PRACtical: Subarray-Level Counter Update and Bank-Level Recovery Isolation for Efficient PRAC Rowhammer Mitigation

Ravan Nazaraliyev Saber Ganjisaffar Nurlan Nazaraliyev Nael Abu-Ghazaleh
University of California, Riverside
{rnaza005,sganj003,nnaza008,naelag}@ucr.edu

Abstract

As DRAM density continues to scale, the Rowhammer vulnerability increases in severity due to heightened charge leakage, which lowers the activation threshold required to induce bit flips. To mitigate this risk, industry-standard solutions have shifted from memory controller-based row activation counters, which require large SRAM storage with significant area and power overheads, to in-DRAM row activation counters. The DDR5 JEDEC standard incorporates a modified DRAM architecture featuring per-row activation counters (PRAC) and an Alert Back-Off (ABO) signal that notifies the memory controller (MC) to trigger mitigation mechanisms. However, PRAC introduces a performance overheads by incrementing counters during the precharge operation, adding an additional delay to the precharge phase. Furthermore, when the ABO signal is triggered upon a row reaching the Alert threshold, $\texttt{RFM}_{\texttt{ab}}$ indiscriminately stalls all memory requests at the memory channel level, even when only a single bank is being accessed heavily, leading to unnecessary performance degradation. In this work, we propose PRACtical, an optimized approach to improving the performance of existing PRAC+ABO mechanisms while maintaining security guarantees. To reduce counter update latency, we introduce a centralized increment circuit, allowing the memory controller to proceed with subsequent activations to other subarrays without suffering the increment delays. To minimize unnecessary memory stalls and make the system resilient against memory performance attacks based on channel stalling upon Alert, we enhance $\texttt{RFM}_{\texttt{ab}}$ with bank-level granularity, enabling the memory controller to selectively stall only the affected banks rather than the entire memory channel. This is achieved by introducing a register in the DRAM that shows the banks under attack upon an Alert signal. Our proposed techniques improve the performance over state of the art opportunistic PRAC and ABO, by 8% (20% max). Additionally, PRACtical saves an average of 19% energy relative to PRAC+ABO. PRACtical is resilient to performance attacks, showing less than 6% slowdown on an aggressive performance attack, while providing the same security as PRAC+ABO.

Keywords

DRAM, Rowhammer, PRAC, ABO

1 Introduction

Rowhammer [13, 15, 36, 47] is a well-known serious vulnerability affecting DRAM: repeated access to one or more target rows can induce bit flips in adjacent rows. The bit flips occur due to charge leakage from repeated activations of a DRAM row, which can accelerate charge leakage in nearby memory cells, causing their state to change. When a row is activated a sufficient number of times within a single refresh interval, it may corrupt the contents of nearby rows. The estimated minimum number of such activations that could induce a bit flip is referred to as the *Rowhammer Threshold*, denoted by N_{RH}. As DRAM technology has scaled to smaller feature sizes, N_{RH} has significantly decreased—from approximately 140K activations in earlier DDR3 devices to as low as 4.8K in LPDDR4 modules [33, 36]; it is expected to drop further in future generations. Concerningly, the significant drop in N_{RH} leads to a noticeable increase in the frequency of DRAM bit flips [40]. Accordingly, mitigation strategies must continuously evolve to remain effective against increasingly demanding and sophisticated threat conditions.

DRAM plays a critical role in determining the performance of memory-intensive workloads, leading to the well known memory wall [18, 19, 74]. As Rowhammer threats continue to increase with memory scaling, the proposed mitigations have come at a substantial cost to DRAM performance. JEDEC [28] has recently introduced a new standard mitigation mechanism for the Rowhammer vulnerability in DDR5 memory devices. This mechanism incorporates two key components: *Per-Row Activation Counters* (PRAC) [29] and *Alert Back-Off* (ABO) protocol. PRAC, inspired by the Panopticon framework [6], implements an in-DRAM architectural enhancement where each row is extended to store an activation counter. When the number of activations to a row exceeds a predefined Alert threshold, the mechanism should trigger mitigation for that row and its neighboring rows. This mechanism enables localized, row-level tracking without incurring the area and power overhead associated with external SRAM-based counters [52, 60]. The ABO protocol [29] complements PRAC by providing a signaling mechanism through which the DRAM device can notify the memory controller (MC) when a critical activation threshold is crossed. The MC sends all-bank RFM ($\texttt{RFM}_{\texttt{ab}}$) triggering targeted refresh operations, thereby preventing potential bit flips. While the PRAC+ABO framework marks a significant improvement in integrated, hardware-supported Rowhammer mitigations under reduced thresholds [9, 53, 71], it also introduces several critical limitations that compromise both performance and scalability, particularly in high-performance or multi-threaded environments.

First, the integration of PRAC requires architectural modifications that impact critical DRAM timing parameters [23, 28]. Specifically, the logic responsible for updating the per-row counters introduces an additional latency of around $5ns$ to the row cycle time (t_{RC}). Even more significantly, the precharge timing (t_{RP}) is extended from $15ns$ to $36ns$ to accommodate the read-modify-write (RMW) operations needed for per-row counter updates. These modifications directly affect memory access latency and throughput, especially in workloads with high row buffer conflicts, as illustrated in Figure 1 (a) (Baseline vs. PRAC+ABO).

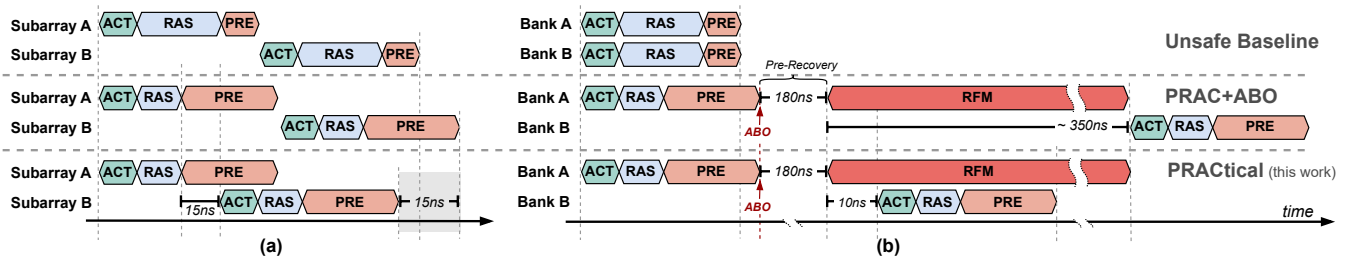

Figure 1: Comparison of baseline DRAM (top), PRAC+ABO (middle), and the proposed design (bottom). PRAC+ABO introduces (a) access latency from counter update overhead and (b) all-bank stalls due to coarse-grained RFM recovery. The proposed mechanism mitigates both by enabling subarray-level PRAC updates and bank-level mitigation. The gray regions illustrate the performance improvements enabled by the proposed approach. Register read operation takes 10ns in PRACtical in part (b).

Second, after ABO is signalled to MC from DRAM, MC sends RFM_{ab} command to DRAM for recovery. This command operates at the memory channel granularity. Therefore, the memory controller must conservatively stall all requests across the entire channel, even if only a single bank is affected, as illustrated in the middle row of Figure 1 (b). Since all banks undergo recovery, even in the absence of an actively hammered (hot) row, the mechanism *opportunistically* refreshes potential victim rows to preemptively mitigate future attacks. Our analysis reveals that this opportunistic strategy results in a threefold increase in recovery refreshes, underscoring the inefficiency and potential redundancy of these additional operations. We further observe that, across a set of memory-intensive benchmarks, on average, only 1.16 out of 64 banks need recovery at any given time (See §3.2). Consequently, the remaining banks are unnecessarily stalled, leading to avoidable performance degradation. This coarse-grained design limits memory-level parallelism, degrading performance and responsiveness in scenarios where finer-grained mitigation would suffice.

While the PRAC+ABO mechanism represents meaningful progress toward an in-DRAM Rowhammer mitigation, it introduces two key performance-related drawbacks: (1) increased memory access latency due to counter update overhead, and (2) channel-wide stalls caused by the coarse granularity of RFM_{ab}. These challenges do not undermine the mechanism's ability to prevent Rowhammer bit flips, but they do limit its efficiency and suitability for performance-sensitive systems. As such, there is a pressing need for a more fine-grained, low-overhead mitigation framework that minimizes latency and preserves DRAM throughput while maintaining effective protection. To address the performance and energy limitations of the existing PRAC+ABO framework, we propose ***PRACtical*** — a minimal and efficient redesign that improves both responsiveness and scalability of PRAC+ABO. This new design introduces two key enhancements. First, it leverages subarray-level increment logic to decouple counter updates from global precharge timing, allowing subsequent activations to proceed in other subarrays without incurring the additional latency associated with PRAC's read-modify-write operations. While prior works have explored and leveraged subarray-level parallelism to enhance memory performance and efficiency [23, 37], the introduction of the PRAC standard imposes new constraints that limit such parallelism. In this work, we demonstrate that with modest modifications to the DRAM circuitry, it is possible to restore a degree of subarray parallelism by overlapping the activation of a new row with the counter update of the previously accessed row. This significantly reduces access delay and enables known subarray-level parallelism for per-row counter updates. Second, PRACtical provides a new RFM command called RFM_MASK that eliminates stalling every bank once ABO is signalled from DRAM to MC, thereby ensuring that only the affected banks are stalled in response to a threshold violation. This fine-grained control minimizes unnecessary interference with unrelated memory traffic and improves overall system throughput. PRACtical makes small modifications to the memory controller and DRAM.

Together, these enhancements enable precise, low-latency and low-energy Rowhammer mitigation with minimal disruption to normal DRAM operations, as demonstrated in the bottom row of Figure 1 (a) and (b). Our performance evaluation, detailed in Section 6, utilizing DRAM simulation with Ramulator [43], demonstrates that PRACtical achieves mean performance improvement of 8% over opportunistic PRAC+ABO for high-performance and memory-intensive applications while maintaining the security guarantees of PRAC+ABO-based mitigations [9, 53, 71]. In summary, the key contributions of this paper are:

- We identify two major performance bottlenecks in the PRAC+ABO mechanism: precharge-induced latency caused by PRAC updates, and coarse-grained channel-wide stalling due to RFM_{ab}.
- We propose a centralized increment circuit that enables subarray-level parallelism, minimizing the counter update overhead during precharge.
- We introduce a bank-level recovery scheme that minimizes unnecessary stalls by isolating mitigation to the affected bank.
- We integrate these two mechanisms into PRACtical, a low-overhead Rowhammer mitigation framework that improves performance while preserving the security guarantees of PRAC+ABO.
- We evaluate PRACtical using Ramulator, showing lower performance overheads compared to PRAC+ABO with minimal hardware modifications.

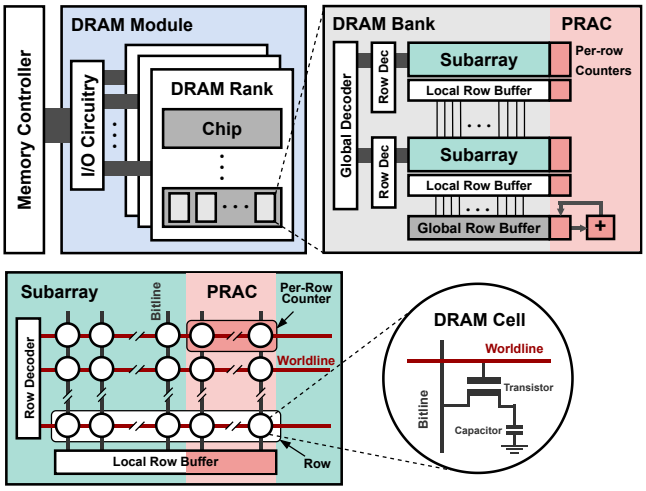

Figure 2: DRAM Organization and Per-Row Activation Counter (PRAC) extension.

2 Background

In this section, we provide an overview of key background topics, including DRAM architecture, the Rowhammer vulnerability, and existing mitigation strategies. We also describe the PRAC+ABO mechanism introduced in the DDR5 standard, which forms the basis for our proposed performance enhancements.

2.1 DRAM Architecture and Parameters

Modern DRAM is organized hierarchically into ranks, banks, subarrays, rows, and columns, as illustrated in Figure 2. At the top of this hierarchy, the memory controller communicates with DRAM modules via a dedicated memory channel. Each module is composed of one or more ranks, which share access to the memory channel in a time-multiplexed fashion. A rank comprises several DRAM chips, and each chip contains multiple banks. Within each bank are numerous subarrays, forming the fundamental building blocks of memory storage. Each subarray is implemented as a two-dimensional array of cells, accessed via rows (wordlines) and columns (bitlines). A single DRAM cell includes a capacitor, which holds a bit of data as an electric charge, and an access transistor that enables read and write operations. Each subarray contains its own local row buffer, which temporarily holds the contents of an activated row. Only one subarray at a time can forward its data to the global row buffer shared across the bank, enabling read and write operations to proceed. To access data, the memory controller issues an ***Activate (ACT)*** command, which opens a specific row by transferring its contents into the row buffer. Subsequent read or write operations are performed on this open row, resulting in a low-latency row hit. If a different row within the same bank must be accessed, the current row must first be closed using a ***Precharge (PRE)*** command, which restores the contents of the row buffer to the array and resets the bitlines.

DRAM access behavior is governed by a set of well-defined timing parameters specified by the JEDEC standard [28], as summarized in Table 1. We use DDR5-3200AN timings. For example, the **Row Access Strobe (RAS)** latency defines the minimum delay between an ACT and a subsequent PRE command within the same bank. Additionally, to maintain data integrity, DRAM cells require periodic refreshing within a retention window denoted as t_{REFW}, typically *32ms*. To amortize the cost of refresh, DRAM divides its cells into 8192 refresh groups, and a **Refresh (REF)** command is issued every t_{REFI} = *3900ns* to refresh one group at a time.

Parameter	Description	Base	PRAC
t_{RAS}	Minimum time a row must be kept open	32ns	16ns
t_{RP}	Time to precharge an open row	15ns	36ns
t_{RC}	Time between successive ACTs to a bank	47ns	52ns
t_{RTP}	Minimum time for a PRE after a RD to the same bank	7.5ns	5ns
t_{WR}	minimum time for a PRE after a WR to the same bank	30ns	10ns

Table 1: DRAM Timing Standards

2.2 Read Disturbance Attacks

Modern DRAM is increasingly vulnerable to read disturbance effects, where the act of accessing one memory row can inadvertently influence the integrity of data stored in nearby rows. This phenomenon arises from the shrinking physical dimensions of DRAM cells and the reduced noise margins between them. As manufacturing processes continue to scale, the isolation between adjacent rows weakens, making DRAM cells more susceptible to charge leakage and electromagnetic coupling.

The most prominent example of a read disturbance attack is Rowhammer [36], where an attacker repeatedly activates (i.e., opens) a single DRAM row, known as the aggressor row, within a refresh interval. If the number of activations exceeds a certain threshold, electrical interference can induce bit flips in adjacent victim rows. Since its discovery, Rowhammer has been shown to compromise system security in various ways, including enabling privilege escalation [65], breaking isolation between virtual machines [56], and subverting browser-based sandboxes [21].

Beyond Rowhammer, newer variants continue to expose the fragility of DRAM. RowPress [41] demonstrates that simply holding a row open for an extended duration—without repeated activations—can cause similar disturbance effects in neighboring rows. This shows that both temporal access frequency and row residency time can lead to data corruption, expanding the threat model beyond traditional Rowhammer.

The security implications of these attacks are severe. Bit flips in sensitive memory structures such as page tables, kernel space, or encryption keys can be exploited to gain arbitrary code execution, bypass memory isolation, or compromise confidentiality [16, 20, 21, 56]. To mitigate read disturbance attacks, a wide range of defenses have been proposed. These typically follow a two-phase approach: (1) Detecting aggressor rows, using techniques such as memory controller-based row activation counters [34], probabilistic sampling [36, 68], or in-DRAM tracking [26, 54]; and (2) Applying preventive measures, such as Target Row Refresh (TRR) [16, 44], domain-aware memory allocation [62], or row remapping [59]. While many of these defenses are effective at reducing bit flips, they often incur significant performance overheads and hardware complexity, especially when mitigation is applied conservatively to

avoid false negatives. As DRAM continues to scale and disturbance thresholds fall, there is a growing need for efficient, low-latency, and fine-grained mitigation mechanisms that preserve performance without compromising reliability or security.

2.3 PRAC and ABO in Modern DRAM Standards

To address the growing threat of Rowhammer attacks, recent JEDEC standards have introduced two complementary in-DRAM mitigation mechanisms: ***Per-Row Activation Counter (PRAC)*** and ***Alert Back-Off (ABO)***. These mechanisms aim to detect and mitigate malicious or excessive row activations efficiently, while remaining scalable for high-density DRAM systems.

2.3.1 PRAC. This feature is designed to mitigate Rowhammer attacks with minimal area and power overhead by embedding activation tracking directly within the DRAM array. Specifically, PRAC extends each DRAM row with a dedicated activation counter. Every time a row is precharged, its corresponding counter is incremented by the increment logic in the global row buffer shared between all subarrays within a bank, as illustrated in Figure 2. This increment operation is performed during the precharge phase and is implemented as a *read-modify-write (RMW)* sequence at the bank level. When a precharge command (*PRE*) is issued, the DRAM bank reads the activation counter of the recently accessed row, updates its value, and writes it back before deactivating the wordline. This sequence introduces additional latency to the precharge operation, requiring DRAM timing parameters to be extended to accommodate the counter update overhead. While PRAC enables fine-grained tracking of row activations without relying on external memory controller storage, its bank-wide RMW implementation creates performance bottlenecks by delaying subsequent accesses to other non-conflicting subarrays until the counter update completes, as illustrated in top and middle rows of the Figure 1 (a). The performance impact of this overhead is analyzed in detail in Section 3.

2.3.2 ABO. This mechanism complements PRAC by providing a signaling interface between the DRAM device and the memory controller. When a PRAC counter in any bank reaches a pre-defined threshold, the corresponding bank sends an ABO signal to notify the memory controller that mitigation is needed. Upon receiving the ABO signal, the memory controller initiates a pre-recovery phase lasting *180ns*, during which it continues to serve memory requests normally as shown in Figure 3. After this window, the controller issues an $\mathsf{RFM}_{\mathsf{ab}}$ command to trigger targeted mitigation. Each $\mathsf{RFM}_{\mathsf{ab}}$ command incurs a latency of *350ns* and applies to all banks in the channel (typically 64 banks). During this period, the memory controller stalls requests to the channel while the banks perform recovery operations, such as refreshing victim rows, as shown in the middle row of Figure 1 (b). If multiple banks reach the threshold in close succession, the memory controller may need to issue multiple RFM commands, each separated by a recovery window. This can amplify the performance impact, especially under aggressive access patterns that frequently trigger mitigation events.

To prevent back-to-back ABO signals, the protocol requires a minimum number of *ACT* commands between two consecutive alerts. In Figure 3, this value is denoted by *n*. It also corresponds to the number of required *RFM* operations, as each *RFM* can service

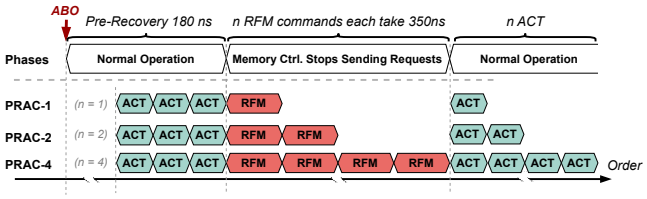

Figure 3: Alert-back-off (ABO) overview

only a limited number of victim rows per invocation. After all *RFM* commands are completed, one additional *ACT* is permitted. The allowed values for *n* are 1, 2, or 4, as illustrated by *PRAC-1*, *PRAC-2*, and *PRAC-4* in Figure 3, as well as in Figures in Sections 3 and 6. Consequently, the minimum and maximum ABO intervals range from *350ns* to *1500ns*, each preceded by a fixed *180ns* pre-recovery window.

2.4 Mitigations Based On PRAC+ABO

Several recent works build on the PRAC+ABO framework to strengthen its security guarantees and reduce its performance overheads. In this paper, we focus on three representative designs—Chronus [9], QPRAC [71], and MOAT [53]—which propose distinct approaches to improving the effectiveness and efficiency of in-DRAM RowHammer mitigation.

Chronus [9] introduces architectural modifications to decouple PRAC counter updates from the critical path of DRAM access. By relocating per-row counters to a dedicated metadata subarray, Chronus reduces the serialization overhead associated with precharge operations. It also proposes enhancements to the ABO protocol by holding the alert signal until all mitigations are complete, ensuring stronger coordination between DRAM and the memory controller.

QPARC [71] revisits PRAC's security model as originally proposed in the Panopticon framework [6], and identifies two new attack vectors that exploit timing gaps in counter updates. To address this, it introduces a priority-based queue that tracks frequently accessed rows and schedules them for mitigation through ABO. This queuing mechanism enables timely and targeted mitigation while preserving compatibility with the existing PRAC+ABO interface.

MOAT [53] focuses on simplifying mitigation logic by replacing queue-based designs with a tracking mechanism. It uses two SRAM registers to monitor one hot row at a time, applying proactive refreshes below a configurable threshold and issuing ABO signals when the threshold is exceeded. While MOAT reduces hardware complexity, it may face limitations under workloads with multiple simultaneous hot rows.

3 PRAC+ABO Limitations and Motivation

While PRAC+ABO provides a low-cost, JEDEC-standardized approach for in-DRAM Rowhammer mitigation, it introduces significant performance bottlenecks and security risks due to its coarse-grained design. This section presents a detailed analysis of the overheads introduced by PRAC updates and ABO-triggered channel-wide stalls. We also show how these issues can be exploited by adversaries to launch memory performance attacks. All insights are derived from cycle-accurate simulations, as described in Section 5.

3.1 Impact of PRAC Updates on DRAM Latency

PRAC extends each DRAM row with an activation counter that is incremented during the Precharge (PRE) command using a *read-modify-write (RMW)* operation, as illustrated in Figure 2. This update is performed at the bank level, using logic typically located near the global row buffer. As a result, when a PRE command is issued, the DRAM must first read, modify, and write back the counter value associated with the activated row before deactivating the wordline and returning to an idle state. This *serialization* delays subsequent memory operations, even when they target non-conflicting subarrays. To accommodate this RMW sequence, DRAM timing parameters are modified. Specifically, the precharge latency increases from *15ns* to *36ns*, and the activation-to-activation interval (t_{RC}) increases from *47ns* to *52ns*. Interestingly, the row active time (t_{RAS}) is reduced from *32ns* to *16ns*, partially offsetting the impact. Nevertheless, the increase in t_{RC} introduces a minimum *5ns* delay between successive row activations — particularly impactful in workloads with high row buffer conflicts.

To quantify this impact, we evaluated a set of memory-intensive workloads under two timing configurations: (1) a baseline DRAM configuration using standard DDR5 timings, and (2) a PRAC-enabled configuration with updated timings, as specified in Table 1. The results, shown in Figure 4, indicate that workloads experience an average slowdown of 6%, with a maximum of close to 20% for *462.libquantum*. These slowdowns are primarily attributed to increased latency introduced by PRAC counter updates, particularly in scenarios involving frequent row activations within the same bank. Although the row access time increases by only 10% (from 47ns to 52ns), the observed system-level performance overhead can reach up to 20%. This discrepancy arises because rows are not always precharged immediately after access. In many cases, a row remains open for extended durations due to row buffer policy, reducing the impact of a longer row access time. However, the increased precharge latency, nearly a 100% increase in the PRE timing, becomes the dominant contributor to performance degradation, especially when frequent rows are not immediately precharged.

We also evaluated multiple Alert thresholds (64, 128, and 256) and observed only minor variations in slowdown across these configurations. This suggests that the performance degradation is largely due to static timing overheads from PRAC's counter update logic, rather than the dynamic triggering of ABO or mitigation procedures. As expected, applications with low row conflict rates, such as *447_dealII* and *444_namd*, exhibited significantly less performance impact.

> ***Observation 1**: The updated DDR5 timings introduce performance overhead, slowing down benign workloads by geometric mean of 6%.*

PRACtical proposes reducing the performance overhead of PRAC by performing counter updates within the local row buffers of individual subarrays. Modern DRAM architectures commonly adopt a density-optimized open-bitline structure [11, 24, 31, 42], which places sense amplifiers on both sides of a subarray and allows adjacent subarrays to share these amplifiers. As a result, when a subarray is performing a counter update, the memory controller must delay not only accesses to that subarray but also to its neighboring subarrays. To evaluate the practical impact of this constraint, we measure the percentage of subarray conflicts relative to total row buffer conflicts. Assuming a configuration with 256 subarrays per bank, Figure 5 presents the ratio of subarray conflicts, including those involving adjacent subarrays, to total row buffer conflicts. The results demonstrate that subarray conflicts are relatively rare, accounting for only 1.24% on average across a diverse set of workloads. This observation indicates that enabling PRAC updates at the subarray level can enhance performance by permitting concurrent accesses to subarrays that are not involved in counter updates.

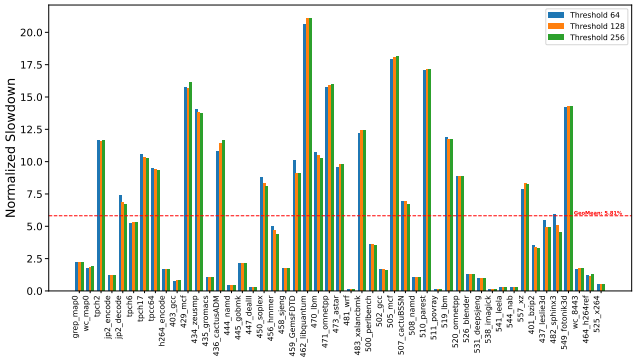

Figure 4: Percentage slow-down due to the increase in the latency of PRE command.

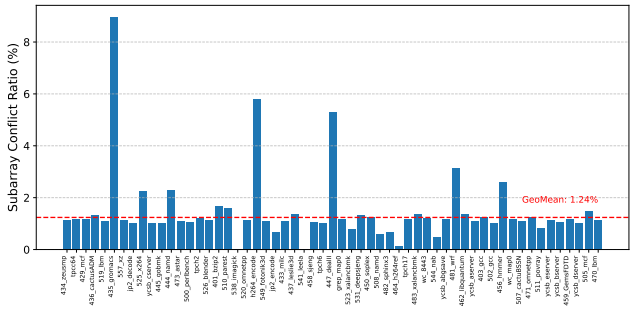

Figure 5: Ratio(%) of Subarray conflicts.

3.2 Inefficiency of Channel-Wide Stall of $\mathtt{RFM_{ab}}$

The ABO mechanism notifies the memory controller when a row's activation count crosses the threshold. After the *180ns* pre-recovery period, the controller issues an $\mathtt{RFM_{ab}}$ command, triggering a *350ns* stall across the entire memory channel even if only a small subset of banks require mitigation.

This coarse-grained stall mechanism is overly conservative. Across a wide range of benign workloads, our measurements show that, on average, only 1.16 out of 64 banks require mitigation when an ABO signal is raised (Figure 7) and maximum of 4 banks need mitigation across all recovery periods (Figure 8). As a result, over 90% of the banks are unnecessarily stalled during each recovery phase, significantly reducing memory-level parallelism and penalizing threads accessing unaffected banks.

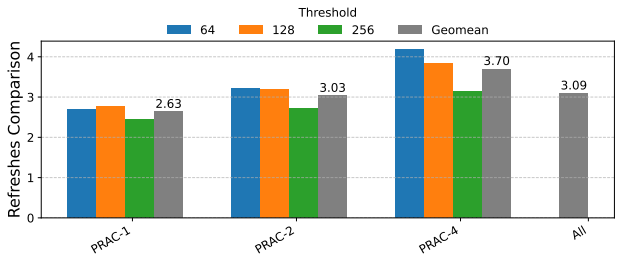

Figure 6: Comparison of number of RFM refreshes for opportunistic mitigation for PRAC-1, 2, and 4 and thresholds of 64, 128, and 256.

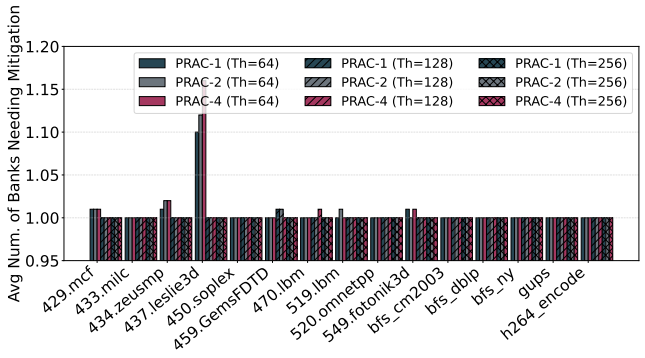

Figure 7: Average number of banks that need mitigation on an ABO signal for benign workloads

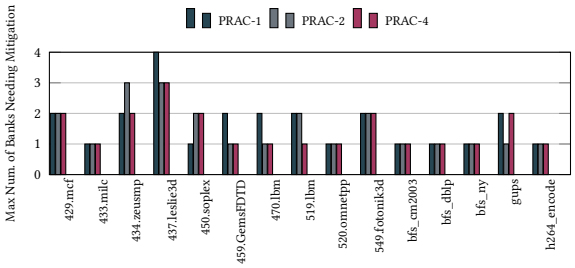

Figure 8: Maximum number of banks that need mitigation on an ABO signal for benign workloads.

Although a bank may not require recovery, i.e., it does not contain any hot rows, the issuance of the RFM_{ab} command blocks memory accesses to all banks within the channel. To utilize this otherwise idle period, prior works [9, 53, 71] adopt *opportunistic* mitigation strategies. These strategies proactively refresh potential victim rows whose counters are likely to reach the critical threshold in the near future. Specifically, upon issuance of RFM_{ab}, each bank refreshes rows adjacent to the row with the highest activation count (i.e., the most likely aggressor), thereby ensuring that banks do not remain idle and reducing the need for issuing additional RFM_{ab} commands in the future.

However, this approach introduces inefficiencies. Not all of the refreshes performed are strictly necessary, leading to redundant operations. To quantify this overhead, we compare the number of total RFM refreshes that are strictly required to those actually performed under opportunistic mitigation. As shown in Figure 6, the results, aggregated as the geometric mean across multiple PRAC variants (PRAC-1, PRAC-2, PRAC-4) and thresholds (64, 128, 256), demonstrate that opportunistic mitigation incurs more than a 3× increase in RFM refreshes. This analysis underscores a significant trade-off: while opportunistic mitigation reduces idle bank time and preempts future violations, it comes at the cost of approximately 70% higher RFM-related energy consumption in DRAM.

> ***Observation 2**: In benign workloads, fewer than 10% of banks require mitigation during an ABO-triggered recovery period, while opportunistic mitigation performs 3x more recovery refreshes than needed.*

3.3 Exploiting ABO: Performance Attacks

The coarse-grained nature of the ABO protocol introduces a new form of unfairness in DRAM systems, making them susceptible to performance slowdown attacks. Because the ABO signal triggers channel-wide stalls without identifying the specific bank responsible for the excessive activation, a malicious actor can exploit this limitation to repeatedly disrupt system-level memory access.

In particular, an attacker can intentionally issue frequent row activations to a single bank to induce an ABO event. Since the memory controller stalls the entire memory channel upon receiving an alert, without visibility into which bank requires mitigation, benign workloads distributed across other banks also suffer from the resulting stall. This lack of spatial granularity makes it extremely difficult to attribute the cause of the alert or to selectively suppress malicious access patterns, rendering existing defenses ineffective against such attacks. MOAT [53] proposed the Torrent-of-Staggered-ALERT (TSA) attack, a performance degradation strategy that carefully coordinates ALERTs across multiple DRAM banks. In this attack, each bank repeatedly activates a small set of rows (e.g., rows A, B, C, D, E) to trigger an ALERT. Crucially, banks issue ACTs in a staggered fashion: a bank only initiates activation once all targeted rows in the previous bank have caused ALERTs and entered the mitigation phase. This serialized activation ensures that when one bank is under mitigation, other banks are forced to stall, as there are no eligible rows to activate without interference. This deliberate serialization of ALERTs forms a torrent of staggered mitigation events, significantly throttling memory concurrency. The attack was shown to reduce system throughput by 24% with four banks and up to 52% with 17 banks, aligning with the tFAW constraint.

BreakHammer [8] introduces a score-based mitigation framework that assigns a score to each hardware thread based on its contribution to Rowhammer mitigation events. This approach is particularly effective when mitigation logic is implemented within the MC, where score attribution can be performed at finer spatial granularity, such as individual banks or row activations. However, performance degradation attacks may evade detection under BreakHammer when mitigations are signaled via coarse-grained mechanisms such as the Alert Back-Off (ABO) signal.

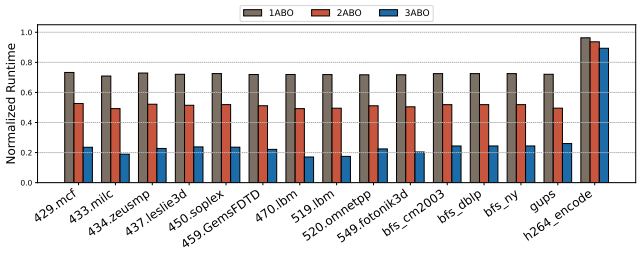

Figure 9: Percentage slowdown of benign applications when there is a 1, 2 and 3 ABO alerts happening in each tREFI

A broader class of memory performance degradation attacks exploits the observation that Rowhammer mitigations, such as recovery operations, are triggered more frequently at lower thresholds. As the mitigation frequency increases, overall system performance degrades significantly. To evaluate the performance cost associated with frequent mitigation, we study the impact of varying thresholds— 64, 32, and 16 as considered in prior works [9, 71]. Assuming a refresh interval (tREFI) of 3900 ns and a refresh operation latency (REF) of 410 ns, approximately 67 row activations can be issued within one tREFI interval. We consider a closed-row memory policy where each activated row is closed after access. For a realistic scenario, an attacker can alternate between two rows to issue repeated activations. Under a threshold of 64, this behavior can typically trigger one Alert per tREFI. For thresholds of 32 and 16, the number of ALERTs increases proportionally, allowing an attacker to induce two to three (or even four) ALERTs within the same interval. Based on this analysis, we evaluate performance degradation under scenarios involving one, two, and three ALERTs per tREFI duration to quantify the impact of aggressive mitigation triggering. As shown in Figure 9, even a single ABO event per interval results in a 20–30% slowdown for most applications. With three alerts per interval, performance degradation exceeds 80% in several cases. Interestingly, workloads such as h264_encode exhibit resilience due to their lower sensitivity to DRAM stalls, but most others are highly vulnerable. These results demonstrate that the lack of bank-level precision in ABO signaling can be exploited by attackers to inflict disproportionate slowdowns on benign threads, emphasizing the need for spatially-aware mitigation mechanisms.

> ***Observation 3**: Coarse-grained ABO signaling enables attackers to trigger repeated channel-wide stalls, leading to excessive performance loss.*

4 PRACtical Design and Implementation

To overcome the limitations of PRAC+ABO, we introduce PRACtical—an enhanced version that builds upon the original framework. PRACtical extends the underlying mechanisms to significantly reduce performance overheads while maintaining the robust security guarantees offered by state-of-the-art Rowhammer mitigation techniques. This section presents an overview of the design and implementation of PRACtical.

4.1 Overview and Design Goals

The core design principle of PRACtical is to minimize the performance overheads inherent in the original PRAC+ABO mechanism. Specifically, the design aims to achieve the following two objectives:

G1: **Reduce PRAC update latency through subarray-level decoupling.** Enable subarray-level counter updates to allow overlapping the counter update time of a row with activation of the next row when rows are non-conflicting at subarray-level, eliminating the *21ns* counter update delay.

G2: **Minimize unnecessary memory stalls with fine-grained recovery command.** Refine RFM_{ab} to operate at bank-level granularity, allowing the memory controller to stall only the affected banks instead of the entire channel.

4.2 Hardware Modifications

PRACtical introduces minimal hardware modifications to enable fine-grained, low-latency RowHammer mitigation by enhancing the PRAC+ABO feature.

First, to support subarray-level PRAC updates, the traditional bank-level increment logic, typically located near the global row buffer, is replaced with a centralized increment circuit that is connected to local row buffers in subarrays through a different bus (8-wire bus for 8-bit counters). This design allows counter updates without delaying the subsequent memory accesses to the memory bank that do not conflict at the subarray level. Correspondingly, the memory controller is extended with subarray mapping logic and an address decoder capable of identifying the target subarray for each memory request.

Second, to enable fine-grained bank-level recovery stalling, the DRAM chip is modified to include a new control register called the Bank Alert (BA) register. This register contains one bit per bank, where each bit indicates whether a given bank has any rows with an activation number higher than the Alert threshold. The contents of the BA register are communicated to the memory controller and serve as a mask, allowing the controller to selectively stall only the requests to the banks under mitigation while continuing to issue commands to unaffected banks.

4.3 PRACtical Mechanism

PRACtical introduces two core mechanisms to reduce the performance overheads associated with the PRAC+ABO framework while maintaining its security guarantees against RowHammer attacks.

4.3.1 Subarray-Level PRAC Updates. This mechanism improves subarray-level parallelism by enabling PRAC counter updates at the subarray level, rather than enforcing bank-wide delays. In the original PRAC design, counter updates introduce additional latency, particularly during precharge operations, because the increment logic is shared at the bank level. This design forces all subarrays within the bank to stall while the update completes. PRACtical addresses this limitation by introducing a centralized PRAC increment logic that connects to the local row buffers of all subarrays, as illustrated in Figure 10.

Upon receiving a memory access request, the memory controller computes the subarray identifier (Subarray ID) for the target row ❶. If the new request targets the same subarray or nearby subarray

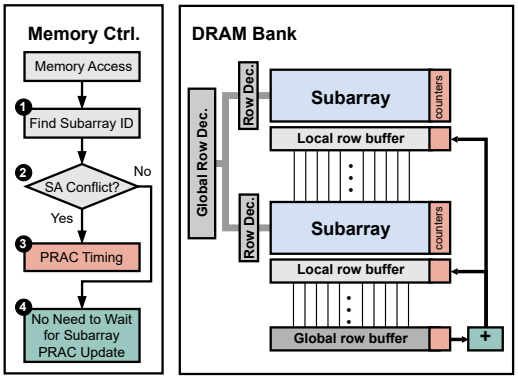

Figure 10: PRACtical adds subarray-level PRAC increment logic to enable independent counter updates and avoid bank-wide stalls across all subarrays.

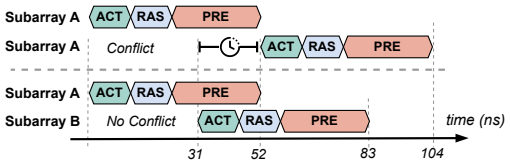

Figure 11: Illustration of conflicting (top row) and non-conflicting (bottom row) memory accesses across subarrays, demonstrating how PRACtical uses parallelism by avoiding unnecessary delay.

that is currently undergoing a PRAC update (i.e. subarray conflict) ❷, the controller applies the required timing constraints to ensure that the update completes before issuing a new activation ❸. In contrast, if the access targets a different non-conflicting subarray, the controller can proceed with the activation command immediately, without waiting for the PRAC update to finish ❹. In this case, counter value of the precharged row is forwarded to the increment unit and the entire row is transferred from the global to the local row buffer. This precharge operation takes 15 ns to complete. After this delay, the increment circuit updates the counter and transmits the new value to the corresponding local row buffer via a dedicated counter data bus. This approach allows subarrays to operate independently, preventing unrelated memory accesses from being delayed by PRAC updates in other subarrays, as illustrated in Figure 11. As a result, PRACtical reduces access latency, particularly in workloads that exhibit diverse subarray access patterns.

4.3.2 Bank-Level Stall for Recovery. PRACtical addresses the limitation of coarse-grained mitigation in the original PRAC+ABO framework by introducing bank-level recovery stall, enabling the memory controller to restrict mitigation only to the affected banks rather than stalling the entire memory channel. This selective mitigation is made possible through the use of a BA register, which functions as a mask to identify and isolate banks requiring Rowhammer mitigation as illuatrated in Figure 12.

When the PRAC counter of a bank exceeds the Alert threshold, the DRAM device sends the ABO signal to the memory controller ①. Upon receiving this signal, the memory controller enters a

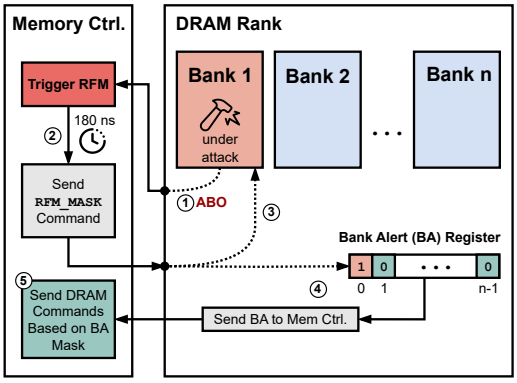

Figure 12: Bank-Level Recovery Stall in PRACtical Using the Bank Alert (BA) Register and RFM_MASK.

pre-recovery window, lasting 180 nanoseconds, during which it continues to process memory requests as normal ②. After this window, the controller issues $\texttt{RFM}_{\texttt{ab}}$ command to the DRAM to initiate recovery. PRACtical repurposes this command into a new command, termed RFM_MASK (Masked Refresh Management), which performs two key functions; it initiates recovery and functions as a register read command. First, the RFM_MASK command triggers the in-DRAM recovery mechanism for the affected bank ③. Second, it returns the contents of the BA register to the memory controller, indicating which banks are currently under mitigation ④. The BA register is reset once it is read. After its reset, a bank can set the corresponding bit in the register. The controller uses this information as a bitmask to manage access granularity during recovery. For any new memory request, the memory controller checks whether the target bank is marked as active in the BA register. If the request targets an unaffected bank, the controller proceeds to issue the command without delay. However, if the request targets a bank under recovery, the controller stalls the request until the mitigation phase completes ⑤.

Figure 13 illustrates how PRACtical leverages bank-level recovery stall to improve memory parallelism during RFM refreshes. In this example, Bank 1 exceeds the Alert threshold and triggers an ABO signal, initiating a short pre-recovery phase followed by the recovery phase. The banks modify their own bits in the register. Bank 1 will set its bit to 1. Upon receiving the RFM_MASK command the DRAM sends the contents of the BA register (`0b1001`) to the MC, which encodes the mitigation status of all banks. The BA mask in this case indicates that Bank 1 is under mitigation. As a result, the memory controller selectively stalls requests to Bank 1 while continuing to issue accesses to unaffected Banks 2 and 3. This fine-grained handling contrasts with the baseline PRAC+ABO design, where all banks would be stalled upon any ABO event. PRACtical thus reduces unnecessary interference and improves memory-level concurrency during mitigation.

5 Experimental Methodology

We utilize a cycle-accurate, open-source memory simulator, Ramulator 2.0 [43, 57]. Our evaluation is on the existing PRAC+ABO framework and our proposed solution, PRACtical. We use the provided

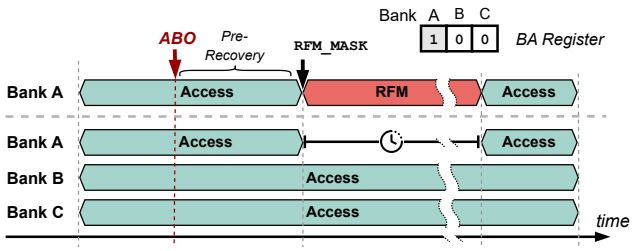

Figure 13: In this example, Bank 1 triggers ABO and enters recovery. Based on the BA mask, accesses to Banks 1 are stalled, while Banks 2 and 3 remain accessible.

PRAC+ABO with RFM in Ramulator2.0. Furthermore, we evaluate Chronus [9] with PRACtical and compare the performance.

System configuration is presented in table 2. We use a pool of traces from SPEC CPU2006 [1], SPEC CPU2017 [2], TPC [70], MediaBench [17] and YCSB [14]. The category of each trace is determined based on row buffer conflict per kilo instructions (RBMPKI). The traces are divided into High (H: ≥10 RBMPKI), Medium (M: 2–10 RBMPKI), and Low (L: <2 RBMPKI) memory usage, as shown in Table 3. For evaluation, we use 4 different traces to form groups of HHHH, MMMM, LLLL, HHMM, MMLL, and LLHH as mixed workloads, represented in Table 4. For evaluation, we use 10 workloads from each group and run 100M instructions for simulation. Our simulation configuration aligns with previous wroks [9, 50, 71].

6 Evaluation

In this section we evaluate the performance improvements and compare results against similar state-of-the-art works. First, PRACtical is compared against standard PRAC with an Alert Back-Off signal to show performance benefit over standard JEDEC specifications.

Table 2: Simulator Configurations

Parameter	Configuration
CPU	4-core, 4.2 GHz, 128-entry instruction window
Last-Level Cache	2MB per core, total 8MB (16-way set-associative)
Memory Controller	32-entry read/write queues Scheduling: FR-FCFS + Cap of 4 Address mapping: MOP
Main Memory	DDR5 DRAM, 1 channel, 2 ranks, 8 bank groups, 4 banks/group, 64K rows/bank

Memory Usage	Benchmarks
High (H)	429.mcf, 433.milc, 434.zeusmp, 450.soplex, 459.GemsFDTD, 462.libquantum, 470.lbm, 482.sphinx3, 483.xalancbmk, 510.parest, 519.lbm, 549.fotonik3d, gups, 520.omnetpp
Medium (M)	436.cactusADM, 473.astar, 507.cactuBSSN, 557.xz, jp2_decode, jp2_encode, tpcc64, tpch17, tpch2, wc_8443, wc_map0, ycsb_aserver, ycsb_bserver, ycsb_cserver, ycsb_eserver
Low (L)	401.bzip2, 403.gcc, 435.gromacs, 444.namd, 445.gobmk, 447.dealII, 456.hmmer, 458.sjeng, 481.wrf, 500.perlbench, 502.gcc, 508.namd, 511.povray, 523.xalancbmk, 526.blender, 538.imagick, 541.leela, 544.nab, grep_map0, tpch6, ycsb_abgsave

Table 3: Grouping of Benchmarks by Memory Usage

Workload	Mixed Types
HHHH	Mix0 to 9
MMMM	Mix10 to 19
LLLL	Mix20 to 29
HHMM	Mix30 to 39
MMLL	Mix40 to 49
LLHH	Mix50 to 59

Table 4: Mapping of Workload Types to Mixed Types

Then, we evaluate PRACtical's effectiveness on one PRAC+ABO-based solution, QPRAC. We also show how resilient PRACtical is to chain attacks 3.3.

6.1 Performance and Energy Evaluation

PRACtical vs. PRAC+ABO. The performance evaluation results comparing PRACtical with the standard (opportunistic) PRAC+ABO framework are presented in Figure 14 and geometric mean results are shown in Figure 15. The evaluation includes both PRACtical and PRAC across configurations with n=1,2, and 4 RFMs as PRAC-1, PRAC-2, and PRAC-4, respectively, with Alert thresholds of 64, 128, and 256. We use a suite of mixed workloads composed of traces with high, medium, and low memory usage characteristics. The results demonstrate that PRACtical outperforms the baseline PRAC+ABO framework. Along the mixes, high and medium memory intensity combinations showcase more speedup (up to 20%). Mean values show that PRACtical provides 7-9% better performance. This performance improvement benefits mostly from subarray-level PRAC updates. The overhead was 6% on average. Hence, PRACtical has 2-3% performance improvement due to bank-level recovery stalling. Although this performance improvement might seem very low, the critical target of this optimization is to eliminate energy inefficiency.

Energy Comparison. In this evaluation, we compare the energy consumption of PRACtical and PRAC+ABO with opportunistic mitigation. Figure 16 depicts the difference in the energy consumption due to extra RFM refreshes of opportunistic mitigation. The results align with our expectations; while the opportunistic mode of PRAC+ABO improves performance by preemptively refreshing rows before they become hot, it also introduces a large number of unnecessary refreshes, significantly increasing energy consumption. On average, across all evaluated recovery and threshold configurations, opportunistic PRAC+ABO consumes 19% more energy compared to PRACtical. Across different configurations of recoveries (1,2, and 4) and thresholds, the values slightly vary. The biggest difference happens with PRAC-4 (rec=4), and the threshold 64, which aligns with the observation of the most unnecessary RFM refreshes in sec 3. These findings underscore a fundamental trade-off: while opportunistic mitigation strategies can minimize performance degradation by anticipating high-activation rows early, this comes at the cost of substantially increased energy usage, reducing the overall efficiency of the memory system.

PRACtical vs Baseline architecture. In this evaluation, we assess the performance of PRACtical in comparison to a baseline architecture, where the baseline represents a DRAM system without any Rowhammer mitigation mechanisms. As illustrated in Figure 17, PRACtical achieves performance levels nearly identical to

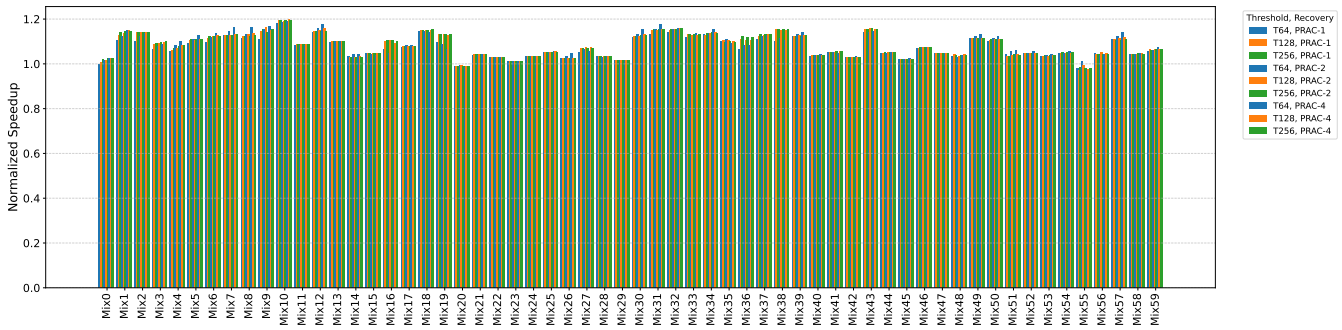

Figure 14: Performance evaluation of PRACtical using normalized speedup over PRAC+ABO

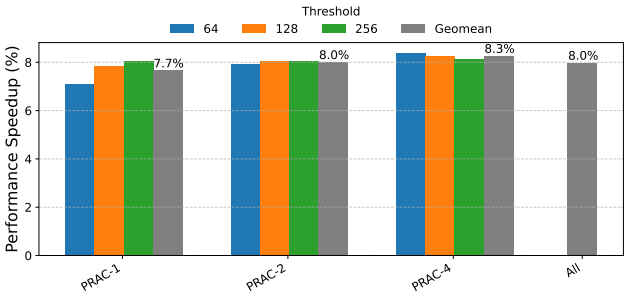

Figure 15: Performance Comparison of PRACtical and opportunistic PRAC+ABO for recoveries of 1, 2, and 4 and thresholds of 64, 128, and 256

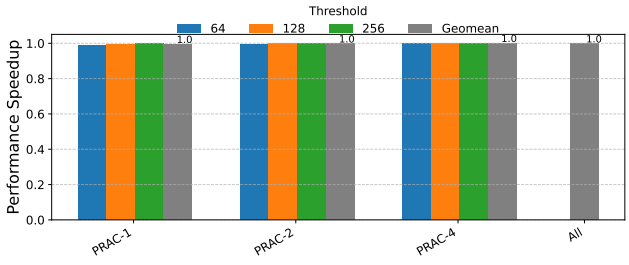

Figure 17: Performance Comparison of PRACtical and Baseline architecture for recoveries of 1, 2, and 4 and thresholds of 64, 128, and 256. Baseline architecture refers to no mitigation.

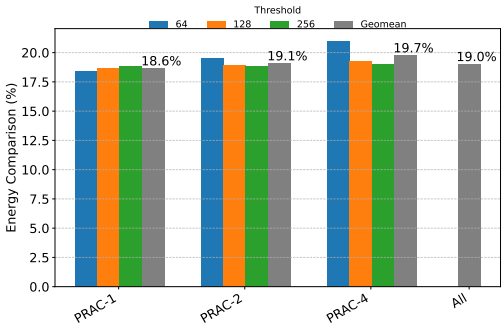

Figure 16: Energy Comparison of PRACtical and PRAC for recoveries of 1, 2, and 4 and thresholds of 64, 128, and 256

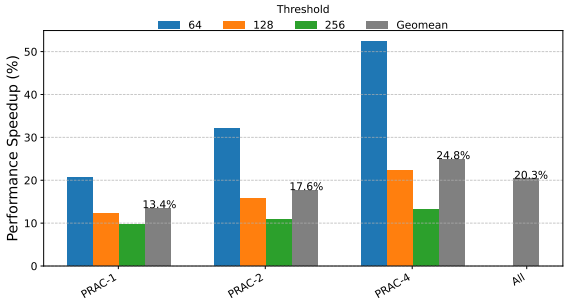

Figure 18: Performance Comparison of PRACtical and non-opportunistic PRAC for recoveries of 1, 2, and 4 and thresholds of 64, 128, and 256

the baseline. The only observable performance degradation is a minor slowdown of approximately 1%, which occurs under the configuration with a recovery count of 1 or 2 RFMs and a threshold of 64. This minimal overhead demonstrates that PRACtical introduces negligible performance penalties, effectively maintaining baseline performance even in the presence of Rowhammer protection.

Non-opportunistic PRAC+ABO vs PRACtical. We have shown that opportunistic mitigation has a wide energy-performance trade-off. In order to save performance, PRAC+ABO can be non-opportunistic, meaning that upon recovery (during $\mathsf{RFM}_{\mathsf{ab}}$), only the banks, that have rows with activation count equal to greater than Alert threshold, are undergoing recovery procedure. As expected, the banks that have all rows with counters less than Alert threshold will stay idle during this period, downgrading the performance. However, this method ensures only necessary number of refresh operations are performed, saving the energy. The results are shown in Figure 15. Our evaluation considers thresholds of 64, 128, and 256, with a configuration of 1, 2, and 4 RFMs per recovery. Across all combinations of recovery mechanisms (denoted as RFMs) and thresholds, we observe an average performance improvement of approximately 20%. The lowest performance gain, around 10%, occurs when the threshold is set to 256 and only one RFM recovery is employed. In contrast, the highest performance improvement is observed under the configuration with a threshold of 64 and one RFM recovery, where performance improves by more than 50%.

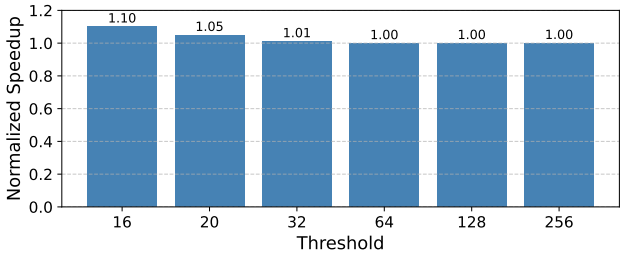

Figure 19: Normalized performance of Chronus implemented with PRACtical over PRAC+ABO.

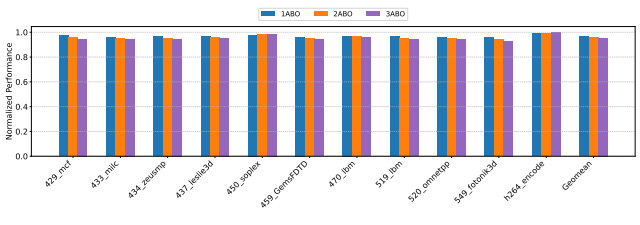

Figure 20: Normalized performance of PRACtical for each single benign benchmark when there is a performance attack.

Chronus equipped with PRACtical vs. PRAC. Chronus [9] optimizes the counter update mechanism in DRAM by relocating per-row activation counters to a dedicated, independent subarray. While this design introduces additional energy overhead, since each row activation results in two separate row accesses (one for the target row and one for the corresponding counter row) and occupies valuable DRAM space, it effectively eliminates the performance penalty typically incurred by in-place counter updates.

In this evaluation, we integrate the PRACtical bank stalling mechanism and subarray-level PRAC update into the Chronus architecture and compare the combined approach against the Chronus with an opportunistic PRAC+ABO framework. As shown in Figure 19, the hybrid Chronus+PRACtical implementation achieves the same performance as the base implementation with PRAC+ABO. These results demonstrate that simple and energy efficient PRACtical can indeed have the same result as expensive and inefficient base design. Note that even though the PRACtical has a small PRAC overhead due to subarray conflicts, this is negligible in complete design. Also note that the Chronus has more energy difference due to double row activation, which incurs an additional 19.07% energy consumption for each row access. We can conclude that PRACtical can achieve same (even better at lower thresholds) performance with very low energy consumption.

6.2 Hardware Overhead and Complexity Analysis

In this section, we discuss the practicality perspective of the proposed solution, PRACtical. Specifically, we discuss the practicality of PRACtical in two aspects: changes to JEDEC standards and the area overhead of changes.

Changes to JEDEC Standards: PRACtical requires several architectural modifications to both the DRAM device and the memory controller (MC) interface. The most critical modification is to decouple the increment circuit from the global circuit and connect it to the subarray's local row buffers. Another required modification stems from the fact that MCs typically lack knowledge of the subarray mapping within each DRAM bank. However, this can be addressed with a dedicated register in DRAM that holds subarray mapping information. Then, at boot time, the MC can read these registers and store the corresponding mapping functions in its internal registers for use during execution. Additionally, another hardware extension involves introducing a single register that maintains one bit per bank to track whether the bank needs mitigation. For a 64-bank channel, this mechanism requires only a 64-bit register, making it a low-overhead enhancement.

Area Overhead of Increment Circuit: PRACtical introduces two critical modifications to the DRAM architecture: (1) an increment unit and its control path integrated into local row buffers, and (2) an n-bit register to represent the bank mask in an n-bank DIMM. In our evaluation setup, the DIMM consists of 64 banks, which requires an 8-byte (64-bit) register to track bank-level status. The first modification decouples the increment logic from the global row buffer and shares it across local row buffers. Since the counter update overlaps with the activation of the next row, it must be routed to the correct subarray. To enable this, we need to have an additional global row address decoder that maps addresses to subarray IDs and row addresses within subarrays, along with an additional comparator in each subarray to resolve subarray ID matches. This ensures that counter updates can be independently delivered, even when the comparator is concurrently needed for the next precharge operation. We evaluate the area overhead of this modification using CACTI [5] and Synopsys Design Compiler [69], and find it to be only 0.03%, making it negligible.

6.3 Security Evaluation

Resilience against Memory Slowdown Attacks. In Section 3.3, we stated that the current PRAC+ABO design is vulnerable to MOAT's Torrent-of-Staggered-Alerts attack [53], which exploit the channel-wide stalling behavior of the ABO protocol. Besides, an attacker can trigger an Alert on a single bank to stall the entire memory channel frequently to cause performance attacks. PRACtical mitigates this vulnerability by restricting the Alert-induced stall to only the affected bank. As a result, the effectiveness of performance attack is significantly diminished, as it relies on a single bank being capable of blocking the entire channel.

To evaluate PRACtical's robustness, we repeated the performance attack experiments on a PRACtical-enabled system. The results, shown in Figure 20, use the *motivation* set—consisting of high memory-intensity, single-trace workloads. Under PRAC+ABO, the attack-induced slowdowns reached over 80%, severely degrading the performance of benign workloads. In contrast, PRACtical demonstrates strong resilience to such memory-based performance attacks. The system experiences an average (geometric mean) slowdown of less than 6%. This is primarily due to the attacker operating on the same banks accessed by these benchmarks. The most slowdown happens when attacker can induce 3 ABO stalls per `tREFI`s. Even in these cases, the maximum observed slowdown remains

below 6%, which is considered a tolerable level for secure memory systems.

Security of PRACtical: While PRAC+ABO alone does not constitute a complete security solution on its own. Any secure Rowhammer mitigation framework built upon PRAC+ABO must use a policy to ensure that any DRAM bank exceeding the activation threshold is properly mitigated. From this perspective, PRACtical is designed to preserve this baseline security requirement while delivering improved performance and energy efficiency. PRACtical targets optimizations in two key components of the PRAC+ABO framework: the per-row activation counter (PRAC) and the Alert-back-off signaling mechanism. PRACtical counter updates are committed in parallel with subarray accesses by leveraging a centralized increment circuitry, in conjunction with a memory controller that maintains subarray mappings. PRACtical provides memory with an alert signal to notify MC to stall requests at the bank level. Since we change channel-level blocking to bank-level blocking, we discuss the following potential security issue.

Since PRACtical allows access to non-mitigated banks, this potentially provides an attacker with an opportunity during the recovery for a period to send more activations to a target bank that is close to the Alert threshold. As an example, assume bank A sets its bit in RFM_MASK and sends alert and bank B is in normal operation of serving requests with its highest activation count close to the threshold, $T-1$. While the mitigation of bank A is performed, an attacker can send up to $N_{\mathsf{ACT}} = \frac{t_{\mathsf{RFM\_MASK}}}{t_{\mathsf{RC}}}$ ACTs to bank B. N_{ACT} is at most 6 since the RFM duration is 350ns and tRC is 52ns. Therefore, bank B will have rows with at most $T+5$ activation counts. PRACtical decreases the Alert threshold by this maximum value (5 in our system) to account for the worst-case scenario. If during recovery, the counter of a row in a bank reaches threshold, it sets the bit and sends the Alert signal. Once this recovery period is finished, in the next recovery, this bank also performs necessary mitigation. Since PRACtical uses a safe threshold, it guarantees the same security as the PRAC+ABO guarantees. In general, PRACtical does not need to lower the threshold. It can keep the ALERT threshold same, and have a second threshold that sets bank bits to 1 in BA register. Once there is row counter reaching the second threshold, its corresponding bank bit is set to 1. Once a row counter reaches the Alert threshold, an alert is sent to MC. For simplicity, we lower the threshold to lower and safe level.

7 Related Work

This paper first eagerly analyzes the efficiency of the existing JEDEC PRAC+ABO using a diverse set of benchmarks and secondly proposes a new solution called PRACtical to improve the performance of memory operations while keeping the same security guarantees provided by PRAC+ABO standards. PRACtical is a transparent hardware solution and can be deeply related to other works, such as counter-tracking mechanisms, hardware and software mitigations, and so on.

Counter-tracking mechanism. The concept of using activation counters was introduced and patented around by many works [4, 30, 32, 39, 55, 66, 67, 73]. Later works [25, 35, 44, 51] proposed secure in-DRAM tracking mechanisms to tackle energy and performance issues. Graphene [51] designs lightweight RowHammer protection to identify the frequent inputs of incoming stream achieving near-zero performance-energy overhead. Mithril [35] is the first to propose DRAM-memory controller cooperative mitigation using in-DRAM tracking. Aamer Jaleel et al. [25] proposes to manage trackers with probabilistic management policies like request-stream sampling and random evictions. PROTRR [44] uses frequent item counting to track aggressor rows in DRAM.

DRAM Performance Mechanisms. Subarray-level parallelism has been investigated in several prior studies. Kim et al. [37] proposed subarray-level parallel memory access mechanisms to improve memory bandwidth and performance. Hassan et al. [23] introduced a self-managing DRAM architecture that leverages the independence of subarrays to enable autonomous management within DRAM devices. Our approach is to eliminate the obscurity for subarray parallelism due to the increment circuit design in PRAC. Specifically, we propose modification of the circuit to enable overlapping only the counter update phase rather than the entire precharge phase with the subsequent activation, thereby minimizing performance overhead. Additionally, Chang et al. [12] explored subarray-level refresh operations, while HiRa [76] presents methods to reduce refresh latency by concurrently refreshing two rows connected to distinct charge restoration circuitries. BreakHammer [8] removes the performance overhead of Rowhammer attacks by identifying and throttling hardware threats that frequently triggter preventive actions.

Slowdown Attacks. Due to the impact of DRAM on performance, the research community has long explored threats that exploit shared memory subsystems to degrade performance, often in the form of Denial-of-Service (DoS) attacks [48]. While recent Rowhammer mitigation mechanisms aim to enhance system reliability, they can inadvertently introduce substantial performance overheads, particularly when relying on aggressive repair strategies such as frequent refreshes [7, 46] or row remapping [59]. Different works [10, 49, 72] describe ways to exploit RH mitigation for performance and side channel attacks.

8 Conclusion

In this paper, we tackle the performance and energy overhead inherent in PRAC+ABO. We propose PRACtical, a novel PRAC+ABO enhancement featuring a two-level optimization that enables subarray-level counter updates and allows DRAM banks to mitigate RowHammer independently without stalling. PRACtical introduces minimal hardware changes—namely, centralized increment circuit connected to subarray and a single global register, called bank-alert register, where each bit indicates whether a specific bank needs a mitigation — enabling the memory controller to continue serving requests to unaffected banks. Our evaluations show that PRACtical improves performance by a geomean of 8% and saves energy by an average of 20% over PRAC+ABO. Its performance is same as the baseline - no mitigation architecture and energy consumption is minimal. Overall, PRACtical provides an efficient and practical enhancement to PRAC+ABO, balancing performance and security with low hardware overhead.

References

[1] 2006. Standard Performance Evaluation Corporation (SPEC) CPU2006 Benchmark Suite. http://www.spec.org/cpu2006/. Accessed: 2025-04-10.

[2] 2017. Standard Performance Evaluation Corporation (SPEC) CPU2017 Benchmark Suite. http://www.spec.org/cpu2017. Accessed: 2025-04-10.

[3] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices* 51, 4 (2016), 743–755.

[4] Kuljit S Bains and John B Halbert. 2016. Distributed row hammer tracking. US Patent 9,299,400.

[5] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.

[6] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. 2021. Panopticon: A complete in-dram rowhammer mitigation. In *Workshop on DRAM Security (DRAMSec)*, Vol. 22. 110.

[7] Ishwar Bhati, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2015. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 235–246.

[8] Oğuzhan Canpolat, A Giray Yağlıkçı, Ataberk Olgun, Ismail Emir Yuksel, Yahya Can Tuğrul, Konstantinos Kanellopoulos, Oğuz Ergin, and Onur Mutlu. 2024. Breakhammer: Enhancing rowhammer mitigations by carefully throttling suspect threads. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 915–934.

[9] Oğuzhan Canpolat, A Giray Yağlıkçı, Geraldo F Oliveira, Ataberk Olgun, Nisa Bostancı, İsmail Emir Yüksel, Haocong Luo, Oğuz Ergin, and Onur Mutlu. 2025. Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance. *arXiv preprint arXiv:2502.12650* (2025).

[10] Oğuzhan Canpolat, A Giray Yağlıkçı, Geraldo F Oliveira, Ataberk Olgun, Oğuz Ergin, and Onur Mutlu. 2024. Understanding the security benefits and overheads of emerging industry solutions to dram read disturbance. *arXiv preprint arXiv:2406.19094* (2024).

[11] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. 2016. Low-cost inter-linked subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 568–580.

[12] Kevin Kai-Wei Chang, Donghyuk Lee, Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. 2014. Improving DRAM performance by parallelizing refreshes with accesses. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 356–367. https://doi.org/10.1109/HPCA.2014.6835946

[13] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 55–71.

[14] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[15] 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)*. 1001–1018.

[16] Pietro Frigo, Emanuele Vannacc, 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. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 747–762.

[17] Jason E Fritts, Frederick W Steiling, Joseph A Tucek, and Wayne Wolf. 2009. MediaBench II video: Expediting the next generation of video systems research. *Microprocessors and Microsystems* 33, 4 (2009), 301–318.

[18] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. 2024. AI and Memory Wall. *IEEE Micro* 44, 3 (2024), 33–39. https://doi.org/10.1109/MM.2024.3373763

[19] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying complex workload-dram interactions: An experimental study. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 3 (2019), 1–50.

[20] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. 2018. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 245–261.

[21] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer. js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 300–321.

[22] Hasan Hassan, Ataberk Olgun, A Giray Yaglikci, Haocong Luo, and Onur Mutlu. 2022. A case for self-managing dram chips: Improving performance, efficiency, reliability, and security via autonomous in-dram maintenance operations. *arXiv* (2022), 2207–13358.

[23] Hasan Hassan, Ataberk Olgun, A Giray Yağlıkçı, Haocong Luo, Onur Mutlu, and ETH Zurich. 2024. Self-Managing DRAM: A Low-Cost Framework for Enabling Autonomous and Efficient DRAM Maintenance Operations. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 949–965.

[24] Kiyoo Itoh. 2001. *VLSI Memory Chip Design*. Springer.

[25] Aamer Jaleel, Stephen W Keckler, and Gururaj Saileshwar. 2024. Probabilistic tracker management policies for low-cost and scalable rowhammer mitigation. *arXiv preprint arXiv:2404.16256* (2024).

[26] Aamer Jaleel, Gururaj Saileshwar, Stephen W Keckler, and Moinuddin Qureshi. 2024. Pride: Achieving secure rowhammer mitigation with low-cost in-dram trackers. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1157–1172.

[27] JEDEC. 2020. JESD79-5: DDR5 SDRAM Standard.

[28] JEDEC. 2024. *JEDEC Updates JESD79-5C DDR5 SDRAM Standard: Elevating Performance and Security for Next-Gen Technologies*. https://www.jedec.org/news/pressreleases/jedec-updates-jesd79-5cddr5-sdram-standard-elevating-performance-and-security Accessed: 2025-03-31.

[29] JEDEC. 2024. JESD79-5C: DDR5 SDRAM Standard.

[30] Ingab Kang, Eojin Lee, and Jung Ho Ahn. 2020. CAT-TWO: Counter-based adaptive tree, time window optimized for DRAM row-hammer prevention. *IEEE Access* 8 (2020), 17366–17377.

[31] Brent Keeth, R. Jacob Baker, Scott Benton, and Brian Johnson. 2007. *DRAM Circuit Design: Fundamental and High-Speed Topics* (2nd ed.). Wiley-IEEE Press.

[32] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. 2014. Architectural support for mitigating row hammering in DRAM memories. *IEEE Computer Architecture Letters* 14, 1 (2014), 9–12.

[33] 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 *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 638–651.

[34] Kwangrae Kim, Jeonghyun Woo, Junsu Kim, and Ki-Seok Chung. 2021. Hammerfilter: Robust protection and low hardware overhead method for rowhammer. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 212–219.

[35] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W Lee, and Jung Ho Ahn. 2022. Mithril: Cooperative row hammer protection on commodity dram leveraging managed refresh. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1156–1169.

[36] 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. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.

[37] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A case for exploiting subarray-level parallelism (SALP) in DRAM. *ACM SIGARCH Computer Architecture News* 40, 3 (2012), 368–379.

[38] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2018. {ZebRAM}: Comprehensive and compatible software protection against rowhammer attacks. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 697–710.

[39] Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. 2019. TWiCe: Preventing row-hammering by exploiting time window counters. In *Proceedings of the 46th International Symposium on Computer Architecture*. 385–396.

[40] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. 2021. Stop! hammer time: rethinking our approach to rowhammer mitigations. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 88–95.

[41] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. 2023. Rowpress: Amplifying read disturbance in modern dram chips. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–18.

[42] Haocong Luo, Taha Shahroodi, Hasan Hassan, Minesh Patel, A Giray Yağlıkçı, Lois Orosa, Jisung Park, and Onur Mutlu. 2020. CLR-DRAM: A low-cost DRAM architecture enabling dynamic capacity-latency trade-off. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 666–679.

[43] Haocong Luo, Yahya Can Tuğrul, F Nisa Bostancı, Ataberk Olgun, A Giray Yağlıkçı, and Onur Mutlu. 2023. Ramulator 2.0: A modern, modular, and extensible dram simulator. *IEEE Computer Architecture Letters* 23, 1 (2023), 112–116.

[44] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. 2022. Protrr: Principled yet optimal in-dram target row refresh. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 735–753.

[45] Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* 2, 2 (1982), 143–152.

[46] Janani Mukundan, Hillery Hunter, Kyu-hyoun Kim, Jeffrey Stuecheli, and José F Martínez. 2013. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 48–59.

[47] Onur Mutlu and Jeremie S Kim. 2019. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2019), 1555–1571.

[48] Thomas Moscibroda Onur Mutlu. 2007. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX security*.

[49] Ravan Nazaraliyev, Yicheng Zhang, Sankha Baran Dutta, Andres Marquez, Kevin Barker, and Nael Abu-Ghazaleh. 2025. Not so Refreshing: Attacking GPUs using RFM Rowhammer Mitigation. In *34th USENIX Security Symposium (USENIX Security 25)*.

[50] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F Oliveira, and Onur Mutlu. 2024. {ABACuS}:{All-Bank} Activation Counters for Scalable and Low Overhead {RowHammer} Mitigation. In *33rd USENIX Security Symposium (USENIX Security 24)*. 1579–1596.

[51] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. 2020. Graphene: Strong yet lightweight row hammer protection. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[52] Moinuddin Qureshi. [n. d.]. AutoRFM: Scaling Low-Cost In-DRAM Trackers to Ultra-Low Rowhammer Thresholds. ([n. d.]).

[53] Moinuddin Qureshi and Salman Qazi. 2024. Moat: Securely mitigating rowhammer with per-row activation counters. *arXiv preprint arXiv:2407.09995* (2024).

[54] Moinuddin Qureshi, Salman Qazi, and Aamer Jaleel. 2024. MINT: Securely mitigating rowhammer with a minimalist in-dram tracker. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 899–914.

[55] 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 *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 699–710.

[56] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)*. 1–18.

[57] SAFARI Research Group. 2021. Ramulator V2.0. https://github.com/CMU-SAFARI/ramulator2.

[58] 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*. 1056–1069.

[59] Anish Saxena, Saurav Mathur, and Moinuddin Qureshi. 2024. Rubix: Reducing the overhead of secure rowhammer mitigations via randomized line-to-row mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1014–1028.

[60] Anish Saxena and Moinuddin Qureshi. 2024. Start: Scalable tracking for any rowhammer threshold. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 578–592.

[61] Anish Saxena, Gururaj Saileshwar, Prashant J Nair, and Moinuddin Qureshi. 2022. Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 108–123.

[62] Anish Saxena, Walter Wang, and Alexandros Daglis. 2024. Preventing Rowhammer Exploits via Low-Cost Domain-Aware Memory Allocation. *arXiv preprint arXiv:2409.15463* (2024).

[63] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM errors in the wild: a large-scale field study. *SIGMETRICS Perform. Eval. Rev.* 37, 1 (June 2009), 193–204. https://doi.org/10.1145/2492101.1555372

[64] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM errors in the wild: a large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems* (Seattle, WA, USA) *(SIGMETRICS '09)*. Association for Computing Machinery, New York, NY, USA, 193–204. https://doi.org/10.1145/1555349.1555372

[65] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15, 71 (2015), 2.

[66] Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. 2016. Counter-based tree structure for row hammering mitigation in DRAM. *IEEE Computer Architecture Letters* 16, 1 (2016), 18–21.

[67] 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)*. IEEE, 612–623.

[68] 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*. 1–6.

[69] Synopsys Inc. 2023. *Synopsys Design Compiler User Guide*. Synopsys Inc. https://www.synopsys.com/ Version Y-2023.03.

[70] Transaction Processing Performance Council. 2025. TPC Benchmarks. http://www.tpc.org/. Accessed: 2025-04-10.

[71] Jeonghyun Woo, Chris S Lin, Prashant J Nair, Aamer Jaleel, and Gururaj Saileshwar. 2025. Qprac: Towards secure and practical prac-based rowhammer mitigation using priority queues. *arXiv preprint arXiv:2501.18861* (2025).

[72] Jeonghyun Woo and Prashant J Nair. 2025. Dapper: A performance-attack-resilient tracker for rowhammer defense. *arXiv preprint arXiv:2501.18857* (2025).

[73] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. 2023. Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 374–389.

[74] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[75] A Giray Yağlikçi, Minesh Patel, Jeremie S Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, et al. 2021. Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 345–358.

[76] A. Giray Yağlikçi, 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. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 815–834. https://doi.org/10.1109/MICRO56248.2022.00062