SMS: Solving Many-sided RowHammer

Samiksha Verma
samiksha@cse.iitb.ac.in
Indian Institute of Technology, Bombay
India

Virendra Singh
singhv@iitb.ac.in
Indian Institute of Technology, Bombay
India

ABSTRACT

Dynamic Random-Access Memory forms the backbone of modern memory systems. However, a critical security vulnerability known as RowHammer threatens the integrity of this cornerstone technology. As DRAM advancements progress, the RowHammer problem worsens due to a concerning trend: the threshold for triggering a bit flip is steadily decreasing. While numerous solutions have been proposed to mitigate RowHammer over the past decade, however these solutions primarily excel at addressing traditional RowHammer scenarios. More complex variations, such as many-sided RowHammer can circumvent existing defenses and compromise the security of main memory. To address this gap, we propose SMS, a solution specifically designed to tackle many-sided RowHammer while also mitigating classic RowHammer. Our model demonstrates notable performance and energy efficiency, with only 3.22% performance overhead and 8.83% DRAM energy overhead, as evidenced by our evaluation on the Spec2017, PARSEC, and LIGRA benchmark suites. In contrast, the current state-of-the-art solution (Hydra+AQUA) incurs a much higher slowdown of 15.61% and a DRAM energy overhead of 32.36%.

KEYWORDS

Microarchitecture security, Secure memory systems, DRAM security, Row hammer attacks

1 INTRODUCTION

Contemporary computing systems place an ever-growing demand on high-capacity primary memory, with Dynamic Random-Access Memory (DRAM) serving as the foundational cornerstone of these modern memory architectures. As our computing technology continues to advance, the capacity of DRAMs experiences a consistent expansion. This expansion necessitates the shrinking and tighter arrangement of DRAM cells, resulting in the emergence of challenges related to inter-cell interference and disturbance errors.

One such vulnerability is known as *RowHammer (RH)* [12]. When a specific row is frequently activated, it can induce bit-flips in the neighboring rows. The repeatedly activated row is aptly termed the aggressor while the adversely impacted adjacent rows are referred as victim rows. The number of activations required to cause bit flips in adjacent rows is called RowHammer threshold (T_{RH}). Unfortunately the RH threshold has seen a significant decrease with newer generations of DRAM, elevating the seriousness of RH as a security threat. Figure 1 illustrates the pronounced decline in the RH threshold since its initial demonstration in older DRAM generations. Following the trend, we can anticipate RH threshold dropping to just a few of hundred for current and future DRAMs.

Moreover, based on the location of aggressors and the specific hammering patterns, more severe variants of RH like many-sided hammer have emerged and they can bypass conventional mitigation techniques designed for classic RowHammer [6, 9, 14]. While numerous solutions effectively address classic RH, many-sided RH poses a severe threat [5, 6] and remains capable of eluding traditional mitigation methods. Therefore, finding an efficient solution for this formidable variant of RH is essential [6, 19]. Although recent literature [29, 32] suggest lowering the thresholds of classic RH solutions will guard DRAM with many-sided hammer, however this invariably results in a notable drop in system performance. Our research aims to overcome these limitations and to provide an efficient solution to many-sided RH. Our major **contributions** are as follows:

(1) Proposing *SMS*, an efficient solution to address many-sided hammering while mitigating classic RH.

(2) Demonstrating through experiments that existing classic RH solutions are unsuitable for addressing many-sided RowHammer.

Driven by the observed decrease in RowHammer threshold over the past decade, in this paper, we search for a solution at an ultra low threshold of 250. We conducted evaluations of our model using SPEC CPU 17, PARSEC, and LIGRA benchmark suites. In comparison to current and previous state-of-the-art solutions, the HYDRA tracker [21] combined with the AQUA mitigator [23], and the Graphene tracker [20] paired with the RRS mitigator [22], SMS consistently outperforms them. Notably, SMS successfully addresses many-sided hammer attacks even at ultra-low thresholds (250 activations), exhibiting only 3.22% performance overhead and 8.83% increase in DRAM energy consumption.

The paper is organized as follows: Section 2 provides an overview of the threat model, the basics of DRAM organization, the types and severity of RowHammer attacks, an introduction to current and previous state-of-the-art solutions, and the motivation for the proposed work. Section 3 offers a detailed explanation of the various structures required in our model and how our model operates. Section 4 covers the evaluation of SMS, including experimental details, results, overhead analysis, sensitivity test, and storage overhead analysis. Section 5 demonstrates security analysis and discusses potential areas for improvement in the proposed model. Section 6 and Section 7 provide related work and conclusion respectively.

2 BACKGROUND & MOTIVATION

2.1 Threat Model

Our threat model assumes an attacker with user-level privileges on a system using DRAM as main memory. The attacker aims to launch untargeted many-sided or classic RowHammer attacks. For a successful attack, the attacker needs frequent access to specific DRAM addresses (achieved by evicting data from cache) and the ability to determine the physical locations of adjacent rows within a row group (through reverse engineering DRAM mapping). In

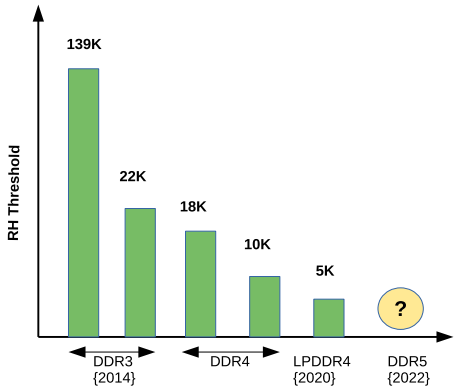

Figure 1: The graph illustrates the declining trend of the RowHammer threshold over the past decade.

Table 1: Average group access count (mean number of activations occurring within each 64ms) and MPKI of SPEC-17 workloads

Workloads	LLC MPKI	Groups ACT-400+	Groups ACT-1600+	Groups ACT-3200+
perlbench	0.430	0	0	0
gcc	24.354	1960	263.333	0
namd	0.828	330.66	7.64	0
bwaves	11.103	6208.857	746.428	0
mcf	23.790	1396.5	1010.5	297.5
cactuBSSN	3.487	2181	1035.25	0
lbm	40.583	38268	16875.5	0
omnetop	10.683	511.5	112	7
wrf	1.531	719	392	0
xalancbmk	7.323	2612	558.5	37.5
x264	0.714	260.6	119.8	0.6
cam4	0.878	272.5	59.75	0
pop2	3.531	1799	1119	121
deepsjeng	0.417	0	0	0
imagick	0.119	130.66	4	0
leela	0.099	88.5	31.5	0
nab	0.659	216	61.2	25.4
exchange2	0.001	0	0	0
fotonik3d	10.398	5551.2	1044.8	0.6
roms	5.709	3308	1173.6	252
xz	0.884	302.5	66.75	0
aster	2.068	517.5	108.5	11
Average	6.800	3028.817	1126.820	34.209

many-sided RowHammer, the attacker's goal is to manipulate access patterns such that the combined access count of a row group surpasses a threshold within 64 milliseconds. Classic RowHammer focuses on hammering a single row beyond its threshold (T_{RH}) within the same time-frame. A successful attack is defined by at least one bit flip occurring in the victim row.

2.2 DRAM Organization

Modern DRAM operates hierarchically, with channels at the top. Memory controllers access these channels in parallel. Dual in-line memory modules (DIMMs) contain multiple DRAM chips, each typically having two ranks. Each rank contains multiple banks, further divided into subarrays. These subarrays consist of rows connected to a local row buffer. A DRAM bank resembles a 2D array of DRAM cells, where cells are connected horizontally via a wordline, and vertically to a local row buffer via bitlines. Figure 2 illustrates basic organisation of a DRAM. Each DRAM cell comprises one transistor and one storage capacitor, where data is stored as charge in these capacitors. Initially, rows are in a closed (precharged) state. To serve read or write requests, the corresponding row must be opened, known as Row Activation. Before activating a row, the previously opened row must be precharged to half V_{DD}. When a row is accessed, the corresponding wordline is enabled, and charge from capacitors flows into the bitline, either charging it to V_{DD} or discharging it to zero. In both cases, the sense amplifier amplifies the voltage change and drives the bitline to either zero or V_{DD}. Repeatedly accessing the same row is a row hit, resulting in low latency. Accessing a different row is a row miss, requiring the disabling of the currently enabled wordline and precharging the bitlines to half V_{DD} before activating another row. Consecutive accesses to an already opened row, without closing it, are row hits and are faster. Accessing a different row in the same bank leads to a row miss, incurring additional time. In addition to read and write operations, DRAM requires periodic row refreshes [2, 28] to maintain data integrity . Due to the leaky nature of DRAM cells, the charge stored in the capacitors gradually dissipates over time, necessitating periodic refreshes to retain the charge and ensure data is not lost.

2.3 Classic & Many-sided RowHammer

Frequent access to a single row in DRAM within a short time frame can result in bit flips in neighboring rows. This phenomenon is referred to as RH. The extent of bit flips depends on the physical location of the aggressor row and the specific hammering pattern, which in turn determines the RH threshold and the severity of the RH threat. Figure 3 illustrates various hammering patterns:

- Figure 3 (i) depicts the classic RH scenario, where a red-colored row serves as the aggressor, causing bit flips in the adjacent blue-colored victim rows. In the case of single-sided hammering, there is a single aggressor responsible for inducing bit flips in its immediate neighboring rows.
- Figure 3 (ii) illustrates double-sided hammering, with aggressors and victims arranged in a sandwich-like configuration. Notably, the middle victim row, colored in dark blue, is particularly susceptible to bit flips as it is affected by two aggressors.
- Figure 3 (iii) showcases a highly dangerous variant of RH known as many-sided hammer. The diagram illustrates the case of four-sided hammering. This pattern, featuring four aggressor rows located alternately, leads to significant higher bit flips, particularly in the dark blue rows. In such hammering patterns, the threshold for having bit flips

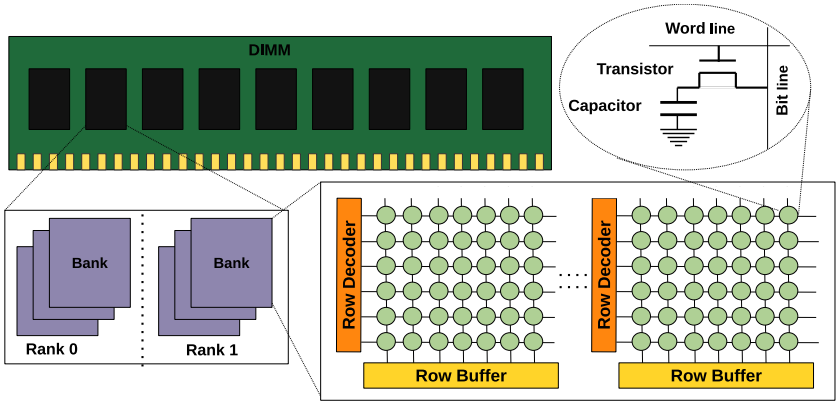

Figure 2: DRAM organization

becomes significantly lower compared to the classic RH threshold, rendering these attacks more threatening [6].

2.4 State-of-the-art RH Detector & Mitigator

Hydra [21] stands as the state-of-the-art solution for detecting RH attacks. Hydra framework introduces a comprehensive approach that utilizes both SRAM and DRAM-based structures to monitor aggressor rows effectively. In the initial stages of tracking, Hydra focuses on monitoring the access count of groups of rows. If access counts to these groups crosses a predefined threshold, Hydra starts tracking accesses at row level. To facilitate this tracking process, Hydra incorporates two key components. First, it maintains a Group Count Table (GCT), located in SRAM within the memory controller. This table serves as a repository for access counts associated with groups of rows. Second, Hydra utilizes a Row Count Table (RCT), stored in DRAM, to keep a detailed record of access counts at the individual row level. Hydra employs a Row Count Cache (RCC), strategically positioned within the memory controller. This cache helps minimize access latency for RCT and optimize the overall efficiency of the row-wise access tracking process.

Aqua [23], the state-of-the-art RH mitigator, employs a strategy to isolate aggressor rows in a designated Quarantine Row Area (QRA). When an aggressor row is identified by the Aggressor Row Tracker, the memory controller copies it to an available location in QRA. Aqua manages row mappings using two tables namely, Forward Pointer Table (FPT) and Reverse Pointer Table (RPT). When an access request reaches the DRAM, the memory controller checks the Forward Pointer Table. If a mapping is found, indicating the row is in the quarantine area, the request is redirected there. Otherwise, it proceeds with the activation command as usual. This proactive approach ensures effective isolation and management of aggressor rows to prevent RH.

2.5 Graphene and Randomised-Row-Swap

Before the emergence of Hydra and Aqua, Graphene [20] and Randomized Row Swap (RRS) [22] were considered state-of-the-art solutions for tracking and mitigating RowHammer respectively. Graphene operates as a tracker utilizing the Misra-Gries algorithm [18], which efficiently identifies frequently occurring items within a stream of data (See Appendix A). Graphene was effective for RowHammer thresholds of around 50K activations (typical for older DRAM versions). However, as modern and future DRAMs exhibit thresholds as low as 500 activations or even fewer, Graphene becomes impractical due to its high Content Addressable Memory (CAM) overhead.

On the other hand, Randomized Row Swap serves as a mitigator by employing aggressive row migration. It swaps the aggressor row with another row in DRAM randomly. However, due to the possibility of inadvertently rediscovering previously attacked rows through random chance (as highlighted by the birthday paradox), RRS must maintain a significantly lower swapping threshold than the actual RowHammer threshold. RRS suggests using one-sixth of the RH threshold as the row-swapping threshold. For recent and future DRAMs, where RH threshold is as low as a few hundred, the row-swapping threshold for RRS becomes a few tens, resulting in RRS continuously swapping rows after only a few activations. So, RRS is impractical and inefficient for newer DRAMs for classic RowHammer. To track many-sided hammer, when per row threshold becomes significantly lower than classic RH then These methods become even more inefficient and difficult to adapt.

2.6 Motivation

The growing threat of RowHammer attacks has spurred the development of numerous mitigation strategies over time. As discussed in Section 2.4, Hydra is an RH detector and Aqua serves as a robust RH mitigator. Together, Hydra and AQUA represent cutting-edge solution for RH. However, it is important to note that these solutions excel primarily in addressing classic RH. The challenges posed by more complex variant many-sided RH lack solutions. many-sided RowHammer can bypass the traditional RH solutions because the frequent activation of numerous nearby rows results in cumulative electrical interference. This cumulative effect can potentially induce bit flips, even if individual rows haven't surpassed the RH threshold. Although some works [5, 6, 9] emphasize the substantial severity of many-sided hammering and its capability to circumvent traditional classic RH solutions, it is noteworthy that there is a clear absence of a dedicated solution to address many-sided hammering [19]. Current RowHammer mitigation techniques optimized for classic RH

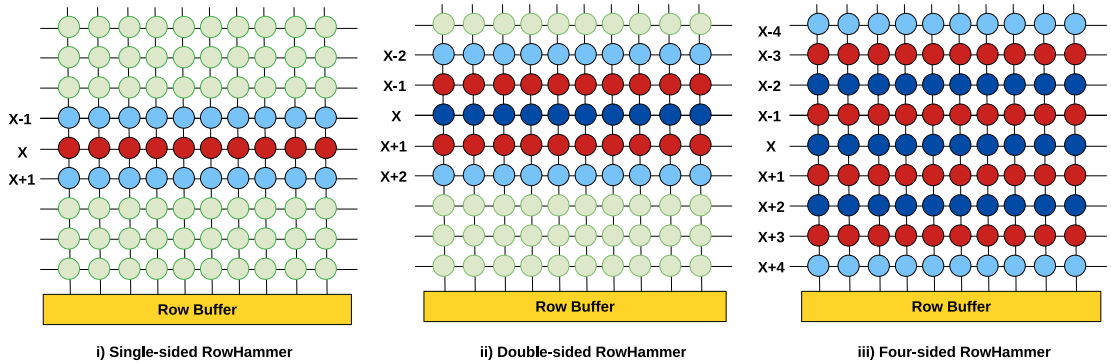

Figure 3: Hammering patterns. Aggressor: red, major victim: dark blue, minor victim: light blue

thresholds are ineffective against Many-sided RowHammer attacks, and simply lowering their thresholds to address Many-sided attacks comes at a significant efficiency cost.

Our research specifically targets the security threat posed by many-sided RowHammer. Our goal is to develop an efficient solution that mitigates many-sided RowHammer attacks with minimal performance and area overhead.

3 PROPOSED WORK

3.1 SMS: Overview and Organization

We propose SMS, a comprehensive model for monitoring and mitigating both the multi-sided hammering and the classic RH vulnerability. Initially, our work focuses on tracking accesses to groups of rows, providing essential insights into addressing the multi-sided RH issue. However, beyond a specified threshold, it refines its tracking approach to the granularity of individual rows, enabling the detection and mitigation of the classic RH as well. In our methodology, we group 16 rows together within a single cluster. As demonstrated in the work by Kim et al.[11], a row subject to hammering can impact up to the sixth row in proximity. To accommodate newer generation DRAM with exceptionally low RH thresholds, we extend our tracking range up to a distance of 8 rows. We have set an aggregate group threshold (denoted as T_A) at 250. When the access count for a group reaches T_A, we commence row-level monitoring to identify the single-sided RH. Setting T_A at 250 is safe to detect classic RH at ultra-low T_{RH} of 250. Even at the row-level tracking stage, we maintain our vigilance over the total accesses directed towards a row group, ensuring the detection of the many-sided hammering phenomenon. Our threshold for detecting the many-sided hammer is set at 3200, calculated as the product of the group size and aggressor's threshold (T_M = 16×200).

The idea of setting the threshold at 200 accesses per row in many-sided RH (i.e. 80% of the classic RH threshold) is driven by the notably lower threshold of aggressors in multi-sided RH [6]. Since Graphene+RRS and Hydra+AQUA do not handle many-sided hammer separately, RH threshold for these model is considered 200. Upon detection of a row reaching to T_R, we initiate mitigation by calling Target Row Refresh (TRR) [6] for victim rows. Additionally, if the access count for a group reaches to many-sided hammer threshold T_M, we blacklist the group for a refresh period, effectively preventing multi-sided hammering within that group. This proactive approach safeguards DRAM against the multi-sided RH threat.

3.2 Structures

Group Count Table (GCT): Tracking accesses at the group level to track classic RH is straightforward, as demonstrated in the Hydra tracker. However, the major challenge lies in efficiently monitoring the activation count of rows when tracking a many-sided hammering pattern. In many-sided hammering, the aggressor rows can originate from various locations within a set of rows. To address this, we propose a method for tracking the access count of groups of rows in a manner that guarantees to detect many-sided hammering, even when it involves aggressor rows from two adjacent groups. As mentioned earlier, we are organizing 16 rows within each row group while meticulously monitoring accesses to these groups. The core concept is to ensure that the next group does not commence where the previous one concludes. This is essential to avoid overlooking the scenario where certain rows belong to one group while others belong to another. This way, we are using a sliding window with an overlap of half the window size. Figure 5 illustrates our model's proposed row grouping scheme. Specifically, rows 1 to 16 form group 1, while group 2 spans from row 8 to row 24. Moving forward, group 3 encompasses row 16 through 32, and so forth. By adopting this approach, we are diligently keeping tabs on all groups that could potentially trigger the many-sided RH. In our model, we employ an SRAM-based group count table in the memory controller, which is similar to Hydra. Each entry stores the aggregated access count of a group.

Row Count Cache & Row Count Table: These components are designed to monitor classic RowHammer. The Row Count Table (RCT) and Row Count Cache (RCC) become active when the access count exceeds the aggregated threshold, shifting tracking from the group level to the row level. The RCT, located in DRAM, stores the access count for each individual row and functions as an untagged table with one entry per row. Since accessing the RCT in DRAM frequently can cause significant slowdown, the RCC serves as a set-associative cache for the Row Count Table, reducing access latency.

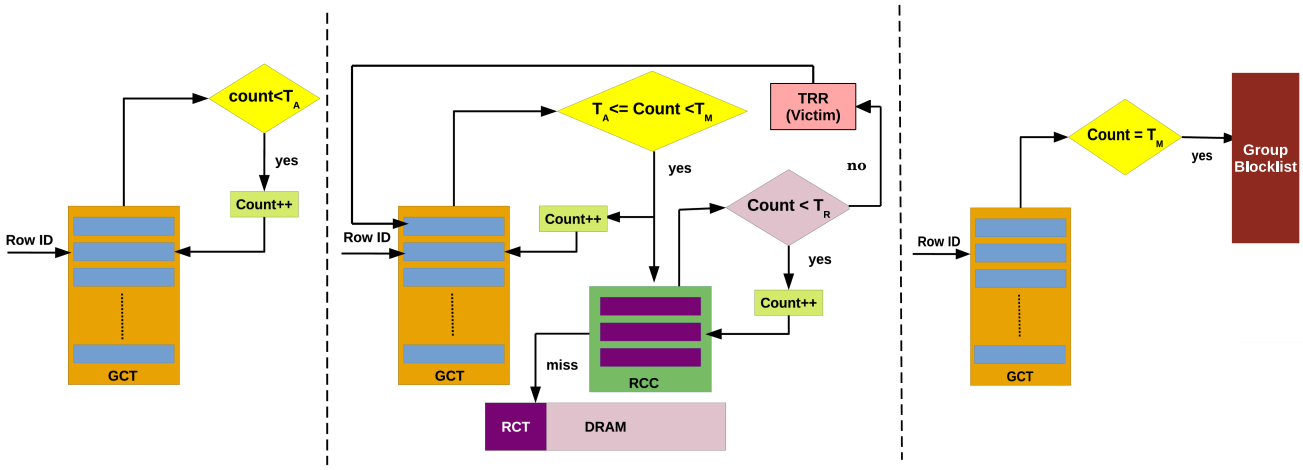

Figure 4: Working of SMS. Case I : When group access count is less than the aggregate threshold, Case II: Tacking is refined at row level for classic RowHammer, Case III: When the access count reaches many-sided hammer threshold.

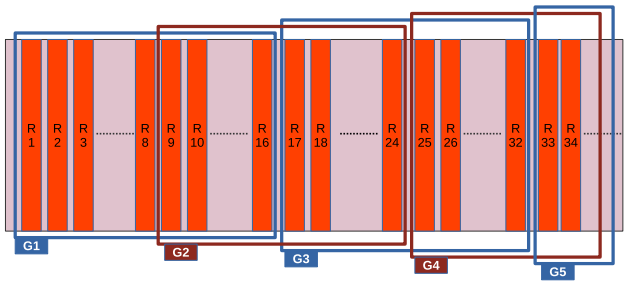

Figure 5: Creating row-groups for Group Count Table. We use a sliding window approach with a window size of 16, where half of the window overlaps.

Group blocklist: The blocklist is a SRAM-based structure integrated in controller. It stores the aggressor groups and, is reset after every refresh period (i.e. 64 ms). Determining the appropriate size for the blocklist is a critical task, as it directly impacts system security and hardware overhead. If the blocklist size is too small, it becomes inadequate for blocking all aggressor groups within a 64 ms window. This deficiency can potentially lead to security vulnerabilities. Conversely, employing an excessively large blocklist consumes unnecessary SRAM resources and contributes to controller area overhead. For security assurances, the size of the blocklist should match the maximum number of groups that can cross many-sided hammer threshold within a refresh window. Internally, DRAM refreshes rows in small batches, with the memory controller issuing refresh commands every 7.8 µs (t_{REFI}), and it takes 350 ns (t_{RFC}) for DRAM to refresh a batch. Consequently, the maximum number of row activations per bank can be calculated using the formula: Maximum activation = Refresh period × (1 - t_{RFC}/t_{REFI})/t_{RC}, where t_{RC} represents the row cycle delay (45 ns for DDR4). Simplifying this equation yields approximately 1360K accesses. Considering a memory space of 32 GB with a RowHammer threshold of 250, the maximum number of aggressors can be 5440 per bank. A total number of groups that can cross many-sided hammer threshold (i.e. 3200) is 425. With a total of 4 million rows, each requiring 22 bits for addressing. Since we have to store group (group size 16, sliding window size 8) address in blocklist, a unique group can be represented using 19 bits. Thus, the blocklist comprises 425 entries, each requiring 19 bits. This computation results in a blocklist size of approximately 1.009 KB per bank and 16.14 KB for 16 banks in a rank.

Whenever the aggregated count of a group surpasses the many-sided hammer threshold (T_M), a blocking action is triggered by adding the group to the blocklist. Groups on the blocklist are inaccessible for a refresh period of 64 ms.

3.3 Working of SMS

Figure 4 illustrates the working of SMS. Upon a DRAM access request, there are three primary cases to consider:

(1) **Case I:** If the access count of the associated group is below the aggregated group threshold (T_A), it increments the group count.
(2) **Case II:** If the access count exceeds T_A but remains below the many-sided hammer threshold (T_M), two tasks are performed: (i) Increment the count of the respective group in GCT. (ii) Switch to classic RH detector: The counter associated with the row is checked in the RCC. If a hit occurs, the counter is compared with the RH threshold (T_R). If the counter is below the threshold, it is incremented. If the counter equals the T_R, it means the row is aggressor and target row refresh (TRR) is issued for neighboring rows. Concurrently, we increment the access count associated with victim rows, as refreshing a row may induce a bit flip in its neighboring rows, given that row refresh requires row activation. In case of a cache miss in RCC, access count

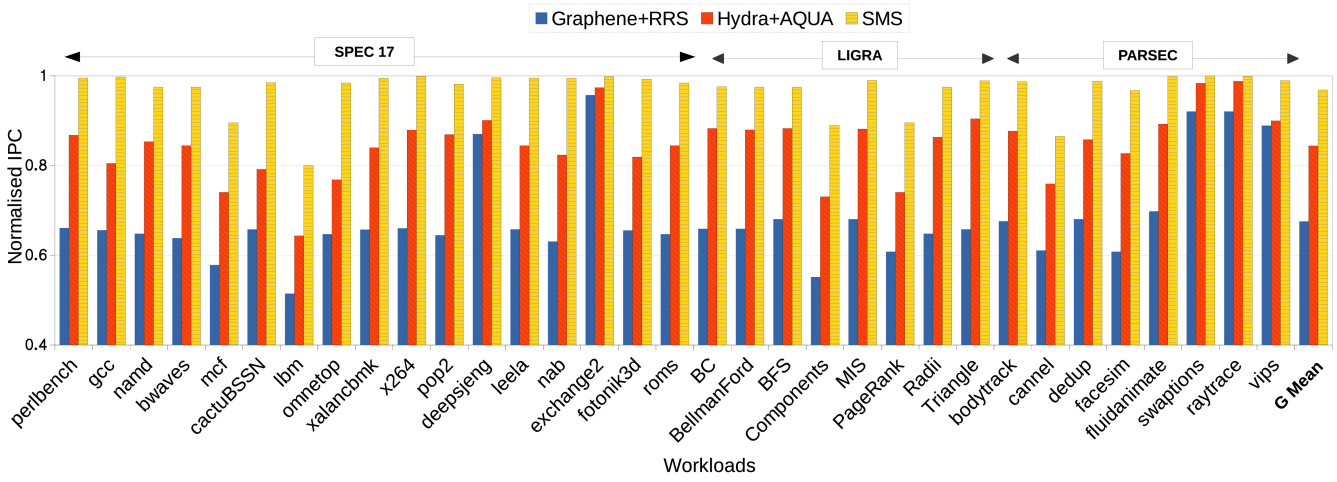

Figure 6: IPC values of the models normalised to standard DRAM. SMS incurs a slowdown of 3.22% where Graphene+RRS and Hydra+AQUA cause slowdowns of 32.43% and 15.61% respectively

of the row is brought to cache from RCT then the rest is similar to cache hit.

(3) **Case III:** If the counter value exceeds T_M, the corresponding group is added to the blocklist and becomes unavailable for the duration of the refresh period.

For each DRAM access request, the controller checks if corresponding row group is on the blocklist. If present, the request is scheduled for processing after the refresh period.

Table 2: Simulation configuration

CPU	Out of Order, 3.2GHz
ROB size	160
LLC	16-way, 1024 sets
bus speed	1.6 GHZ
main memory	32 GB, DDR4, 3.2GHz
$T_{RCD} - T_{RP} - T_{CAS}$	14-14-14 ns
Channel, Rank, Banks	2,1,16
DRAM Row size	8 KB

4 EXPERIMENTS AND RESULTS

We assess our model using DRAMSim3 [17] simulator interfaced with Champsim [7]. We use SPEC CPU 17 [4], LIGRA [1], and PARSEC [3] benchmark suites. Table 1 displays the characteristics of SPEC CPU 17 benchmarks. The table provides the MPKI at LLC and the average number of row groups surpassing various thresholds within a 64-millisecond time-frame. Specifically, it includes data on the number of row groups exceeding one-fourth, half, and the full many-sided threshold (i.e., 400+, 1600+, and 3200+ activations, respectively). Details on the characteristics of the LIGRA and PARSEC benchmarks are provided in Appendix A. We compare SMS with Hydra & AQUA combination (a state-of-the-art detector and mitigator) and Graphene+RRS, As discussed in Section 2, it is important to note that HYDRA+AQUA and Graphene+RRS do not provide a comprehensive solution for addressing the intricate problem of many-sided RH. To overcome this limitation, we adopted a commonly suggested approach of configuring a lower RH threshold. We re-implemented HYDRA+AQUA and Graphene+RRS, adjusting their parameters to function with this lower threshold (T_{RH}=250), and subsequently compared their performance with our model.

Table 2 illustrates the system configuration used in our experiments. We evaluate the efficiency of our model by comparing slowdown, total DRAM energy, sensitivity to lower RH thresholds, and hardware overhead against both current and previous state-of-the-art solutions.

4.1 Performance

Figure 6 illustrates the normalised IPC (Instructions Per Cycle) for Hydra+AQUA, Graphene+RRS, and our model. The graph shows that our model outperforms Hydra+AQUA and Graphene+RRS. The performance challenges observed with Hydra+AQUA may be attributed to its approach of migrating aggressor row to a reserve DRAM area (referred as quarantine area). However, in many-sided RH scenarios, where multiple aggressors are involved, migrating an entire group of rows becomes necessary. The process of migrating a row entails reading data from one location and writing it to another, introduces a significant number of additional read and write operations. This influx of operations, in turn, results in a reduction in IPC, negatively impacting overall system performance. Similarly, Graphene and RRS exhibit substantial performance overhead due to their designs being inefficient for current and future RH thresholds. RRS initiates row swaps after just a few tens of activations, resulting in considerable additional reads and writes, ultimately leading to diminished IPC. The experimental results indicate that Graphene+RRS incurs a performance overhead of 32.43%, while Hydra+AQUA exhibits a slightly lower overhead of 15.61%. In stark

contrast, our model demonstrates a mere 3.22% performance overhead, all values being normalized to the baseline system. These results unequivocally highlight the superior performance of our model compared to both current and prior state-of-the-art solutions.

4.2 Energy Consumption

Figure 7 illustrates the normalized total energy of Hydra+AQUA, Graphene+RRS, and our model. The graph shows Hydra+AQUA and Graphene+RRS, come at a significant energy cost. Compared to standard DRAM, Hydra+AQUA consumes 32.36% more energy, while Graphene+RRS suffers from an even higher overhead of 58.23%. In contrast, our model only requires 8.83% more energy than standard DRAM. The higher energy consumption of Hydra+AQUA is attributed to the additional read and write operations involved in aggressor migration. Similarly, Graphene+RRS consumes excessive energy due to the significant increase in reads and writes required for swapping.

4.3 Sensitivity to RowHammer Threshold

We conducted experiments to assess the performance of SMS across different RowHammer thresholds. Figure 8 illustrates the normalized IPC for Hydra+AQUA, Graphene+RRS, and our model at threshold values of 500, 250, and 125, respectively. At a threshold value of 500, SMS incurs a performance overhead of 2.4%, which increases to 3.14% at a threshold value of 250, and further to 5.46% at a threshold value of 125. Comparatively, Hydra+AQUA experiences slowdowns of 6.74%, 15.6%, and 30.65% for RH threshold values of 500, 250, and 125, respectively. Similarly, Graphene+RRS exhibits slowdowns of 17.46%, 32.43%, and 71.05% for the same threshold values. It's notable that as we decrease the RH threshold, the performance loss incurred by SMS, compared to Hydra+AQUA and Graphene+RRS also decreases. This observation underscores the scalability of our model, demonstrating its ability to adapt and maintain efficient performance for present and future DRAMs.

4.4 Storage Overhead

As previously calculated for a RowHammer threshold of 250, the maximum number of aggressors can be 5440 per bank and 87040 per rank. Since Graphene+RRS and Hydra+AQUA does not handle many-sided hammer separately, RowHammer threshold for theses model is reduced to 200. For the given configuration of a 32 GB DRAM with 2 channels and an 8 KB row size, the storage overhead of Hydra+Aqua is as follows: Hydra requires GCT of 154 KB for the given threshold. Additionally Hydra requires 24 KB RCC with 8k entries 24-bit (valid+tag+ 8-bit counter) each. GCT size is 4 KB DRAM. AQUA requires 108 KB for FPT, 64 KB for RPT. They require 360 MB of quarantine area in DRAM for given configuration. Hydra+AQUA together incurs 252 KB SRAM and 364 MB DRAM storage overhead. given a total of 4 million rows, each requiring 22 bits for addressing. Graphene requires the doubling of tracker state due to periodic resets, leading to the loss of tracking information. To mitigate the vulnerability resulting from these resets, Graphene operates the tracker at half of the threshold. Consequently, Graphene operates with a threshold of 100. However, for this lower threshold, the storage overhead of Graphene exceeds 2.6 MB of CAM. RRS requires row indirection table (RIT) in controller that stores the row mapping of swapped rows. one entry in RIT contains (valid+lock+source+destination). They require swap buffers for swapping. SRAM overhead of 0.65 MB grows up to around 13 MB as threshold decreases from 4K to 200. This significant increase in overhead renders Graphene+RRS nearly impossible to adapt.

Our work requires a group count table of 512K entries of 12-bit counters that conclude in 786 KB, RCC has 8K entries with 24-bit entry size. We require GCT in DRAM of size 4 MB. Blocklist has 425 entries per bank each size of 19 bits. So blocklist becomes of size 16.144 KB for a rank with 16 banks . Our model causes around 816 KB SRAM and only 4 MB DRAM overhead. Clearly, the SRAM overhead of our model is slightly higher than Hydra+Aqua, but the DRAM overhead is significantly lower.

5 DISCUSSION

Many-sided RowHammer poses a challenge to traditional RH solutions because the frequent activation of numerous nearby rows results in cumulative interference, potentially inducing bit flips even if individual rows haven't surpassed the RH threshold. In response, we've reduced the per-row threshold to 80% of the classic RH threshold when rows form many-sided groups. As demonstrated in the study by Kim et al. [11], a row under hammering can affect up to the sixth neighboring row. To address the needs of newer generation DRAMs with exceptionally low RH thresholds, we've extended our tracking range to encompass up to 8 rows. These adaptations showcase our model's efficacy in safeguarding current and future DRAMs from many-sided RH attacks. Our model initially monitors access at the group level. When transitioning to per-row tracking, the access count begins from the group level threshold, considering the worst-case scenario where all accesses converge onto a single row within a group. Upon detecting a RowHammer event, the victim row undergoes a Target Row Refresh (TRR). TRR commands hold the highest priority among other read-write operations, ensuring that the victim row is refreshed without data loss. To prevent the victim row refresh from inadvertently causing RH in its neighboring rows [14], we treat the issuance of TRR for victim rows as an access for the row itself, thereby incrementing its access count. This mechanism ensures protection against classic RowHammer attacks.

While TRR safeguards data, it introduces performance overhead due to refresh latency of victim rows. To address this challenge, a recently proposed technique HiRA [30] can be immensely beneficial. HiRA operates by mitigating the latency of refresh operations, accomplishing this by concurrently refreshing a row while accessing or refreshing another row within the same bank. By enabling victim row refresh to occur in parallel with other memory access commands, HiRA effectively enhances the performance by eliminating additional latency.

The HiRA refresh technique can enhance system resilience against Denial-of-Service (DoS) attacks leveraging classic RowHammer. By eliminating the extra latency incurred during victim row refresh, HiRA ensures that other access requests are serviced without delays. This significantly hinders the attacker's ability to disrupt system performance through classic RowHammer techniques.

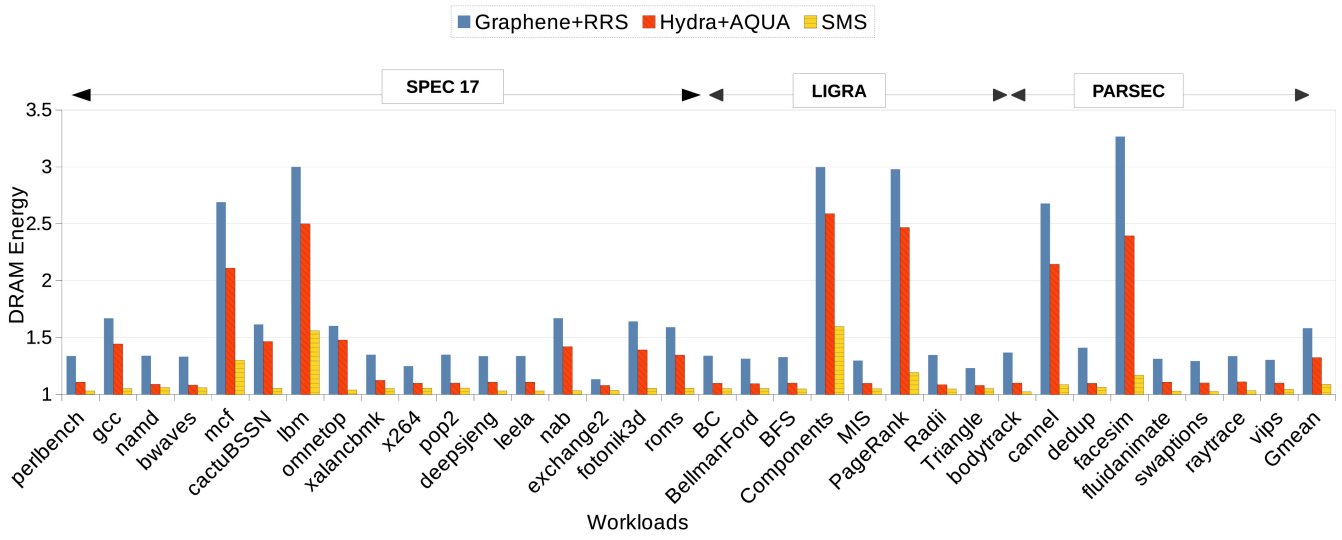

Figure 7: Total DRAM energy consumption normalised to standard DRAM. Graphene+RRS and Hydra+AQUA incur DRAM energy overheads of 58.23% and 32.36%, respectively, whereas SMS results in a significantly lower DRAM energy overhead of only 8.83%.

Table 3: Comparison of storage overhead of SMS with Hydra+Aqua and Graphene+RRS

Model	Structures	Cost	Total SRAM	Total CAM	Total DRAM
Hydra+AQUA	GCT	154 KB	358 KB	0	364 MB
	RCT	4 MB			
	RCC	32 KB			
	FPT	108 KB			
	RPT	64 KB			
	RQA	360 MB			
Graphene+RRS	CAM	2.6 MB	13.02 MB	2.6 MB	0
	RIT	13 MB			
	Buffer	16 KB			
SMS (Proposed)	GCT	768 KB	816 KB	0	4 MB
	RCT	4 MB			
	GCC	32 KB			
	Blocklist	16.14 KB			

6 RELATED WORK

RowHammer remains an active area of research, with ongoing investigations focusing on trackers, mitigators, and even security threats that exploit this vulnerability. Numerous trackers, both SRAM and DRAM-based, have been proposed to detect RowHammer vulnerabilities. SRAM-based trackers offer speed but come with higher area and cost overheads, while DRAM-based alternatives are more cost-effective and area-efficient but suffer from performance inefficiencies. Prior works [16, 20] use SRAM-based table to track the access count of DRAM rows. Some solutions [10, 26] propose to use DRAM based trackers. Hydra [21] is a hybrid tracker that uses both SRAM and DRAM based structures to track activations.

RowHammer mitigators generally fall into two categories: mitigation through victim row refreshes and aggressor migration. Previous works [12] [20, 24, 25] propose refreshing victim rows before bit flips occur. Alternatively, mitigators like RRS[22] and Aqua [23] migrate aggressors to prevent them from affecting neighboring rows. Such as CROW [8] creates copies of rows to safeguard against RowHammer attacks. BlockHammer [29] functions by blocking rows that are frequently accessed once they surpass a certain threshold, thereby preventing them from causing RowHammer issues.

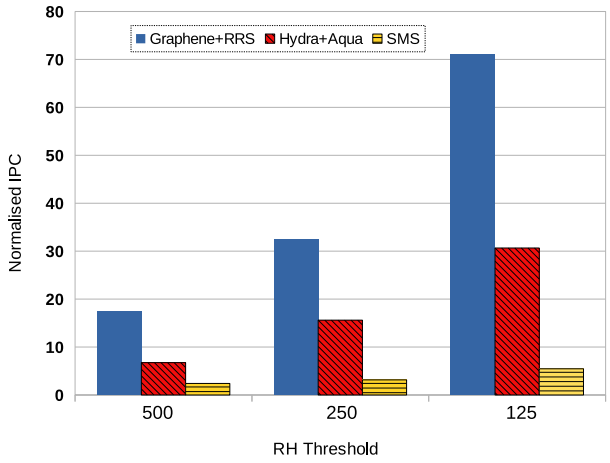

Figure 8: Normalised IPC for different RH thresholds

Numerous severe security threats exploiting RowHammer vulnerabilities have been proposed in the literature. For example, PThammer [31] induces bit-flips in page tables to achieve privilege escalation. SpecHammer [27] combines the transient attack Spectre [13] with RowHammer to form a new speculative attack. Rambleed [15] demonstrates that RH not only compromises data integrity but also poses a significant threat to data confidentiality by analyzing the tendency of bit-flips.

Several works [5, 9, 14] have proposed even more dangerous variants of RH with different hammering patterns that are difficult to mitigate. TRRespass [6] demonstrates that by using many-sided hammering, the widely deployed Target Row Refresh (TRR) mitigation can be bypassed.

In a recent study by Onur et al. [19], the research on RowHammer highlights enduring challenges and unanswered questions, underscoring the ongoing need for exploration and innovation in this field.

7 CONCLUSION

RH is a significant security concern in modern DRAM technology. Various hammering patterns have been developed that result in RH variants that are more threatening and challenging to mitigate. Many-sided hammering is one such variant that can bypass the mitigation techniques deployed in DRAMs to guard against RH. In response to the evolving threat landscape, we introduce SMS, a solution crafted to effectively counter many-sided Hammer attacks while concurrently addressing the conventional RH. SMS outperforms current and previous state-of-art solutions. Our model effectively addresses many-sided hammer with only 3.22% performance and 8.83% energy overhead. We show that SMS is a robust and scalable solution for both current and future DRAMs.

REFERENCES

[1] André Bauer, Marwin Züfle, Simon Eismann, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev. 2021. Libra: A Benchmark for Time Series Forecasting Methods. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Virtual Event, France) *(ICPE '21)*. Association for Computing Machinery, New York, NY, USA, 189–200. https://doi.org/10.1145/3427921.3450241

[2] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2016. DRAM Refresh Mechanisms, Penalties, and Trade-Offs. *IEEE Trans. Comput.* 65, 1 (jan 2016), 108–121. https://doi.org/10.1109/TC.2015.2417540

[3] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 72–81.

[4] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 41–42.

[5] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2021. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1001–1018.

[6] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the Many Sides of Target Row Refresh. arXiv:2004.01807 [cs.CR]

[7] Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jimenez, Elvira Teran, Seth Pugsley, and Jinchun Kim. 2022. The Championship Simulator: Architectural Simulation for Education and Competition. arXiv:2210.14324 [cs.AR]

[8] Hasan Hassan, Minesh Patel, Jeremie S. Kim, A. Giray Yaglikci, Nandita Vijaykumar, Nika Mansouri Ghiasi, Saugata Ghose, and Onur Mutlu. 2019. CROW: a low-cost substrate for improving DRAM performance, energy efficiency, and reliability. In *Proceedings of the 46th International Symposium on Computer Architecture* (<conf-loc>, <city>Phoenix</city>, <state>Arizona</state>, </conf-loc>) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 129–142. https://doi.org/10.1145/3307650.3322231

[9] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. 2022. BLACKSMITH: Scalable Rowhammering in the Frequency Domain. In *2022 IEEE Symposium on Security and Privacy (SP)*. 716–734. https://doi.org/10.1109/SP46214.2022.9833772

[10] Dae-Hyun Kim, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. Architectural Support for Mitigating Row Hammering in DRAM Memories. *IEEE CAL* 14, 1 (2015), 9–12. https://doi.org/10.1109/LCA.2014.2332177

[11] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 638–651. https://doi.org/10.1109/ISCA45697.2020.00059

[12] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. *SIGARCH Comput. Archit. News* 42, 3 (jun 2014), 361–372.

[13] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1–19. https://doi.org/10.1109/SP.2019.00002

[14] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. 2022. Half-Double: Hammering From the Next Row Over. In *USENIX Security 22*. USENIX Association, 3807–3824.

[15] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAMBleed: Reading Bits in Memory Without Accessing Them. In *2020 IEEE Symposium on Security and Privacy (SP)*. 695–711. https://doi.org/10.1109/SP40000.2020.00020

[16] Eojin Lee, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. 2019. TWiCe: Preventing Row-hammering by Exploiting Time Window Counters. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 385–396.

[17] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. https://doi.org/10.1109/LCA.2020.2973991

[18] J. Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming* 2, 2 (1982), 143–152. https://doi.org/10.1016/0167-6423(82)90012-0

[19] Onur Mutlu, Ataberk Olgun, and A. Giray Yağlıkcı. 2023. Fundamentally Understanding and Solving RowHammer. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '23)*. Association for Computing Machinery, New York, NY, USA, 461–468. https://doi.org/10.1145/3566097.3568350

[20] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae Lee. 2020. Graphene: Strong yet Lightweight Row Hammer Protection. 1–13. https://doi.org/10.1109/MICRO50266.2020.00014

[21] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J. Nair. 2022. Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking. In *ISCA* (New York, New York) *(ISCA '22)*. ACM, New York, NY, USA, 699–710.

[22] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J. Nair. 2022. Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 1056–1069. https://doi.org/10.1145/3503222.3507716

[23] Anish Saxena, Gururaj Saileshwar, Prashant J. Nair, and Moinuddin Qureshi. 2022. AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In *MICRO 2022*. 108–123.

[24] Seyed Mohammad Seyedzadeh, Alex K. Jones, and Rami Melhem. 2018. Mitigating Wordline Crosstalk Using Adaptive Trees of Counters. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 612–623. https://doi.org/10.1109/ISCA.2018.00057

[25] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Making DRAM Stronger Against Row Hammering. In *Proceedings of the 54th Annual Design Automation Conference 2017* (Austin, TX, USA) *(DAC '17)*. Association for Computing Machinery, New York, NY, USA, Article 55, 6 pages. https://doi.org/10.1145/3061639.3062281

[26] § TanjBennett, Stefan Saroiu, Alec Wolman, Lucian Cojocar Microsoft, and Avant-Gray Llc. 2021. Panopticon: A Complete In-DRAM Rowhammer Mitigation. https://api.semanticscholar.org/CorpusID:235420813

[27] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. 2022. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*. 681–698. https://doi.org/10.1109/SP46214.2022.9833802

[28] Samiksha Verma, Shirshendu Das, and Vipul Bondre. 2023. Hybrid Refresh: Improving DRAM Performance by Handling Weak Rows Smartly. In *Proceedings of the 2022 International Symposium on Memory Systems* (Washington, DC, USA) *(MEMSYS '22)*. Association for Computing Machinery, New York, NY, USA, Article 7, 11 pages. https://doi.org/10.1145/3565053.3565060

[29] A. Giray Yağlikçi, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. 2021. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *HPCA 202*. 345–358. https://doi.org/10.1109/HPCA51647.2021.00037

[30] Abdullah Giray Yağlıkçı, Ataberk Olgun, Minesh Patel, Haocong Luo, Hasan Hassan, Lois Orosa, Oğuz Ergin, and Onur Mutlu. 2022. HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. arXiv:2209.10198 [cs.AR]

[31] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. 2020. PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses.

[32] Zhi Zhang, Yueqiang Cheng, Minghua Wang, Wei He, Wenhao Wang, Surya Nepal, Yansong Gao, Kang Li, Zhe Wang, and Chenggang Wu. 2022. SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh. In *USENIX ATC 22*. USENIX Association, Carlsbad, CA, 399–414. https://www.usenix.org/conference/atc22/presentation/zhang-zhi

APPENDIX-A

Characteristics of PARSEC and LIGRA

Table 4 provides an overview of the characteristics of LIGRA and PARSEC workloads. The table provides the MPKI at LLC and the average number of row groups surpassing various thresholds within a 64-millisecond time-frame. Specifically, it includes data on the number of row groups exceeding one-fourth, half, and the full many-sided threshold (i.e., 400+, 1600+, and 3200+ activations, respectively). This information offers insights into the memory access patterns and workload intensities of the PARSEC and LIGRA benchmarks, aiding in the analysis and comparison of their performance implications on memory systems.

Target Row Refresh (TRR)

In our model, we employ Target Row Refresh (TRR) as a mitigative measure to address classic RowHammer For DDRx memory technology, Target Row Refresh (TRR) emerges as a pivotal mitigation strategy against the pernicious RowHammer vulnerability. TRR operates by strategically deploying additional refresh commands to memory rows identified as potential targets for RowHammer attacks.

By subjecting these vulnerable rows to more frequent refreshing, TRR aims to disrupt the repetitive accessing patterns that can lead to bit flips and memory corruption. While TRR's efficacy in curbing RowHammer attacks on modern DDR4 systems is widely acknowledged, its inner workings and specific implementation details remain somewhat opaque. This underscores the need for further exploration and understanding of TRR's mechanisms to ensure robust protection against emerging memory-related security threats.

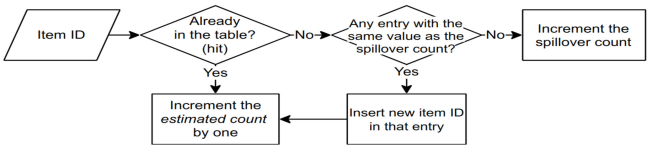

Figure 9: Working of Misra-Gries algorithm

Table 4: Characteristics of PARSEC and LIGRA.

Workloads	LLC MPKI	Groups ACT-400+	Groups ACT-1600+	Groups ACT-3200+
xz	0.88	302.50	66.75	0
BC	2.58	2,771.50	814.50	0
BellmanFord	2.94	2,860.50	1,000	0
BFS	2.96	2,654	810.25	0
Components	14.16	5,286	2,290.25	585.75
MIS	4.98	4,722	1,384	0
PageRank	51.79	20,349	3,350	197
Radii	3.83	3,505.50	1,384	0
Triangle	10.05	9,878.50	0	0
bodytrack	0.19	145	2	0
cannel	6.08	4,753	2,779.50	14.50
dedup	1.26	827	39	1
facesim	3.07	2,731.50	1,582.50	171
fluidanimate	2.59	668.50	0	0
swaptions	0	0	0	0
raytrace	0.03	0	0	0
vips	0.10	35	1	0
Average	**6.87**	**3,440.63**	**1,084.29**	**46.24**

Working of Misra-Gries algorithm

Graphene utilizes the Misra-Gries algorithm to identify frequently accessed rows from incoming row activations. Figure 9 provides an explanation of how Misra-Gries operates in identifying frequently occurring items. The algorithm maintains a spillover counter: when a new entry arrives, it searches for it in the table. If the entry is found, its counter is incremented; if not, the algorithm checks if any other item's counter equals the spillover count. If so, the algorithm replaces that entry with the new one and increments its counter by one; if not, it simply increments the spillover counter.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009