START: Scalable Tracking for Any Rowhammer Threshold

Anish Saxena
Georgia Tech
Atlanta, USA
asaxena317@gatech.edu

Moinuddin Qureshi
Georgia Tech
Atlanta, USA
moin@gatech.edu

Abstract—The Rowhammer vulnerability is worsening, with the Rowhammer Threshold (T_{RH}) reducing from 139K to 4.8K activations over the last decade. As thresholds reduce further, the number of possible aggressor rows increases inversely, making it difficult to reliably track such rows in a storage-efficient manner for typical Rowhammer defenses. To be secure at lower thresholds, academic trackers like Graphene must dedicate prohibitively high storage (hundreds of KBs to MBs) at the chip's design time. Recent in-DRAM trackers from the industry, such as DSAC-TRR, perform approximate tracking and sacrifice guaranteed protection for reduced storage overheads, leaving DRAM vulnerable to Rowhammer attacks. Ideally, we seek a configurable tracker that is secure and precise, incurs negligible dedicated storage and performance overheads, and scales at deployment to track arbitrarily low thresholds.

To that end, we propose *START - a Scalable Tracker for Any Rowhammer Threshold*. Rather than relying on dedicated SRAM structures, START dynamically repurposes a small fraction of the Last-Level Cache (LLC) to store tracking metadata. START leverages the observation that while the memory contains millions of rows, typical workloads touch only a small subset of rows within a refresh period of 64ms. Thus, allocating tracking entries on demand reduces storage significantly. If the application does not access many rows in memory, START does not reserve any LLC capacity. Otherwise, START dynamically uses 1-way, 2-way, or 8-way of the cache set based on demand. START consumes, on average, 9.4% of the LLC capacity to store metadata, which is 5× lower compared to dedicating a counter in LLC for each row in memory. We also propose START-M, a memory-mapped START for large-memory systems. Our designs require only 4KB SRAM for newly added structures and perform within 1% of idealized tracking even at T_{RH} of less than 100.

I. Introduction

DRAM scaling enables large-capacity memory that powers modern computing. DRAM cells become smaller and come closer to each other with successive process nodes. Unfortunately, such close packing leads to inter-cell interference. A prominent mode of this interference is *Rowhammer* [23], [28], wherein frequent activations to a DRAM row cause bit flips in nearby rows. Rowhammer remains a severe security threat [4], [11], [14], [17], [18], [20], [29], [29], [40], [42]. For example, flipping bits in page tables leads to privilege escalation attacks.

Alarmingly, Rowhammer keeps getting worse with each technology generation. When the phenomenon was characterized in 2014, the *Rowhammer Threshold (T_{RH})*, which denotes the activations required to an aggressor row within a 64ms refresh period to induce a bit-flip in a nearby row, was 139K (for DDR3). As shown in Figure 1(a), T_{RH} has steadily reduced, and the most recent study from 2020 reported T_{RH} of only 4.8K (for LPDDR4). The T_{RH} for current generation (DDR5)

and future generation (DDR6) devices is expected to be much lower. Between 2014 and 2020, the threshold reduced by 30X, and if the trend continues, we can expect sub-100 thresholds by the end of this decade. As systems remain deployed for several years, effective Rowhamer solutions must handle not only current but also future thresholds. Our goal is to develop a practical and configurable Rowhammer defense that works for a range of Rowhammer thresholds. In line with prior works [32], [37], we focus on low future thresholds of less than 500.

The typical solution for mitigating Rowhammer consists of (i) a *tracking mechanism* to identify an aggressor row (that is estimated to reach T_{RH} activations), and (ii) a *mitigating action*, such as refreshing neighboring victim rows. In this paper, our focus is the tracking mechanism. As the threshold reduces, the number of rows that can become aggressors increases, therefore the required tracking resources must increase in inverse proportion to the threshold (doubling when the threshold gets halved). Tracking aggressor rows with low storage and performance overhead in a secure manner has been a key subject of Rowhammer research, as shown in Figure 1(b).

At thresholds above 10K, tracking resources can be obviated by issuing mitigations probabilistically, as is done in PRA [23] and PARA [28]. However, probabilistic solutions incur considerable performance overheads at lower thresholds due to unnecessary mitigations. For a threshold of 10K and lower, tracking and issuing mitigation selectively when a row reaches T_{RH} activations reduces performance overheads. An *Ideal Tracker* provisions one SRAM counter for each memory row. However, it incurs significant SRAM overheads. We note that while the memory system contains millions of rows, the fraction of rows that are likely to reach TRH is still fairly small. Recent proposals dedicate SRAM tables to identify hot rows by tracking only a small subset of rows, either at the memory controller (e.g., Graphene [36]) or inside the DRAM-chip (e.g., Mithril [26]). To illustrate, Graphene, a storage-efficient tracker, requires 170KB at T_{RH} of 8K for 64GB DDR5 memory with 8 million rows. Unfortunately, as the thresholds reduce, the number of rows that can reach T_{RH} increases, so the number of rows to track also increases. For example, for the same 64GB memory, at T_{RH} of 1K and 256, Graphene requires 1.4MB and 5.2MB, respectively.

Recent industrial solutions store the aggressor row counters in-DRAM (e.g., TRR [14], DSAC-TRR [19]). Unfortunately, the tracker deployed in DDR4 does not track all aggressors and is vulnerable [14]. Recent white papers from JEDEC [21] [22] clearly state that "in-DRAM mitigations cannot eliminate
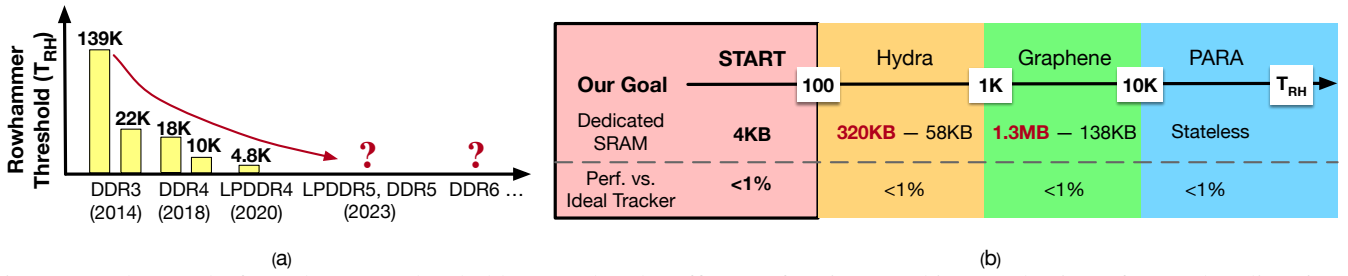
Fig. 1: (a) The trend of Rowhammer Threshold (T_{RH}) (b) The efficacy of various tracking mechanisms for our baseline 64GB memory system. Current solutions are not scalable to T_{RH} of sub-100. Our proposed design, START, efficiently tracks any T_{RH}.

all forms of Rowhammer attacks". Thus, the systems remain vulnerable even in the presence of these in-DRAM TRR mitigations. Furthermore, recent research from the industry on developing trackers for newer versions of in-DRAM TRR focuses mainly on doing so in an approximate manner to reduce storage overheads while still suffering from significant escape probability (for example, the recent DSAC-TRR [19] incurs 13.9% probability of aggressor escaping detection between two mitigations), rendering such upcoming schemes insecure, and leaving future systems still vulnerable to Rowhammer attacks.

It is possible to lower the storage overhead of tracking by placing the tracking table (one counter for each row) within the DRAM and caching the entries on demand [23]. The small counter-cache is susceptible to thrashing, so the recently proposed Hydra tracker [37] uses an SRAM filter to track rows at a group level, eliminating unnecessary accesses to the cache of per-row entries and minimizing the performance penalty. Hydra incurs a modest SRAM overhead of 186KB at T_{RH} of 256, but the slowdown increases to more than 10% at ultra-low T_{RH} of 64. Ideally, we need a solution which **(1)** precisely tracks activation counts at an arbitrarily low Rowhammer threshold, **(2)** is configurable to any Rowhammer threshold without being restricted by the size of the dedicated structures provisioned at design time, **(3)** incurs negligible SRAM overheads for newly added structures, and **(4)** performs similar to idealized tracking. We develop such a solution in this paper.

This paper proposes *Scalable Tracking for Any Rowhammer Threshold (START)*, which precisely tracks activations of each row in memory and is well suited to thresholds of 256 and lower. We leverage the observation that applications that utilize the last-level cache (LLC) well typically do not access millions of rows within 64ms, and applications that access a large number of rows within 64ms tend to have poor locality and are less sensitive to LLC capacity. Our key insight is to obviate the dedicated SRAM storage of tracking by leveraging the LLC to store the per-row counters dynamically. Typical workloads access only a small fraction of the memory rows within a period of 64ms, so the storage overhead is reduced significantly by tracking only the accessed rows.

If the application does not access rows within 64ms, START does not reserve any LLC capacity. Otherwise, START dynamically allocates ways within a cache set when new rows are accessed. A 64-byte line can store tracking metadata of 32 rows (including the row-tag). With 16MB of LLC capacity, reserving just 1-way across all sets holds the tracking entries of

up to 512K rows, which is sufficient in the common case (our system contains 64GB memory with 8 million rows). When a set requires more than 32 tracking entries, the allocation of that set is increased from 1-way to 2-way, and finally from 2-way to 8-way – sufficient to track all 512 rows that map to the set with untagged counters. We require just 2 bits of state per set (4KB of SRAM, 0.02% overhead) to track the per-set allocation. On average, START requires just 9.4% of the LLC capacity for counters, minimizing performance loss and performing within 1% of an ideal per-row tracker.

The structures for tracking the state of aggressor rows must be provisioned at the design time for prior works. The chip designer must decide what would be the Rowhammer threshold during the system's lifetime, and this information may not be available. It is, therefore, desirable for a solution to be configurable (say, at boot time) to the correct Rowhammer threshold, without being constrained by the size of the dedicated structures. Unlike prior schemes, START enables such reconfigurability, as the tracking state is created dynamically based on need. START uses a two-byte register, which is configured at boot time, allowing START to track any threshold, while still performing within 1% of an ideal tracker.

To support large-capacity memory systems and higher thresholds, we further propose *Memory-Mapped START (START-M)*, where the tracking data for all memory rows is stored in the DRAM and accessed only when the number of tracking entries exceeds the dedicated fraction of LLC capacity (8-ways per set). As START-M stores up to 2.75 million tagged tracking entries in the 8-ways of LLC, the number of memory accesses for obtaining tracking entries is negligible. We evaluate START-M with 64GB of DRAM per core and observe that START-M uses less than 12% of the LLC capacity for tracking and performs within 1% of an idealized tracker. Finally, our open-source simulation infrastructure is available at https://github.com/Anish-Saxena/rowhammer_champsim.

Overall, our paper makes the following contributions:

1) To the best of our knowledge, we are the first to propose a configurable tracker which scales to sub-100 threshold.
2) We propose START, which obviates the dedicated SRAM tracking overheads by leveraging the LLC.
3) We reduce the storage consumed for tracking by dynamically allocating per-set space based on demand.
4) Our memory-mapped START design scales to large-memory systems and supports higher thresholds.

II. Background and Motivation

A. DRAM Organization and Timing

Modern DRAM-based memory is organized logically into channels, sub-channels, ranks, banks, and rows. In DDR5, each 64-bit channel consists of two independent sub-channels which are 32-bit wide with a burst length of 16 to supply a 64B line. Each sub-channel has 32 banks organized as a 2D array of rows and columns with typical row size of 8KB. The bank contains a *row buffer* that caches the most recently opened row. To access data from DRAM, a row must be activated, which brings the data into the row buffer. To access data in another row, the bank clears the row buffer using the *precharge* command, followed by activation of the given row. DRAM cells also require periodic refresh operations (at 64ms).

An important DRAM timing parameter is t_{RC} *(Row Cycle Time)*, which determines the time between consecutive activations for a given bank. The T_{RC} for DDR5 systems is approximately 45ns, which means a bank can encounter up-to 1.36 million activations (ACT_{max}) in the refresh window of 64ms, after discounting the time spent in refresh.

B. Rowhammer and Security Threat

Rowhammer occurs when frequently activated rows cause bit-flips in nearby rows. *Rowhammer Threshold (T_{RH})* denotes the number of activations required to any row, using any access pattern, to induce bit-flips in the nearby row. When the Rowhammer phenomenon was first discovered in 2014, T_{RH} was 139K, whereas it has reduced by 30X to 4.8K [24] in 2020. T_{RH} is likely to reduce even further for future DRAM technology. For example, if the trends hold, then a similar reduction of 30X would render a T_{RH} of less than 100 by the end of the decade. Therefore, it is important that the solutions for Rowhammer are designed to tolerate not just the current T_{RH} but also T_{RH} for future nodes.

Rowhammer poses a serious threat to system security and gives the attacker a powerful attack vector to flip bits in Page-Tables for privilege escalation [46] or exploit the data-dependence of Rowhammer to read confidential data [29].

C. Threat Model

We assume an unprivileged attacker that can run code natively on the system that is vulnerable to Rowhammer. The attacker can run process(es) under *user* privilege and exploit Rowhammer to flip bits in the page-table or in another program's data to corrupt it [46]. We assume the Rowhammer bit-flip occurs at the victim location when any row in memory incurs more than T_{RH} activations within the refresh interval of 64ms. Thus, the attack is successful if no mitigation is issued when a row has encountered more than T_{RH} activations.

D. Scaling Challenges for SRAM Trackers

The typical method to mitigate Rowhammer is to track the activations and issue a *victim refresh* when a row reaches T_{RH} activations. Prior studies have developed sophisticated algorithms to intelligently track aggressor rows by provisioning the tracking entries for only a small subset of memory rows.

The minimum storage for tracking depends on the number of rows that can encounter at-least T_{RH} activations within the refresh period. As T_{RH} reduces, rows that can reach the threshold increase and the storage for the tracking structures increases proportionately. In this paper, our goal is to develop a Rowhammer tracker that works at thresholds lower than 500. Table I shows the storage requirement of recent state-of-the-art trackers as T_{RH} is reduced from 4K to 64, for a 64-GB memory.

TABLE I: SRAM/CAM storage required for 64 GB memory (two 32-GB DIMM, 128-Banks, 8KB-Row, 8M Rows).

T_{RH}	Graphene (CAM)	DSAC-TRR (CAM)	Ideal Tracker (SRAM)	Goal
64	>8 MB	N/A	6 MB	
256 (target)	5.2 MB	N/A	8 MB	4 KB
1K	1.4 MB	68 KB	10 MB	
4K (current)	340KB	16 KB	12 MB	
Secure?	Yes	**No**	Yes	Yes

Ideal Tracker (One-Counter-Per-Row) dedicates one SRAM counter for each row. For a system with R rows and threshold of T_{RH}, it needs R entries, each of $log2(T_{RH})$ bits. The storage requirement of the ideal tracker ranges from 12MB to 6MB, as the threshold is varied from 4K to 64 (lower storage due to smaller counters). Ideal trackers are traditionally considered impractical due to prohibitive storage requirements.

Graphene [36] is a state-of-the-art tracker. It uses the Misra-Gries algorithm to identify top-N frequently accessed rows, where N is based on T_{RH}. While Graphene is effective at T_{RH} of 4K (requiring 340KB), its storage overhead grows to more than 8MB at sub-100 threshold. For example, at T_{RH} of 64, a 5-bit counter and 17-bit row-id for 40K potential aggressors takes up more SRAM (109KB per bank) than storing 128K 5-bit untagged counters (80KB), making Graphene worse than an ideal tracker. While more space-efficient Misra-Gries based trackers have recently been proposed (like ABACuS [35]), they still require high storage at T_{RH} of 64 (800KB) and use imprecise group-tracking, leading to excessive mitigations. Finally, to remain secure, such trackers must provision worst-case storage for lowest supported threshold at design-time, dedicating hundreds of KBs to several MBs of SRAM, even if such thresholds are never encountered by the system.

DSAC-TRR [19] is a recent tracker proposed by Samsung. It combines space-efficient tracking with stochastic insertions to minimize counters required to defend against *known* adversarial access patterns that employ decoy rows. DSAC-TRR trades off security for area efficiency with a 13.9% probability of escape for aggressor between mitigations during an attack at T_{RH} of 10K. Moreover, since the effective threshold of DSAC is $T_{RH}/2 - ACT_{tREFI}$, where ACT_{tREFI} can be as high as 255, it does not scale to T_{RH} below 500.

> **Key Takeaway:** The storage requirements of *intelligent* trackers like Graphene balloon at ultra-low thresholds compared to an idealized tracker. As trackers must provision worst-case storage at design-time, we need to make ideal tracking viable in this regmine for practical Rowhammer mitigation.

E. Scaling Challenges for Hybrid Tracker

The SRAM overheads of an ideal tracker can be reduced by placing the counter table within the DRAM and caching the entries on demand in a metadata cache, as proposed in *Counter-Based Row Activation (CRA)* [23]. Unfortunately, even in presence of large metadata-caches (64KB-256KB), CRA experiences a significant number of extra accesses for fetching counter lines because of poor spatial locality, causing drastic slowdown (averaging 25% [37]), limiting practical adoption.

A recent proposal, *Hydra* [37], uses a hybrid design where an SRAM filter reduces the DRAM accesses for per-row counters. Hydra contains an SRAM structure that performs aggregated tracking for a group of rows until a subset of the Rowhammer threshold is reached. Per-row tracking is enabled only for rows for which the group-level threshold is breached. Hydra at $T_{RH} = 500$ was shown to have low overheads and slowdown.

The SRAM overhead of Hydra depends on T_{RH} and the number of channels. For our baseline system with two DDR5 DIMMs, Hydra incurs an SRAM overhead of 186KB for a threshold of 256. However, as the threshold reduces, the number of entries in the Hydra SRAM structures must be increased in direct proportion (that is, 4X more entries if the threshold is reduced by 4X). So, at thresholds of 64 and 16, Hydra incurs an SRAM overhead of 544KB and 1.3MB, respectively. If the storage overheads are not increased, then Hydra incurs significant slowdowns at lower thresholds.
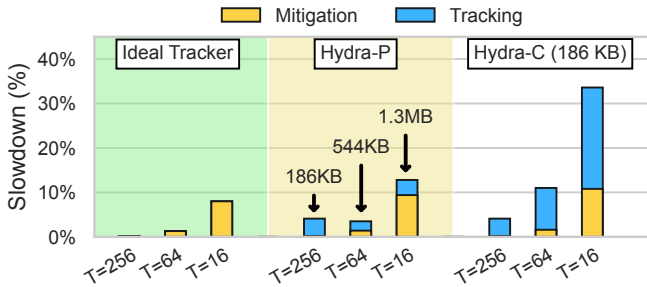

Fig. 2: Slowdown of Ideal Tracker, Hydra-C (186KB), Hydra-P (proportional storage) for thresholds of 256 to 16. Hydra incurs a slowdown due to both mitigation and tracking.

Figure 2 shows the slowdown of ideal tracker, *Hydra-P* (proportional storage), and *Hydra-C* (constant 186KB) as T_{RH} is varied from 256 to 16. Ideal tracker incurs slowdown only due to mitigation, whereas Hydra suffers from both mitigation and metadata memory accesses. The overhead of ideal tracker due to mitigation alone is relatively small, 0.2% at $T_{RH} = 256$, 1.3% at $T_{RH} = 64$ and 8% at $T_{RH} = 16$, because modern memory devices have a large number of banks and concurrency, hiding the impact of victim refresh in a bank. The overhead of Hydra-P is within 2% of the ideal tracker. However, if we do not provide the proportional SRAM storage to Hydra, then the constant storage configuration (Hydra-C) incurs significant slowdowns, from 4.2% at $T_{RH} = 256$ to 34% at $T_{RH} = 16$. Thus, Hydra at sub-100 thresholds incurs either significant SRAM overhead or significant slowdown.

F. Our Goal

We observe that at ultra-low thresholds, existing proposals either require prohibitive SRAM overheads, or performance overhead, or both. Furthermore, for all prior proposals, the SRAM structures are provisioned to target a particular Rowhammer threshold, and this decision is taken at design time. Therefore, the system becomes incapable of handling a memory module that is known to have a lower threshold.

> **Goal of Our Paper:** We aim to develop a scalable tracking mechanism with the following attributes: (1) Precise row tracking at an arbitrarily low threshold (2) Configurable to a given threshold without being restricted by the size of the tracking structures (3) Incurs negligible SRAM overheads for newly added structures, and (4) Incurs negligible slowdown compared to an ideal tracker.

III. EVALUATION METHODOLOGY

A. Simulation Framework

We use ChampSim [15], a cycle-level multi-core simulator, interfaced with DRAMSim3 [31], a detailed memory system simulator. We modified DRAMSim3 to include the DDR5 configuration, wherein each DIMM supports two sub-channels that can be operated independently and provides a 64-byte line with a burst length of 16. We use the DRAM-based power model provided by Micron [34]. Table II shows the configuration for our baseline system.

TABLE II: Baseline System Configuration

Out-of-Order Cores	8 cores at 4GHz
ROB size	352
Fetch, Dispatch, Retire width	6, 6, 5
L1-I/D and L2 (Private)	32KB and 512KB, 8-way
Last Level Cache (Shared)	16MB, 16-Way, 64B lines, SRRIP
Memory size	64GB – DDR5
Memory bus speed	2.4 GHz (4800 MT/s)
t_{RCD}-t_{CL}-t_{RP}-t_{RC}	16.6 - 16.6 - 16.6 - 48.6 ns
Channels	2 (one 32GB DIMM per channel)
Banks x Ranks x Sub-Channels	32×1×2
Rows per bank	64K
Size of row	8KB
Sub-Channel width and BL	4B and 16

We evaluate performance using 8 out-of-order cores with private L1 and L2 caches and shared L3 cache. The L3 is non-inclusive, with 128 MSHRs/core, 32 entry/core read and write queues, 4 read and write ports, 30-cycle hit-latency, no prefetcher, and SRRIP replacement policy. Our memory system contains two channels, each with a 32GB DDR5 DIMM (total of 64GB containing 8 million 8KB rows).

For evaluating prior Rowhammer mitigation schemes, all SRAM structures associated with tracking are incorporated into the memory controller. For the mitigating action, without loss of generality, we assume victim refresh of one neighboring row on each side using *Directed Refresh Management (DRFM) command*, where the memory controller supplies the aggressor row address to the memory, and the memory internally refreshes the victims rows. Unless specified otherwise, we assume a default Rowhammer threshold of 256.

B. Workload Characterization

We evaluate our design using the publicly available Champ-Sim traces, which includes 10 from SPEC2017 [1], 13 from LIGRA [41] (graph processing), and 5 from PARSEC [8]. These traces have been collected after fast-forwarding the workload to a region-of-interest. We perform a warm-up period of 50 million instructions for each workload. Eight copies of the same workload runs on 8 cores and continue executing until all 8 cores complete 200 million instructions each[1].

Table III shows workload characteristics, including the average per-core IPC, LLC-Misses Per 1000 Instructions (MPKI), workload footprint (number of unique 4KB pages touched), and Unique-Rows touched within a period of 64ms, on average. The last row of table captures the geometric mean of IPC and arithmetic mean of other values across all 28 traces.

TABLE III: Workload Characteristics: IPC, MPKI, footprint, and Unique Rows Touched (average within 64ms).

Workload	IPC (per-core)	MPKI (LLC)	Footprint (8-core)	Unique Rows Touched (64ms)
fotonik3D	0.49	19.7	16.1 GB	2,126K
mcf	1.1	14.4	4.7 GB	1,170K
gcc	0.31	17.8	1.9 GB	184K
omnetpp	0.53	10.9	1.6 GB	396K
bwaves	0.67	14.4	1.2 GB	260K
roms	0.89	6.2	511 MB	130K
cactuBSSN	1.59	7.8	473 MB	121K
wrf	0.83	11.7	277 MB	71K
pop2	1.92	3.5	219 MB	56K
xalancbmk	1.12	2.1	157 MB	40K
CF	1.08	9.6	2.7 GB	677K
BC	0.47	31	2.2 GB	438K
PR-Delta	0.41	24.6	2.1 GB	389K
BFSCC	0.71	23.4	2 GB	471K
BFS	0.67	19.5	1.9 GB	431K
Radii	0.59	26.7	1.2 GB	220K
Triangle	0.79	16.5	1 GB	258K
Components	0.59	40.8	920 MB	162K
Comp-SC	0.57	40	915 MB	162K
PageRank	0.47	54.7	878 MB	151K
BFS-BV	1.1	12.6	763 MB	194K
BellmanFord	1.09	8.4	751 MB	191K
MIS	1.47	7.1	700 MB	178K
canneal	0.33	15	1.3 GB	301K
fluida	0.88	6.7	789 MB	201K
raytrace	1.11	5.7	453 MB	116K
facesim	0.83	6.4	182 MB	46K
streamc	1.04	13.6	69 MB	18K
Average	0.76	16.8	1.7 GB	327K

C. Figure of Merit

Our primary figure of merit is the normalized performance compared to an unprotected baseline. We estimate performance by measuring the IPC averaged across all 8 cores. As all cores run the same workload, the IPC variation across cores is small.

We also consider secondary metrics such as (a) SRAM overhead for newly added structures, (b) loss in LLC capacity, (c) change in LLC misses, (d) impact on system power consumption of DRAM and the LLC, (e) sensitivity to LLC capacity, and (f) the blast radius of the mitigation.

[1]Appendices A to C provide results for multi-programmed workloads and additional multi-threaded CloudSuite [13] workload evaluations.

IV. SCALABLE ROWHAMMER TRACKING

To enable practical Rowhammer mitigation at ultra-low thresholds, we propose *Scalable Tracking for Any Rowhammer Threshold (START)*. START performs precise tracking of row activations of memory rows without requiring dedicated SRAM structure for storing the tracking entries. The key insight of START is to obviate the dedicated SRAM storage of tracking, by leveraging the last-level cache (LLC) to store the per-row counters. For our baseline system with 64 GB memory (8 million rows), even with a 1-byte counter per row, we would need 8MB of space to store the counters.

A naive design of START, which we call *START-Static* or *START-S*, simply reserves 8 ways of the 16MB 16-way LLC to provision the counters. However, doing so reduces the LLC capacity considerably, causing a significant slowdown. Therefore, we develop a dynamic scheme, which we call *START-Dynamic* or *START-D*, which adaptively allocates tracking entries only for the rows that get accessed within 64ms. We observe that only a small fraction of memory rows (about 4% on average) are touched within 64ms, so START-D consumes significantly less storage. In this section, we provide an overview of START, using START-S as a simplified example, then describe START-D, and provide results and analysis.

A. START-S: The Naive Design

Figure 3 shows an overview of START-Static (START-S) design. Even though START-S is inefficient, we use it to provide an overview of START owing to its simplicity. START-S reserves 8 ways of the 16-way LLC to store the counters for the 8 million rows. Let the ways reserved for storing the counters be ways 0 through 7. Then, on an LLC miss, these ways do not participate in the LLC replacement algorithm, so these lines cannot be removed from the cache.

With 1-byte counter per row, each cache line of 64 bytes stores the tracking entries for 64 rows. As each row has a dedicated tracking entry, these entries are untagged. To obtain a tracking entry for a given row, we hash the row address to the cache set, then use 3-bits of the row address to select one of the reserved ways, and then use 6 bits of the row address to identify the byte-in-line that stores the tracking information.

When a demand access probes the LLC and encounters an LLC miss, it gets routed to the memory controller to perform DRAM access. If this access results in row activation, the memory controller provides the row address to the cache controller, so that the controller can obtain the tracking entry and increment the counter. If the counter reaches the threshold, the counter is reset and a signal for performing mitigation for the given row is provided to the memory controller.

START-S consumes half of the LLC for Rowhammer tracking, therefore, it incurs significant performance overheads (on average 7.4% in our evaluations). We observe that while the memory system contains 8 million rows, a workload would typically not touch all these rows within the refresh period of 64ms. In fact, based on the workload characterization in Table III, we observe that on average about 300K rows get touched within 64ms (4% of the total memory rows) and only
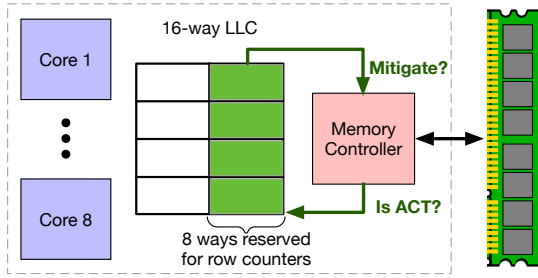
Fig. 3: Overview of START using START-S.

3 out of the 28 workloads touch more than 500K rows. If we could provide the space only to the rows that get touched at least once during the 64 ms period, then we can greatly reduce the storage consumption of tracking. Our dynamic design, START-D, achieves this goal.

B. START-D: The Optimized Design

START-Dynamic (or START-D) varies the number of ways reserved for the tracking entries based on demand. Figure 4 shows the overview of START-D. At the start of every 64ms period, START-D reserves no ways in the LLC and if the application does not access memory within this period (as the working set might already be cached), no LLC capacity is consumed. Otherwise, tracking entries get allocated on demand, and initially we use a tagged entry that identifies the row and the counter value. For our memory system with 8 million rows (23-bit row-id) and a cache with 16K sets (14-bit set index), the row-tag is 9-bits. Without loss of generality, we use a 7-bit counter, to form a 2-bye tracking entry. Thus, a 64-byte line can store up to 32 tracking entries.
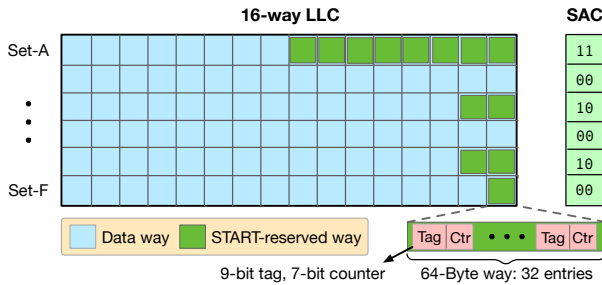

Fig. 4: Dynamic allocation of START-D. The number of ways reserved for each set varies based on demand.

When a set in LLC receives the first request to update a row-counter mapped to it, the way-allocation is increased (from 0) to 1-way, thereby enabling storage of up-to 32 tagged counters. If all sets in LLC transition to this state, START-D can hold up-to 512K tracking entries. As only 3 workloads (fotonik3D, mcf, and CF) out of 30 touch more than 500K rows within a period of 64ms, this state is sufficient for most workloads in the common case, reducing the storage consumption of tracking by 8X (from 8-ways reserved to 1-way reserved).

If the LLC encounters a request for updating the counter for a given row, and all the tracking entries of the given set are in use, the allocation for that set is increased from 1-way to

2-way. The entries already stored in the first way are rehashed such that the even entries are retained in the first way and the odd tag entries are placed in the second way. The incoming entry is then allocated into one of the two ways depending on the row address (even tag or odd tag). Finally, in rare cases, if two ways are insufficient, then the allocation is increased to 8-way. Note that 8-ways are sufficient to hold all tracking entries for the 512 rows that map to the given set, and all the tracking entries of the set are read and restored in an untagged format (each row has a designated byte for its tracking entry). Such reorganizations are also rare and not in the critical path.

1) The Newly Added SAC Table: While START-D obviates dedicated SRAM for precise tracking information, it does require the state to indicate the number of ways that are reserved in each set to store tracking information. Given that we have four possible allocations, we need two bits per set, which we call the *Set Allocation Counter (SAC)*. If SAC is 00, the set has the default reservation of 0-way (no capacity reserved). If SAC is 01, the set has 1-way reserved (with 32 tagged entries). Similarly, if SAC is 10, the set has 2-ways of 32-entries each reserved (with each containing 32 tagged entries). Finally, if SAC is 11, the set has reserved 8 ways, and it would have 8 lines each storing 64 untagged entries, for a total of 512 entries. Figure 5 shows the transitions of SAC entries from 00 state to different states. Every 64ms, the SAC entry of each set is reset to 00, so the allocation of a set remains valid only within the current refresh period. As each set requires a 2-bit SAC, and we have 16K sets, the table storing SAC entries (SAC Table) requires 4KB of SRAM.
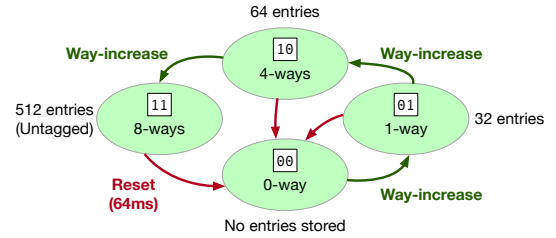

Fig. 5: SAC transitions and the resulting allocation to a set.

2) Operations: When the memory controller issues a row activation, it sends an update for that row to the LLC. The LLC uses the top 14-bits of the 23-bit row-tag to identify the cache set that stores the given row's tracking entry. The cache controller checks the SAC entry for the set to find the number of ways reserved for the set. If the SAC entry is 00, a way is allocated for tracking entries, and a tracking entry is allocated with the designated row-tag and counter value of 1. The SAC value is increased to 01. For subsequent row-updates to this set, the incoming row tag is compared with the row tag of all entries (for which the counter is nonzero). If the entry is found, the counter is updated. If the counter reaches the Rowhammer threshold, the counter is reset, and a signal is sent to the memory controller for issuing mitigation for the given row. If the entry is not found, then a new tracking entry is allocated, unless all 32 entries are allocated. In this case, SAC transition occurs, followed by tracking entry allocation
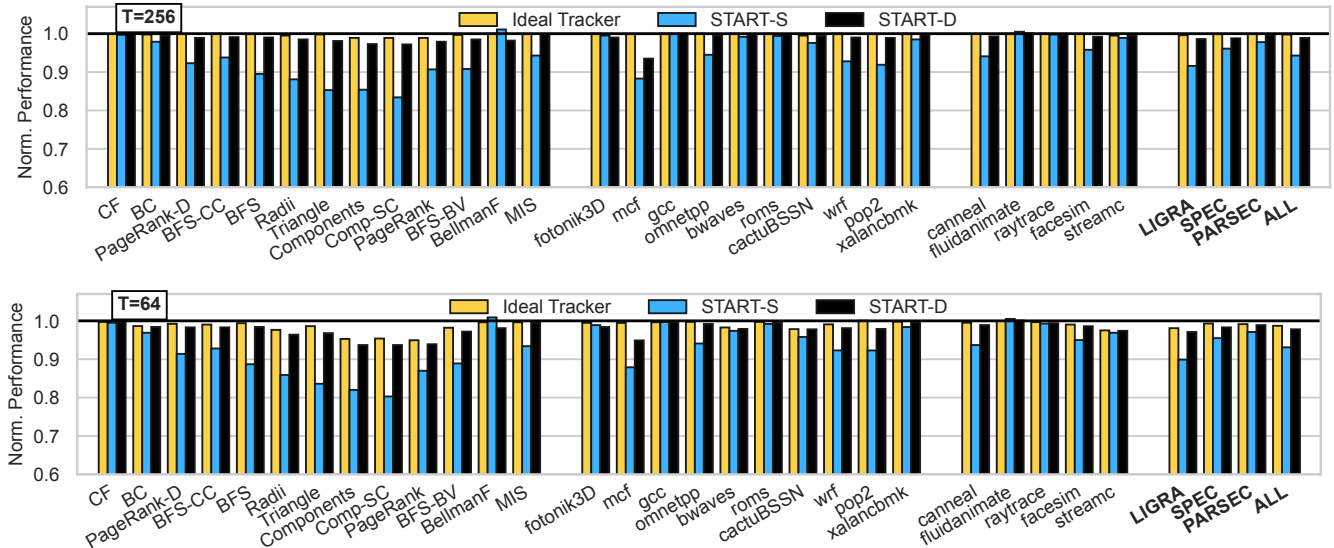
Fig. 6: Performance of ideal tracker, START-Static, and START-Dynamic normalized to unprotected baseline. START-D performs within 1% of an ideal tracker: slowdown of 1.1% vs. 0.2% at T_{RH} of 256 (top), and 2.2% vs. 1.3% at T_{RH} of 64 (bottom).

in the appropriate way. To obtain the way-index, the-row-tag is hashed (1-bit, 3-bit for 2-way, 8-way respectively) and the process remains same as SAC value of 01.

While START-D requires changes to the lookup and replacement policy of the cache, row-counter lookups are outside the critical path of demand accesses. On an LLC access or miss, the SAC of the given set is consulted, and depending on the SAC values, between 1 to 8 ways are removed from consideration of lookup or replacement. This also ensures that valid tracking entries do not get evicted from the LLC.

3) Periodic Reset and Impact on Threshold: We want to track the activation counts within 64ms, so, every 64ms, we reset the SAC table and the ways allocated in sets are released. This allows all ways to participate in cache replacement policy.

After SAC reset, the implicit row counts of all the rows are zero. As the reset of START-D may not be synchronized with refresh operations, the attacker could potentially perform $(T-1)$ activations to the row before the reset and $(T-1)$ activations after reset and still not encounter any mitigation with a Rowhammer threshold set to T. Thus, resetting causes the actual threshold tolerated by START to be $(2 \cdot T - 1)$. Therefore, to tolerate a threshold of 256, we set the effective T to be 128. The phenomenon of halving of effective threshold due to reset is common in prior trackers [36], [37].

C. Impact on Performance

Fig. 6 shows the performance of START-S, START-D and Ideal Tracker normalized to the unprotected baseline. Ideal Tracker incurs only mitigation overheads and no tracking overhead. START-S suffers from considerable slowdown due to 50% LLC capacity loss. In contrast, START-D closely follows the performance of ideal tracker for all workloads. The average slowdown of START-D is 1% compared to 0.2% for ideal tracker (T_{RH} of 256, top). At T_{RH} of 64 (bottom), START-D incurs 2.2% average slowdown compared to 1.3% with the ideal tracker (within 1%). START-D scales to arbitrarily low threshold, and performs within 1% of an ideal tracker.

D. Analysis on LLC Capacity Loss

While START-S incurs constant 50% capacity loss, the space consumed by START-D is proportional to the unique rows activated by the workload within 64ms (see Table III), as shown Fig. 7. On average, START-D only incurs 9.4% capacity loss (5x lower than START-S), with 25 out of 28 workloads consuming less than 10% of LLC capacity.
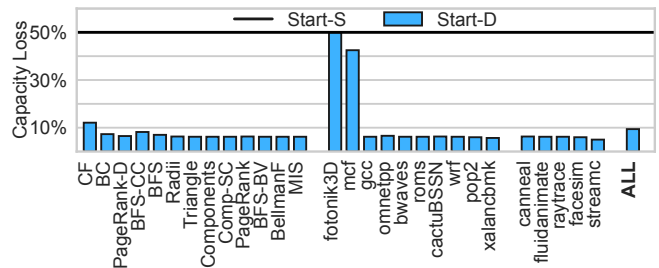

Fig. 7: START-D requires 9.4% of LLC capacity on average.

E. Impact on Cache Misses

Fig. 8 shows the increase in LLC misses due to START at T_{RH} of 256. START-S significantly increases cache misses by 21% on average. In contrast, START-D only incurs a negligible 2.3% additional misses compared to the baseline.
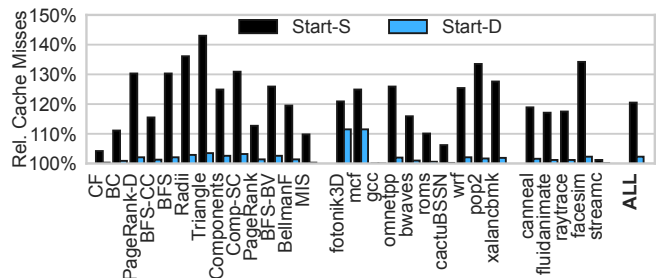

Fig. 8: START-D increases Last Level Cache misses by just 2.3% compared to the baseline, almost one-tenth of START-S.

F. Sensitivity to Cache Size

Fig. 9 shows the performance of ideal and START-Dynamic trackers at different cache sizes compared to our default configuration (16MB, 16-way). In the non-default cache configurations, START-D dynamically reserves up-to 8-ways for 12MB 12-way LLC and up-to 4 ways for 24MB 12-way LLC. START-D incurs similar performance overheads, even at reduced cache sizes, because reservation of more than 1-way within a set remains exceedingly rare.

Fig. 9: Impact on slowdown as LLC size is varied. START-D performs similar to ideal tracker at different cache sizes.

G. Impact of Blast Radius

Non-adjacent rows may also be impacted by activations to an aggressor row [2]. Recent proposals, therefore, increase the *blast radius* of the mitigation by refreshing two or four adjacent rows on either side of the aggressor. We evaluate ideal and START-D with blast-radii from 1 to 4 in Fig. 10. While the overheads of mitigation increase considerably with blast-radius, especially at T_{RH} of 64, START-D maintains a slowdown similar to the ideal tracker with 11.2% average slowdown compared to 10.2% with the ideal tracker at BR=4.
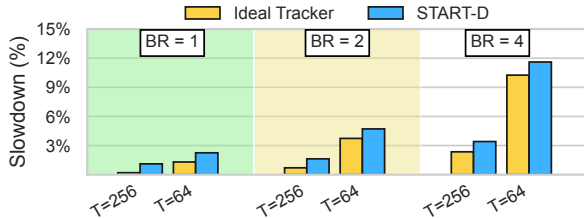

Fig. 10: Slowdown with varying Blast Radius. START-D continues to perform within 1% of ideal tracker.

H. Storage and Power Overheads

START-D requires 4KB SRAM for the SAC table (2 bits per set). The size of SAC depends only on the LLC capacity and not on the Rowhammer threshold. We also need two-bytes to store the Rowhammer threshold.

We use Micron's power calculator tool [33] to compute the DRAM power requirement. START-D increases DRAM power by 105mW at a negligible 0.3% overhead. START incurs a LLC read and write for the row-counter on every DRAM activation. We compute SRAM power overheads using CACTI 7.0 [5] with 22nm technology. START-D incurs a dynamic cache power overhead of 93mW, an 11.5% increase over baseline. However, taking the LLC leakage power into account, the overall cache power increases by only 0.9%.

I. Security Analysis

For successful Rowhammer mitigation, START must ensure that it issues a mitigation before a row receives a threshold (T_{RH}) number of activations. We define T_{RH} as the minimum number of per-row activations to **at-least** one row that are sufficient to cause a bit-flip via any attack pattern. To prove that START is secure, we make one assumption:

> A successful row hammer attack requires activating **at-least** one row more than T_{RH} times within a refresh period.

START is reset every 64ms. We call the period between consecutive reset as the *tracking window*. As DRAM refresh is uncoordinated, a given DRAM row can experience two tracking windows within a single refresh period of 64ms. So, START provides a stronger security guarantee, as follows:

> **Theorem-1:** *START issues mitigation for a row (a) at $T_{RH}/2$ activations and (b) at each $T_{RH}/2$ activations since its past mitigation, in a tracking window.*

1) Proof of Security for Tracking by START: Let T_{true} be the exact or true count of a row's activations. We prove Theorem-1 analyzing two phases. Phase-1 is from reset to issuing the first mitigation. Phase-2 is between each consecutive mitigations.

In Phase-1, the activation counter entry associated with a given row is incremented whenever the row has an activation. So in Phase-1, the value of the counter is always equal to T_{true} of any row. Therefore, if the first mitigation for an aggressor row in a tracking-window is performed at $T_{RH}/2$, the activation count of the row (T_{true}) must reach $T_{RH}/2$. This proves part (a) of Theorem-1. In Phase-2, the counter is reset to 0 upon a mitigation, and subsequently the tracking continues to be exact. The counter reaches $T_{RH}/2$ again only after performing $T_{RH}/2$ activations for the row after the mitigation. Therefore, the aggressor row is mitigated before receiving $T_{RH}/2$ activations (threshold is set to $T_{RH}/2$). This proves part (b) of Theorem-1.

2) Adaptive Attacks on START: The attacker may try to dislodge the cache lines that store the tracking entry. However, this approach is not viable as the ways reserved for the tracking entries do not participate in the replacement algorithm. As LLC accesses due for tracking are outside the critical path of demand accesses, START does not introduce new timing side channels.

The mitigative action of performing refreshes of neighboring victims rows itself causes activations on victim rows. Recent Half-Double [2] attack exploits activations arising from re-freshes of distance-1 neighbors to cause bit-flips in distance-2 neighbors. To be resilient to such attacks, START also includes any activation encountered due to victim refresh as part of the overall activation counts of the row. Note that we assume either the DRAM mapping is available to the memory controller, or DRFM is modified to provide victim row-IDs. Finally, START is simply a tracking mechanism and can be used with any mitigating action. We evaluate with victim-refresh and default blast radius of 1 and assume that the mitigating action would be configured appropriately for the DRAM module in use.

V. MEMORY-MAPPED START (START-M)

Thus far, we have considered baseline system with 8GB of memory-per-core (64GB for 8-cores), 2MB of LLC-per-core (total of 16MB), and T_{RH} of 256 and below. The tracking metadata for our system fits within a subset of the LLC (8MB) because each tracking entry is 2-bytes (9-bit row-tag and 7-bit counter) and the 512 untagged counters mapping to a set fit within 8-ways. However, modern systems might have much larger memory capacity, or might work with current and old memory, which has a higher threshold than 256. For example, an 8-core system with 512GB memory (64 GB-per-core) at threshold of 256 needs tracking metadata of 60MB, much larger than our 16MB LLC. Similarly, if such a system operates at threshold of 4K (12-bit row-tag and 11-bit counter) would need more than 5X the LLC capacity of metadata. START-D would be unable to handle such systems. For such systems, we propose *Memory-Mapped START (START-M)*, which maintains a counter table in memory and uses tracking entries in the LLC to virtually eliminate all of the memory accesses for tracking[2].

A. Overview

Consider a large-memory system with 512GB of memory and 8 cores (64GB of memory-per-core), operating at a threshold of 4K. We maintain the other parameters similar to the previous baseline (16MB LLC, 2 DDR5 channels). As our baseline contains 64 million rows and 11-bit tracking entry for each row, it would require 82MB storage, well beyond LLC's capacity.
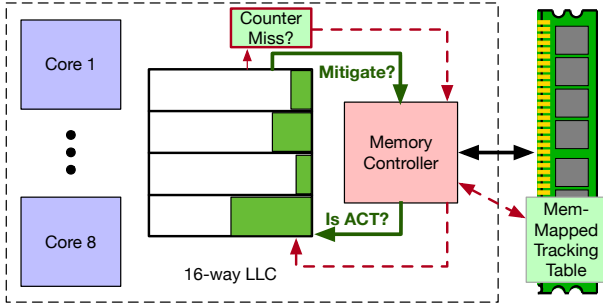

Fig. 11: Overview of START-M. The dotted red arrows denote the rare case of metadata accesses to the DRAM.

Figure 11 provides an overview of START-M. START-M reserves the required memory for untagged counters (82MB) in the addressable space of the main memory to store the *Memory-Mapped Tracking Table (MTT)*. But accessing the MTT to obtain the tracking metadata would require memory access and hence cause slowdowns. Rather than using a dedicated metadata cache (as done in CRA [23]) or a filter (as done in Hydra [37]), START-M simply uses the LLC as the expandable area to store the tracking entries. Similar to START-D, by default, START-M starts with no LLC capacity reserved (SAC of the set is set to 00). If a set requires entries, then the allocation is increased to 1-way, and then 2-ways, and finally 8-ways, on-demand, which is the maximum allocation allowed by our design.

[2]Multi-socket and disaggregated memory systems can be supported by provisioning tracking-entries in memory controller's parent socket.

B. Cache Changes: START-D to START-M

The two key changes in START-M, compared to START-D are: (1) larger tracking entries, as memory capacity increases by 8x and counter size by 16x (12-bit row-tag and 11-bit counter), so each tracking entry in the LLC is 3 bytes (the changes from 1-way to 2-way now happens when there are 21 entries mapped to the set), and (2) always using tagged organization, as 8-ways (maximum allocation) are insufficient to hold the tracking entries for all the rows mapping to a set. As shown in Figure 12, even with 8-ways we use a tagged organization.
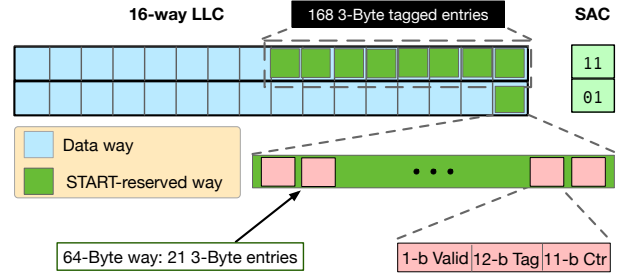

Fig. 12: Set organization of START-M. Each tracking entry needs 3 bytes, and all allocations use tagged entries.

If all sets are in state-1 (1-way reserved), START-M provides 344K tracking entries, which can increase to 2.75 million tracking entries at 8-ways. As workloads typically do not access these many unique rows within 64ms, virtually all of the memory accesses for obtaining tracking entries for START-M are cold misses (after the cache state is reset). Next, we develop an optimization that avoids cold misses for the counters.

C. Avoiding Cold Misses in START-M

Every 64ms, the allocation of START-M reverts to 0-way reserved (SAC value of 00), similar to START-D. Thus, any unique row that gets accessed after the reset will not find the tracking entry in the LLC, and will access the memory.

We leverage the observation that if there is space for the entry (e.g. allocation is less than 8-way or invalid tracking entries are present in the indexed way), then the given row is being accessed for the first time during the 64ms period. Otherwise, either the entry will be present, or the entry was evicted to accommodate another entry due to limited capacity. Therefore, we do not access the MTT on such *first-time* accesses and simply install the row in the LLC with a counter value of 1. At reset, START-M also requires resetting the MTT in memory. We do this lazily by resetting all row-counters mapping to a set only when that set encounters its first-row eviction. As each entry contains a valid-bit, the information to conduct this per-set reset is available without any extra overhead. The episode of MTT accesses are also extremely rare, and we avoid the MTT reset overheads in the common case.

With this optimization, START-M accesses the MTT only when there is no space for the tracking entry even with 8-ways, which cumulatively store approximately 2.75 million tracking entries in the LLC. As our workloads touch less than 2.2 million unique rows within 64ms, we observe negligible (less than 0.1%) memory accesses for the MTT in our evaluations.
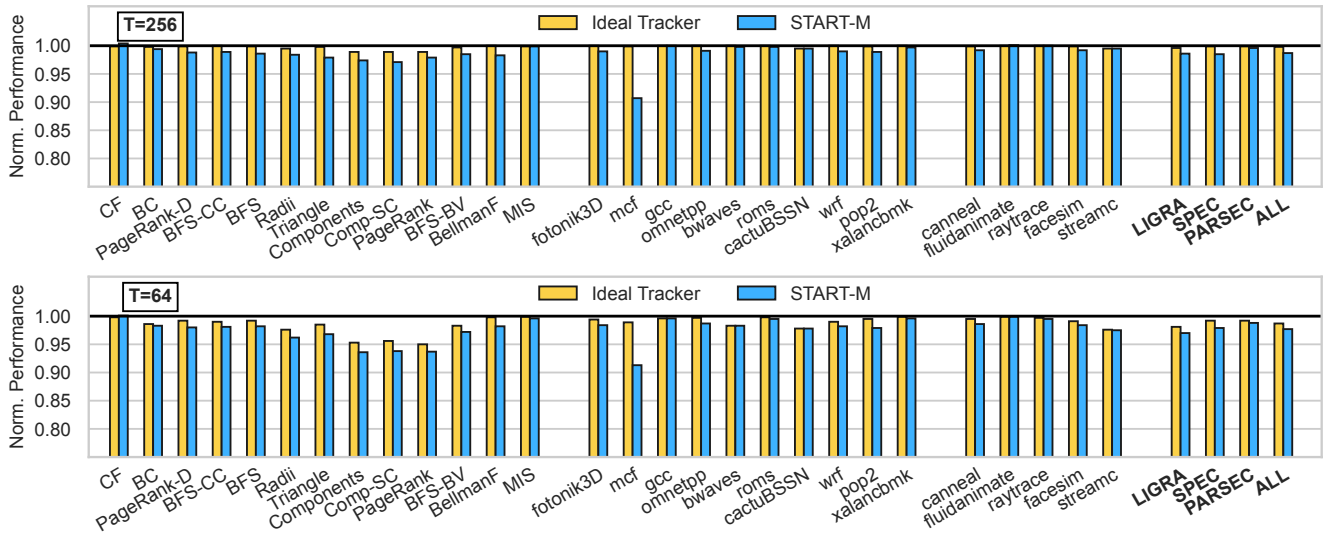
Fig. 13: Performance of ideal tracker and START-M normalized to an unprotected baseline. START-M performs within 1% of an ideal tracker: slowdown of 1.3% vs. 0.2% at T_{RH} of 256 (top), and 2.3% vs. 1.3% at T_{RH} of 64 (bottom).

D. Impact on Performance

Figure 13 shows the performance of ideal tracker and START-M at thresholds of 256 and 64. At T_{RH} of 256, START-M incurs slowdown of 1.3%, similar to 0.2% for ideal tracker. At T_{RH} of 64, START-M incurs an average slowdown of 2.3% (within 1% of the ideal tracker). START-M performs virtually identically to START-D while supporting a much larger memory capacity.

E. Analysis of Cache Capacity Loss

Fig. 14 shows the loss in LLC capacity by START-M at T_{RH} of 256. As START-M utilizes 3-Byte tagged counters and can store up-to 168 tagged counters within 8-ways, the cache capacity loss is 11.4% compared to 9.4% for START-D. All workloads, except fotonik3D and mcf (both access >1 million rows in 64ms), avoid memory accesses for tracking.
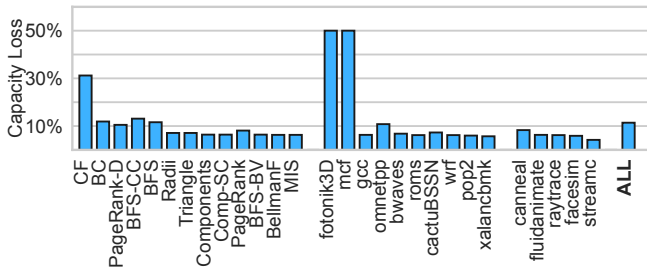

Fig. 14: START-M requires just 11.4% of the LLC capacity on average even with 1TB of memory provisioned per core.

F. Sensitivity to Rowhammer Threshold

START's seamlessly scale to lower thresholds within a system's lifetime. Fig. 15 plots the overheads of START and ideal tracker as threshold is varied from 4K to 16. START-M is used for thresholds of 1K and 4K. START incurs 1% overhead at T_{RH} of 4K (ideal incurs negligible overhead). Even at the extremely low threshold of 16, START is within 1% of ideal tracker with 9% overhead compared to 8% for ideal.
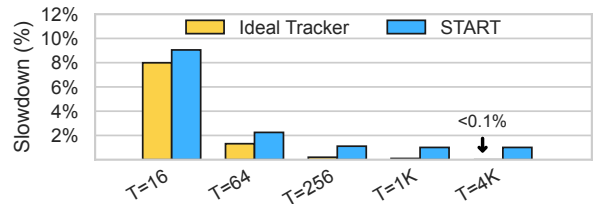

Fig. 15: START scales from current threshold of 4K to extreme threshold of 16, remaining within 1% of ideal tracker.

G. Security Considerations

START-M maintains accurate row-counts as the memory-mapped tracking table (MTT) simply provides a larger backing store for entries. Thus, Theorem-1 is applicable to START-M. To eliminate threat of bit flips in the MTT itself, activation counts for rows storing the MTT are maintained in START and mitigations are issued when it reaches $T_{RH}/2$ (just like data rows). An adversary can also access several million rows randomly to trash the tracking entries in LLC and cause 2X extra activations for each activation in the baseline, causing bandwidth overheads. As the adversary can cause performance degradation attacks even in the baseline by flooding the memory with requests, memory system isolation solutions for such problems are also applicable to START-M.

VI. DISCUSSION

A. Reduction of Rowhammer Threshold

Over the past decade, the Rowhammer threshold has been characterized over thousands of DRAM devices [25], with a clear trend of significant threshold reduction over successive process nodes, as discussed in Section II-B. It is likely that T_{RH} will continue to reduce, which has triggered recent works to develop solutions for sub-500 threshold [32], [35], [37]. Per this trend, sub-100 threshold will be reached in the next few years (or within the next decade) unless the DRAM organization changes fundamentally or DRAM vendors mitigate

Rowhammer. Unfortunately, after a decade of efforts, neither option has materialized, as stated by JEDEC [21], [22] and recent industry papers [19], [27]. Systems designed today must deploy defenses that work several years in the future on devices with unknown characteristics. To this end, START protects against arbitrary Rowhammer thresholds at low overheads, irrespective of when such thresholds arrive.

B. Pitfalls of Hybrid Tracking with Hydra

Hydra tracks at a group-level until a group-threshold is reached, followed by row-level tracking by caching recently used row-counters in a dedicated cache. Unfortunately, the SRAM the filter and counter-cache must scale proportionally with increase in aggressors. Moreover, the dedicated SRAM structures must be provisioned at design-time. Hydra's structure sizes depend on the range of thresholds (row-counter bits) and maximum memory supported (row-tag bits) [37]. For example, Hydra-544KB provisions 5-bit counters at threshold of 64, while 7-bit counters are needed support thresholds ranging from 64 to 256, requiring 700KB SRAM. Hydra is also not an exact tracker, as all row-group entries are initialized to the group-threshold when it is reached, even if many rows in the group encounter no activations, leading to spurious mitigations. Limited configurability, dedicated SRAM structures, and imprecise tracking limit Hydra's feasibility. Table IV compares Hydra with START-D at T_{RH} of 64.

TABLE IV: Comparison of Hydra with START at T_{RH} of 64.

Attribute	Hydra-544KB	START-D
Dedicated SRAM	544KB	4KB
Memory-mapped Storage	5MB	0
Performance Overhead	3.2%	1.9%
SRAM Provisioned Dynamically	✗	✓
Scales to Arbitrary T_{RH}	✗	✓
Precise Tracking	✗	✓

C. A Case for Configurability via START

Hydra incurs low performance overhead only if hundreds of KBs of SRAM is provisioned at design time, requiring additional chip area, power, and higher cost. As systems remain deployed for several years, designers must provision worst-case SRAM today for thresholds of the future. The dedicated storage can be rendered wasteful if lower thresholds are not reached within the system lifetime. Whereas, if ultra-low thresholds arrive in the absence of adequate dedicated storage, the system would experience a significant slowdown.

Our design solves the dichotomy with negligible dedicated SRAM overhead (4KB), while integrating with existing cache hierarchy, at negligible performance loss. Unlike Hydra, START can be configured for different use-cases *at deployment*, for example, precise tracking within the LLC without a memory-mapped table with START-D or large-memory systems with higher thresholds up-to 4K with START-M (Section-V).

In Appendices A to C, we extend evaluations to include multi-programmed and multi-threaded workloads for START, Hydra, and Ideal tracker, and present a new START policy that limits the LLC consumption to at-most 1 way.

VII. RELATED WORKS

A. New Mitigating Actions for Rowhammer

Our paper focuses on tracking activations; an orthogonal problem is the mitigative action. We evaluate the *victim refresh* mitigative action. Recently, alternative mitigative actions have emerged, such as row migration (Randomized Row-Swap [38], AQUA [39], Scalable Row-Swap (SRS) [16], SHADOW [43]) and rate control (Blockhammer [45]). Of these, SRS caches heavily swapped rows (containing data), while we use LLC to store metadata (row-counters). Moreover, these solutions still need a tracker and can use START as a practical and scalable tracker, as START is compatible with any mitigative action.

B. Modifying DRAM to Reduce Rowhammer

Several recent proposals modify the DRAM substrate to mitigate or reduce Rowhammer. For example, REGA [32] changes the DRAM substrate to generate extra refresh operations when a row is activated. SHADOW [43] modifies the DRAM microarchitecture with an extra row per sub-array to perform row swaps (although it does not scale to ultra-low thresholds due to limited randomization). *Panopticon* [6] proposes to redesign DRAM sub-array to store the counter alongside the DRAM row and increments this counter on each activation. Our goal is to mitigate Rowhammer without needing to redesign DRAM arrays. HiRA [44], can hide the refresh operations latency by refreshing a row concurrently with another access or refresh to the given bank. While HiRA may help with the mitigative action (such as victim refresh), it still needs a mechanism to identify aggressor rows.

C. Virtualizing Predictors and Metadata

Virtualizing a hardware structure by placing it in the cache space is a powerful paradigm [30] and has been used in prior academic and industrial proposals. Such techniques have previously been applied to virtualize the prefetcher state [10] and the Branch Target Buffer (BTB) [9]. AMD Magny-Cours processor uses part of the L3-Cache to store a probe filter [12]. However, our proposal (START-D) not only virtualizes tracking to the LLC, but also dynamically allocates the storage required, to reduce the space required by almost 5X compared to a design that stores the full tracking table within the LLC (START-S).

VIII. CONCLUSION

As Rowhammer thresholds continue to reduce with each technology generation, we seek solutions effective over a range of current and future thresholds. Tracking activation counts is critical to mitigate Rowhammer. At ultra-low sub-100 thresholds, all prior tracking techniques incur either significant SRAM overheads, or performance overheads, or both. In this paper, we propose *Scalable Tracking for Any Rowhammer Threshold (START)*, which enables practical and precise tracking of row activations. START obviates dedicated SRAM overheads by placing tracking metadata in the LLC using dynamic mechanism to reserve ways on-demand. START requires only 4KB SRAM and performs within 1% of an idealized tracker, even for thresholds of less than 100.

REFERENCES

[1] "Spec cpu2017 benchmark suite," in *Standard Performance Evaluation Corporation*. [Online]. Available: http://www.spec.org/cpu2017/

[2] "half-double": Next-row-over assisted rowhammer. https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf.

[3] M. Alian, Y. Yuan, J. Zhang, R. Wang, M. Jung, and N. S. Kim, "Data direct i/o characterization for future i/o system exploration," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 160–169.

[4] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

[5] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

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

[7] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1121–1137.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[9] I. Burcea and A. Moshovos, "Phantom-btb: A virtualized branch target buffer design," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV. New York, NY, USA: Association for Computing Machinery, 2009, p. 313–324. [Online]. Available: https://doi.org/10.1145/1508244.1508281

[10] I. Burcea, S. Somogyi, A. Moshovos, and B. Falsafi, "Predictor virtualization," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII. New York, NY, USA: Association for Computing Machinery, 2008, p. 157–167. [Online]. Available: https://doi.org/10.1145/1346281.1346301

[11] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 55–71.

[12] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *IEEE Micro*, vol. 30, no. 2, pp. 16–29, 2010.

[13] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," *Acm sigplan notices*, vol. 47, no. 4, pp. 37–48, 2012.

[14] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 747–762.

[15] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.

[16] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.

[17] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.

[18] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2016, pp. 300–321.

[19] S. Hong, D. Kim, J. Lee, R. Oh, C. Yoo, S. Hwang, and J. Lee, "Dsac: Low-cost rowhammer mitigation using in-dram stochastic and approximate counting algorithm," *arXiv preprint arxiv:2302.03591*, 2023.

[20] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, 2017, pp. 1–6.

[21] JEDEC, "Near-term dram level rowhammer mitigation (jep300-1)," 2021.

[22] JEDEC, "System level rowhammer mitigation (jep301-1)," 2021.

[23] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *IEEE CAL*, vol. 14, no. 1, pp. 9–12, 2014.

[24] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th ISCA*. IEEE, 2020, pp. 638–651.

[25] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *ISCA*. IEEE, 2020, pp. 638–651.

[26] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative row hammer protection on commodity dram leveraging managed refresh," *arXiv preprint arXiv:2108.06703*, 2021.

[27] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang, S. Lee, D. Kim, S. Ku, D. Choi, N. Joo, S. Yoon, J. Noh, B. Go, C. Kim, S. Hwang, M. Hwang, S.-M. Yi, H. Kim, S. Heo, Y. Jang, K. Jang, S. Chu, Y. Oh, K. Kim, J. Kim, S. Kim, J. Hwang, S. Park, J. Lee, I. Jeong, J. Cho, and J. Kim, "A 1.1v 16gb ddr5 dram with probabilistic-aggressor tracking, refresh-management functionality, per-row hammer tracking, a multi-step precharge, and core-bias modulation for security and reliability enhancement," in *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, 2023, pp. 1–3.

[28] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ISCA*, 2014.

[29] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 695–711.

[30] T. S. Lehman, A. D. Hilton, and B. C. Lee, "Maps: Understanding metadata access patterns in secure memory," in *2018 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2018, pp. 33–43.

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

[32] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "Rega: Scalable rowhammer mitigation with refresh-generating activations," in *44rd IEEE Symposium on Security and Privacy (SP 2023)*. IEEE, 2023.

[33] Micron Technology Inc., "System Power Calculators," https://www.micron.com/support/tools-and-utilities/power-calc.

[34] Micron Technology Inc., "DDR5 SDRAM Datasheet," 2022. [Online]. Available: https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/ddr5_sdram_core.pdf

[35] A. Olgun, Y. C. Tugrul, N. Bostanci, I. E. Yuksel, H. Luo, S. Rhyner, A. G. Yaglikci, G. F. Oliveira, and O. Mutlu, "Abacus: All-bank activation counters for scalable and low overhead rowhammer mitigation," *arXiv preprint arXiv:2310.09977*, 2023.

[36] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 1–13. [Online]. Available: https://ieeexplore.ieee.org/document/9251863/

[37] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 699–710.
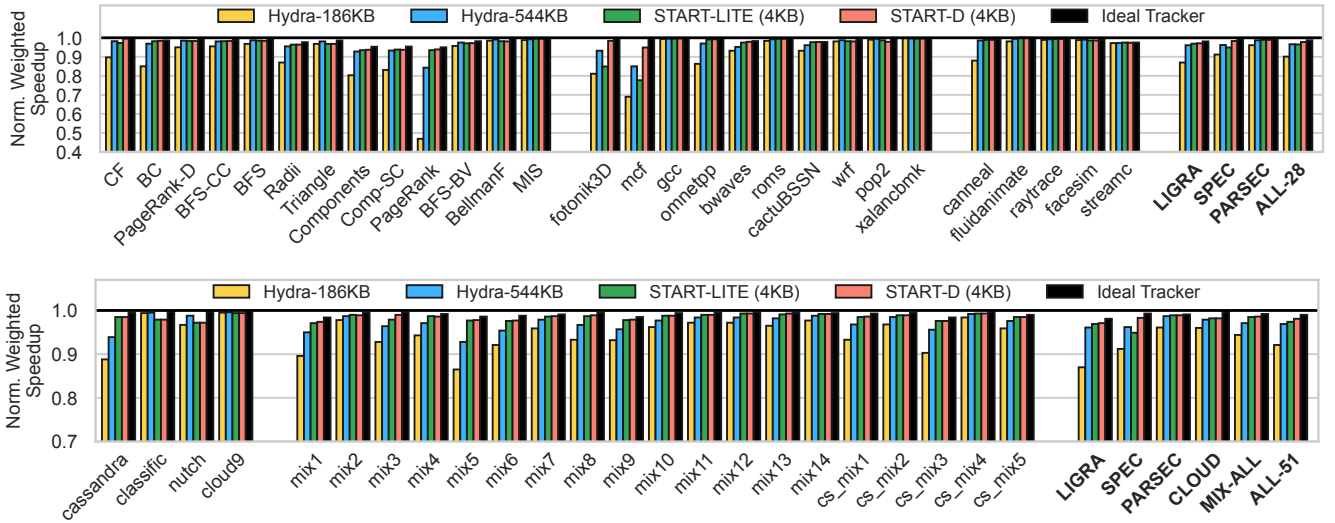
Fig. 16: Performance of START, Hydra, and Ideal trackers normalized to unprotected baseline at T_{RH} of 64 for single (top) and mix (bottom) 8-core workload configurations. On average, START-D incurs average slowdown of 1.9% compared to 1% for ideal tracker, while START-LITE incurs 2.7% slowdown, similar to Hydra-544KB's 3.2%, while requiring 136x less SRAM.

[38] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized row-swap: mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1056–1069.

[39] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 108–123.

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

[41] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 135–146. [Online]. Available: https://doi.org/10.1145/2442516.2442530

[42] V. Van Der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1675–1689.

[43] M. Wi, J. Park, S. Ko, M. J. Kim, N. S. Kim, E. Lee, and J. H. Ahn, "Shadow: Preventing row hammer in dram with intra-subarray row shuffling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 333–346.

[44] A. G. Yağlıkçı, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "Hira: Hidden row activation for reducing refresh latency of off-the-shelf dram chips," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 815–834.

[45] A. G. Yağlıkçı, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 345–358.

[46] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 28–41.

APPENDICES

APPENDIX-A: ADDITIONAL WORKLOADS

In addition to 28 SPEC, LIGRA, and PARSEC workloads (details in Section III-B), we evaluate multi-threaded, multi-programmed, and cache-sensitive workloads:

Multi-programmed Workloads: We generate 14 workload mixes by randomly selecting sets of 8 workloads from 28 SPEC, LIGRA, and PARSEC traces to run on 8 cores. We label them as *mix1* to *mix14*.

Multi-threaded CloudSuite Workloads: We evaluate 4 Cloud-Suite workloads [13] using two copies of the workload (4 unique traces per workload) running on 8 cores. Table V shows characterisitics of the workloads, which are cache-sensitive (average LLC MPKI of 3.7 compared 16.8 for workloads in Table III). We also generate 5 mixes of CloudSuite, SPEC, LIGRA, and PARSEC workloads by randomly selecting 8 workloads from the 44 traces (labeled as *cs_mix1* to *cs_mix5*).

TABLE V: CloudSuite Workload Characteristics.

Multi-Threaded Workload	Weighted Speedup	MPKI (LLC)	Footprint (8-core)	Unique Rows Touched (64ms)
cassandra	4.21	6.9	1.4 GB	365K
classification	4.4	2.8	373 MB	95K
nutch	5.42	3.1	203 MB	52K
cloud9	4.65	2.2	110 MB	28K
Average	4.65	3.7	528 MB	135K

APPENDIX-B: START-LITE: LIMITING LLC USAGE

START-D can dynamically allocate up-to 8-ways (50% of LLC capacity) for tracking entries. Each LLC access consults the Set Allocation Counter (SAC), so START's maximum allocation can be lowered by limiting the maximum SAC value. This is especially useful if START is co-running with other optimizations that require LLC resources, like way-partitioning

or Data Direct I/O [3]. As all tracking entries cannot fit in the LLC (in the worst-case), memory-mapped START can be used to store counters for each row in the memory and access them on-demand, while avoiding cold counter misses (Section V).

We evaluate such a design, termed START-LITE, where maximum SAC value is `01` (1-way reserved), requiring just 6.25% of LLC capacity in the worst case. START-LITE accesses the memory for metadata only when there is no space in the allocated way (32 tracking entries). Despite 8x lower LLC allocation than START-D in the worst case, the overhead is low because the evaluated workloads activate about 330K rows within 64ms on average (cf. Table III) and 1-way allocation in the LLC (Set Allocation Counter value of state-1) accommodates up-to 512K row-counters, making metadata memory accesses infrequent, as we show next.

APPENDIX-C: SLOWDOWN OF START-D AND START-LITE

Figure 16 shows the weighted speedup of START-D, START-LITE, Hydra and Ideal tracker normalized to an unprotected baseline at T_{RH} of 64. Hydra with 186KB of SRAM incurs a significant slowdown (8.6%) that reduces to 3.2% by provisioning 3x more dedicated SRAM (Hydra-544KB).

START-LITE requires only 4KB of dedicated SRAM and incurs only a 2.7% slowdown. START-D further reduces the slowdown to 1.9% (within 1% of ideal) and does not require a memory-mapped tracking table. START-D increases cache misses by just 2.2% while START-LITE increases them by 2.6% (including counter-misses). Across 51 single and mixed workloads, START-D's maximum slowdown is just 6.7%, compared to 18.6% for Hydra-544KB. Thus, START provides low-overhead protection for memory-intensive, cache-intensive, multi-programmed and multi-threaded workloads while avoiding significant dedicated storage structures.

APPENDIX-D: ARTIFACT

A. Abstract

This artifact presents the code and methodology to reproduce evaluation results for *START*, a Scalable Tracker for Any Rowhammer Threshold. START and other Rowhammer defenses are implemented in ChampSim, a cycle-level multi-core simulator, interfaced with DRAMSim3, a detailed memory system simulator. We provide the complete code-base for our experiments. All traces used in our paper are publicly available and accessible. The code-base includes documentation and scripts to compile ChampSim and DRAMSim3, download traces, launch experiments (for unprotected baseline, START, Hydra, and ideal tracker), parse results, and plot graphs. Most of the simulator code is in C++, scripts for launching experiments are in Bash, meta-scripts for creating job-files and collecting stats are in Perl, and plotting scripts are in Python. This artifact enables recreation of Figures 2, 6, 7, 8, 9, 10, 13, 14, 15, and 16. If compute resources are limited, representative figures are 6, 7, 8, 13, 14, and 16, while the rest are sensitivity studies.

B. Artifact check-list (meta-information)

- **Algorithm:** START, Hydra, and Ideal Rowhammer trackers.
- **Program:** ChampSim multi-core simulator interfaced with DRAMSim3 memory-system simulator and 44 publicly available execution traces from SPEC2017, LIGRA, PARSEC, and CloudSuite workloads.
- **Compilation:** Tested with cmake v3.23.1 and gcc v10.3.0.
- **Binary:** ChampSim simulator binary and DRAMSim3 simulator as a dynamically loaded library.
- **Data set:** 44 publicly accessible dynamic execution traces from 10 SPEC2017, 13 LIGRA, 5 PARSEC, and 4 CloudSuite workloads.
- **Run-time environment:** All experiments were run on RHEL Server 7.9 running Linux kernel v3.10.0 on x86_64 processors. Additionally tested on ARM-based server running CentOS 8 with Linux kernel 4.18.0.
- **Hardware:** Requires many-core server with atleast 4GB memory per core. We used a scale-out HPC cluster with hundreds of cores and TBs of memory.
- **Run-time state:** 4GB of memory per core required to store the dynamic execution state of simulations.
- **Execution:** One processor core required per workload simulation experiment. All workloads and configurations run independently of each other and can be fully parallelized. The paper includes 48 configurations with 28 workloads each (some with 51 workloads), for a total of 1528 experiments. If compute is limited, there are 609 representative experiments.
- **Metrics:** Most graphs use normalized IPC (for same-workload experiments) or weighted speedup (for Mix and CloudSuite workload experiments) as the performance metric. Analysis graphs use LLC capacity loss or cache misses as key metric.
- **Output:** Recreating Figures 2, 6, 7, 8, 9, 10, 13, 14, 15, and 16. For limited resources, representative figures are 6, 7, 8, 13, 14, and 16.
- **Experiments:** Instructions to set-up and run experiments, parse results, and plot graphs are available in the README file.
- **How much disk space required (approximately)?:** 10GB for the traces and less than 100MB for the simulators and scripts.
- **How much time is needed to prepare workflow (approximately)?:** Downloading traces might take a few hours (depends on network bandwidth). Compiling the simulators takes less than a minute per configuration, and there 48 configurations (so less than an hour).
- **How much time is needed to complete experiments (approximately)?:** Each experiment runs for about 6 hours on average, so recreating all 1528 experiments require 9,000 core-hours (approximately 1-2 days on four 64-core servers). Recreating the 334 representative experiments require about 3,600 core-hours (approximately 1-2 day on a single 64-core server). Note that some experiments can take up to 12 hours.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Apache License 2.0.
- **Data licenses (if publicly available)?:** MIT License.
- **Workflow framework used?:** We extend run-scripts, stat-collection scripts, and trace download utility of Pythia [7], which is a prefetching framework that used ChampSim as the simulator.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.10247141.

C. Description

1) How to access: The ChampSim simulator code and instructions on how to evaluate the artifact are available at publicly at https://doi.org/10.5281/zenodo.10247141. They are also present on GitHub at https://github.com/Anish-Saxena/rowhammer_champsim.

2) Hardware dependencies: The artifact requires many-core server(s) to run all configurations and workloads. There are 1528 workload simulations stemming from 48 configurations with 28 workloads (51 workloads in some cases). As all workloads can run in parallel, it would take about about 1-2 days of runtime on four 64-core servers. If compute is limited, the 609 representative simulations require about 1-2 days of runtime on a single 64-core server (the rest are sensitivity studies). Atleast 4GB of memory per core is required.

3) Software dependencies: Compilation requires gcc/ g++, cmake, and make. Launch scripts use Bash. Job creation scripts require Perl, although we supply default job-files (for slurm cluster manager) that can be easily adapted to the experimental system. Trace download is streamlined using Megatools utility, although they can also be downloaded using wget. The plotting scripts use Python (specifically, matplotlib library) and Jupyter Notebook.

4) Data sets: SPEC2017, LIGRA, PARSEC, and CloudSuite workload dynamic execution traces that are publicly accessible online.

D. Installation

Please clone the GitHub repository (or download from the Zenodo archive) and follow the step-by-step instructions available in the README file.

E. Experiment workflow

The workflow setup includes downloading the 44 execution traces, cloning simulator repositories, compiling simulator binaries, and making changes to run-scripts (either using helper-scripts or manually) as required. Once set up, experiments are launched in parallel (depending on compute resources). Finally, the simulation results are parsed and graphs are plotted to recreate relevant figures.

F. Evaluation and expected results

The artifact provides scripts to parse the simulation results to derive the normalized IPC, weighted speedup, cache miss-rate, or cache capacity loss metrics, as required. The relevant commands are provided in the README. The Python scripts, available within the Jupyter Notebook, plot the relevant graphs. This artifact enables recreation of Figures 2, 6, 7, 8, 9, 10, 13, 14, 15, and 16.

G. Experiment customization

Running all configurations discussed in the paper (including sensitivity studies) require significant compute resources (about 9,000 core-hours). The artifact provides instructions on prioritizing the representative figures, which reduce compute resources significantly (about 3,600 core-hours). Although further customization is not expected, the experiments can be sped up by reducing the simulated instructions or by running a sub-set of workloads. This requires changing the run-scripts (or job-creation scripts).

H. Notes

Please reach out to the authors in case of any questions or issues.

I. Methodology

Submission, reviewing and badging methodology:
- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html