resource	Value
Processor	Intel Core i9-7940X @ 3.10 GHz
Org.	1 socket, 14 cores
L3 $	19.25 MB, Shared
Main memory	4 channels, 1 DIMM per channel
DIMM type	DDR4-2666, 2 ranks, 16 GB
Timings (cycles)	19-19-19 (tCL-tRCD-tRP)
	467 (tRFC), 10400 (tREFI)

We compared the SHADOW with PARFM, Mithril, RRS, and BlockHammer. ***PARFM*** [44] is a probabilistic RH-protection scheme (PARA [45]) with RFM. It performs TRR on a randomly sampled row whenever RFM commands are issued. We used the RAAIMT values of SHADOW and PARFM, resulting in a RH-induced bit-flip probability of 1% per year. ***Mithril*** [44] uses the CAM structure based on the Counter-based Summary (CbS) algorithm to track the ACT counts and the row addresses efficiently. Mithril has the flexibility of setting different tracking table sizes and RAAIMT values for a given target H_{cnt}. We use two different classes of configurations for Mithril: performance-optimized Mithril (***Mithril-perf***) with 10KB per bank CAM table, which is expensive in terms of area and power for DRAM manufacturing technology [17], [24], and area-optimized Mithril (***Mithril-area***) with RAAIMT set to 32. ***BlockHammer*** [94], a representative throttle-based solution, is configured based on the memory access profiling results of the workloads and the target H_{cnt}. ***RRS*** [72], a state-of-the-art row-shuffle-based RH mitigation scheme, is favorably configured with a row-swap threshold of ⅙ of H_{cnt}. We consider the blast radius of 3 as the baseline; a recent study [43] reported that the blast radius could be as wide as 6.

RFM command emulation: We emulated the RFM interface by replacing RFM commands with additional auto-refresh commands. To calculate the desired number of auto-refreshes, we calculated the number of RFM operations during tREFW (N_{RFM}) by dividing the measured ACT counts by RAAIMT.

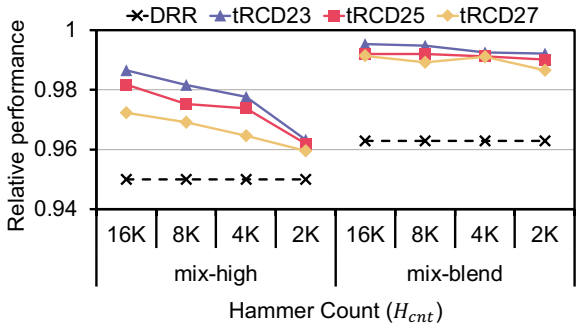

Fig. 9. Relative performance (weighted speedup) of SHADOW with various tRCD parameters, tRCD23, tRCD25, and tRCD27, on mix-high and mix-blend multi-programmed workloads when varying H_{cnt} from 16K to 2K. Each weighted speedup is normalized to the baseline (tRCD19, the default tRCD parameter).

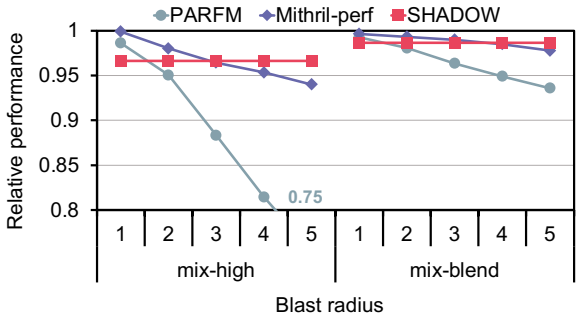

Fig. 10. Relative performance (weighted speedup) of SHADOW when the blast radius ranges from 1 to 5 on mix-high and mix-blend multi-programmed workloads. RAAIMT is fixed at 2K. Each weighted speedup is normalized to the baseline (system without RH mitigation).

Then, we increased the number of additional refreshes during tREFW (N_{REF}) by reducing the tREFI parameter. We obtained the necessary value of tREFI′ for each experiment through N_{RFM} and N_{REF} as the following equation:

$$tREFI' = tREFI \times \frac{tRFC}{tRFC + tRFM \times N_{RFM}/N_{REF}} \quad (1)$$

Actual system experimental results: Figure 8 depicts the relative performance of SHADOW on multi-programmed, single-threaded SPEC CPU2017 workloads and memory-intensive multi-threaded workloads (GAPBS and NPB) in comparison with the PARFM, Mithril, and double refresh rate (DRR) approaches. First, when the system executes the single-threaded applications, SHADOW and the other schemes rarely increase the execution time. Despite the longer tRCD of SHADOW, the impact of SHADOW on the system performance is insignificant, being less than 2% even on spec-high. Even when H_{cnt} decreases, the change in such a trend is small because single applications incur neither a large number of ACTs nor RFM commands. With multi-threaded and multi-programmed workloads, SHADOW demonstrates comparable or superior performance compared to PARFM, especially when H_{cnt} is low. For Mithril-perf, the performance overhead is smaller than that of SHADOW, but it requires around 10KB CAM per DRAM bank. Lastly, the gap between Mithril-area and SHADOW shrinks at a lower H_{cnt}, but the required CAM structure of Mithril-area reaches 5KB per bank when H_{cnt} is 2K. SHADOW generally shows less performance overhead due to its lower RAAIMT than PARFM and shows comparable performance compared to Mithril-area/perf.

Even under the worst-case synthetic, adversarial pattern of performance that is not observed in actual applications, SHADOW ensures robustness with less than 3% and 9% performance degradation compared to the baseline with and without the RFM command, respectively. The two main factors that degrade the performance of SHADOW are the increase in tRCD and the periodic RFM command. Thus, we set a random stream microbenchmark that issues frequent activations as the baseline. It is sensitive to tRCD changes and can frequently trigger RFM. Upon this, SHADOW's tRCD increase results in a performance degradation of only 3%. Moreover, even when the effect of the theoretically most frequent RFM commands is imposed, calculated based on the maximum possible number of activations under the JEDEC interface, less than 9% performance degradation is observed. Such an impact of the RFM is not only applied to SHADOW to but all RFM-compatible schemes.

tRCD sensitivity in SHADOW: We conducted a sensitivity study of the tRCD timing parameters by changing the tRCD values to 23 tCK and 27 tCK from SHADOW's default value of 25 tCK, where tCK is the DRAM cycle time (0.75ns for DDR4-2666). Figure 9 illustrates the normalized weighted speedup in each case with the baseline parameter of 19 tCK tRCD without SHADOW (see Table IV). The performance effect of a higher tRCD value is noticeable when H_{cnt} is 16K, especially in the memory-intensive mix-high case. However, when H_{cnt} is as low as 2K, the effect of a different tRCD shrinks due to the more frequent RFM commands that are issued. In all cases, the performance overhead of SHADOW is less than 4%, which is much lower than doubling the refresh rate.

Blast radius sensitivity in SHADOW: SHADOW has less performance degradation than previous RFM-compatible schemes at a high blast radius. Figure 10 shows the performance overhead of SHADOW, PARFM, and Mithril according to the blast radius. When the blast radius is greater than two, SHADOW's performance surpasses those of the other schemes.

Architectural simulation results: Figure 11 shows that SHADOW demonstrates robust performance in a wide range of workloads and H_{cnt} values, which is better than other schemes when H_{cnt} is less than 4K. In particular, the other row-shuffle-based RH mitigation scheme (RRS) blocks the entire memory channel to perform row-shuffles, which incurs high performance overhead, as in [53], [74]. SHADOW's row-shuffle operation utilizes the DRAM chip's internal bandwidth, which is more efficient, especially when frequent row-shuffles occur at a low value of H_{cnt}. BlockHammer also incurs high overhead with a low H_{cnt}, as the necessary delay becomes too long [44], [67] and the possibility of misidentifying a normal memory access also increases.

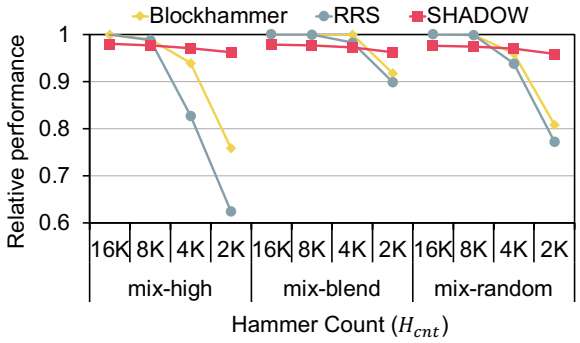

Fig. 11. Relative performance (weighted speedup) of SHADOW with BlockHammer [94] and RRS [72] in the mix-high, mix-blend, and 32 mix-random cases (which consists of randomly chosen SPEC CPU 2017 applications) when sweeping H_{cnt} from 16K to 2K. Each weighted speedup is normalized to the baseline (no RH mitigation).

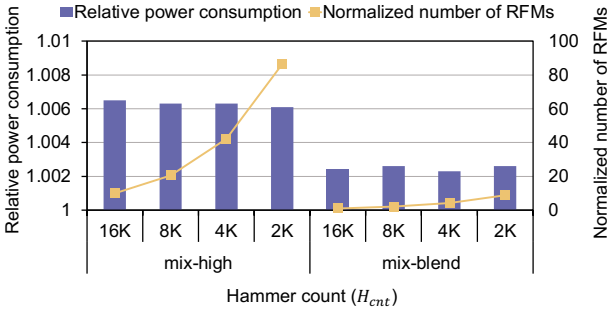

Fig. 12. Relative system-level power consumption of SHADOW compared to the baseline and the number of RFMs normalized to the number of refreshes of the baseline on mix-high and mix-blend multi-programmed workloads across varying H_{cnt} values ranging from 16K to 2K.

D. Area and Power Analysis

Methodology: To analyze the area and power overhead of SHADOW, we designed the SHADOW controller and additional logic in each subarray using Verilog. We synthesized them with CMOS 40nm technology using the Synopsys Design Compiler [48]. We scaled the area overhead of the logic in SHADOW to a 22nm DRAM process, considering that a DRAM process is 10× less dense than an ASIC process due to the inferior driving current and fewer metal layers [17], [24]. We calculated the DRAM power using the Micron DDR4 calculator [56] and by referring to the Micron DDR5 datasheet [57]. We regarded the TDP (Thermal Design Power) of the CPU as the power of the processor in our experiments.
Synthesis results: The total area overhead of the SHADOW logic in a DRAM chip is 0.35mm^2, which is equivalent to 0.47% of a DDR5 chip [42], and the DRAM capacity overhead for additional rows is 0.6% considering empty rows and remapping-rows. These additional rows are feasible given the existing spare-row implementation scheme [40], [81]. The per-bank SHADOW controller includes an ACT counter, six 9-bit row address latches, a 7-bit subarray index latch, a MUX for a column decoder, and a control logic component. In each subarray, a MUX and a DEMUX are included. We assumed that the area overhead of three additional wire paths for the signals would be negligible compared to the numbers of wordlines and bitlines in a subarray. Lastly, SHADOW utilizes one PRINCE block cipher-based RNG unit per chip [8], [71], [72], which occupies less than 0.1% of the DRAM chip area, even when using the DRAM process.

The power consumption of SHADOW is negligible on the system-wide scale, being less than 0.63% even when H_{cnt} is 2K and with a memory-intensive workload. Figure 12 shows the relative system-level power consumption of SHADOW compared to that of the baseline system, with both calculated based on the number of memory commands measured in the actual system environment (Table IV). The number of issued RFM commands increases as H_{cnt} decreases. However, the fact that the overall power consumption of SHADOW changes minimally suggests that it is dominated by the additional remapping-row accesses rather than the row-shuffle operations.

VIII. Discussion

Hardware RNG unit in DRAM: We can implement the RNG unit with negligible overhead based on block ciphers, as in [8], [38], [71], [72]. The required per-chip random number throughput of SHADOW is 126Mbits/s, with 4K H_{cnt} and a conservative 32-bit random number for Row_{aggr} and Row_{rand} in each case. The PRINCE block cipher can generate more than a Gbit/sec of random numbers, even under the slow DRAM core frequency. It can also be transferred via a 1-bit signal path in advance and buffered at each bank's SHADOW controller. Additionally, a linear-feedback shift register-based (LFSR) RNG with its seed being periodically randomized can also be used to ensure much lower area/power overhead [64], [97]. Recent DDR5 memory chips are already equipped with LFSR to generate read training patterns [35].

The security of the RNG unit also affects the security of SHADOW. We can enhance the RNG unit with boot-time or periodic key and counter initialization strategies utilizing CPU-side true RNG units.

Soft post package repair (sPPR): Since DDR4, the JEDEC interface [33], [35] defines an sPPR mechanism that allows an OS or a user to replace a faulty row address during runtime. The fact that the tRCD value is unmodified even after sPPR implies that there is a low-overhead address relocation path that is reusable by SHADOW. Also, the number of possible sPPR replacements per bank-group has continually increased over recent generations [70], suggesting that SHADOW can exploit such resources in the future or even provide the required mechanism for such an enhanced sPPR.

Optimizing RFM interface with a filtering counter structure: Earlier suggested efficient memory-access filtering structures can be orthogonally adopted to SHADOW by optimizing the RFM interface. Prior works such as BlockHammer [94] and Hydra [67] utilize efficient random projection-based counter structures [60] such as a dual-counting Bloom filter (D-CBF) or a group-count table (GCT), which allows them to filter out the majority of the memory accesses from the normal workloads. When such a filtering process exists before issuing the RFM command, it has the potential greatly to decrease the number of unnecessary RFM issues and thus mitigate the ensuing performance degradation.

IX. Related Work

Tracker-based RH protection schemes: Graphene [62] exploits a CAM structure based on the Misra-Gries [58] algorithm to track the number of ACTs to each DRAM row efficiently. RRS [72] also uses the Misra-Gries tracker and a row indirection table of a matching size, which is optimized by a collision-avoidance-table [71]. BlockHammer [94] uses a dual-counting Bloom filter [12] to track the ACT counts of each row. Mithril [44] and PROTRR [55] are two concurrent DRAM-side approaches that both use a modified Misra-Gries tracker and an RFM interface. However, under a low H_{cnt} and a broad blast radius, such SRAM- or CAM-based structures incur the scalability issue.

Tracker-less RH protection schemes: Panopticon [6] employs a per-row ACT counter by modifying two MAT structures per subarray. However, its TRR-based RH mitigation scheme is inefficient against blast-attacks compared to row-shuffle. PARA [45] uses a stateless logic component to trigger TRR with a low probability whenever an ACT command is issued. However, the performance overhead is exacerbated with high sensitivity under a low H_{cnt}.

X. Conclusion

We have proposed SHADOW, an RFM-compatible in-DRAM row-shuffle-based Row Hammer mitigation solution that provides superior protection capability against the non-adjacent Row Hammer attacks. SHADOW adopts the newly introduced JEDEC RFM interface to trigger row-shuffle operations in DRAM without adding new DRAM commands. The row-shuffle operations of SHADOW dynamically randomize the physical adjacency between any row pair within a DRAM subarray so that it efficiently prevents non-adjacent Row Hammer attacks and attackers from generating sophisticated RH attack patterns. By populating an additional DRAM row per subarray, SHADOW efficiently tracks all of the remapping information of shuffled DRAM rows. The proposed SHADOW optimizes the access latency to the remapping information by leveraging and repurposing the DRAM internal circuitry. SHADOW identifies the row to shuffle without an additional tracking table, which would otherwise incur high area overhead. Our evaluation shows that SHADOW provides superior protection against non-adjacent Row Hammer attacks in comparison with other Row Hammer mitigation schemes and incurs little area/energy overhead.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-01300), the National Research Foundation of Korea (NRF) grant by the Korea government (MSIT) (NRF-2022R1F1A1062826), and NSF (CNS 1705047). Nam Sung Kim has a financial interest in Samsung. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Minbok Wi, Jaehyun Park, Seoyoung Ko, and Michael Jaemin Kim are with the Department of Intelligence and Information, Seoul National University (SNU), Seoul, South Korea. Jung Ho Ahn, the corresponding author, is with the Department of Intelligence and Information and the Interdisciplinary Program in Artificial Intelligence, SNU, Seoul, South Korea.

XI. Appendix: RH-induced Bit-Flip Probability of Three Attack Scenarios for SHADOW

The number of rows in a single subarray is denoted as N_{row} (e.g., 512). We also define W_{sum} as the weighted sum of the effects all aggressors within the blast range can have on a victim, for which we set to 3.5 as the default value.

Attack scenario I: We represent scenario I by exploiting the buckets and balls model [72]; rows in the subarray and attack rounds are represented as buckets and balls, respectively. The number of buckets is N_{row}, as its definition suggests. Meanwhile, the number of balls is also N_{row}, due to the incremental refresh technique that limits the duration of attack scenario I under an incremental refresh window (i.e., N_{row} number of RFM commands). We define M_1 as the minimum number of RFM intervals required to activate the aggressor rows for an attacker to cause RH bit-flips on a victim row. At this point, the bit-flip probability of SHADOW for scenario I, denoted as P_1, can be described as the probability of one or more buckets containing M_1 balls when throwing N_{row} balls into N_{row} buckets. The success probability of one trial (p) is W_{sum}/N_{row}. We determine P_1 conservatively as follows using the Bernoulli trial method:

$$P_1 = N_{row} \times \binom{N_{row}}{M_1} \times p^{M_1} \times (1-p)^{N_{row}-M_1} \tag{2}$$

Attack scenario II: We can create a recurrence formula for the bit-flip probability of scenario II, denoted as $P_2[n]$, where n stands for the n-th RFM command. We denote the number of different aggressor rows that are activated as $\mathrm{N_{Aggr}}$. We also define m as $\mathrm{RAAIMT/N_{Aggr}}$, which means that m ACTs are executed for each aggressor during a single RFM interval. Similar to scenario I, M_2 denotes the necessary number of RFM commands by which an aggressor can evade a SHADOW row-shuffle to cause an RH bit-flip from the perspective of any single aggressor. The recurrence equation for $P_2[n]$ is as follows:

$$P_2[n] = P_2[n-1] + (1 - P_2[i - M_2 - 1]) \frac{1}{N_{Aggr}} \left(1 - \frac{1}{N_{Aggr}}\right)^{M_2} \tag{3}$$

We sweep all possible variations of $\mathrm{N_{Aggr}}$, from 1 to RAAIMT. The incremental refresh of SHADOW sets an additional constraint for a successful attack; $m \times N_{row} < H_{cnt}$. This creates an upper bound of M_2, decreasing the failure possibility. We calculate $P_2[N_{row}]$ and conservatively report $N_{Aggr} \times P_2[N_{row}]$, finding that there are $\mathrm{N_{Aggr}}$ different aggressor rows.

Attack scenario III: We again define M_3 with the same meaning, which is also expressed as H_{cnt}/m. The recurrence equation for the bit-flip probability of scenario III, $P_3[n]$, is identical to Equation 3 only with M_2 replaced by M_3. We conservatively ignore the possible benefit of an incremental refresh. With the last $P_3[last]$ bit-flip probability in a tREFW window calculated, we again conservatively report $N_{Aggr} \times P_3[last]$.

REFERENCES

[1] M. T. Aga, Z. B. Aweke, and T. Austin, "When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2017.

[2] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *ISPASS*, 2013.

[3] "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors," https://www.amd.com/system/files/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf, AMD, 2013.

[4] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, and M. Yarrow, "The nas parallel benchmarks 2.0," Technical Report NAS-95-020, NASA Ames Research Center, Tech. Rep., 1995.

[5] S. Beamer, K. Asanović, and D. Patterson, "The GAP Benchmark Suite," *arXiv preprint arXiv:1508.03619*, 2015.

[6] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," in *Workshop on DRAM Security (DRAMSec)*, 2021.

[7] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of Rowhammer: Flipping Secret Exponent Bits using Timing Analysis," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[8] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE–A Low-Latency Block Cipher for Pervasive Computing Applications," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2012.

[9] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

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

[11] L.-P. Chang, "On Efficient Wear Leveling for Large-Scale Flash-Memory Storage Systems," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, 2007.

[12] M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," in *International Colloquium on Automata, Languages, and Programming*, 2002.

[13] 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 *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[14] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[15] S. P. E. Corp., "SPEC CPU2017," https://www.spec.org/cpu2017.

[16] 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.

[17] F. Devaux, "The true Processing In Memory accelerator," in *2019 IEEE Hot Chips 31 Symposium*, 2019.

[18] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.

[19] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[20] P. Frigo, E. Vannacc, 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 *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[21] J. C. Gealow, "Impact of Processing Technology on DRAM Sense Amplifier Design," Ph.D. dissertation, Massachusetts Institute of Technology, 1990.

[22] 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 *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[23] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.

[24] T. R. Halfhill, "Upmem Nails RowHammer," *Microprocessor Report, The Linley Group*, 2021.

[25] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu, "CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," in *ISCA*, 2019.

[26] 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.

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

[28] "Intel Xeon Processor 7500 Series," https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-processor-7500-series-vol-2-datasheet.pdf, Intel, 2010.

[29] ITRS, "ITRS Reports," http://www.itrs2.net/itrs-reports.html.

[30] B. Jacob, D. Wang, and S. Ng, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2010.

[31] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, 2017.

[32] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[33] JEDEC, "DDR4 SDRAM Standard," 2017.

[34] JEDEC, "LPDDR5 Standard," 2019.

[35] JEDEC, "DDR5 SDRAM," 2020.

[36] JEDEC, "Near-Term DRAM Level RowHammer Mitigation," 2021.

[37] JEDEC, "System Level RowHammer Mitigation," 2021.

[38] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "CSI:Rowhammer-Cryptographic Security and Integrity against Rowhammer," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[39] I. Kang, E. Lee, and J. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, 2020.

[40] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*. John Wiley & Sons, 2007, vol. 13.

[41] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *IEEE Computer Architecture Letters*, 2014.

[42] D. Kim, M. Park, S. Jang, J.-Y. Song, H. Chi, G. Choi, S. Choi, J. Kim, C. Kim, K. Kim, K. Koo, S. Song, Y. Kim, D. U. Lee, J. Lee, D. Kim, K. Kwon, M. Han, B. Choi, H. Kim, S. Ku, Y. Kim, J. Kim, S. Kim, Y. Seo, S. Oh, D. Im, H. Kim, J. Choi, J. Chung, C. Lee, Y. Lee, J.-H. Cho, J. Chun, and J. Oh, "A 1.1V 1ynm 6.4 Gb/s/pin 16Gb DDR5 SDRAM with a Phase-Rotator-Based DLL, High-Speed SerDes and RX/TX Equalization Scheme," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2019, pp. 380–382.

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

[44] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022.

[45] 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.

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

[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] P. Kurup and T. Abbasi, *Logic Synthesis Using Synopsys®*. Springer Science & Business Media, 2012.

[49] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory without Accessing Them," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[50] D. Lee, D. Jung, I. Fang, C.-C. Tsai, and R. Popa, "An Off-Chip Attack on Hardware Enclaves via the Memory Bus," in *USENIX Security*, 2020.

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

[52] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. Ahn, "TWiCe: Preventing Row-hammering by Exploiting Time Window Counters," in *ISCA*, 2019.

[53] S. Lee, K. Lee, M. Sung, M. Alian, C. Kim, W. Cho, R. Oh, S. O, J. Ahn, and N. Kim, "3D-Xpath: High-Density Managed DRAM Architecture with Cost-Effective Alternative Paths for Memory Transactions," in *PACT*, 2018.

[54] H. Luo, T. Shahroodi, H. Hassan, M. Patel, A. Yağlıkçı, L. Orosa, J. Park, and O. Mutlu, "CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off," in *ISCA*, 2020.

[55] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "PROTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[56] Micron, "Calculating Memory Power for DDR4 SDRAM," 2017.

[57] Micron, "16Gb DDR5 SDRAM Addendum," 2021. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/16gb_ddr5_sdram_diereva.pdf

[58] J. Misra and D. Gries, "Finding Repeated Elements," *Science of Computer Programming*, 1982.

[59] M. Murugan and D. H. Du, "Rejuvenator: A Static Wear Leveling Algorithm for NAND Flash Memory with Minimized Overhead," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies*, 2011.

[60] S. Muthukrishnan, *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005.

[61] S. O, Y. H. Son, N. S. Kim, and J. Ahn, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *ISCA*, 2014.

[62] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *MICRO*, 2020.

[63] M. Patel, J. S. 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.

[64] A. Perais and A. Seznec, "Practical Data Value Speculation for Future High-end Processors," in *HPCA*, 2014.

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

[66] PTM, "Predictive Technology Model," http://ptm.asu.edu/.

[67] 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.

[68] Rambus, "DRAM Power Model," http://www.rambus.com/energy, 2010.

[69] 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.

[70] R. Rooney and N. Koyle, "Micron® DDR5 SDRAM: New Features," *Micron Technology Inc., Tech. Rep*, 2019.

[71] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design," in *USENIX Security*, 2021.

[72] 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.

[73] S. Saroiu, A. Wolman, and L. Cojocar, "The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses," in *2022 IEEE International Reliability Physics Symposium (IRPS)*, 2022.

[74] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime," in *MICRO*, 2022.

[75] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[76] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating Wordline Crosstalk using Adaptive Trees of Counters," in *ISCA*, 2018.

[77] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, 2002.

[78] H.-C. Shih, P.-W. Luo, J.-C. Yeh, S.-Y. Lin, D.-M. Kwai, S.-L. Lu, A. Schaefer, and C.-W. Wu, "DArT: A Component-Based DRAM Area, Power, and Timing Modeling Tool," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2014.

[79] K. Sohn, T. Na, I. Song, Y. Shim, W. Bae, S. Kang, D. Lee, H. Jung, S. Hyun, H. Jeoung, K.-W. Lee, J.-S. Park, J. Lee, B. Lee, I. Jun, J. Park, J. Park, H. Choi, S. Kim, H. Chung, Y. Choi, D.-H. Jung, B. Kim, J.-H. Choi, S.-J. Jang, J.-B. Lee, and J. S. Choi, "A 1.2 V 30 nm 3.2 Gb/s/pin 4 Gb DDR4 SDRAM with Dual-Error Detection and PVT-Tolerant Data-Fetch Scheme," *IEEE Journal of Solid-State Circuits*, 2012.

[80] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *2017 54th ACM/IEEE Design Automation Conference (DAC)*, 2017.

[81] Y. H. Son, S. Lee, S. O, S. Kwon, N. S. Kim, and J. Ahn, "CiDRA: A Cache-inspired DRAM Resilience Architecture," in *HPCA*, 2015.

[82] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.

[83] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2018.

[84] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. P. Jouppi, "CACTI 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[85] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[86] 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[87] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. Padmanabha Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-based Rowhammer Attacks on ARM," in *In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2018.

[88] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.

[89] C. Wang and W.-F. Wong, "Observational Wear Leveling: an Efficient Algorithm for Flash Memory Management," in *2012 49th ACM/IEEE Design Automation Conference (DAC)*, 2012.

[90] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[91] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi, and O. Mutlu, "Figaro: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching," in *MICRO*, 2020.

[92] 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.

[93] A. G. Yağlıkçı, J. S. Kim, F. Devaux, and O. Mutlu, "Security Analysis of the Silver Bullet Technique for RowHammer Prevention," *arXiv preprint arXiv:2106.07084*, 2021.

[94] A. G. Yaglikci, M. Patel, J. Kim, R. AziziBarzoki, J. Park, H. Hassan, A. Olgun, L. Orosa, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021.

[95] 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.

[96] K. S. Yim, "The Rowhammer Attack Injection Methodology," in *IEEE 35th Symposium on Reliable Distributed Systems*, 2016.

[97] S. F. Yitbarek, M. T. Aga, R. Das, and T. Austin, "Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors," in *HPCA*, 2017.

[98] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-hammering Based on Memory Locality," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[99] 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.