2026 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

MIRZA: Efficiently Mitigating Rowhammer with Randomization and ALERT

Hritvik Taneja
Georgia Tech
htaneja3@gatech.edu

Ali Hajiabadi
ETH Zurich
ahajiabadi@ethz.ch

Michele Marazzi
ABB Research
michele.marazzi@ch.abb.com

Kaveh Razavi
ETH Zurich
kaveh@ethz.ch

Moinuddin Qureshi
Georgia Tech
moin@gatech.edu

***Abstract*—In-DRAM Rowhammer mitigation requires three resources: *space* (to track aggressor rows), *time* (to perform mitigation), and *energy* (to refresh victim rows). An ideal in-DRAM mitigation must minimize all three overheads. Recent randomized trackers, such as MINT, can perform tracking with negligible storage overheads. However, they perform mitigation *proactively* and frequently, which incurs significant performance and energy overheads at low thresholds. Recently, JEDEC introduced *Per-Row Activation Counters (PRAC)* and *ALERT Back Off (ABO)* protocol to obtain the time for mitigation *reactively*, as needed. While PRAC+ABO minimizes the time and energy overheads of mitigation, PRAC incurs significant changes to the DRAM array and significant performance overhead (6.5% on average) due to increased memory timings to update the PRAC counters. Our goal is to develop an efficient in-DRAM mitigation that has low storage, performance, and energy overheads.**

Our paper proposes *MIRZA*, the first low-cost *reactive* in-DRAM mitigation. MIRZA relies on MINT to track aggressor rows. However, instead of *proactively* doing mitigation at regular intervals (via REF or RFM), MIRZA uses ABO to *reactively* obtain the time required for mitigation. To avoid frequent ABO, MIRZA employs *Coarse-Grained Filtering* to disable mitigations if the activation count is below a certain *Filtering Threshold*. To tolerate a threshold of 1K, MIRZA requires a storage overhead of only 196 bytes of SRAM per bank. Compared to MINT, MIRZA reduces the mitigation overheads by 28.5x. Compared to PRAC, MIRZA has 45x lower area overheads and negligible slowdown (0.36% average slowdown vs. 6.5% for PRAC).

I. Introduction

Rowhammer is a disturbance error that occurs when rapid activations of a DRAM row cause bit-flips in neighboring rows [20]. The *Rowhammer Threshold (TRH)*, which is the number of activations required to induce a bit-flip, has continued to decrease, lowering from 140K [20] (in 2014) to 4.8K [17] (in 2020). Due to the lack of publicly available characterization data for DDR5 modules, the current and future trend of the Rowhammer threshold is less clear. However, research on architectural solutions against lower Rowhammer thresholds is still vital as it can mitigate the risk posed by low-threshold devices if and when such devices arrive. This is a preferable approach rather than waiting for devices to be attack and then develop a design for vulnerable devices.

Typical mitigation for Rowhammer relies on *tracking* to identify the aggressor rows and *refreshing* the victim rows [10]. We focus on in-DRAM mitigations, as they can solve Rowhammer transparently. In-DRAM mitigation requires three types of resources: storage for tracking, and time and energy for performing mitigation.

The Storage Overhead: In-DRAM mitigation requires storage for tracking aggressor rows. DDR4 modules were equipped with *Targeted Row Refresh (TRR)* tracker with 4-28 entries [12] per bank. However, TRR is not secure and can be easily broken with specialized patterns [7], [12]. Optimal in-DRAM trackers, such as Mithril [19] and ProTRR [24], provide sufficient entries to track all aggressor rows for a given threshold, however, they require significant storage overheads (for example, 4.5KB CAM per bank at TRH of 1K). Recent works, such as PrIDE [11] and MINT [35], propose principled randomized in-DRAM trackers that can securely mitigate Rowhammer while incurring negligible SRAM overheads. For example, MINT is a single-entry tracker that selects one activated row (using uniform random sampling) between two refreshes (see Figure 2 for more details). Unfortunately, randomized Rowhammer solutions require frequent mitigations.

The Mitigation Overhead: In-DRAM trackers require time to perform mitigative refreshes on the victim rows associated to an aggressor row. To transparently perform victim refresh, in-DRAM mitigation is typically performed by borrowing the time reserved for performing demand refresh (REF). Borrowing time from REF takes away the time the DRAM chip has to perform regular refreshes (mitigating a row takes 280ns [28] and REF time is 410ns). Therefore, existing in-DRAM trackers mitigate one aggressor row every 4 to 8 REF [10].

Conventional in-DRAM trackers perform mitigation *proactively* at each mitigation opportunity, even if no row has incurred TRH activations. The threshold tolerated by in-DRAM trackers depends on the rate of mitigation. For example, while MINT can tolerate a threshold of 1.5K if one aggressor row is mitigated at every REF, for realistic mitigation rates of one per 4 to 8 REF, the tolerated threshold increases to 6K to 12K.

To reduce the threshold tolerated by in-DRAM trackers, DDR5 introduced *Refresh Management (RFM)*, which counts the number of activations per bank and *proactively* stalls the Memory Controller (MC) and allows DRAM chips additional time for mitigation. As RFM stalls the memory system, frequent use of RFM causes a slowdown and incurs energy overhead, as each mitigation typically requires a refresh of multiple victim rows. Our evaluations show that such proactive mitigations increase the overall DRAM refresh energy by 4%-

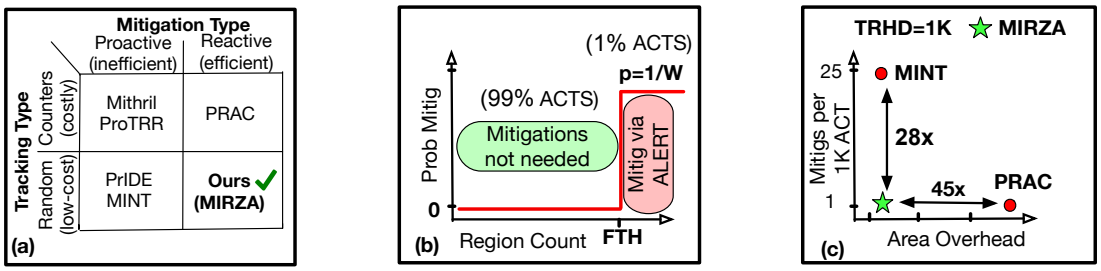

Fig. 1. (a) The time for in-DRAM mitigation can be provisioned proactively with REF/RFM or obtained reactively with ABO. Ideally, we want the storage-efficiency of randomized tracking and the mitigation-efficiency of reactive mitigation (b) Our proposal, MIRZA, uses coarse-grained filtering to skip 99% of ACTs and uses MINT+ABO for the remaining 1%. (c) MIRZA requires 28x fewer mitigations than MINT and 45x lower area overhead than PRAC.

16% for thresholds between 500 and 2K. The key problem for proactive in-DRAM trackers is the frequent mitigations.

Reactive Mitigation with PRAC+ABO: Recently, JEDEC announced an extension to DDR5 specification, which now includes *Per-Row Activation Counting (PRAC)* and *ALERT-Back-off (ABO)*. PRAC extends the DRAM array with a per-row activation counter that is updated on each activation. The ABO protocol allows the DRAM chip to *reactively* request the MC to pause and allow the DRAM chips additional time to perform Rowhammer mitigation. For example, an ABO is triggered when a given row reaches the specified number of activations. As ABO is asserted reactively, in response to the row activation counts, ABO facilitates more efficient *reactive in-DRAM mitigation*, whereby mitigation time/energy is consumed only when needed. For thresholds of higher than 500, PRAC+ABO incurs no mitigations, thus eliminating the time and energy spent on performing mitigations.

As PRAC adds a counter with each row, it incurs area overheads. PRAC also changes the DRAM timings to facilitate counter-update operations. For example, PRAC increases the *Row Precharge Time (tRP)* from 14ns to 36ns. The increased timing causes slowdown (on average, 6.5%) for typical applications, even if ABO is not triggered. Thus, while reactive mitigation by PRAC+ABO is efficient in terms of mitigations, it suffers from area overheads and performance overheads. The goal of our paper is to develop an in-DRAM mitigation that has low overheads for all three: performance, area, and energy.

Enabling Low-Cost Reactive In-DRAM Trackers: Figure 1 (a) classifies in-DRAM mitigations based on tracking and mitigation. Trackers can be counter-based trackers (e.g., Mithril or ProTRR) or randomized (e.g., MINT or PrIDE). Mitigation can be done *proactively* using REF/RFM or *reactively* using ABO. Prior in-DRAM mitigations, except PRAC, use proactive mitigation. Currently, PRAC is the only *reactive* in-DRAM mitigation. While it is straightforward to use counter-based trackers with reactive mitigation (when the counter reaches a threshold, trigger ABO to mitigate), it remains unclear how one would use reactive mitigation with randomized trackers (they do not have per-row state and need to rely on periodic mitigation). To the best of our knowledge, no prior work has explored the efficacy of low-cost randomized trackers with reactive mitigation. Our paper proposes the *first* low-cost reactive in-DRAM mitigation, called *MIRZA (Mitigating Rowhammer with Randomization and ALERT)*.

Solution: Our first insight towards making randomized trackers mitigation-efficient is to make such trackers reactive, for example, by triggering an ABO when the tracker (e.g., MINT) wants to perform a mitigation. However, simply using ABO instead of RFM does not itself reduce the mitigation overheads. As the tracker still requires periodic mitigations, it will invoke frequent ALERT and incur performance and energy overheads. Thus, using MINT with ABO still results in similar energy overheads and slowdowns as RFM. Our second insight is to use *Coarse-Grained Filtering (CGF)*, so that randomized mitigation is invoked infrequently.

We observe that over a refresh window (32ms), each region of DRAM bank (e.g. subarray of 1K rows each) typically receives only about 1000 or fewer activations, on average. So, MIRZA keeps track of per-region counts (128 counters of 11 bits, so 176 bytes per bank) and uses randomized mitigation only if the subarray receives more than a *Filtering Threshold (FTH)* activations. For CGF to be effective, it is important that the activations to a bank are not focused on only a few regions. To achieve an equal distribution of activation to regions, we propose to use *Strided Row-to-Subarray* mapping. With this mapping, CGF is highly effective. At FTH of 1500, we observe that more than 99% of the activations do not require any mitigation for a TRHD of 1K (per each side of a double-sided attack). Only if activation counts exceed FTH, MIRZA uses MINT to perform randomized mitigation, as shown in Figure 1 (b). Overall, MIRZA reduces the mitigation overhead of MINT by 28x. For a TRHD of 1K, MIRZA requires SRAM overhead of only 196 bytes per bank (45x lower area overhead than PRAC) and an average slowdown of only 0.36%.

Contributions: Our paper makes the following contributions:

1) This is the first paper to enable *Low-cost Reactive in-DRAM Mitigation* by using ABO. We enable in-DRAM mitigation with low area and performance overheads.
2) We employ *Coarse-Grained Filtering (CGF)* and *Strided Row-to-Subarray mapping* to reduce the mitigation overheads by invoking randomized mitigation only if the memory region incurs more than specified activations. CGF reduces the mitigations of MINT by 28x.
3) Our proposal, *MIRZA*, uses ABO and filtering to tolerate a threshold of 1K with only 196 bytes SRAM per bank (45x lower than PRAC) and incurring only 0.36% slowdown and 0.3% refresh power overheads.

II. Background and Motivation

A. Threat Model

Our threat model assumes that an attacker can issue memory requests for arbitrary addresses. The attacker knows the defense algorithm but not the outcome of the random number generator. We declare an attack to be successful when any row receives more than the threshold number of activations without any intervening mitigation or refresh. We do not consider RowPress [23] as it can be mitigated with *Row-Buffer Decoupling* [23], [26] or by converting the row-open time into equivalent number of activations [34], [38].

B. DRAM Architecture and Parameters.

DRAM chips are organized as banks with rows and columns. To access data from DRAM, the memory controller must first issue an activation (ACT) to open the row. To access a conflicting row, the bank must first be precharged (RP). To ensure data retention, the data in DRAM gets refreshed every tREFW. A REF operation is performed every tREFI and lasts for tRFC (see Table I).

TABLE I
DRAM Timings (DDR5 specs for 6000AN).

Parameter	Description	Value	PRAC
tRCD	Time for performing ACT	14 ns	14 ns
tRP	Time to precharge an open row	14 ns	36 ns
tRAS	Time between activate and precharge	32 ns	16 ns
tRC	Time between successive ACTs	46 ns	52 ns
tREFW	Refresh Period	32 ms	
tREFI	Time between successive REF Cmds	3900 ns	
tRFC	Execution Time for REF Command	410 ns	

C. DRAM Rowhammer and Trend

Rowhammer [20] occurs when a row is activated frequently, causing bit-flips in nearby victim rows. Rowhammer is a serious security threat [4], [7], [8], [21], [25], [40], [48]. The minimum number of activations to an aggressor row to cause a bit-flip in a victim row is called the *Rowhammer Threshold (TRH)*. TRH can be for a single-sided pattern *(TRHS)* or a double-sided pattern *(TRHD)*. As DRAM devices become smaller, the leakage increases, causing TRH to drop from 139K (TRHS) in 2014 [20] to 4.8K (TRHD) in 2020 [17]. To ensure our designs are applicable to lower thresholds, our work focuses on developing low-cost solutions for TRHD of 2K to 500.

D. In-DRAM Rowhammer Mitigation

Rowhammer mitigations rely on a mechanism to identify the aggressor rows and refresh the victim rows. We focus on in-DRAM solutions. There are three resources required for an in-DRAM mitigation: **Space** (for tracking aggressor rows), and **Time** and **Energy** (for doing mitigation). Identification of aggressor rows can be done using counters (e.g., Mithril [19] and ProTRR [24]) or probabilistically (e.g., PrIDE [11] or MINT [35]). As counter-based trackers require impractical SRAM overheads, we focus on probabilistic trackers to reduce space overheads. Without loss of generality, we use MINT as a representative low-cost in-DRAM tracker.

E. Minimizing Storage Overhead with MINT

Minimalist In-DRAM Tracker (MINT) [35] is a recent design that can perform secure mitigation while requiring only a single tracking entry. Figure 2 shows an overview of MINT. MINT operates on a window of size W, where W is the maximum number of activations between consecutive mitigations. At each mitigation, MINT mitigates the selected entry (by refreshing the corresponding four victim rows) and then randomly picks which of the N activations in the upcoming window will be picked for mitigation. The process then repeats for each successive window. Thus, for a MINT with a window of W, the *mitigation overhead* is one mitigation per W activations. While MINT requires only a single entry for tracking, it can still tolerate a TRHD of 1.5K with a window size of 75. Throughout the paper, we refer to the window of W activations for MINT as MINT-W.

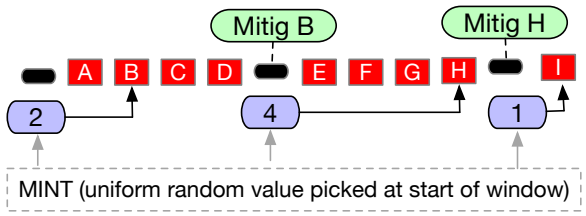

Fig. 2. Overview of MINT with W=4. A-I are activations. During the first window, the 2nd entry (B) is selected.

F. Mitigation Needs of Proactive Solutions

Typical In-DRAM trackers perform mitigation *proactively* at periodic intervals, whereby one aggressor row is mitigated, even if no row has encountered threshold number of activations. In-DRAM trackers typically perform mitigation transparently during the time provided for refresh (REF). As the time to mitigate an aggressor row is significant (280 ns for *bounded refresh* [14]), performing mitigation within REF reduces the time available to perform regular refreshes and degrades DRAM reliability. Thus, DRAM chips typically perform a mitigation every 4-8 REF [10]. Table II shows the TRHD tolerated by MINT (single entry per bank) and Mithril (2K entries per bank) as the time per mitigation varies from 1 REF to 8 REF (*Refresh Cannibalization* is the fraction of REF time consumed by mitigations). At practical mitigation rates (1 per 4-8 REF), MINT cannot tolerate even current thresholds (4.8K). Even Mithril (with high storage overheads) cannot tolerate current TRHD at 1 mitigation per 8 REF.

TABLE II
TRHD tolerated by MINT and Mithril (both performing proactive mitigation) with varying Mitigation Rate

Mitigation Rate	Refresh Cannibalization	MINT (1-entry/bank)	Mithril (2K-entry/bank)
1 aggressor per REF	68%	1.5K	1K
1 aggressor per 2 REF	34%	2.9K	1.7K
1 aggressor per 4 REF	17%	5.8K	2.9K
1 aggressor per 8 REF	**8.5%**	**11.6K**	**5.4K**

Refresh Management (RFM): As DRAM cells move to smaller technology nodes, DRAM chips face challenges due to retention failures, so stealing more time from REF to

perform Rowhammer mitigation can adversely impact DRAM reliability. DDR5 introduced the *Refresh Management (RFM)* to provide DRAM chips additional time for mitigation. To facilitate RFM, the MC provisions one counter per bank, which is incremented for each activation to the bank. When the counter reaches a specified *Bank Activation Threshold (BAT)*, an RFM signal is sent to the DRAM and the MC resets the counter and stalls for a specific time (similar to refresh). We note that BAT-RFM does not decrement the per-bank counter on a REF, to avoid taking time away from normal refresh. RFM is sent *proactively* by the MC at regular intervals, regardless of whether DRAM needs the time to mitigate.

Unfortunately, RFM stalls the DRAM and reduces performance. Furthermore, even with RFM, the DRAM device still incurs energy overheads to perform frequent mitigations. Figure 3 shows the slowdown and refresh power with RFM when MINT is configured to tolerate TRHD from 500, 1K, and 2K (RFM every 24-96 activations). The slowdown with RFM ranges from 2.9% (at TRHD=2K) to 11.1% (at TRHD=500). Similarly, the victim refresh increases the DRAM refresh power by 16.4% to 4.1% (we compute the refresh power overheads as the ratio of the number of rows undergoing victim refreshes to rows undergoing demand refresh).

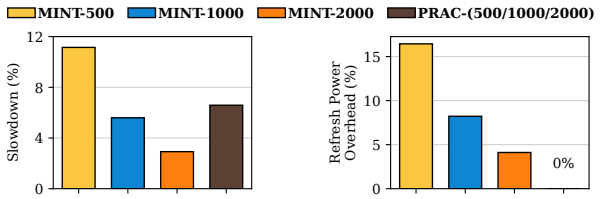

Fig. 3. Slowdown and Refresh Power Overhead. The slowdown of MINT ranges from 11.1% to 2.9% (PRAC+ABO remains 6.5% across thresholds). The Refresh Power Overhead of MINT ranges from 16.4% to 4.1% (PRAC+ABO is 0%)

Inefficiency of Proactive Mitigation: With proactive mitigation, the DRAM tries to proactively and periodically perform one mitigation at every opportunity (under REF or under RFM), regardless of whether any row has reached the threshold number of activations. Proactive mitigations are inefficient, as they incur time and energy overheads of mitigation at a conservative rate, usually dictated by the worst-case pattern (e.g. Feinting Attack [24] or circular pattern [35]). Ideally, we need in-DRAM mitigation that can dynamically obtain time for mitigation as and when the DRAM chip needs it, for example, based on counter values. A new feature from JEDEC enables such a *reactive* way to obtain mitigation time.

G. Reactive Mitigation with PRAC+ABO

JEDEC recently updated DDR5 specifications to support *Per-Row Activation Counting (PRAC)* and *Alert-Back-off (ABO)*. PRAC extends the DRAM array to have per-row counters. ABO extends the ALERT signal so that DRAM chips can pause the Memory Controller (MC) to get additional time to do Rowhammer mitigation. Figure 4 shows an overview of ABO. When ALERT is asserted, the MC can perform normal operations for 180ns, after which the MC must stall all operations and *reactively* issue an RFM. The latency of ALERT is 530ns, out of which DRAM is unavailable for 350ns. PRAC+ABO is an example of *reactive in-DRAM mitigation*, whereby the mitigation time is obtained as needed (based on value of per-row counter).

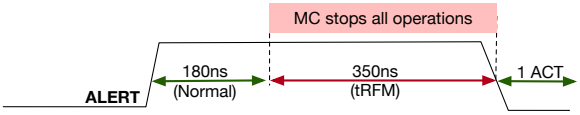

Fig. 4. Overview of Alert-Back-Off (ABO) with 1 RFM.

The Good of PRAC: One of the key advantages of PRAC+ABO is the mitigation efficiency. For thresholds of 500 or more, PRAC+ABO performs almost no mitigations (the rate of mitigation is less than 1 per 1 million activations), thus eliminating the time and energy overheads consumed in mitigation. This occurs because typical workloads tend to spread their activations over a large number of rows, so rows do not incur hundreds of activations within the refresh interval (32ms) and do not require any mitigation.

The Bad of PRAC: PRAC not only requires area overhead for counters and significant changes to the DRAM array, but it also increases memory timings to facilitate the counter update. For example, the *Row Cycle Time (tRC)* increases from 46ns to 52ns (13% increase), and this latency is in the critical path of serving conflicting requests from the same bank.

The higher latency of PRAC causes considerable slowdowns. Figure 3 shows the slowdown of PRAC+ABO at TRHD of 500 to 2K (all three values are identical). At our thresholds, PRAC+ABO encounters zero ALERTs, so the slowdown is only due to higher memory timings. On average, PRAC+ABO incurs a slowdown of 6.5%[1]. We want a reactive in-DRAM mitigation that avoids the area and latency overheads of PRAC.

H. Goal of Our Paper

We note that ABO enables reactive in-DRAM trackers that are mitigation-efficient. However, such reactive mitigation seems to be viable only for counter-based trackers (as they can trigger ABO when the per-row counter reaches the specified threshold). It is unclear how one could use ABO-based reactive mitigation with low-cost randomized trackers, as such trackers do not have a per-row state and rely on periodic mitigations.

Ideally, we want in-DRAM mitigation that has the storage efficiency of randomized trackers (thus requiring low storage) and the mitigation efficiency of reactive mitigation (to reduce the time and energy for mitigations). The goal of our paper is to enable such a space-time-energy efficient in-DRAM mitigation. Our key insight is to use a randomized tracker (to reduce SRAM overheads) with ABO (to reduce the mitigation time and energy) and use *coarse-grained filtering* to reduce the frequency of ALERTs.

[1]Note that our PRAC-based slowdowns are similar/lower than the recent work [2], [44] and we compare our slowdowns with a recent work [18] that evaluates the slowdown of PRAC on a real system in Section X.

III. Experimental Methodology

A. Configuration

We use DRAMSim3 [22] updated with a detailed memory model per DDR5 specifications. Table III shows our configuration. We assume that the OS performs a virtual-to-physical mapping at 4KB page granularity using the clock-style paging algorithm [3]. We use the *Minimalist Open Page (MOP)* [16] policy with 4 lines per row, as it is the best-performing policy for our setup. To close a row, we use a soft close page policy that closes a row after tRAS unless there are pending requests to the opened row (if so, these requests are served first).

TABLE III
Baseline System Configuration

Out-of-Order Cores Last Level Cache (Shared)	8 core, 4GHz, 4-wide, 392 entry ROB 16MB, 16-Way, 64B lines
Memory Specs t$ALERT$ Banks x Sub-channel x Rank Rows	32 GB, DDR5 180ns (normal) + 350ns (RFM) 32×2×1 128K rows per bank, 4KB rows
DRAM Address Mapping	MOP4 [16] Mapping
Page Closure Policy	Soft Close-Page Policy

B. Workloads

We use all 12 benchmarks from SPEC-2017 with at least 1 L3-MPKI, all six from *GAP* [37], and six *mixed* workloads. We run the workloads in 8-core rate-mode, until each core completes 250 million instructions (simpoint). We measure performance using weighted speedup. Table IV shows workload characteristics, including DRAM bandwidth utilization and average and the standard deviation of ACTs per subarray (1024 rows each) per tREFW (32ms).

TABLE IV
Workload Characteristics.

Workloads	L3 MPKI	ACT-PKI (Mean)	Bus Util. (%)	ACT/subarray ($\mu \pm \sigma$)
bc	58.8	29.7	82	572 ± 191
bfs	30.9	16.1	80.6	642 ± 278
cc	57.9	51.5	77.7	1037 ± 542
pr	57.7	29.5	83.1	620 ± 204
sssp	27.2	13	79.9	518 ± 149
tc	87.8	40.7	85.5	558 ± 118
blender	1.1	0.7	16	84 ± 46
bwaves	41.6	15.5	77.8	680 ± 224
cactuBSSN	3.5	3.3	44.6	395 ± 242
cam4	3.7	2.9	42.1	267 ± 204
fotonik3d	26.6	34.1	62.3	1469 ± 388
lbm	27.7	39.5	64.4	1413 ± 343
mcf	19	12.6	76.9	1056 ± 465
omnetpp	9.2	11.4	54.3	1015 ± 445
parest	26.5	12.8	84.6	965 ± 440
roms	7.8	5.1	58.5	551 ± 279
xalancbmk	1.6	2.3	26.1	281 ± 169
xz	5.2	8.3	48.1	914 ± 523
mix_1	18.6	17	72.7	1085 ± 397
mix_2	22.6	18.6	68.4	956 ± 304
mix_3	15.1	18.6	62.3	1006 ± 375
mix_4	10	19.1	57.7	1074 ± 373
mix_5	12.3	23.4	52.4	1182 ± 370
mix_6	13.6	18.7	62.9	1008 ± 340
Average	24.4	18.5	63.4	806 ± 309

IV. MIRZA: Low-Cost Reactive Mitigation

To enable in-DRAM mitigation at both low storage overheads and low mitigation overheads, we propose *MIRZA (Mitigating Rowhammer with Randomization and ALERT)*. Our key insight is that SRAM overheads can be minimized using a randomized tracker, such as MINT. However, instead of proactively doing a mitigation using REF or RFM, we use ABO to reactively obtain the time required for mitigation, as needed. To the best of our knowledge, ours is the first paper to explore the viability of ABO with randomized trackers, as ABO is currently considered compatible only with counter-based trackers. Unfortunately, simply combining MINT with ABO (MINT+ABO) still results in significant performance overheads akin to RFM (5%-15%), as we show next.

A. The Naive MIRZA Design: MINT+ABO

Our first step in enabling MIRZA is to use ABO for mitigation. MINT operates on a fixed-sized window (e.g., with a window of W, MINT would select one of W upcoming activations for mitigation). However, ALERT is channel-wide, and different banks may encounter a different number of activations; hence, their windows may end at different times. To help synchronize the mitigations of ALERTs across different banks of a channel so that they can be serviced with one ALERT, we use queues (MIRZA-Q) to buffer the addresses selected for mitigation. Furthermore, we add *Tardiness Counters* with each MIRZA-Q entry that counts the number of activations a row has encountered since entering the queue (there are no duplicates). An ALERT is triggered when any bank has a full queue or if the *Tardiness Counter* of any MIRZA-Q entry exceeds a defined *Queue Tardiness Threshold (QTH)*. On ALERT, each bank mitigates the address with the highest counter value in the queue. Figure 5 shows an overview of *Naive MIRZA*. Our default design uses 4-entry MIRZA-Q per bank (so it avoids having large CAMs).

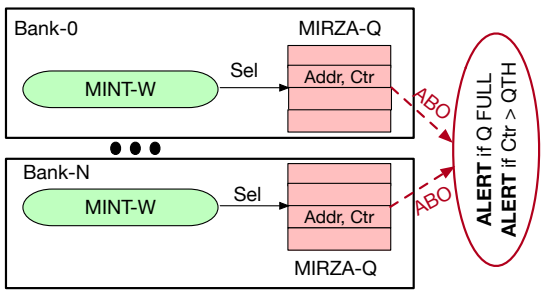

Fig. 5. Overview of Naive MIRZA: MINT selects one address out of MINTW activations to a bank and buffers it in MIRZA-Q. ALERT is triggered (for mitigation) if MIRZA-Q is full or if the queued entry exceeds QTH activations.

B. Pitfall of Naive MIRZA

Table V shows the slowdown of Naive MIRZA for MINT with a window size of 12/24/48 as the number of MIRZA-Q entries is varied from 1 to 8. There are two main takeaways. First, the buffering of entries in the per-bank queue reduces the slowdown significantly, as it helps ensure that when an ALERT happens, a larger number of banks will have something

to mitigate, making ALERT more efficient (more rows get mitigated across all the banks). Second, the naive method of combining randomization with ABO still causes a significant slowdown. On average, at MINT with a window of 24/48/96, which corresponds to TRHD of 500/1000/2000, the average slowdown, even with a large queue, remains approximately 5% to 15%, which is similar to 5%-17% for MINT with RFM. Furthermore, Naive MIRZA still incurs refresh power overheads similar to those of RFM. Thus, Naive MIRZA still has high mitigation overheads.

TABLE V
SLOWDOWN OF NAIVE MIRZA FOR MINTW OF 24/48/96 (TRHD OF 500/1000/2000) AS MIRZA-Q SIZE IS VARIED.

MINTW	Num. Entries in MIRZA-Q			
	1	2	**4**	8
24	151.83%	14.21%	**10.95%**	10.49%
48	102.18%	7.02%	**5.81%**	5.62%
96	64.07%	3.52%	**3.08%**	3.01%

C. Insight: Coarse-Grained Filtering

The security of in-DRAM trackers (be it deterministic or randomized) is determined by a worst-case pattern that continuously performs the maximum number of activations focused on a single bank. These patterns tend to do ACTs at a high rate and focus their ACTs on only a small set of rows. However, for benign workloads, the access pattern is more spread out over a large number of rows and typically does not perform the maximum number of ACTs. This divergence between the worst-case pattern and benign workloads is highly pronounced when observed at a coarser granularity. Consider a bank divided into regions of 1024 rows each (e.g. subarray), so there are 128 regions per bank. For the worst-case pattern, for every tREFW, we can get 621K activations per bank, and these activations could be in the same subarray. On the other hand, for benign workloads, we get fewer activations compared to the worst case,[2] and if these activations get spread across all the subarrays, then the number of activations on each subarray would further get reduced by 128x. Figure 6 compares the number of ACTs per subarray (per tREFW, see Table IV) for our workloads and the worst-case pattern. Our workloads incur between 219-1243 ACTs per subarray per tREW, which is 500x smaller than the 621K in the worst-case.

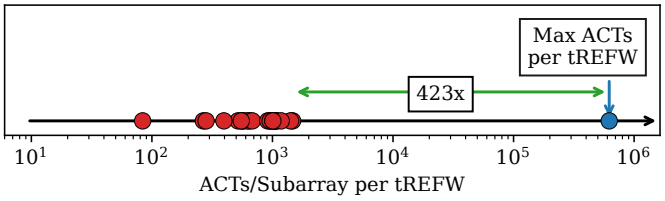

Fig. 6. Avg. ACTs/subarray per tREFW. Rowhammer trackers are designed to tolerate the worst-case (e.g., 621K ACTs per tREFW/subarray). However, workloads average 100-1500 ACTs/subarray per tREFW (423x lower).

[2]DRAM channels impose power constraints on the rate of activations using tFAW (time for four bank activation window). Given tFAW of 13ns, a channel can perform a maximum of 8.8 Million activations per tREFW, which if spread over 32 banks, results in average 275K activations per bank per tREFW. Our workloads perform about 25%-50% of this theoretical maximum number of activations per bank per tREFW.

Our key insight is to do coarse-grained (e.g. per-subarray) activation counting and enable randomized mitigation only if the activation count exceeds a *Filtering Threshold (FTH)*. Figure 7 shows an overview of *Coarse-Grained Filtering (CGF)*. As CGF tracks at a coarse granularity (subarray), it needs only a few counters (128 per bank), so SRAM overhead is small (176 bytes per bank).

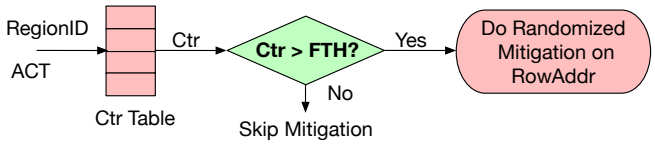

Fig. 7. Overview of Coarse-Grained Filtering

D. Impact of Row-to-Subarray Mapping

The filtering of CGF will be effective only if the activations to a bank get distributed over a large number of subarrays. This distribution of activations is highly influenced by the *Row-to-Subarray (R2SA)* mapping. We consider two mappings: *Sequential R2SA* and *Strided R2SA*. Sequential mapping maps consecutive 128 rows to the same subarray. Strided mapping maps consecutive rows to different subarrays such that every 128th row gets placed in the same subarray.

Table VI shows the percentage of ACTs that get filtered by CGF, as the FTH is varied for the two mappings. We note that under *Sequential R2SA* mapping, CGF is not effective as most of the activations are focused on only a small number of subarrays. This occurs because of the spatial locality of accesses across consecutive pages. We observe that only 7% of the ACTs get filtered, requiring that the remaining 93% of ACTs still need randomized mitigation.

TABLE VI
EFFECTIVENESS OF COARSE-GRAINED FILTERING

Filtering Threshold	Sequential Row-to-SA Filtered	Sequential Row-to-SA Remaining	Strided Row-to-SA Filtered	Strided Row-to-SA Remaining
1400	5.16%	94.84%	98.34%	1.66%
1500	**5.55%**	**94.45%**	**99.12%**	**0.88%**
1600	5.94%	94.06%	99.62%	0.38%
1700	6.31%	93.69%	99.85%	0.15%

With *Strided R2SA* mapping, CGF becomes highly effective. For example, at FTH of 1500, CGF filters out 99.5% of the activations. Only the remaining 0.5% of the ACTs participate in the randomized mitigation. Thus, CGF can reduce the mitigation overhead of our workloads by almost 200x. The ACTs that escape filtering participation still get selected probabilistically (e.g., 1/12 for a MINT window of 12). We propose to use *Stride Row-to-Subarray* mapping, as DRAM vendors have the flexibility to choose the mapping, and it incurs no additional cost (just different address bits for indexing).

We note that the CGF-based filtering is effective only for benign workloads. If a workload continuously causes ACTs to the same subarray, then it can evade CGF quickly. Therefore, to ensure security, our design, MIRZA handles workloads that escape CGF.

V. MIRZA: Overview and Design

MIRZA reduces the mitigation overhead of MINT by using both filtering and ABO. Figure 8 shows an overview of MIRZA for one bank (the structure is replicated for each bank). We call the filtering structure *Region Count Table (RCT)*, where the *region-size* denotes the number of rows per region. For our default setting of subarray-sized region, the region-size is 1024 rows, and the number of regions equals 128. An ACT command first looks up the RCT (based on *region id*) and obtains the RCT-Counter (Ctr). If the RCT counter is greater than FTH, the row address goes through MINT-based probabilistic selection, and if selected, it is inserted into the MIRZA-Q. When the queue is full or if a certain queued entry has accrued QTH activations, an ABO is triggered. On receiving ALERT, the bank mitigates one entry from the MIRZA-Q and dequeues it.

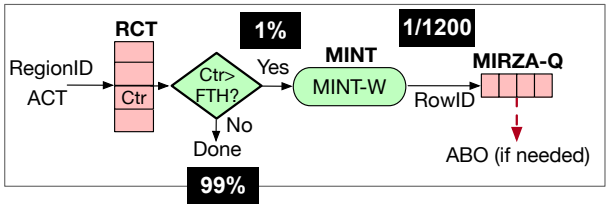

Fig. 8. Overview of MIRZA. MIRZA contains four parts (a) RCT for filtering (b) MINT for probabilistic selection (c) MIRZA-Q, and (d) ABO for time.

A. Structures

MIRZA requires three structures: (1) RCT, for filtering, (2) MINT for probabilistic selection, and (3) MIRZA-Q for queuing the row selected by MINT until mitigated via ALERT.

RCT: RCT is an untagged table that contains one counter per region. Each RCT entry is reset every tREFW. An ACT looks up the RCT entry. If the entry is less than or equal to FTH, the counter value is incremented, and the activated row does not participate in any mitigation (this occurs for more than 99% of the ACTs for our default setting). If the RCT entry is greater than FTH, then it is not updated, and the row participates in MINT-based probabilistic selection (this occurs for 1 out of 100 ACTs for our default setting). Once the RCT counter reaches FTH+1, it remains in that state until the region gets refreshed and the RCT counter is reset.

MINT: MINT with a window size of W selects exactly one of the W rows sent to it. Thus, for our default setting, MINT receives only 1/100 ACTs for probabilistic selection, and it selects only 1/12 (so, overall, 1 out of 1200). The selected entry is inserted into MIRZA-Q with a counter value of 1.

MIRZA-Q: The queue entries buffer the row addresses selected by MINT. They also keep track of the ACTs received by the buffered row since insertion into the queue. If the queue is full or the ACTs to any buffered entry exceeds QTH, MIRZA sends an ALERT. On receiving an ALERT, MIRZA picks the valid entry with the highest value of ACT counts and mitigates it (mitigation is done by refreshing two victim rows on either side of aggressor row). The entry is removed from MIRZA-Q.

B. Operation

An ACT operation arriving at the DRAM bank can result in one of three cases: (1) RCT updated and counter does not exceed FTH, so no mitigation is needed. (2) RCT counter exceeds FTH, and the row is contained in MIRZA-Q. The matching counter in MIRZA-Q is increased. (3) RCT counter exceeds FTH, row is not in MIRZA-Q, so the row participates in MINT. If the row gets selected by MINT, it gets buffered in MIRZA-Q. In all three cases, the ACT performs the activation of the given row. MIRZA does not require any changes to all other DRAM operations (such as read, write, precharge). MIRZA leverages refresh to reset the RCT counters.

C. Safe Reset of RCT Counters

As RCT is in-DRAM, this reset is synchronized with the REF operation. We want the RCT counter associated with the region to be reset when the region gets refreshed. However, as a region contains hundreds of rows, refreshing a region takes several REF operations.

Without loss of generality, we assume that refreshes are performed sequentially, one subarray at a time (16 rows per REF). To refresh a subarray with 1K rows, we need 64 REFs. Resetting RCT-entry on the 1st REF to a subarray is insecure as activations to the later rows in the region can get undercounted. Resetting RCT-entry when the last row in the region gets refreshed is insecure, as the activations of the earlier rows in the region can get undercounted.

To do a safe reset, we keep track of the number of activations to the region undergoing refresh (between the refresh of the first row to the last row), and instead of resetting the RCT entry to zero, we initialize the RCT entry to the number of activations incurred during the refresh. This ensures that RCT-entry still correctly accounts for activations (to earlier rows during earlier REF) at the end of the region refresh. We need only one extra counter per bank if the chip refreshes one subarray at REF (if the chip interleaves refreshes of two subarrays, we would need two such counters per bank).

D. Impact of ACTs During ABO

Until now, we have treated ABO as an instantaneous command that simply provides time for mitigation. However, ABO is not instantaneous: it requires a 180ns prologue (activity before stall), a 350ns stall period, and one mandatory ACT (epilogue) before another ALERT can be issued (see Figure 4). This allows up to 4 ACTs to the same bank between consecutive ALERTs. Since each ALERT can evict only 1 entry from MIRZA-Q, steady-state insertion into MIRZA-Q is limited to 1 entry per ALERT. This constraint is satisfied if MINT-W$\geq$4. In our default setting, TRHD=1K and MINT-W=12.

E. Minor Changes to JEDEC Specifications

Current JEDEC specifications provide PRAC+ABO as a combined package selected by a mode register. For implementing MIRZA, we need JEDEC specifications to enable only ABO and not PRAC. In this mode, the DRAM timing parameters remain unchanged (not inflated due to PRAC). And, we use ABO with 1 RFM per ALERT.

VI. Security Analysis of MIRZA

In this section, we determine the TRH that can be safely tolerated by MIRZA. We first determine the safe-TRH for single-sided attacks and use it to determine the safe-TRH for double-sided attacks.

A. Analyzing Safe-TRH in Single-Sided Attack

To obtain Safe-TRH ($TRHS_{Safe}$) under a single-sided attack, we analyze the maximum number of unmitigated activations that can be performed on a row before the row is mitigated or naturally refreshed. A row gets refreshed every tREFW. Furthermore, RCT-entries get reset when the row (region) gets refreshed (Appendix B). Hence, we want to determine the number of unmitigated activations on a row in the tREFW interval since the last refresh.

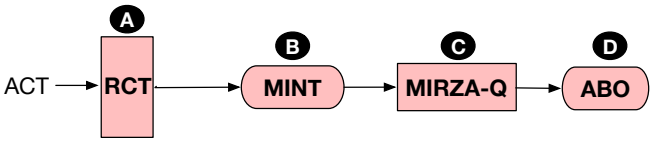

Fig. 9. Determining Safe-TRH of MIRZA by analyzing the four phases of accruing ACTs before mitigation.

Let $ACT_{nomitig}$ be the number of unmitigated activations on a given row since the last refresh or mitigation. We determine $ACT_{nomitig}$ by analyzing the number of activations received by the row in different parts of MIRZA, as shown in Figure 9.

Phase-A (RCT): Initially (after reset), the activation counts of the row are tracked by the RCT (albeit in a coarse-grained manner). The RCT entry is incremented whenever the given row (or any other row within the same region) has an activation. This phase ends when the RCT count reaches the filtering threshold (FTH). So, during Phase-A, $ACT_{nomitig} \leq FTH$

Phase-B (MINT): This phase begins when the RCT-entry exceeds FTH, so the row participates in MINT-based selection. The row can accrue unmitigated ACTs if it continues to escape selection of MINT. The unmitigated ACTs during exclusively this phase equals to the TRHS of MINT ($MINT_{TRHS}$). Therefore, the total unmitigated ACT until this phase is bounded by: $ACT_{nomitig} \leq (FTH + MINT_{TRHS})$.

Phase-C (MIRZA-Q): This phase begins when the row is selected by MINT and is inserted in MIRZA-Q. If the queue is not full, the entry will continue to accrue unmitigated ACTs while queued. MIRZA-Q tracks these ACTs and triggers an ABO when the count exceeds a *Queuing Threshold (QTH)*. So, the row can accrue at most QTH unmitigated activations. So, the total unmitigated ACT until this phase is bounded by: $ACT_{nomitig} \leq (FTH + MINT_{TRHS} + QTH)$

Phase-D (ABO): As ABO is not instantaneous, the row still receives a few ACTs (ABO_{ACTS}), even after triggering an ABO. For example, even after triggering ABO, we can do 3 more ACTs to the row during the prologue phase. In fact, an attacker could keep multiple rows in MIRZA-Q and cause even more ABO_{ACTS}, as we can remove only one entry per ABO (similar to *Feinting Attack* [24]). Figure 10 shows this example for a queue-size of 4 entries, and QTH is denoted as Q (for brevity). A, B, C are at Q ACTs, we insert D, which triggers mitigations for A, then B, and then C. C can get QTH+7 ACTs. Thus, the total number of unmitigated ACTs is bounded by: $ACT_{nomitig} \leq (FTH + MINT_{TRHS} + QTH + ABO_{ACTS})$.

$TRHS_{Safe}$ for MIRZA is any value greater than $ACT_{nomitig}$.

$$TRHS_{Safe} > (FTH + MINT_{TRHS} + QTH + ABO_{ACTS})$$

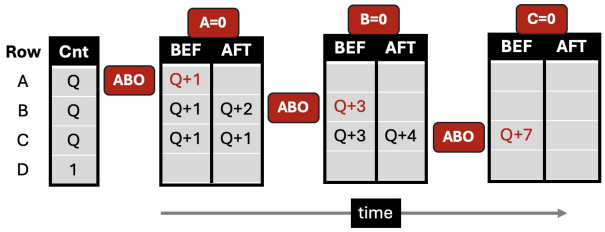

Fig. 10. Extra ACTs incurred due to non-instantaneous ABO (BEF/AFT is before/after stall). Row-C get Q+7 ACTs.

B. Analyzing Safe-TRH in Double-Sided Attack

A double-sided attack sandwiches a victim row between two aggressor rows. We are interested in determining the number of unmitigated ACTS ($ACT_{nomitig}$) on the two aggressors. Similar to the single-sided case: (1) In Phase-A, each row can accrue (FTH/2) ACTs[3] (2) In Phase-B, each row can get as many activations as the double-sided threshold of MINT ($MINT_{TRHD}$), (3) In Phase-C, each row can get QTH ACTs, and (4) In Phase-D, for simplicity, we assume that both rows accrue ABO_{ACTS}. $TRHD_{Safe}$ is any value greater than $ACT_{nomitig}$.

$$TRHD_{Safe} > (FTH/2 + MINT_{TRHD} + QTH + ABO_{ACTS})$$

C. MIRZA Configurations and Storage

We use a default 4-entry MIRZA-Q, and QTH equals 16. Table VII shows the MIRZA configurations we use for TRHD of 2K/1K/500 (target TRHS would be 2x higher). We note that MIRZA can tolerate a TRHD of 1K with less than 200 bytes of SRAM per bank. To determine the threshold of MINT, we use its publicly available security model [33].

TABLE VII
MIRZA Configurations for Target TRHD

TRHD	FTH	MINT-W	Regions/Bank	SRAM/Bank
2000	3330	16	64	116
1000	1500	12	128	196
500	660	8	256	340

[3]If a victim row is at the edge of a region, the two aggressors can each accrue FTH activations, and increase the tolerated TRHD. To ensure security, for ACTs on the edge row, we increment the RCT counter of both regions. This is a concern only if the region size is less than the subarray size.

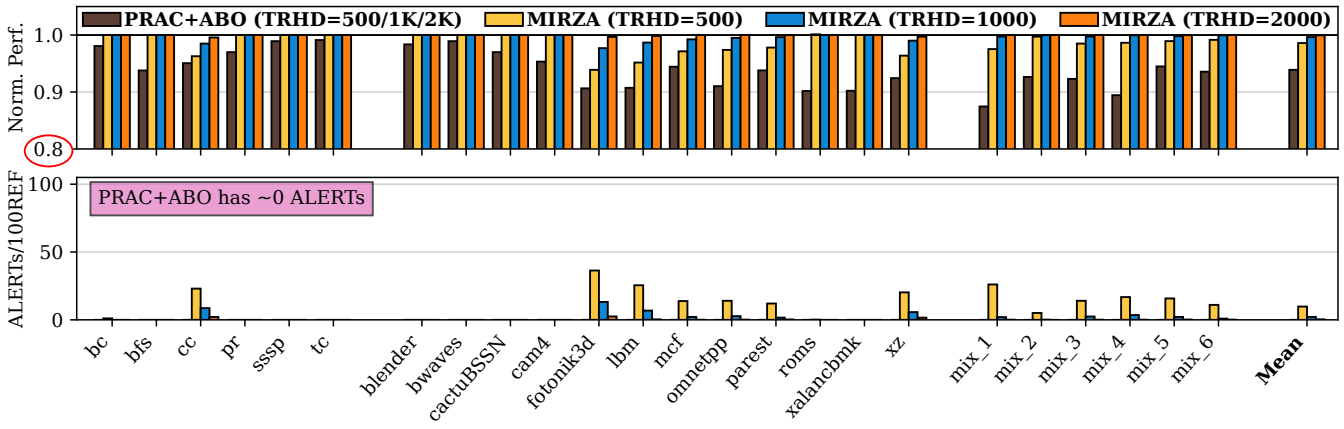

Fig. 11. (a) Performance impact due to MIRZA and PRAC+ABO. MIRZA incurs significantly lower slowdown than PRAC (1.43%, 0.36%, and 0.05%) vs 6.5% on average for TRHD=500/1K/2K, respectively. (b) ALERTs-per-100×tREFI (per sub-channel). For TRHD=1K, MIRZA triggers ALERTs in only 2.16% of tREFIs, while PRAC triggers almost none, indicating that PRAC's slowdown is exclusively due to increased memory timings.

VII. Results and Analysis

We present the results of MIRZA for the three target configurations (see Table VII) corresponding to a TRHD of 500, 1000, and 2000. We also compare MIRZA with a system using PRAC+ABO, which can tolerate similar TRHD values. We assume that PRAC+ABO uses the recently proposed MOAT [34] design (for our thresholds, PRAC+ABO incurs negligible mitigations, so other designs, such as Panopticon [1] and QPRAC [46], would yield similar results).

A. Impact on Performance

Figure 11 (a) shows the performance impact of MIRZA and PRAC normalized to our unprotected baseline system. MIRZA incurs an average slowdown of 1.43%, 0.36%, and 0.05% for TRHD values of 500, 1K, and 2K, respectively. For TRHD=500/1K `fotonik3d` experiences the highest slowdowns (6.5%, 2.3%). For TRHD=2K, `cc` experiences the highest slowdown (0.43%). This happens because the escape probability of `fotonik3d` from the filtering mechanism (CGF) is the highest for the corresponding TRHD.

PRAC+ABO incurs an average slowdown of 6.5% for all three configurations. This is due to increased memory timings (e.g. tRC increase of 10%). MIRZA can tolerate low thresholds using ABO, and without the overheads of PRAC.

B. Rate of ALERT

MIRZA uses ALERTs to obtain mitigation time. Frequent invocation of ALERT can cause a slowdown with MIRZA. Figure 11 (b) shows the ALERTs-per-100×tREFI for both MIRZA and PRAC+ABO. At TRHD of 1K, MIRZA triggers 2.16 ALERTs per 100×tREFI, while PRAC triggers almost 0. This further indicates that the slowdown of PRAC is exclusively due to increased memory timings.

C. Analysis of Mitigation Overheads

MIRZA uses a 3x-4x shorter window (W) than MINT but employs filtering to reduce mitigations. The mitigation overhead of MIRZA is a product of (a) the Escape probability of filtering and (b) the Selection probability of MINT (1/W). Table VIII shows the mitigation overheads of MIRZA and MINT. For TRHD=500, 1K, and 2K, MINT incurs mitigation overheads of 1/24, 1/48, and 1/96, respectively. The mitigation rate of MIRZA is 10x, 28.5x, and 125x lower than MINT, respectively. Thus, the coarse-grained filtering used by MIRZA is highly effective at lowering the need for mitigation.

TABLE VIII
Mitigation Overhead of MINT vs. MIRZA.

TRHD	MINT (1/W)	MIRZA (RCT Escape Prob. × 1/W)	Difference
2000	1/96	$1/751 \times 1/16 = 1/12016$	125x
1000	1/48	$1/114 \times 1/12 = 1/1368$	28.5x
500	1/24	$1/30 \times 1/8 = 1/240$	10x

VIII. Sensitivity Analysis

Table IX shows the slowdown incurred by MIRZA for different MINT-W and FTH pairs at TRHD of 1000. Higher FTH filters more ACTs and reduces slowdown, but requires a lower MINT-W to stay within the same TRHD (see Section VI-B). Lower MINT-W increases ABO frequency, offsetting the benefits of filtering.

As MINT-W increases, ABO overhead drops, but FTH must also decrease, reducing filtering. For TRHD=1000, this tradeoff results in a higher overall slowdown. For example, increasing MINT-W from 8 to 12 reduces MINT slowdown by 1.5x, but quadruples unfiltered ACTs (0.21% to 0.88%), more than doubling the total slowdown (0.13% to 0.36%).

TABLE IX
Average slowdown (%) from MIRZA with varying FTH and MINT-W values for TRHD=1K. Our proposed configuration is shown in bold.

MINT-W	4	8	12	16
FTH	1820	1660	**1500**	1350
SRAM/Bank (B)	196	196	**196**	196
Slowdown (%)	0.1	0.13	**0.36**	0.6
Remaining ACTs (%)	0.06	0.21	**0.88**	2.29

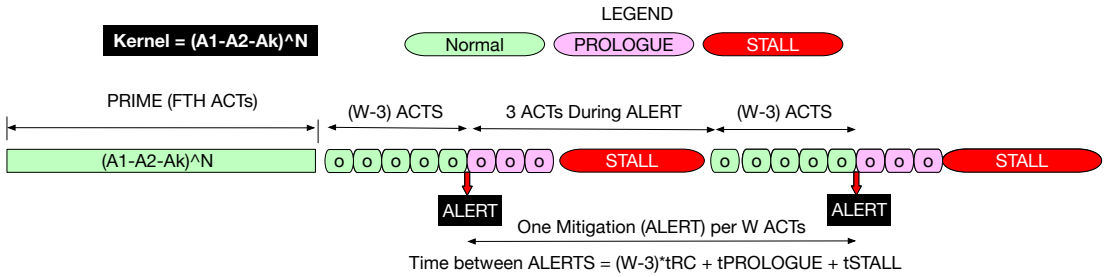

Fig. 12. Kernel for Performance Attack. Slowdown depends on MINT window (W).

A. Analysis of Relative Area Overheads

At TRHD=1K, MIRZA requires 1 (SRAM) counter per subarray. As PRAC provisions 1 counter per row, it would require 1K (DRAM) counters per subarray, in addition to the bulky sense-amplifier-and-precharge units for the counter cells.

To compare the relative areas of DRAM versus SRAM, we use a simple model [5], [45] with each DRAM cell requiring $6F^2$ area and SRAM cell requiring $120F^2$ area (F is the feature size). Table X compares the relative area overheads per subarray for MIRZA and PRAC. As both MIRZA and PRAC require per-bank queues, that area is common to both. At TRHD=1K, MIRZA requires a 45x lower area than PRAC.

TABLE X
RELATIVE AREA OF MIRZA AND PRAC (PER SUBARRAY).

TRHD	MIRZA	PRAC
1K	11-bit SRAM	10-bit x 1K = 10Kb DRAM (45x more area)
500	20-bit SRAM	9-bit x 1K = 9Kb DRAM (22.5x more area)
250	36-bit SRAM	8-bit x 1K = 8Kb DRAM (11.2x more area)

MIRZA also requires a much lower area overhead than Mithril. To tolerate a TRHD of 1K, even if Mithril is provisioned with 1 mitigation per REF, Mithril requires 2K entries (28 bit each, so 7KB SRAM) per bank, whereas, MIRZA requires only 196 bytes SRAM per bank (37x lower).

B. Analysis of Refresh Power Overheads

Figure 13 shows the relative increase in refresh power due to mitigations for MINT and MIRZA for TRHD=500 to 2K. As MIRZA reduces the mitigation requirements (by 10x-125x), it also reduces the relative refresh power consumed by mitigative refreshes.

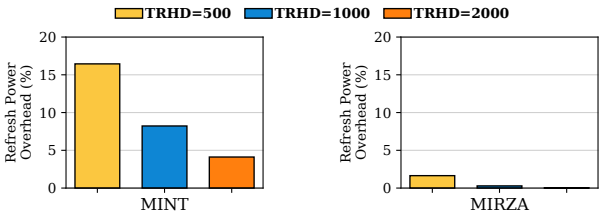

Fig. 13. Refresh Power Overhead for MINT and MIRZA.

The SRAM structures required for MIRZA incur negligible power overheads. Using CACTI-7.0, we estimate that MIRZA structures incur 0.6 milliwatts (per chip) which is approximately 0.25% of the overall DRAM power consumption (240 milliwatts per chip).

IX. ANALYZING PERFORMANCE ATTACKS

MIRZA relies on ALERT, which stalls an entire channel and affects all co-running applications. A major concern with any ALERT-based solution is their susceptibility to *Denial-of-Service (DoS)* attacks, where an attacker can cause significant system-wide slowdown on benign applications.

A. Model for Benign Workloads and ALERT

For this analysis, we use *ACT-Throughput* (ACTs per unit time) as the key metric to measure the system performance. Consider a benign workload that continuously performs reads by striping the accesses over 16 banks, and each access incurs an activation. Given tFAW of 12ns (and bus time per request of 3ns), the workload can perform one ACT every 3ns. So, the ACT throughput is 1 per 3ns. Under ALERT, the benign application can still do activations for the first 134ns (180ns - tRC), then precharge, the stall for 350ns. As the application can do 44.66 ACTs during the 530ns of ALERT, the ACT throughput is 1 per 11.35 ns (3.8x slowdown).

B. Kernel for Triggering Performance Attack

For a performance attack, we form a circular pattern containing K addresses that each map to a different row within the same RCT region, and do continuous ACTs to this circular pattern, as shown in Figure 12. The attack spends the initial time priming the RCT to exceed FTH. For our design, this time is small (less than 1% of tREFW). Once the RCT entry is primed, for a given MINT-W, the attack performs 3 ACTs during the prologue period and W-3 ACTs outside of ALERT.

Table XI shows the relative throughput of benign application under peformance attack. At MINT-W of 12 (our default), the maximum slowdown is 1.8x. Even with MINT-W of 8, the slowdown remains 2.25x. We compare the worst-case slowdown of MIRZA with MINT and PRAC in Appendix A.

TABLE XI
RELATIVE ACT THROUGHPUT AND SLOWDOWN

MINT-W	ACT-Throughput	Slowdown
16	63.4%	1.6x
12	**55.9%**	**1.8x**
8	44.5%	2.25x

We note that the 1.6x-2.25x slowdown from the performance attack (for MINT-W of 16 to 8) on MIRZA is similar in range to other memory contention attacks, such as row-buffer conflicts [29], [31]. Thus, the performance attacks on MIRZA do not represent a worse way for *Denial-of-Service*.

X. Related Work

Target Row Refresh (TRR): To tolerate Rowhammer, current devices are equipped with TRR. While the details of TRR trackers are not public, the designs have been reverse engineered (containing 4-28 entries [12]) and broken with specialized patterns [7], [12]. Table XII compares TRR with MINT and MIRZA (both configured for the current TRHD of 4.8K). We configure TRR with 28 entries (3 bytes each, including a row-id and counter) and do one aggressor row mitigation (280ns) per 4 REF. We configure MINT with *Delayed Mitigation Queue [35]* (to handle refresh postponement) and perform one aggressor row mitigation per 3 REF. For MIRZA, we use CGT with 32 regions (so, 32 counters per bank) and do not do any victim refreshes under REF.

TABLE XII
Storage and Mitigation Overhead (TRHD=4.8K)

Tracker	Storage (per bank)	Secure Tracking?	Refresh Cannibalization
TRR	84 bytes	No	17%
MINT	20 bytes	Yes	23%
MIRZA	72 bytes	Yes	0%

At the current threshold of 4.8K, all three designs incur low SRAM overheads (below 100 bytes per bank). However, TRR is not secure. As both TRR and MINT are proactive trackers, they proactively perform mitigations under REF, thus cannibalizing the REF time and reducing the time available for refreshes (by 17%-23%). MIRZA avoids refresh cannibalization, leaving the entire REF time for demand refresh. Thus, MIRZA is appealing for adoption even at current thresholds.

Per-Row Activation Counting: PRAC is a framework and the security is still determined by the underlying implementation. Recent works, such as MOAT [34] and QPRAC [46] propose secure mitigations using PRAC. However, they still incur the area and performance overheads of PRAC. Chronus [2] proposes to have a specialized subarray for counters, such that the counter update and demand activations can happen concurrently. Unfortunately, this design still incurs the area overheads of counters, requires significant design overheads for architecting heterogeneous subarrays, and imposes significant restrictions on concurrent activations (for example, the tFAW time may need to be doubled, as each demand activation now consumes the power of two activations). MoPAC [44] reduces the slowdown of PRAC by probabilistically updating its counters, but it still requires the area overheads of counters, and takes time away from refresh to update counters. MIRZA avoids the area, power, design complexity, and performance overheads of Chronus (higher tFAW) and MoPAC. Hydra [36] uses SRAM filters to reduce the memory lookups for activation counters (stored in the addressable space of DRAM) but still incurs the DRAM storage overheads. Furthermore, Hydra requires double lookups for some accesses (one for demand and the second for counter), which makes it incompatible with the in-DRAM setting. The in-DRAM setting also makes Blockhammer [47] impractical, as DRAMs are deterministic devices and cannot delay a request for an arbitrary time.

Comparison of PRAC Slowdowns: Kim et al. [18] evaluated PRAC on a real system and reported an average slowdown of 1% for SPEC2017 [42], significantly lower than the 6.5% slowdown we observe. This discrepancy stems from differences in workload selection and system configuration. First, we only include SPEC workloads with MPKI>1, which are more memory-intensive. Second, their system is over-provisioned with 8MB cache per core (L2 + LLC), as they run only 8 P-cores (SMT disabled) while utilizing the entire LLC, which is also provisioned for the E-cores, whereas our setup uses 2MB per core (similar to current systems). Third, their baseline tRC is 48ns compared to our 46ns, reducing the delta with PRAC (52ns). Finally, they use 2 memory channels for 8 cores, while we use only one channel for 8 cores.

When our simulation setup is modified to match their configuration, we obtain PRAC slowdown of 2.39% over all SPEC2K17, within 1% of their results. However, as their setup underutilizes the available cores, it is not representative of typical server-class CPUs, which provides 2MB LLC and 1MB L2 per physical core (halved under SMT), and 8 cores per channel. Also, if given 8MB cache-per-core, most SPEC2K17 benchmarks fit in the cache (20 out of 26 have MPKI<1).

ABO: MIRZA relies on ABO to obtain mitigation time. Two recent proposals, *Self-Managing DRAM* [9] and *AutoRFM* [33], propose to do Rowhammer mitigations transparently within the DRAM chips, and redefine ABO to decline an activation that conflicts with a subarray undergoing mitigation. These solutions still incur energy overheads of frequent mitigations. Furthermore, these designs also redefine ABO to quickly react to signal a conflicting ACT, imposing significant timing restrictions on the ALERT signal. MIRZA avoids the power overheads of these designs, does not impose restrictions on address mapping at the MC, and does not require JEDEC to redefine the timing and specifications of ABO.

Error Correction: SafeGuard [6], CSI-RH [15], PT-Guard [39] use codes to correct Rowhammer failures. However, uncorrectable failures can still occur and cause data loss. TAROT [41] uses profiling to proactively access rows that are vulnerable to Rowhammer. As Rowhammer behavior changes over time [30], imperfect profiling can cause errors.

MC-side Defenses: Recent works such as DREAM [43] and MIST [32] use the DRFM [28] command to build MC-side rowhammer defenses. DRFM refreshes victims across 8 or 32 banks in parallel. DREAM delays DRFM, and MIST ensures a sampled aggressor is always available, allowing each DRFM command to mitigate several banks concurrently.

Rowhammer in DDR5: Recent work has shown that DDR5 remains vulnerable to Rowhammer despite stronger in-DRAM TRR. Zenhammer [13] reported the first DDR5 bit flips using non-uniform patterns, though only on a single DIMM. Phoenix [27] extends this, reverse-engineering DDR5 TRR behavior and introducing a self-correcting synchronization method that reliably triggers bit flips across all tested DIMMs.

XI. Conclusion

For in-DRAM mitigations to be practical, they need to be both space-efficient (area for tracking) and mitigation-efficient (time and energy for mitigations). In this paper, we develop MIRZA, the first low-cost reactive in-DRAM mitigation that incurs low overheads for storage, time, and power. MIRZA uses a randomized tracker (MINT), coarse-grained filtering for reducing mitigation requirements, and ABO for reactively obtaining the mitigation time as needed. MIRZA requires 45x lower area overheads than PRAC and 28.5x lower mitigation overheads than MINT. MIRZA can tolerate a threshold of 1K with only 196 bytes SRAM per bank and incurring only 0.36% slowdown (much lower than 6.5% with PRAC). Thus, MIRZA represents a strong practical alternative to PRAC. The code for MIRZA is available at https://github.com/hritwik567/mirza-ae

Acknowledgements

We thank Salman Qazi and Stefan Saroiu for feedback on an earlier draft of our paper. We also thank the anonymous reviewers of ASPLOS-2025, MICRO-2025, and HPCA-2026 for their suggestions and feedback. This work was supported by NSF grant 233304 and the Swiss State Secretariat for Education, Research and Innovation under contract number MB22.00057 (ERC-StG PROMISE).

Appendix A
Performance Under Attacks

We configure the default parameters for MIRZA for optimizing not only the performance for benign workloads (typically lower MINT-W is better) but also to ensure that MIRZA has acceptable slowdowns even under worst-case patterns (hence higher MINT-W for increasing thresholds). Table XIII compares the slowdown of MIRZA with PRAC+ABO and MINT+RFM for both worst-case patterns (performance attacks) and benign workloads (average slowdowns).

TABLE XIII
Average and worst case slowdown for PRAC, MINT and MIRZA for TRHD=500/1000/2000.

TRHD	Tracker	Perf-Attack Slowdown	Average Slowdown
500	PRAC+ABO	1.2x	6.5%
	MINT+RFM	1.4x	10.95%
	MIRZA	2.25x	**1.43%**
1000	PRAC+ABO	1.1x	6.5%
	MINT+RFM	1.2x	5.81%
	MIRZA	1.8x	**0.36%**
2000	PRAC+ABO	1.05x	6.5%
	MINT+RFM	1.1x	3.08%
	MIRZA	1.6x	**0.05%**

We note that PRAC+ABO has the best performance under worst-case patterns. However, it suffers from high slowdowns for the average case (benign workloads). In general, systems are primarily designed to provide good performance for typical non-adversarial workloads, while still ensuring acceptable performance for worst-case patterns.

Our design ensures that the slowdown caused by MIRZA is no worse than that other non-RH-related performance attacks, such as row buffer conflicts and memory contention attacks [29], [31]. Thus, the overall worst-case slowdown of the system is still determined by the non-RH attacks.

Appendix B
Safe Reset of RCT Counters

To ensure that the RCT counters maintain a low value, they must be periodically reset. As RCT is in-DRAM, this operation can be synchronized with the refresh operation. Ideally, we want the RCT counter associated with the region to be reset when the region gets refreshed. However, a region contains several hundred rows, and refreshing these rows takes multiple REF operations. The activations incurred between the first and last REFs in the region can cause MIRZA to be insecure, if the RCT entry is reset too early or too late.

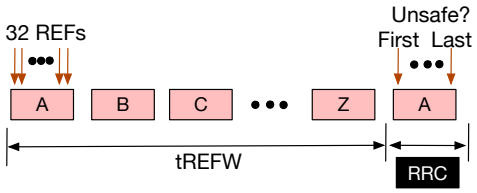

Fig. 14. Need for Safe Reset of RCT counters in MIRZA.

Figure 14 shows an overview of the refresh operations of a bank. The bank contains 128 subarrays. We assume one subarray per region for RCT. With 8K refreshes per tREFW, each region can be refreshed with 64 REFs. Without loss of generality, we assume that at any time, only one subarray is getting refreshed, so each bank has one register *RefPtr* that keeps track of the subarray (and group within the subarray) being refreshed. We now consider how resetting the RCT entry can affect the security of MIRZA.

Eager/Lazy Reset (Unsafe): An eager policy resets the RCT counter at the first REF in the region. And lazy policy resets the RCT counter at the last REF on the region. However, both these policies are insecure. For the eager policy, an attacker could target the last row and issue FTH-1 activations just before the *first* REF and another FTH-1 activations *between* the first and the last REF. Similarly, for the lazy policy, the attacker could target the first row and issue FTH-1 activations *between* the first and the last REF and another FTH-1 activations after the RCT counter gets reset. For both policies, a row can receive 2·(FTH-1) activations without participating in randomized mitigation (to accommodate this, we would need to increase the TRHD of MIRZA by FTH/2).

Safe Reset: To do a safe reset, when the region starts getting refreshed, we copy the RCT entry in a *Refreshed-Region-Counter (RRC)* register, and reset the RCT entry. If an ACT maps to the region undergoing a refresh, it updates both the RCT and the RRC, and only the RCC is used for the filtering decision. This safe-reset design ensures security of MIRZA. This design requires one RRC per bank, if the chip refreshes one subarray at REF (if the chip interleaves refreshes of two subarrays, we would need two RRC registers per bank).

REFERENCES

[1] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A complete in-dram rowhammer mitigation," in *Workshop on DRAM Security (DRAMSec)*, 2021.

[2] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, N. Bostancı, İ. E. Yüksel, H. Luo, O. Ergin, and O. Mutlu, "Chronus: Understanding and securing the cutting-edge industry solutions to dram read disturbance," *HPCA*, 2025.

[3] F. J. Corbató, "Paging experiment with the multics system," 1969, MIT Press.

[4] 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 21*, 2021.

[5] R. Dorrance, F. Ren, Y. Toriyama, A. A. Hafez, C.-K. K. Yang, and D. Markovic, "Scalability and design-space analysis of a 1t-1mtj memory cell for stt-rams," *IEEE Transactions on Electron Devices*, vol. 59, no. 4, pp. 878–887, 2012.

[6] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection," in *HPCA*, 2022.

[7] 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)*. IEEE, 2020, pp. 747–762.

[8] 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)*. IEEE, 2018, pp. 245–261.

[9] H. Hassan, A. Olgun, A. G. Yaglikci, H. Luo, O. Mutlu, and E. Zurich, "Self-Managing DRAM: A Low-Cost Framework for Enabling Autonomous and Efficient DRAM Maintenance Operations ," in *MICRO*, 2024.

[10] 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-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1198–1213.

[11] A. Jaleel, G. Saileshwar, S. W. Keckler, and M. Qureshi, "Pride: Achieving secure rowhammer mitigation with low-cost in-dram trackers," in *ISCA*. IEEE, 2024.

[12] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," in *43rd IEEE Symposium on Security and Privacy'22 (Oakland)*, 2022.

[13] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: rowhammer attacks on AMD zen-based platforms," in *33rd USENIX Security Symposium (USENIX Security)*, 2024.

[14] JEDEC, "Ddr4 sdram standard (jesd79-4b)," 2017.

[15] 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)*. IEEE Computer Society, 2022, pp. 236–252.

[16] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *MICRO*, 2011, pp. 24–35.

[17] 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, pp. 638–651.

[18] J. Kim, S. Baek, M. Wi, H. Nam, M. J. Kim, S. Lee, K. Sohn, and J. H. Ahn, "Per-row activation counting on real hardware: Demystifying performance overheads," *IEEE Computer Architecture Letters*, 2025.

[19] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative row hammer protection on commodity dram leveraging managed refresh," in *HPCA*, 2022, pp. 1156–1169.

[20] 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," *ISCA*, 2014.

[21] 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)*. IEEE, 2020, pp. 695–711.

[22] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. L. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 110–113, 2020.

[23] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Rowpress: Amplifying read disturbance in modern dram chips," in *ISCA*, 2023.

[24] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "Protrr: Principled yet optimal in-dram target row refresh," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 735–753.

[25] M. Marazzi and K. Razavi, "Risc-h: Rowhammer attacks on risc-v," in *4th Workshop on DRAM Security (DRAMSec)*, 2024.

[26] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023.

[27] D. Meyer, P. Jattke, M. Marazzi, S. Qazi, D. Moghimi, and K. Razavi, "Phoenix: Rowhammer attacks on ddr5 with self-correcting synchronization," 2026.

[28] "DDR5 SDRAM Datasheet: Directed Refresh Management (DRFM), Page-290," Micron Technology Inc., 2022. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/ddr5_sdram_core.pdf

[29] T. Moscibroda and O. Mutlu, "Memory performance attacks: denial of memory service in multi-core systems," in *Proceedings of 16th USENIX Security Symposium*, ser. SEC'07, 2007.

[30] A. Olgun, F. N. Bostanci, I. E. Yuksel, O. Canpolat, H. Luo, G. F. Oliveira, A. G. Yaglikçi, M. Patel, and O. Mutlu, "Variable read disturbance: An experimental analysis of temporal variation in DRAM read disturbance," *HPCA*, 2025.

[31] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "Drama: exploiting dram addressing for cross-cpu attacks," ser. SEC'16, 2016.

[32] S. Qazi and M. Qureshi, "DRFM and the Art of Rowhammer Sampling," *DRAMSec*, 2025.

[33] M. Qureshi, "AutoRFM: Scaling low-cost in-dram trackers to ultra-low rowhammer thresholds," in *HPCA*, 2025.

[34] M. Qureshi and S. Qazi, "MOAT: securely mitigating rowhammer with per-row activation counters," *ASPLOS-2025*.

[35] M. Qureshi, S. Qazi, and A. Jaleel, "Mint: Securely mitigating rowhammer with a minimalist in-dram tracker," in *MICRO*. IEEE, 2024.

[36] 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, pp. 699–710.

[37] K. A. S. Beamer and D. Patterson, "The gap benchmark suite," in *arXiv preprint arXiv:1508.03619*, 2015.

[38] A. Saxena, A. Jaleel, and M. Qureshi, "Impress: Securing dram against data-disturbance errors via implicit row-press mitigation," in *MICRO*, 2024, pp. 935–948.

[39] A. Saxena, G. Saileshwar, J. Juffinger, A. Kogler, D. Gruss, and M. Qureshi, "Pt-guard: Integrity-protected page tables to defend against breakthrough rowhammer attacks," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[40] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[41] C. Song, M. J. Kim, T. Wang, H. Ji, J. Huang, I. Jeong, J. Park, H. Nam, M. Wi, J. H. Ahn, and N. S. Kim, "Tarot: A cxl smartnic-based defense against multi-bit errors by row-hammer attacks," ser. ASPLOS '24, 2024.

[42] "SPEC CPU2017 Benchmark Suite," Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2017/

[43] H. Taneja and M. Qureshi, "DREAM: Enabling low-overhead rowhammer mitigation via directed refresh management," in *ISCA*, 2025.

[44] S. Vittal, S. Qazi, P. Das, and M. Qureshi, "MoPAC: Efficiently mitigating rowhammer with probabilistic activation counting," in *ISCA*, 2025, pp. 723–738.

[45] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

[46] J. Woo, C. S. Lin, P. J. Nair, A. Jaleel, and G. Saileshwar, "Qprac: Towards secure and practical prac-based rowhammer mitigation using priority queues," *HPCA*, 2025.

[47] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, and O. Mutlu, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA*, 2021.

[48] 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, pp. 28–41.