Rowhammer Cache: A Last-level Cache for Low-Overhead Rowhammer Tracking

Aman Singh
Indian Institute of Technology Bombay
and the University of Illinois Urbana-Champaign *
Email: aman14@illinois.edu

Biswabandan Panda
Indian Institute of Technology Bombay, India
Email: biswa@cse.iitb.ac.in

***Abstract*—The rowhammer attack on modern DRAM systems is here to stay as the number of row activations required to induce a DRAM bit flip (rowhammer threshold) is following a trend of concern: 100K activations in 2014 to a few hundred activations in recent years. Hardware mitigations of rowhammer attacks need a rowhammer tracker that can track the DRAM row activations, and trigger the rowhammer mitigation. The state-of-the-art rowhammer tracker named Hybrid Tracker (HYDRA) is a lightweight hardware approach. HYDRA incurs a performance slowdown of less than 1%. However, our evaluations show that HYDRA fails to deliver its promise in terms of performance and storage overhead. For SPEC CPU2017 homogeneous 8-core workloads, HYDRA incurs a performance slowdown of 23.63%. We find that the simulation infrastructure used in HYDRA does not simulate a modern processor with a detailed cache hierarchy with hardware prefetching. The workloads used do not capture all the regions of interest (sim-points) of a benchmark. To mitigate this problem of performance slowdown without compromising security, we propose rowhammer cache, a storage-efficient and low-performance overhead HYDRA that provides the sweet spot in terms of storage overhead, and performance overhead. rowhammer cache uses existing last-level cache (LLC) space instead of additional SRAM storage for rowhammer tracking. Rowhammer cache improves the effectiveness of HYDRA by improving the performance slowdown from 23.6% to 2.25% for 55 8-core SPEC CPU2017 homogeneous mixes. For 20 8-core GAP mixes, it improves the performance slowdown from 36.47% to 5.61%. For 45 representative heterogeneous mixes, the Rowhammer cache improves performance slowdown from 28.68% to 3.14%. Rowhammer cache provides these performance improvements with negligible storage of 512 bytes.**

I. Introduction

Rowhammer interference [22] [13] [46] [27] [28] [15] [16] poses a significant reliability and security threat as frequent access to DRAM rows can cause bit flips at neighbouring rows. The severity of the rowhammer attack is characterized by the rowhammer threshold (T_{RH}) that denotes the number of row activations required for a given row to cause bit flips at neighbouring rows. The threshold has decreased rapidly with newer generations of DRAM technology. In the latest DDR5 DRAM chips, the threshold is speculated to be around 500 (ultra-low), whereas a decade back T_{RH} used to be 139K [34]. So, a rowhammer tracker should be agile enough to track the row activations and trigger the mitigation well ahead of time so that rowhammer-induced bit flips can be avoided.

* The author contributed to the work as a UG student at IITB.

Additionally, a recently discovered attack RowPress [26] has shown the increased susceptibility of DRAM devices to bit flips if rows are kept open for a long time. This attack significantly lowers the number of aggressor row activations needed to induce bit flips in nearby rows. Under extreme cases with high values of t_{AggON} (time for which aggressor row is on), bit flips are observed even with a single activation. A way to mitigate RowPress, as discussed in [26], is by using rowhammer tracking and mitigation mechanisms together with limiting the maximum row-open time. Consider a DRAM device that requires only T_{RP} activations to induce bit flips with t_{AggON} value of Xns. In such a case limiting the row-open time to Xns and making sure no row is accessed more than T_{RP} times in a refresh interval prevents bit flips. However, limiting row-open time to small values significantly increases the slowdown caused by the prevention mechanism. On the other hand, allowing larger row-open times lowers the activation threshold ($T_{RP} < T_{RH}$) for bit flips. Hence, allowing larger row-open times demands the need for tracking mechanisms that are performant even at extremely low tracking thresholds.

Rowhammer tracking and mitigation. rowhammer prevention mechanisms usually involve tracking and mitigation mechanisms [13], [33], [34], [37], [41], [44]. It works by keeping track of the number of times each row is accessed using the tracking mechanism. The tracking mechanism detects rows that cross a set Tracking Threshold (T_T) number of accesses. Such rows are known as *aggressor* rows, while the nearby rows susceptible to bit flips are known as *victim* rows. The focus of the paper is rowhammer tracking.

HYDRA. The state-of-the-art rowhammer tracker Hybrid Tracker (HYDRA [34]) uses a hybrid of SRAM and DRAM-based techniques to scale to ultra-low RH thresholds. The design of HYDRA is inspired by the observation that in most applications, only a few rows receive hundreds of accesses. HYDRA handles the common case of accessing DRAM rows with a small access count through group-based tracking using an SRAM structure. Only aggregate group-based tracking is performed in SRAM to meet total SRAM size constraints. Accurate per-row tracking is done in DRAM but is used only for rows with lots of accesses. In summary, HYDRA uses four hardware structures that are as follows:

Group Count Table (GCT). The GCT is an SRAM structure that enables aggregate tracking of row access counts. Each

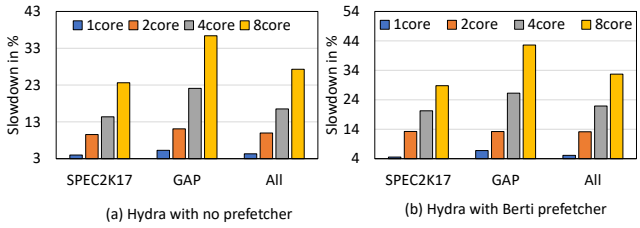

Fig. 1: Performance slowdown with HYDRA normalized to a non-secure baseline.

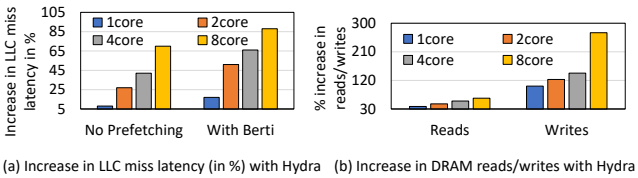

Fig. 2: Average increase in LLC miss latency and increase in reads/writes to DRAM with HYDRA.

entry of the GCT corresponds to a group of rows and stores the sum of access counts of rows in that particular group. T_G is the threshold for group-based aggregate tracking. Once the GCT entry reaches T_G, a separate count for each row of the group is maintained by two structures: the RCT and the RCC.

Row Count Table (RCT). RCT is a DRAM structure that tracks the per-row access count. A region of the DRAM is reserved for storing the RCT. The reserved region's size equals the total number of DRAM rows times the size of an RCT entry.

Row Count Cache (RCC). The RCC is an SRAM structure that caches RCT entries. Consequently, the cache is only used for rows whose corresponding group count has reached T_G. Each entry of the RCC corresponds to a single-row access count. The RCC is a set-associative cache. A few bits of the row ID are used to index into a set in the RCC, where tags are compared to find the row access count.

RCT Activations (RCT-ACT). The RCT-ACT is an SRAM structure that maintains the access counts of rows in the DRAM region reserved for RCT.

Workloads and simulation framework. We use memory-intensive simpoint traces from SPEC CPU2017 and GAP workloads [5], [7]. In contrast to the prior evaluation, we use all the simpoints and not one simpoint per benchmark. We use ChampSim [6] simulator to simulate the processor and a cache hierarchy and integrate it with DRAMsim3 [25] for a detailed DRAM simulation. This is in contrast to HYDRA's evaluation on a DRAM-only simulator (USIMM) [2] with no detailed modeling of processors, multi-level caches, or prefetching and has its limitations [1]. Note that the LLC and DRAM behaviors are different from what is reported in HYDRA as there are many memory-intensive sim-point traces and we use a processor with large ROB entries of 352 as compared to 160 [34]. Table III shows the baseline parameters and Table IV and V provide the details of benchmarks used.

The problem. Figure 1 shows performance slowdown with HYDRA for 1, 2, 4, and 8-core simulated systems averaged across 75 homogeneous mixes created out of memory-intensive

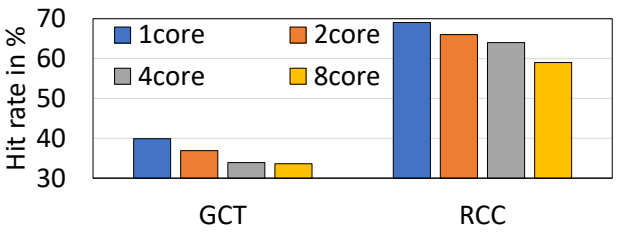

Fig. 3: Average hit rates of GCT and RCC with HYDRA.

SPEC CPU2017 and GAP traces [5] [7]. We show the effectiveness of HYDRA with and without a state-of-the-art highly accurate L1 prefetcher called Berti [29]. We use one DRAM channel for up to eight cores as commercial many-core systems like 60-core Intel Xeon Platinum [42], 64-core AMD EPYC Rome 7702P [11], and a 64-core AMD Threadripper 3990X [35] support eight DDR4-3200 channels; one DRAM channel for eight cores.

On average for an 8-core system, the average slowdown goes to 23.63% for SPEC CPU2017 and 36.47% for GAP workloads. With the hardware prefetcher ON, the slowdown goes up to 28.79% and 42.57% for SPEC CPU2017 and GAP workloads, respectively. Figure 2 (a) shows the increase in LLC miss latency because of HYDRA averaged across SPEC CPU2017 and GAP workloads. The increase in LLC miss latency happens because of additional HYDRA DRAM reads/writes generated to the DRAM delaying demand reads (Figure 2 (b)). The number of HYDRA reads and writes to the DRAM is driven by the GCT and RCC hit rates that go down with the increase in core counts (Figure 3). In summary, for many-core systems, the hit rates of GCT and RCC drop, causing more HYDRA requests to DRAM increasing the LLC load miss latency, and causing performance slowdowns. In the presence of a prefetcher, the effectiveness of a prefetcher drops as HYDRA increases prefetch lateness.

Trivial solution. One of the trivial solutions to improve performance is to increase the capacity of GCT and RCC. To achieve a performance slowdown of less than 3%, GCT has to scale up to MBs keeping RCC unchanged or RCC has to be increased to 384KB with 512KB of GCT for an 8-core system with one DRAM channel.

Our approach and observations. We borrow a few LLC ways to store group access counts and row access counts corresponding to GCT and RCC entries. We provide an RCC hit rate of more than 90% even for an 8-core system with an 8MB shared LLC. We find that to minimize the performance slowdown, the RCC hit rate plays a major role as a high hit rate at the RCC alone makes sure that HYDRA does not access DRAM. We find that two LLC ways per LLC slice are enough to achieve an average hit rate of 90%. So for an 8-core system with eight 1MB LLC slices, we borrow two ways from each LLC slice storing metadata for HYDRA. Note that stealing two ways from an LLC incurs marginal performance degradation compared to an LLC with all the ways available. We observe that a minor increase in the GCT hit rate coupled with a high RCC hit rate reduces the performance slowdown.

Our contributions. We make a case for end-to-end performance evaluations of a state-of-the-art rowhammer attack tracker. We show that a higher RCC hit rate is the key to decreasing the performance slowdown. We show that a trivial solution of increasing GCT and RCC capacities is not a storage-efficient solution (Section III). We propose rowhammer cache, a simple and intuitive solution that can help rowhammer tracking with negligible performance and storage overheads (Section IV). Rowhammer cache outperforms HYDRA by improving the performance slowdown from 23.6% to 2.25% for SPEC CPU2017 8-core homogeneous mixes. For GAP mixes, it improves the slowdown from 36.47% to 5.61%. These improvements come with marginal SRAM storage overhead (Section V).

II. Background

DRAM Organisation. DRAM is organized in the form of ranks that contain multiple DRAM devices. Each DRAM device is a collection of banks where a bank is organized as a combination of rows and columns. To fetch the data from the DRAM, an activate command is used that puts the data into a row buffer. DRAM cells are capacitor-based cells that need a precharge (refresh) after a fixed interval else they leak data. Modern DRAMs typically use a refresh interval of 64ms. For DDR4 memory systems, there can be millions of activations (1.36 million if we consider a row cycle time of 45ns) within a refresh interval.

Threat model. A rowhammer bit flip can occur at a DRAM row if the neighboring rows have been activated frequently and the number of activations within a refresh interval has crossed the rowhammer Threshold (T_{RH}). The degree to which a DRAM device is susceptible to RH attack is characterized by T_{RH}. This number has decreased rapidly with newer generations of DRAM technology. In the latest DDR-5 chips, the threshold is speculated to be around 500 whereas a decade back T_{RH} used to be 139K. It is interesting to note that even benign applications can frequently surpass such ultra-low T_{RH} thresholds. A rowhammer attacker can corrupt any data including data corresponding to page tables [46] located at any unspecified DRAM locations. A rowhammer attack can also be induced remotely [17]. So, a rowhammer tracker should be agile and should track the row activations and trigger the mitigation, well ahead of time so that rowhammer induced bit flips can be avoided.

Rowhammer tracking and mitigation. Rowhammer prevention mechanisms usually involve tracking and mitigation mechanisms. It works by keeping track of the number of times each row is accessed using the tracking mechanism. The tracking mechanism detects rows that cross a set Tracking Threshold (T_T) number of accesses. Such rows are known as aggressor rows, while the nearby rows susceptible to bit flips are known as victim rows. The number of victim rows on each side of the aggressor row is called the blast radius. The mitigation mechanism prevents bit flips in victim rows when the tracking mechanism detects an aggressor row. Various mitigation mechanisms [37], [36], [14], [9] have been proposed.

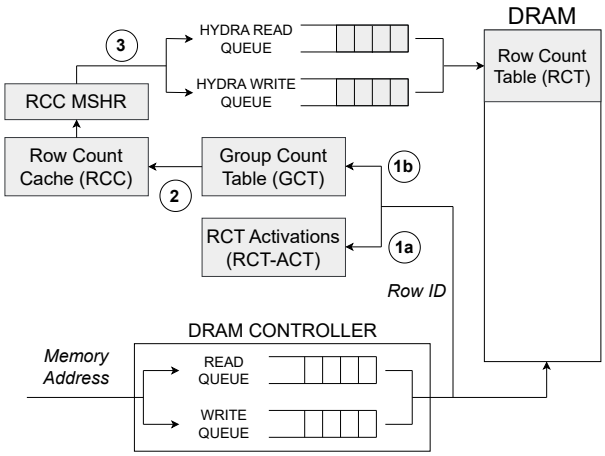

Fig. 4: HYDRA overview: (1a) Only the RCT-ACT is accessed when the row belongs to the RCT. (1b) Otherwise, the GCT is accessed. (2) Then, the RCC is accessed if the GCT threshold of the row is reached. (3) After this, the RCT is accessed if an RCC miss is encountered.

However, the focus of the paper is tracking that triggers the mitigation mechanism.

Rowhammer tracking mechanisms. SRAM and DRAM-based tracking mechanisms such as TWiCe [23], Graphene [33], D-CBF [43], CAT [20], HYDRA [34] have been proposed to track the aggressor rows. Out of all the proposals, HYDRA is the one with low storage and low performance overhead even for a low Rowhammer threshold value of 500. The state-of-the-art Rowhammer tracker HYDRA [34] uses a hybrid of SRAM and DRAM-based techniques to scale to ultra-low RH thresholds.

HYDRA. The design of HYDRA is inspired by the observation that in most practical applications, only a few rows receive hundreds of accesses. HYDRA handles the common case of accessing DRAM rows with a small total access count through group-based tracking using an SRAM structure. Group-based tracking is performed to meet total SRAM size constraints. Accurate per-row tracking is done in DRAM but is used only for rows with lots of accesses. In Section I, we explain four structures of interest (GCT, RCC, RCT, and RCT-ACT). Now, we show how they work in unison. HYDRA structures are accessed on an LLC miss that leads to issuing an ACT command. Figure 4 shows the sequence of events with the HYDRA organization. In the first step, the access count stored in the RCT-ACT is accessed and incremented if the row belongs to the RCT region of the DRAM (1a). Otherwise, the appropriate entry of the GCT is accessed and incremented (1b). Step (2) comes into the picture if the GCT threshold of the group of rows is reached. In this step, the RCC is searched for a matching row's access count data and incremented if found. Finally, HYDRA executes step (3) where it goes to the DRAM if it gets a miss in the RCC. The RCT entry at the DRAM is accessed by issuing a request to the priority HYDRA DRAM read queue. When the request is served, the access count is incremented and put into the RCC.

Tracking threshold. It is important to note that T_T is not equal to T_{RH}, but rather it depends on both the tracking

mechanism used and T_{RH}. All DRAM rows are refreshed once every refresh interval, regardless of using an RH mitigation mechanism. So, for a successful RH attack, an aggressor row has to be accessed more than T_{RH} times within a refresh interval, i.e. before the victim rows get refreshed. Tracking mechanisms can exploit this fact and reset access counts for rows in the refreshed region.

Tracking mechanisms may reset all the row access counts once every refresh interval without synchronization with the DRAM refreshes. However, with this mechanism, a DRAM row can be accessed $T_T - 1$ times before its count is reset and $T_T - 1$ times after reset but before the victim rows get refreshed as resets and refreshes are not in sync. This allows an aggressor row to be accessed $2T_T - 2$ times without the tracking mechanism's detection. Hence, the tracking threshold must be set to $T_{RH}/2$ to prevent a rowhammer attack. HYDRA, too, resets row access counts every refresh interval by setting group counts to zero and invalidating RCC entries, and hence must set $T_T = T_{RH}/2$.

III. Limitations of HYDRA

A. Capacity bottleneck

An appropriately sized GCT and RCC are critical for performance as they hide the large latency of cache misses. RCC plays an important role in the design of HYDRA. With an inadequate RCC, many DRAM activation commands (for rows in groups that have reached T_G) would have to perform a read and a write to the DRAM for updating the RCT. This increases the amount of work the DRAM has to perform for every activation command, especially because reads and writes to the RCT are prioritized over reads and writes to other memory addresses.

To evaluate the effectiveness of GCT and RCC size, we calculate the hit rate. Table I shows the GCT hit rates and the corresponding performance slowdowns with different numbers of GCT entries keeping RCC size fixed to 8k entries as suggested in the HYDRA paper. Even with a sixteen-fold increase in GCT entries, the hit rate improves from 34.87% to 46.09%. The performance slowdown improves marginally from 27.29% to 23.84%. A 512k entry GCT incurs a storage overhead of 512KB but delivers marginal performance improvement.

Next, we evaluate RCC's hit rate for various configurations by varying its associativity and number of entries. The hit rate is averaged over SPEC CPU2017 and GAP traces. Based on Figure 5, the hit rate saturates at around 95% for an RCC with 32k entries and 64-way associativity. On the contrary, fixing the associativity to 16 and increasing the RCC size barely improves the hit rate. An RCC with 64k entries and with 128 as the associativity incurs a performance slowdown of 1.53% only. However, a 64k entry RCC with 16 as the associativity incurs a performance slowdown of 24.59%. In terms of reasonable associativity and number of entries, a 128k entry with 32 as the associativity provides the best performance. However, this incurs storage of 384KB. In summary, a 512KB GCT and a 384KB RCC together deliver a performance slowdown of 1.76% and with just 32KB GCT and 384KB RCC, HYDRA incurs a performance slowdown of 2.92%.

TABLE I: GCT hit rates for different GCT sizes in HYDRA averaged across 75 SPEC CPU2017 and GAP mixes.

#GCT entries	GCT hit rate	Slowdown
32k	34.87%	27.29%
64k	37.97%	26.27%
128k	40.28%	25.63%
256k	42.67%	24.97%
512k	46.09%	23.84%

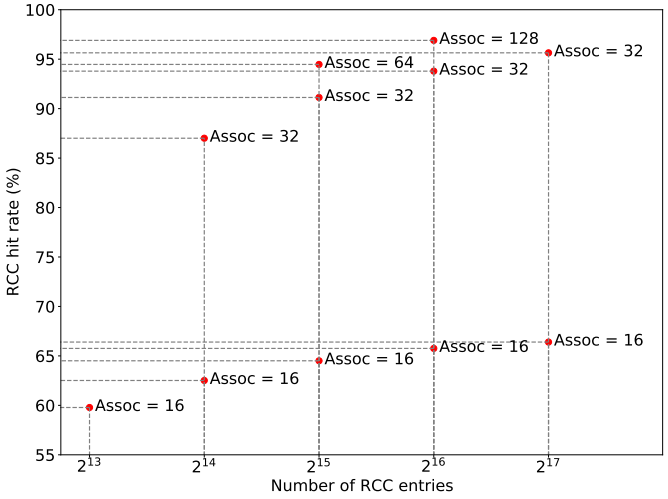

Fig. 5: Average RCC hit rate with different associativity and number of entries. HYDRA uses 8k entries (16-way).

B. Scaling with number of cores

Modern servers have large multi-core chips with a huge amount of main memory. The use of newer-generation DRAM devices with ultra-low RH thresholds in such servers makes the performance of RH prevention mechanisms essential.

Increasing the number of CPU cores connected to a DRAM causes an increase in the number of activation commands issued per unit of time. This, in turn, increases the number of rows with an activation count greater than a threshold per unit time or refresh interval. We validate this by evaluating the effect of increasing the number of cores on the number of rows reaching a threshold number of activations per refresh interval, averaged over memory-intensive SPEC CPU2017 and GAP traces. Table II shows an increase in the number of activations with an increase in core count. More ACTs lead to more RCT accesses. Averaging over memory-intensive SPEC CPU2017 and GAP traces, the number of times RCT is accessed per kilo instructions executed by each core increases from 6.5 in one core to 15.5 in two cores, 35.9 in four cores and 83.1 in eight cores. As each RCT access requires accessing the DRAM, this explains why the performance plummets with increasing CPU cores in a multi-core system. Figures 1, 2, and 3 show the effect of core count on performance, LLC miss latency, and hit rates of GCT and RCC.

C. Effect of hardware prefetchers

Prefetching is a common technique modern processors use to hide the latency of cache misses. It is critical to the

TABLE II: Average number of DRAM rows that cross different threshold numbers of activation (ACT) commands within a single refresh interval.

No. of cores + L1D prefetcher	Average no. of rows with ACTs greater than 0	100	250	500	1000
1 + No-pref	20K	3.5K	1.6K	0.2K	0.1K
1 + Berti	26K	4.4K	2.1K	0.4K	0.2K
2 + No-pref	38K	6.3K	2.6K	0.5K	0.3K
2 + Berti	47K	8.1K	3.6K	0.8K	0.4K
4 + No-pref	67K	11K	4.0K	1.1K	0.6K
4 + Berti	83K	14K	4.7K	1.5K	0.8K
8 + No-pref	118K	18K	5.8K	2.0K	1.2K
8 + Berti	129K	19K	6.4K	2.4K	1.3K

TABLE III: Baseline system parameters.

Core	Out-of-order, Hashed-perceptron, 4 GHz, 4-retire width, 352-entry ROB
L1I	32 KB, 8-way, 4 cycles, LRU
L1D	48 KB, 12-way, 5 cycles, LRU
L2	512 KB, 8-way, 10 cycles, LRU
LLC	1MB/core, 16-way, 64B lines, 20 cycles, SRRIP [18], non-inclusive
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
DRAM size - T_{CK}	32 GB (DDR4) - 0.68 ns
T_{RCD} - T_{RP} - T_{CAS}	13.6 ns - 13.6 ns - 13.6 ns
T_{RC} - T_{RAS} - T_{WR}	45.56 ns - 31.96 ns - 14.6 ns
T_{RFC} - T_{REFI}	349.52 ns - 7.78 μs
Rows - Columns in a bank	131072 - 1024, size of the row: 8 KB
Banks - Ranks - Channels	16 - 2 - 1

performance of several applications. Prefetchers learn memory access patterns and fetch data before time so that future memory accesses do not stall the processor. Prefetching techniques can be deployed at any level of the cache hierarchy. However, prefetchers only speculate about future cache accesses and may pollute the cache by fetching useless data. The metric that tracks this is called prefetcher accuracy. As there is no hardware prefetcher with 100% accuracy; with hardware prefetcher ON, additional DRAM accesses increase row activation counts, leading to more RCT activity per kilo instructions. Berti [29] is a recently proposed state-of-the-art L1D prefetcher with high prefetch accuracy (almost 90%). We evaluate the performance improvement due to the Berti prefetcher at L1D in different system configurations, averaging over memory-intensive SPEC CPU2017 and GAP traces. The results depicted in Figure 1 show a non-marginal increase in performance slowdown for a system with hardware prefetcher. One of the primary reasons for this performance slowdown is prefetch timeliness. On average, HYDRA increases the prefetch lateness by 4%, 5%, 9%, and 13%, for 1, 2, 4, and 8-core systems respectively. *So, in summary, not only does HYDRA cause significant performance overheads in the baseline system, but it also reduces the effectiveness of prefetchers. Combining these effects, we observe that using HYDRA for multicore systems with prefetchers has significant performance degradation.*

IV. Rowhammer Cache

For a baseline system with a 1 MB LLC slice and 16-way associativity (1024 sets), two ways from each set of each LLC slice are reserved for usage by rowhammer cache. This much space is enough as per the sensitivity study shown in Figure 5. With this setup, rowhammer cache reserves 128 KB of LLC slice for GCT, RCC, and RCT-ACT which is 12.5% of the total LLC size. Although reducing the LLC size available for normal applications comes with a performance penalty, it is far outweighed by the reduction in slowdown by using a scalable tracker like rowhammer cache. Figure 6 depicts an overview of the structures and working of rowhammer cache. Based on our experiments, if we steal two ways (way-0 and way-1) from LLC for GCT and RCC, compared to a non-secure system with no rowhammer tracker, we see a performance drop of 1.28%, 1.22%, and 0.82% for 8-core, 4-core, and 1-core systems, respectively. However, compared to HYDRA, it reduces the performance slowdown by more than 20% for an 8-core system.

GCT: The GCT structure in the rowhammer cache occupies cache set numbers 512 to 1023 of each LLC slice. Each cache line is 64 bytes long and holds 64 GCT entries. For an 8-core baseline system, the GCT consumes 512KB for eight LLC slices (64KB per LLC slice), and 8 DRAM rows map to a single GCT entry. Note that one GCT entry is of 1 byte that stores the group count. Just like its counterpart in HYDRA, GCT entries store the sum of activation counts of a group of rows and initiate per-row tracking when a GCT entry reaches the group-based tracking threshold (T_G). We use $T_G = 200$ for the case where $T_T = 250$.

RCC: RCC is a structure in rowhammer cache that occupies cache set numbers 0 to 511 of each LLC slice. Each cache line is 64 bytes long and holds 16 RCC entries. The RCC entries in 2 reserved cache lines (ways) of each LLC set form an RCC set. This makes the RCC a 32-way associative cache that is 512 KB in size for an 8-core baseline system with eight 1MB slices. As shown in Figure 7, 4-byte long RCC entries contain 8 bits for row access count data, 10 for the tag, 1 for the valid flag, 2 for the replacement state, and the remaining 11 bits are unused. RCC follows the SRRIP [18] replacement policy with 4 different replacement states. Figure 8 shows the RCC and GCT sets for a rowhammer cache with a 1MB LLC slice.

RCT: RCT stores access counts for rows whose per-row tracking has been initiated in the DRAM. As in HYDRA, it is accessed by issuing DRAM commands. With 2^{22} total rows in the baseline DRAM and one byte for each entry, 4 MB of space is reserved for the RCT. This sets aside less than 0.02% of 32 GB memory space and therefore causes minimal impact on performance.

RCT-ACT: RCT-ACT's function and working are the same as HYDRA. We use additional storage of 512 bytes for 512 RCT-ACT entries.

Indexing. Accounting for 2^{22} rows in the baseline DRAM, Row IDs are 22 bits wide. The upper 19 bits of the Row ID identify the group a row is part of. This group number is used to index into the LLC to get a GCT entry. Starting from the most significant bits, the indexing mechanism uses three bits to identify the LLC slice, nine bits to identify the set number (from 512 to 1023) and one bit for choosing the

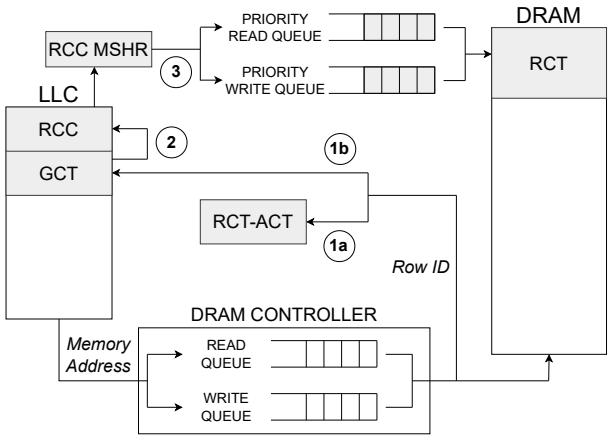

Fig. 6: Overview of rowhammer cache: On an LLC miss, the DRAM controller gets the DRAM row ID and then accesses rowhammer cache. (1a) Only the RCT-ACT is accessed when the row belongs to the RCT (1b) Otherwise, the appropriate entry of the GCT residing in the LLC is accessed (2) Then, the RCC is searched for a matching row access count data if the GCT threshold of the row is reached (3) If an RCC miss is encountered, the RCT is accessed by adding a request to the Priority Read Queue.

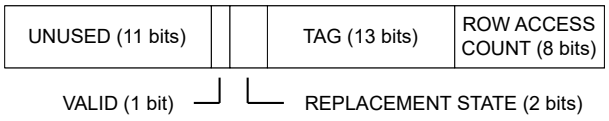

Fig. 7: One RCC entry.

reserved way/cache line (as two reserved ways in each set to store GCT entries). Once a particular cache line is found, we use the lowermost six bits of Row ID as the byte offset into the cache line to access the GCT entry (64-byte cache line store 64 GCT counters). Figure 9 shows this indexing.

The RCC is searched for cached access counts for rows which have crossed the group threshold. Similar to the GCT indexing mechanism, starting from the most significant bits, the 3 most significant bits of the 22-bit-wide Row ID are used to identify the LLC slice. The next nine bits are used to identify the set number from 0 to 511. This leads us to the 2 reserved ways per set that store a total of 32 RCC entries (an RCC entry is 4 bytes). The remaining 10 bits of the Row ID are compared among the tags of 32 RCC entries to find a matching entry. This design provides the benefit of a 32-way cache. As shown in Figure 5, a 32-way RCC provides higher hit rate than a 16-way RCC. As the 32 comparisons needed for 32 RCC entries can affect the probe time and also the demand loads waiting to access the LLC, we use a small buffer of 128 bytes to store the RCC entries. We copy the 128 bytes from the cache lines into this buffer and then perform 32 concurrent comparisons.

Additional design functionality. With rowhammer cache, we need an additional port per LLC slice for the HYDRA accesses to GCT and RCC sets so that it won't affect the demand

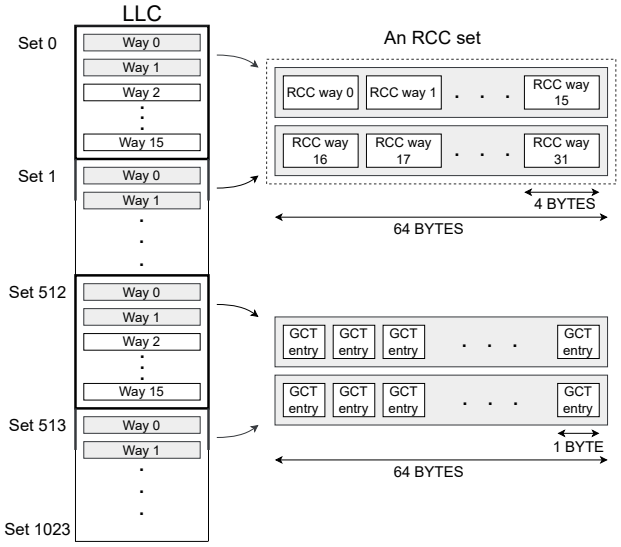

Fig. 8: An LLC slice in rowhammer cache with 2 ways reserved in each set for storing GCT and RCC.

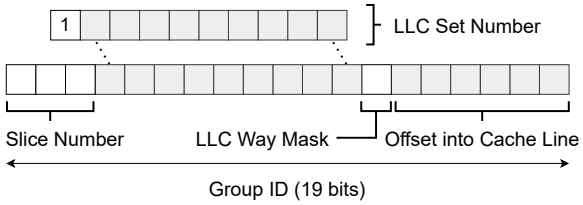

Fig. 9: Indexing mechanism: From Row ID to LLC GCT entry.

requests coming from the processor. For rowhammer cache-related requests, the DRAM controller sends a message back to the LLC with a `HYDRA-flag` set. The LLC controller reserves two ways (way-0 and way-1) exclusively for rowhammer cache-related requests. Note that a possible design is to reserve one way for GCT and one for RCC across all 1024 sets. However, as we need the behavior of 32-way associativity for RCC, we design two ways for RCC with 512 sets. HYDRA in the baseline design reserves a DRAM area for RCT which is not used by the OS. The DRAM controller is aware of that region and based on the address and Row ID.

Security guarantee. By design, the rowhammer cache ensures that no row gets activated more than T_T (T_T is set to $T_{RH}/2$) times within a refresh interval without its nearby rows being refreshed.

V. Evaluation

We use ChampSim [6], a trace-based CPU microarchitecture simulator coupled with DRAMsim3 [25], a cycle-accurate DRAM simulator to carry out all the experimental evaluations. ChampSim is used for the 2nd and 3rd Data Prefetching Championships (DPC-2 [3] and DPC-3 [4]). Recent caching and prefetching proposals [8], [10], [29], [30], [38] are also coded and evaluated on ChampSim. The recently modified ChampSim extends the one provided with the DPC-3 with a decoupled front-end and a detailed memory hierarchy support for address translation that further improves the baseline performance.

TABLE IV: SPEC CPU2017 workloads with metrics of interest. Total number of activations: rows × average-ACTs/row.

Workload	LOAD MPKI	LOAD APKI	Unique rows	250+ ACT rows	ACTs per row
gcc_s-1850B	17.8	17.8	73.2K	7	0.18
gcc_s-2226B	69.8	69.8	96.7K	11	1.73
gcc_s-2375B	0.8	2	42.3K	264	0.3
gcc_s-734B	8.6	8.7	37.9K	1951	0.99
bwaves_s-1740B	18.5	18.5	186.3K	0	1.29
bwaves_s-2609B	18.5	18.5	185.3K	0	1.29
bwaves_s-2931B	7.7	8.7	182.6K	5	3.08
bwaves_s-891B	25.4	25.4	146.7K	69	2.19
mcf_s-1152B	36.5	56.8	91.1K	5023	1.79
mcf_s-1536B	46.5	57.2	283.6K	1283	3.38
mcf_s-1554B	154.7	154.7	15.5K	6004	4.18
mcf_s-1644B	31.4	36.9	972.1K	6145	4.2
mcf_s-472B	52.3	52.4	81K	8704	3.3
mcf_s-484B	23.3	23.3	85.6K	9318	2.63
mcf_s-665B	12.4	19.3	38.2K	4986	1.16
mcf_s-782B	93.5	132.4	116.6K	5916	2.52
mcf_s-994B	20.3	24	716.8K	33	2.05
cactuBSSN_s-2421B	8	10.1	87.6K	1	2.59
cactuBSSN_s-3477B	2.1	4.2	44.6K	16	0.99
cactuBSSN_s-4004B	2.1	4.2	44.3K	24	0.99
lbm_s-2676B	6.3	6.3	89.5K	19101	4.25
lbm_s-2677B	11.8	11.8	88.5K	21927	4.06
lbm_s-3766B	11	11	88.3K	21789	4.03
lbm_s-4268B	11.4	11.4	88.5K	21810	3.9
omnetpp_s-141B	10.4	12.3	246.3K	1302	1.67
omnetpp_s-874B	9.3	11.1	227.4K	1166	1.62
wrf_s-6673B	12.2	12.7	43.8K	16318	3.18
wrf_s-8065B	9.2	9.6	40.3K	15650	3.15
wrf_s-8100B	1.5	1.7	24.5K	2812	1.21
xalancbmk_s-10B	23.4	24.4	53.6K	6039	2.03
xalancbmk_s-165B	6.4	12.2	4.6K	1817	1.13
xalancbmk_s-202B	17.3	20.6	6.9K	5026	1.99
xalancbmk_s-325B	0.3	0.6	12.6K	2147	0.59
xalancbmk_s-592B	0.3	2.1	26.7K	1726	1.11
xalancbmk_s-700B	0.4	2.2	21.5K	1598	0.66
x264_s-18B	0.6	0.7	17.2K	0	0.96
x264_s-39B	0.9	1	25.1K	2976	1.37
cam4_s-490B	3.8	6.1	23.4K	6877	3.49
cam4_s-573B	0.5	0.7	37.1K	9573	2.26
pop2_s-17B	2.5	4.1	50.8K	4385	2.18
leela_s-1083B	1.2	1.2	5.9K	2759	0.55
nab_s-12521B	1.6	1.6	44.6K	0	1.21
fotonik3d_s-10881B	21.1	21.3	939.7K	208	3.5
fotonik3d_s-1176B	9.1	9.9	65.4K	718	2.21
fotonik3d_s-7084B	16.4	16.4	117.1K	99	2.54
fotonik3d_s-8225B	9.1	10	65.9K	682	2.23
roms_s-1007B	5.7	6.3	79.2K	3598	2.1
roms_s-1070B	14.5	14.5	115.8K	16056	3.54
roms_s-1390B	19.1	19.1	171.3K	6973	3.82
roms_s-1613B	1.9	1.9	34.5K	4660	1.15
roms_s-293B	10.6	11.5	256.4K	85	2.51
roms_s-294B	11.7	12.7	259.6K	93	1.79
roms_s-523B	16.5	16.5	140.9K	8581	3.69
xz_s-2302B	1	1.5	40.7K	153	1.07
xz_s-3167B	1.1	1.5	43.6K	1491	1.23

We verify all the timing constraints in the integrated ChampSim+DRAMsim3 simulator as per the bandwidth and latency stack [12]. We use an RH threshold of 500 for evaluation. So, T_T is set to 250, and T_G is set to 200 for HYDRA. However, we show sensitivity studies for lower thresholds too. The blast radius is set to 2. Table III shows the simulation parameters used for evaluation. We use the sum of IPCs as the throughput (performance) metric.

TABLE V: GAP workloads with the metrics of interest.

Workload	LOAD MPKI	LOAD APKI	Unique rows	250+ ACT rows	ACTs per row
bc-0	55.8	66.2	91.1K	8933	3.08
bc-12	57.3	67.9	86.4K	8499	3.19
bc-3	53.4	63.6	80.3K	9526	3.18
bc-5	51.4	62.4	110.6K	8271	3.75
bfs-10	18.3	19.5	113.5K	10712	1.77
bfs-14	16.7	17.2	123.1K	9032	1.59
bfs-3	19.1	20.7	114.1K	10490	1.83
bfs-8	16.8	17.4	118.9K	9177	1.6
cc-13	41.4	55.2	17.3K	4928	2.39
cc-14	40	53.4	18K	4912	2.46
cc-5	38.6	51.5	18.8K	5092	2.55
cc-6	45.5	60.7	17.5K	4955	2.48
pr-10	92.7	122	28.3K	13885	1.59
pr-14	92.6	122	29.1K	13858	1.59
pr-3	92.7	121.9	28.1K	14112	1.59
pr-5	92.7	121.9	29.5K	14051	1.59
sssp-10	31	39.1	167.9K	5165	3.14
sssp-14	31	39.1	172.1K	5196	3.14
sssp-3	31.1	39.1	166.9K	5219	3.14
sssp-5	31	39.1	166.8K	5364	3.14

Workloads. We run the evaluation on traces from the SPEC CPU2017 [5] and GAP [7] benchmarks. Tables IV and V show all the simpoints used with their respective LLC accesses per kilo instructions (APKI), misses per kilo instructions (MPKI), and metrics of interest for a rowhammer tracker. We collect statistics for 500M instructions in the region of interest for each sim point after a warmup of 50M instructions per core. We limit our study to memory-intensive SPEC and GAP traces (75 traces in total), with an 1MB LLC/core in our modeled baseline system. We provide a detailed analysis of 75 8-core homogeneous mixes with one DRAM channel where all the cores of a many-core system run the same benchmark trace in the SPEC RATE mode. We also show results for 45 8-core heterogeneous mixes. For each homogeneous and heterogeneous mix, when a core finishes its 500M instructions, it gets replayed until all the cores finish their respective 500M instructions. We report performance in terms of weighted speedup [39]. Weighted speedup is equivalent to system throughput which accounts for the number of programs completed per unit of time. The choice of using a large number of instructions was taken so that each trace encounters at least an entire refresh interval during the simulation. The performance degradation associated with enabling the RH tracking mechanism is with respect to the same but vulnerable system without an RH tracking mechanism.

A. Performance and hit rates at RCC and GCT

SPEC CPU2017 8-core homogeneous mixes. Figure 10 shows the performance slowdown for SPEC CPU2017 with HYDRA and rowhammer cache normalized to a non-secure system with Berti prefetcher ON. For SPEC CPU2017, rowhammer cache improves the performance slowdown from 28.79% to 3.05%. Note that Berti is a highly accurate prefetcher with low prefetch accuracy of less than 50% for a few benchmarks like `mcf`. To understand the performance benefits of rowhammer cache, Figure 11 shows the RCC hit rates with HYDRA and rowhammer cache for SPEC CPU2017 8-core homogeneous

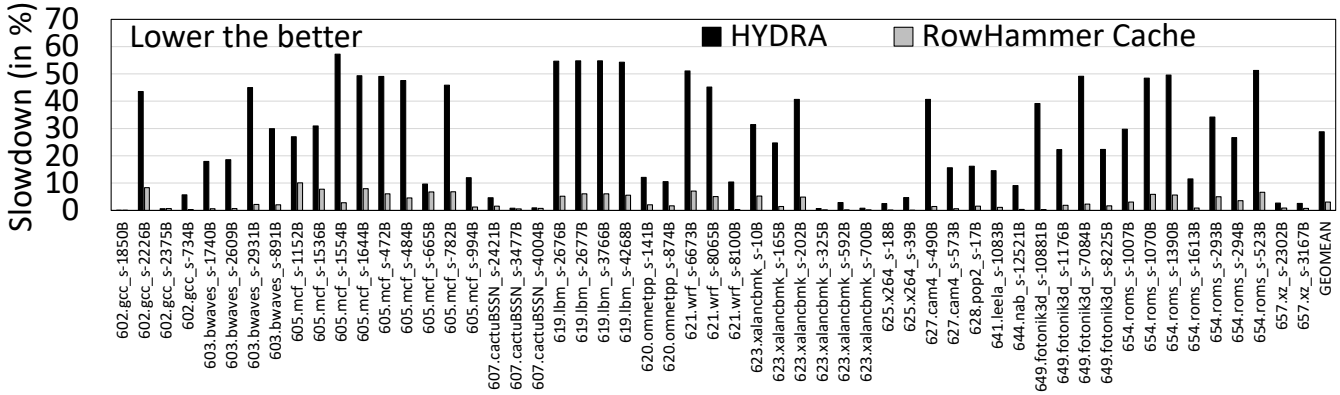

Fig. 10: Performance slowdown: SPEC CPU2017 homogeneous mixes.

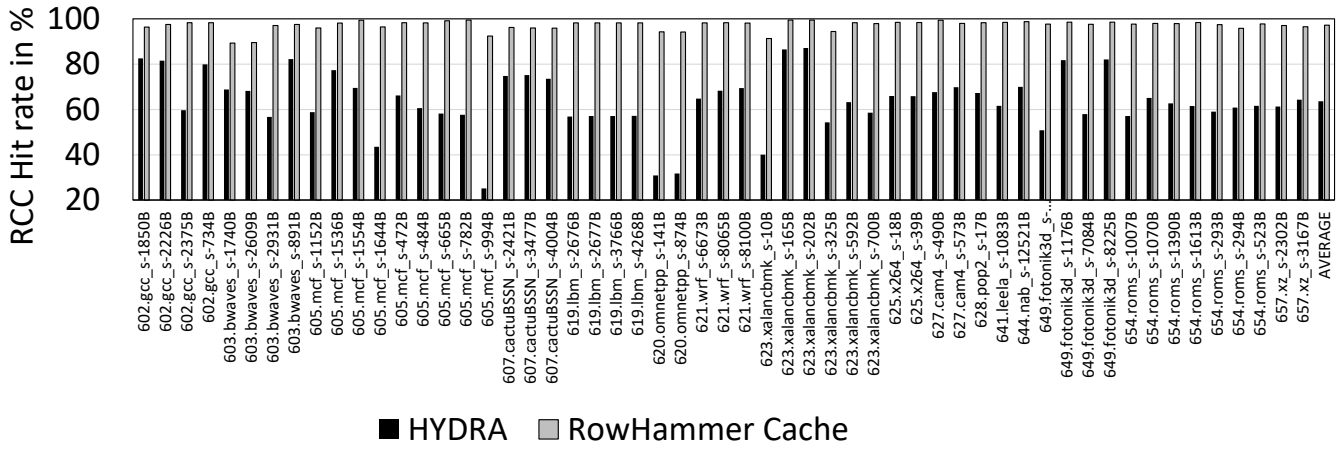

Fig. 11: RCC hit rate for SEPC CPU2017 with rowhammer cache.

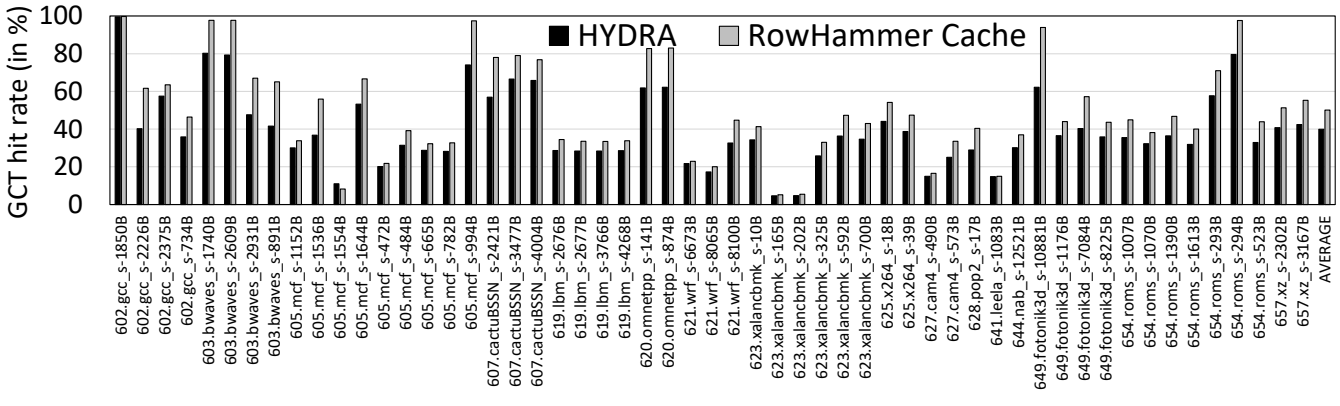

Fig. 12: GCT-only hit rate for SEPC CPU2017 with rowhammer cache.

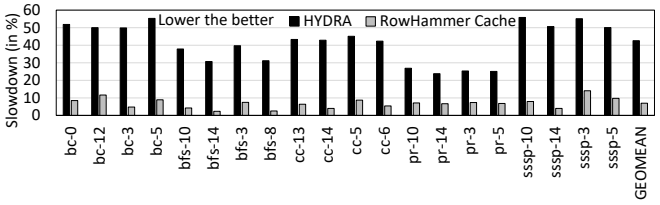

Fig. 13: Performance slowdown: GAP homogeneous mixes

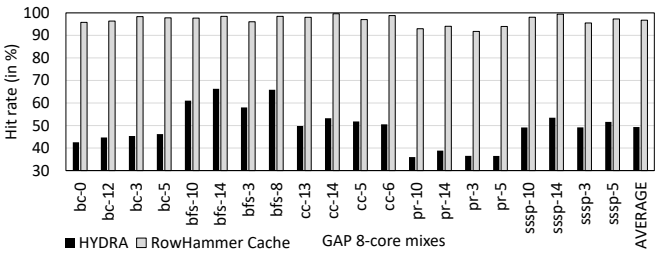

Fig. 14: RCC hit rate for GAP with rowhammer cache.

mixes. Rowhammer cache improves the RCC hit rate from an average of 63.58% to 97.16%. Note that rowhammer cache does not improve the hit rate of GCT, significantly (34.87% becomes 42.01% as shown in Figure 12). However, a minor increase in the hit rate of GCT coupled with a high RCC hit rate contributes significantly to performance. There are applications with no slowdown (`cactuBSSN`) as the count of 250+ ACT rows is low (Table IV).

GAP 8-core homogeneous mixes. For GAP mixes, the rowhammer cache improves the performance slowdown from 42.57% to 6.97% (Figure 13). To understand the performance benefits, next we show the RCC and GCT hit rates. For GAP, the rowhammer cache improves the RCC hit rate from an average of 49.33% to 96.75% (Figure 14). There is a drop in hit rate by an average of 1.23% with prefetching. Figure 15 shows the marginal increase in the GCT hit rate. One of the primary reasons for performance slowdown with rowhammer cache and prefetcher ON is the reliance on the agility of the tracker for security. If the access count maintained by rowhammer cache goes out of sync with the actual access count due to delays in servicing reads to the RCT, some rows may breach the rowhammer threshold and induce bitflips. Hence, DRAM requests from rowhammer cache are always prioritized over demand loads and prefetch requests. With prefetching ON, some of the accurate prefetch requests get delayed causing additional slowdown. Also, prefetching introduces bursty traffic that exacerbates the problem with HYDRA. In summary, there is a performance cost that the system has to pay in order to get the rowhammer tracking right. Overall, rowhammer cache improves the effectiveness of HYDRA significantly with minor changes to the LLC organization.

Heterogeneous mixes. Figure 16 shows performance slowdowns with HYDRA and rowhammer cache for 45 heterogeneous mixes. We create mixes based on their LLC MPKI values (a few mixes with low MPKI, a few with high MPKI, and a few random mixes). These mixes have performance slowdowns of

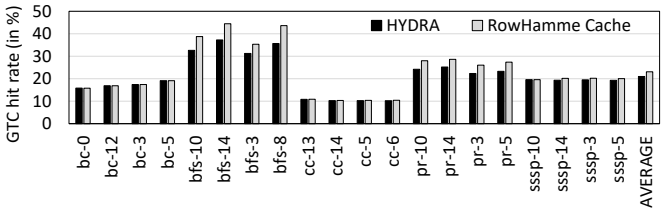

Fig. 15: GCT-only hit rate for GAP with rowhammer cache.

3.52% to 49.87% with HYDRA. On average, the rowhammer cache improves the performance slowdown from 28.68% to 3.14%. The insights that we see for homogeneous mixes are also applicable to heterogeneous mixes.

Storage and performance tradeoff. HYDRA incurs an SRAM storage of 56.5KB that leads to a performance slowdown of 23.63% and 36.47% for 8-core homogeneous SPEC and GAP mixes, respectively. HYDRA with 32KB of GCT and 384KB of RCC results in a performance slowdown of less than 3% and 6% for SPEC CPU2017 and GAP mixes, respectively. Rowhammer cache uses 512 bytes of SRAM storage and leads to performance slowdowns of only 2.25% and 5.61% for SPEC and GAP mixes, respectively. The baseline HYDRA outperforms the rowhammer cache if we use 512KB of GCT and 384KB of RCC.

B. Sensitivity studies

Per-core LLC size. Figure 17 shows the effect of rowhammer cache on an 8-core system with different sizes of LLC-slice per core (1MB/core to 4MB/core). Intuitively, the performance slowdown improves with larger LLC per core as fewer misses go to DRAM. Nevertheless, rowhammer cache shows its effectiveness in improving the slowdown, significantly. Note that we use the same storage (storage used for 1MB LLC) for GCT and RCC even with 2MB and 4 MB LLC per core. So far, we have shown the effectiveness of rowhammer cache with the highly accurate state-of-the-art Berti prefetcher. We also evaluate rowhammer cache with other state-of-the-art L1 and L2 prefetchers like IPCP [30], Bingo [8], and SPP [10]. In case of aggressive prefetchers causing more DRAM traffic and hence more performance slowdowns, then ideas based on criticality [19], [31], [32] can be used. Rowhammer cache is resilient to all the prefetchers, delivering improvements at the same scale across all the prefetchers.

Rowhammer tracking threshold. Figure 18 shows the effect of the rowhammer tracking threshold (T_T) on the effectiveness of rowhammer cache. We use extremely low T_T values like 62 and 31 to check the effectiveness of the rowhammer cache. Rowhammer cache is equally effective across all thresholds as it significantly bridges the gap between a non-secure system and a HYDRA tracker.

VI. Related Work

SRAM based trackers. Graphene [33] is a tracking mechanism that uses the Misra-Gries algorithm to find the top-N frequently activated rows. The algorithm uses gives a space-efficient way to compute the frequently accessed elements. However,

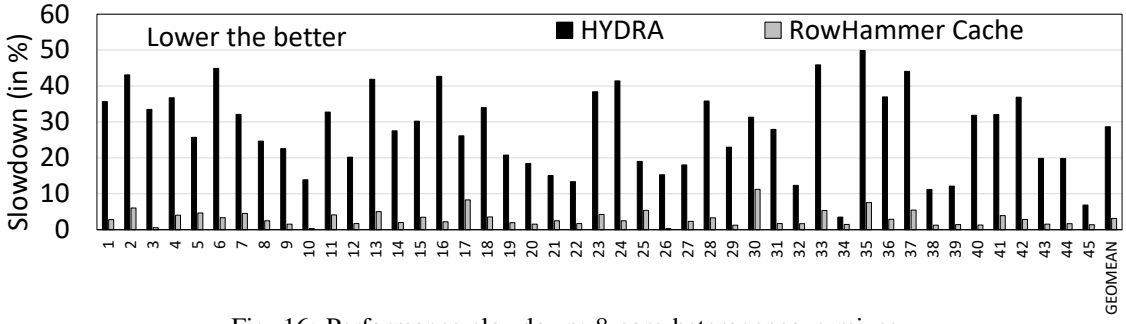

Fig. 16: Performance slowdown: 8-core heterogeneous mixes

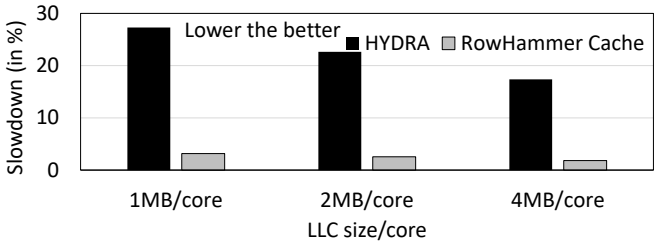

Fig. 17: LLC size sensitivity

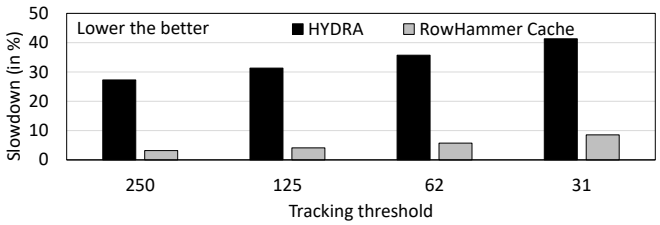

Fig. 18: Tracking threshold sensitivity

the number of entries the algorithm needs to keep track of to ensure rowhammer mitigation is inversely proportional to T_{RH}. Consequently, the storage overhead increases to around 500KB of expensive CAM at $T_{RH} = 500$. D-CBF [44] uses area-efficient bloom filters to track the row activation counts. The counting bloom filters map multiple DRAM rows to a single entry that stores the total activation counts of these rows. Although more area efficient than trackers with similar SRAM overhead, D-CBF still uses over 400KB of SRAM and over 50KB of CAM even for $T_{RH} = 1K$. D-CBF must maintain the same false-positive rate even at even lower rowhammer thresholds to limit the additional refresh operations. However, this quickly becomes impractical due to the increased SRAM overhead. TWiCe [23] keeps a table of counters to track the row activations counts of frequently accessed rows. This is done by pruning the counts for rows that are activated infrequently and hence unlikely to cross T_{RH}. However, pruning becomes ineffective at low thresholds as the pruning threshold (th_{PI}) drops to 0. At thresholds like T_{RH} = 500, TWiCe tables have to store activation counts for each DRAM row, hence demanding extremely high SRAM overhead. Nohammer [24] mitigates eviction-based RH attacks by temporarily extending the associativity of the cache set that is being targeted by utilizing another cache set as the extended set. This keeps the cache lines of aggressor rows on the extended set under the eviction-based RH attack.

DRAM based trackers. The CRA [21] scheme maintains row activation counts for each row in the DRAM. In the baseline DRAM system with 2^{22} rows and $T_T = 250$, this would require 4MB of space. As it would be too expensive to implement in SRAM, the counters can be stored in a reserved region of DRAM. However, even with a dedicated cache for the counters, this mechanism induces a high-performance overhead.

Probabilistic mechanisms. Unlike the previously discussed works, PARA [22] is a probabilistic rowhammer mitigation mechanism. This means that PARA does not guarantee rowhammer prevention but rather makes bit flips extremely unlikely to occur in practice. PARA works by refreshing adjacent rows with a small probability each time a row is opened and closed. As it is stateless, implementing PARA requires little hardware overhead. However, the probability with which adjacent rows must be refreshed to ensure safety increases as T_{RH} decreases. Due to these extra refreshes, PARA's energy and performance overhead increases rapidly at lower rowhammer thresholds. Other probabilistic mechanisms like ProHIT [40], and MRLoc [45] have been shown to be vulnerable to specific adversarial patterns, which significantly increase the probability of a successful attack [33].

VII. Conclusion

Rowhammer attack is here to stay and with ultra-low rowhammer tracking threshold, rowhammer tracking, and mitigation techniques play an important role in designing a secure memory system. The state-of-the-art HYDRA tracker is a promising technique. However, HYDRA when evaluated with realistic simulation parameters and memory-intensive workloads, incurs significant performance slowdown. We proposed the rowhammer cache, a simple and lightweight enhancement to HYDRA that steals space from the last-level cache for the SRAM structures of HYDRA. rowhammer cache uses two ways from LLC, which is good enough in improving the slowdown significantly.

VIII. Acknowledgement

We would like to thank all the anonymous reviewers for their insightful comments and suggestions. We would also like to thank members of the CASPER group especially Sumon, Shubham, and Anubhav, for their feedback on the initial draft.

REFERENCES

[1] "Utahsim Documentation," https://users.cs.utah.edu/~rajeev/pubs/usimm.pdf, May 2012.

[2] "Utahsim simulator," https://github.com/pranith/usimm, May 2012.

[3] "The 2nd data prefetching championship (dpc-2)," Jun. 2015. [Online]. Available: https://comparch-conf.gatech.edu/dpc2/

[4] "The 3rd data prefetching championship (dpc-3)," Jun. 2019. [Online]. Available: https://dpc3.compas.cs.stonybrook.edu/

[5] "SPEC CPU 2017 traces for champsim," https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/, Feb. 2019.

[6] "ChampSim simulator," http://github.com/ChampSim/ChampSim, May 2020.

[7] "GAP traces for champsim," https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561, Mar. 2021.

[8] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *25th*, Feb. 2019, pp. 399–411.

[9] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A complete in-dram rowhammer mitigation," in *Workshop on DRAM Security (DRAMSec)*, vol. 22, 2021, p. 110.

[10] E. Bhatia, G. Chacon, S. H. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *46th*, Jun. 2019, pp. 1–13.

[11] A. EPYC, "Amd epyc 7702," https://www.amd.com/en/products/cpu/amd-epyc-7702p, May 2019.

[12] S. Eyerman, W. Heirman, and I. Hur, "Dram bandwidth and latency stacks: Visualizing dram bottlenecks," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 322–331.

[13] P. Frigo, E. Vannacci, 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 *S&P*, May 2020, best Paper Award, Pwnie Award for Most Innovative Research, IEEE Micro Top Picks Honorable Mention, DCSR Paper Award.

[14] Z. Greenfield and L. Tomer, "Throttling support for row-hammer counters," Feb. 2 2016, uS Patent 9,251,885.

[15] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. Ox27;Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2018, pp. 245–261. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2018.00031

[16] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, ser. DIMVA 2016. Berlin, Heidelberg: Springer-Verlag, 2016, p. 300–321. [Online]. Available: https://doi.org/10.1007/978-3-319-40667-1_15

[17] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, ser. DIMVA 2016. Berlin, Heidelberg: Springer-Verlag, 2016, p. 300–321. [Online]. Available: https://doi.org/10.1007/978-3-319-40667-1_15

[18] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 60–71, 2010.

[19] N. S. Kalani and B. Panda, "Instruction criticality based energy-efficient hardware data prefetching," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 146–149, 2021.

[20] I. Kang, E. Lee, and J. H. Ahn, "Cat-two: Counter-based adaptive tree, time window optimized for dram row-hammer prevention," *IEEE Access*, vol. 8, pp. 17 366–17 377, 2020.

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

[22] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. IEEE Press, 2014, p. 361–372.

[23] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "Twice: Preventing row-hammering by exploiting time window counters," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 385–396. [Online]. Available: https://doi.org/10.1145/3307650.3322232

[24] S. Lee, K. Kang, G. Park, N. Kim, and D. Kim, "Nohammer: Preventing row hammer with last-level cache management," *IEEE Computer Architecture Letters*, vol. 22, no. 02, pp. 157–160, jul 2023.

[25] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[26] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Rowpress: Amplifying read disturbance in modern dram chips," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–18.

[27] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1116–1121.

[28] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2020.

[29] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibánez, V. Viñals-Yúfera, and A. Ros, "Berti: an accurate local-delta data prefetcher," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 975–991.

[30] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *47th*, Jun. 2020, pp. 118–131.

[31] B. Panda, "Clip: Load criticality based data prefetching for bandwidth-constrained many-core systems," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 714–727. [Online]. Available: https://doi.org/10.1145/3613424.3614245

[32] B. Panda and S. Balachandran, "Introducing thread criticality awareness in prefetcher aggressiveness control," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6.

[33] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. Ho Ahn, and J. W. Lee, "Graphene: Strong yet lightweight row hammer protection," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1–13.

[34] 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.

[35] A. Ryzen, "Amd ryzen threadripper," 2019.

[36] 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.

[37] 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)*, 2022, pp. 108–123.

[38] I. Shah, A. Jain, and C. Lin, "Effective mimicry of belady's min policy," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 558–572.

[39] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreading processor," in *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000*, L. Rudolph and A. Gupta, Eds. ACM Press, 2000, pp. 234–244. [Online]. Available: https://doi.org/10.1145/378993.379244

[40] M. Son, H. Park, J. Ahn, and S. Yoo, "Making dram stronger against row hammering," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[41] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*. IEEE, 2023, pp. 374–389. [Online]. Available: https://doi.org/10.1109/HPCA56546.2023.10070999

[42] I. Xeon, "Xeon platinum," 2023.

[43] A. G. Yağlikçi, 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.

[44] A. G. Yaglikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed DRAM rows," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 2021, pp. 345–358. [Online]. Available: https://doi.org/10.1109/HPCA51647.2021.00037

[45] J. M. You and J.-S. Yang, "Mrloc: Mitigating row-hammering based on memory locality," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[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)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2020, pp. 28–41. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MICRO50266.2020.00016