Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations

Loïc France, Florent Bruguier, David Novo, Maria Mushtaq, Pascal Benoit

Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations

Loïc France, Florent Bruguier, David Novo, Maria Mushtaq, and Pascal Benoit

Abstract—Modern computer memories have shown to have reliability issues. The main memory is the target of a security threat called Rowhammer, which causes bit flips in adjacent victim cells of aggressor rows. Numerous countermeasures have been proposed, some of the most efficient ones relying on row access counters, with different techniques to reduce the impact on performance, energy consumption and silicon area. In these proposals, the number of counters is calculated using the maximum number of row activations that can be issued to the protected bank. As reducing the number of counters results in lower silicon area and energy overheads, this can have a direct impact on the production and usage costs.
In this work, we demonstrate that two of the most efficient countermeasures can have their silicon area overhead reduced by approximately 50% without impacting the protection level by changing their counting granularity.

Index Terms—Security, Rowhammer, DRAM.

1 Introduction

Memory is a key component in modern computers. The largest portion of the data used at run-time is stored in a Random-Access Memory (RAM) built with Dynamic RAM (DRAM) CMOS technology. In the past decades, CMOS process technology became more efficient and smaller. DRAM manufacturers have been able to put more memory cells in much smaller spaces, resulting in better performance and lower cost. However, making DRAM smaller resulted in higher vulnerability to what Kim et al. [1] described as disturbance errors: activating a DRAM row creates electrical disturbances that slightly affect adjacent rows. Repeated activation of neighbors of a victim row can imply a loss of charge for capacitors in this row, effectively deleting the stored data. This became an important security threat with the emergence of Rowhammer (RH) attacks [1] that exploit this vulnerability. Using precisely targeted bit-flips, this type of attack is able to perform privilege escalation [2] or to retrieve sensitive information [3]. RH became even more important as state-of-the-art attacks were successfully mounted in JavaScript from a web browser sandbox [4], [5], or even without malicious code running on a victim server, using only network requests [6].

Therefore, this problem is being widely studied in academia and industry at present. In order to counter the RH attack, multiple mitigation techniques have been proposed [1], [7], [8], [9], [10], [11], [12]. Some mitigations rely on probability to refresh the victims before they get corrupted, while others count activations of DRAM rows to detect attacks. Detecting the aggressor before acting reduces the performance overhead as the protection does not disturb the normal operation of the memory. However, it implies additional memory to store counters, which incurs silicon area and energy consumption overheads.

In this paper, we propose to change the scope at which row activations are counted from the bank level to the rank level to reduce the storage requirements of two state-of-the-art mitigation proposals by 40% to 50%, without affecting the protection level. While a previous publication studied the effect of this change for two mechanisms [13], this proposal intends to generalise the impacts of a granularity change to what we consider to be the most important counter-based Rowhammer protections.

• *Loïc France, Florent Bruguier, David Novo, and Pascal Benoit are with LIRMM, University of Montpellier, CNRS, France. E-mail: {firstname}.{lastname}@lirmm.fr*
• *Maria Mushtaq is with LTCI, Télécom Paris, Institut Polytechnique de Paris, France. E-mail: maria.mushtaq@telecom-paris.fr*

2 Background

2.1 DRAM architecture and operation

Modern computer architectures use a DRAM main memory to store run-time data, and several cache levels to speed up access to frequently-used data. Addressing a data in the main memory is done using multiple levels: channel, rank, bank group (BG), bank, row, and column. Fig. 1 illustrates the different levels for the main memory.

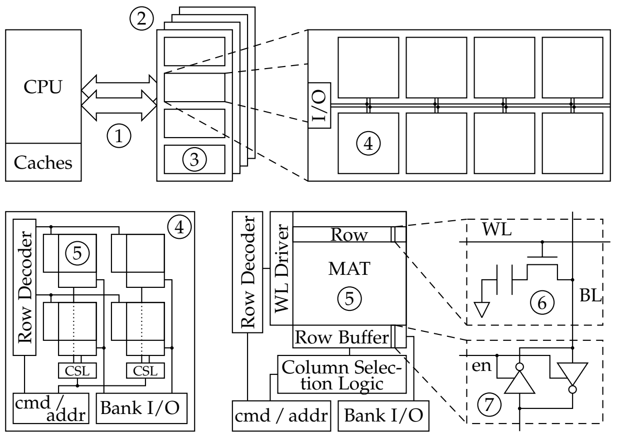

Fig. 1. DRAM internal architecture.

The CPU has multiple channels ① to communicate with the DRAM ranks ②. Multiple DRAM ranks can use the same channel. A DRAM rank contains multiple DRAM chips ③ that are accessed in parallel. The data bytes of a memory access are split across all the chips of the rank, and processed in parallel. Each rank contains multiple banks ④ which can be further organized in BG of e.g., 4 banks each.

In a bank, the data is organized in memory arrays (MAT) ⑤, containing a matrix of memory cells ⑥. A memory cell stores one bit of data using the charge of a capacitor. An access transistor acts as a relay controlled by a wordline (WL) to connect this capacitor to a bitline (BL). All cells of a row share the same WL, and all cells of a column share the same BL. BLs are connected at one end to a sense amplifier (SA) ⑦, which reads the voltage of the BL and acts as a row buffer (RB) to store the last accessed row.

To access an address in the main memory, the memory controller first finds the appropriate bank where the requested data is stored. Second, if the requested data is not in the row buffer, the controller issues an ACTIVATE command (ACT) to transfer the target bank row to the row buffer. The bank charges all the BL with $\frac{V_{DD}}{2}$. Third, the WL is used to open all transistors of the row to raise (lower) the voltage of the BL by discharging (charging) the capacitor into (from) the BL. Finally, the SA is enabled, interprets and stores the value as a 1 or a 0 in the RB. After a short period of time or when the RB is needed for another row, the value is stored back in the MAT using the PRECHARGE command (PRE). The WL is kept open, and the BL is set to either V_{DD} or GND to restore the capacitor to its proper voltage. The WL is then set to GND to close the access transistors.

As capacitors are not perfect insulators, they leak charge over time. To maintain the data during long periods of time, the memory controller cyclically restores all cells at a row-level granularity. This operation, known as REFRESH (REF), is akin to an ACT followed by a PRE.

2.2 Rowhammer and existing mitigation techniques

As DRAM technologies become denser, wordlines tend to get closer to each other. This proximity makes them more vulnerable to electrical disturbance: Due to multiple factors such as electromagnetic coupling and charge pumping through the silicon, activating a row slightly depletes the capacitors in neighboring rows. If repeated enough times, this operation can lower the charge of victim capacitors under the threshold at which it is considered high, therefore deleting the stored value. This is known as the Rowhammer attack. Due to the still increasing density, the number of ACTs an aggressor must send to the neighbors of a victim row has gone from 130k for DDR3 to around 9.6k in the latest LPDDR4 [14]. Fig. 2 illustrates the evolution of the voltage across a capacitor that stores one bit of data under normal operation and under RH attack. The stored value is originally set to 1, and after the disturbance, it is changed to 0 without any write operation on it.

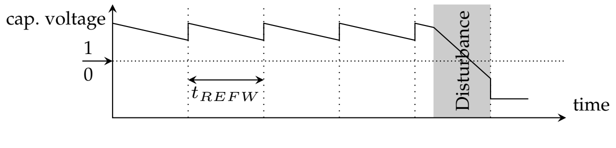

Fig. 2. Memory cell capacitor voltage evolution under normal behavior and attack. Stored value starts at 1, and changes to 0 after corruption.

Since the discovery of the RH problem, multiple mitigation techniques have been proposed with different advantages. Among the existing mitigations, we can distinguish two categories: probability- and counter-based.

The former aims at making corruption statistically improbable by randomly refreshing the neighbors of activated rows, often using light but imprecise detection to reduce the performance overhead. They offer a very low area overhead at the expense of a relatively high False Positive (FP) rate and a non-zero False Negative (FN) rate [1], [10], [11].

The latter relies entirely on a detection mechanism that counts the ACTs of each row during the refresh window. When the count reaches a threshold, the row is considered as an aggressor and the mechanism either refreshes the victim rows or prevents further accesses to the aggressor rows. They use different algorithms to reduce the number of counters to use. They offer a very low FP rate and a FN rate of zero at the expense of a higher area overhead to store the counters [7], [8], [9], [12]. In this paper, we will focus on counter-based mitigations.

2.3 DRAM timing parameters

For counter-based mitigation, the configuration (i.e., the number of counters) depends on the bandwidth of the memory, which can be calculated using its timing parameters. Table 1 lists the relevant timing parameters for three models of DDR3, DDR4 and DDR5, and Fig. 3 illustrates them.

TABLE 1
Relevant timing parameters (in ns unless specified otherwise) for DDR3 1600 [15], DDR4 2400 [16], and DDR5 4000 [17] 8Gb x8 modules.

Name	Description	DDR3	DDR4	DDR5
t_{RC}	Same-bank ACT interval	48.75	45.8	46
t_{RRD_S}	Diff.-BG ACT interval	6.25	3.3	4
t_{RRD_L}	Same-BG ACT interval	-	4.9	5
t_{FAW}	Four activate window	30	21.67	16
t_{REFW}	REF window (ms)	64	64	32
t_{REFI}	REF interval (µs)	7.8	7.8	3.9
t_{RFC}	REF command duration	350	350	195

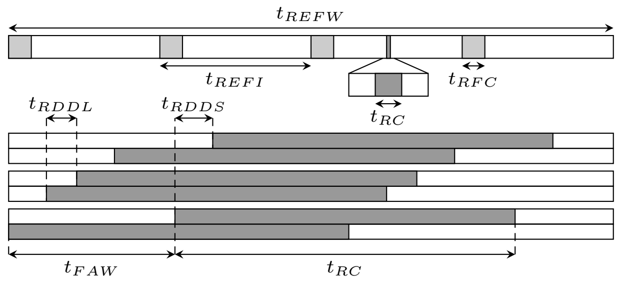

Fig. 3. Illustration of DDR timings. Refreshes are in light grey and ACTs are in darker grey. Banks are grouped by pairs in 3 bank groups.

The minimum interval between two ACTs is defined by t_{RC}, t_{RRD_L} or t_{RRD_S} if the two rows are in the same bank, in different banks but the same BG, or in different BG, respectively. A maximum of four ACTs can be issued to the rank during t_{FAW}. t_{FAW} is the most limiting parameter for frequent row activations in different banks. DRAM rows are periodically refreshed every t_{REFW}. Refreshes of all rows are spread across this period. REF commands are triggered every t_{REFI} and keep the bank busy during t_{RFC}.

3 Counting at a different granularity

The objective of the RH attack is to send enough ACTs to an aggressor row within a refresh cycle (t_{REFW}) to corrupt some bits in neighboring victim rows. We define the RH detection threshold (D_{RH}) as the ACT count at which the detection mechanism must trigger a mitigation technique to prevent the corruption. Usually, the storage overhead needed to guarantee the detection of a RH attack mostly depends on (1) the bitwidth of each counter S (that directly depends on D_{RH}) and (2) the total number of counters. In principle, we would need to keep count of each row separately. However, having one counter per row is excessively expensive (e.g., a DDR4 bank can have 64K rows per bank, which would require about 1Gb of counter storage for a typical D_{RH} of 16384 [14]). Thus, most existing counter-based proposals include methods to minimize the number of counters while still guaranteeing detection. These methods take D_{RH} and W, the maximum number of ACTs that can be issued to the considered memory during t_{REFW}, as inputs. D_{RH} determines the vulnerability of the memory to corruption and is fixed after DRAM fabrication. Hence, in this work we aim at reducing the memory overhead of a detection mechanism by lowering W which in turn reduces the number of counters. Our key observation is that the value of W varies depending on the granularity at which ACTs are counted. While existing mitigation techniques propose to keep the count at a bank granularity, a previous publication calculated the benefits of having a rank-level counting granularity [13] for two proposals. We propose to extend this study to other mitigation proposals.

Considering the traditional bank granularity, W_B (W at bank-level) is limited by the minimum interval between two ACTs t_{RC} and periodic refreshes (t_{RFC} every t_{REFI}). The value of W_B can be calculated using the following equation:

$$W_B = \left\lceil \frac{t_{REFW} \times \left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{RC}} \right\rceil . \qquad (1)$$

Alternatively, considering a rank granularity, the frequency of ACTs is not limited by the maximum frequency of individual banks. It is limited by t_{FAW}, which restricts the ACTs count in a rank to four per t_{FAW}. As a consequence, not all banks of a rank can be accessed at their maximum frequency. The periodic refreshes also impact the maximum ACT frequency of a rank. DDR3 and DDR4 use the all-bank refresh command, which refreshes all banks simultaneously, keeping the rank busy for t_{RFC} every t_{REFI}. DDR5 standard introduced the single-bank refresh command [18] which allows the banks of a rank to be refreshed individually, allowing the available banks of a rank to be used while some banks are being refreshed. Hence, the value of W at rank-level W_R can be calculated using

$$W_R = \left\lceil \frac{t_{REFW}(1 - \frac{t_{RFC}}{t_{REFI}})}{t_{FAW} \div 4} \right\rceil \qquad (2)$$

for DDR3 and DDR4. Instead, DDR5 follows the following equation:

$$W_R = \left\lceil \frac{t_{REFW}}{t_{FAW} \div 4} \right\rceil . \qquad (3)$$

Considering N_{bank} as the number of banks per rank, the total number of counters in a rank for bank-level detection mechanism is N_{bank} times the number of counters per bank. If the number of counters is proportional to W, the effective value of W for the sum of all bank-level detection mechanisms in a rank is $N_{bank} \times W_B$. Hence, when $N_{bank} \times W_B > W_R$, a rank-level detection mechanism has a lower effective W than the sum of all bank-level detection mechanisms. As a consequence, moving to rank-level granularity could reduce the number of counters needed to protect against RH attacks.

4 Results

In this section, we calculate the effective reduction in W when changing the granularity of the RH detection, and its storage benefits on two state-of-the-art countermeasures.

Table 2 lists the different values of W_B and W_R, the number of banks per rank, and the reduction in W when changing the detection level from bank to rank, for DDR3, DDR4 and DDR5. The reduction in W is calculated using the following formula: $1 - \frac{W_R}{W_B \times N_{bank}}$. Interestingly, the more recent DDR technologies achieve a more important reduction, from 19% for DDR3 to 62% for DDR5, due to the trend of including more banks per rank.

TABLE 2
W reduction for DDR3, DDR4 and DDR5

	W_B	W_R	N_{bank}	W reduction
DDR3	1.25×10^6	8.15×10^6	8	19%
DDR4	1.33×10^6	11.3×10^6	16	47%
DDR5	6.61×10^5	8.00×10^6	32	62%

To quantify the memory reduction resulting from the reported W reductions, we consider two recently published RH countermeasures, namely Graphene [7] and BlockHammer [8]. Since these countermeasures were only dimensioned for DDR4 memories, the results in the rest of the paper are restricted to DDR4. All changes on the parameters are calculated to maintain the original detection accuracy.

Graphene. Graphene [7] stores the most activated rows in a Content-Addressable Memory (CAM) and uses the Misra-Gries algorithm to only count the ACTs on the N_{entry} most activated rows. According to the algorithm, the minimum number of entries in the CAM is calculated using the following equation:

$$N_{entry} = \left\lfloor \frac{W}{D_{RH} \div 2} \right\rfloor . \qquad (4)$$

Every entry includes a key-value pair, where the key is the row address and the value is a counter, and one overflow bit for the counter, which is used as a trigger to detect the aggressors and refresh the victim rows. The counter needs to go up to $D_{RH} \div 2$, which corresponds to 13 bits for a typical $D_{RH} = 16384$ [14]. When counting at the bank level, as proposed in the original paper, the total entry size is 30 bits. This includes a row address of 16 bits to address the 64K rows included in a bank (for the considered DDR4 memory), the 13 bits of the counter and the overflow bit. According to Eq. 4, a prototypical DDR4 memory needs 162 CAM entries. Thus, the total CAM size is 16 banks$\times 162$ entries$\times 30$ bits $=$ 9.49KiB per rank (5.06KiB for the keys, 4.43KiB for the rest). Alternatively, counting at a rank granularity, as we propose in this paper, requires extending the row address to 20 bits to cover the 16 banks in a rank. As a result, the total entry size becomes 34 bits. Considering W_R in (4), the number of CAM entries becomes 1394. Thus, the total CAM size is 1394 entries $\times$ 34 bits $=$ 5.71KiB per rank (3.36KiB for the keys, 2.35KiB for the rest), which is 40% lower than when counting at the bank level.

BlockHammer. BlockHammer [8] uses Counting Bloom Filters (CBF) to count the number of ACTs for each row. A CBF uses hashing to associate each possible entry to a unique set of k counters out of a total of m counters. When a row is activated, the CBF increments the corresponding k counters. When all k counters are above a pre-defined threshold N_{BL}, BlockHammer considers the row as an aggressor row and blocks further access to this row until the end of the refresh window to stop the attack. To maintain the same detection precision when counting at the rank level, we adapt the CBF architecture to achieve a similar False Positive Probability (P_{FP}). According to CBF theory [19], the P_{FP} of BlockHammer can be calculated as follows:

$$P_{FP} = \left(1 - \sum_{l<N_{BL}} \binom{kW}{l} \left(\frac{1}{m}\right)^{l} \left(1 - \frac{1}{m}\right)^{kW-l}\right)^{k}. \quad (5)$$

The implementation proposed by the authors of BlockHammer, which counts at the bank level, includes $m = 1024$ counters and $k = 3$ hash functions. This corresponds to a $P_{FP} \approx 10^{-108}$ for the considered DDR4 memory. To maintain the P_{FP} when W increases to count at the rank level, m and/or k should be changed. For simplicity, we choose to keep the value of k and only change the number of counters m to the closest power-of-two leading to the desired P_{FP}, which corresponds to 8192. The authors of Blockhammer propose a maximum counting value $N_{BL} = 8192$. They use two CBFs in a time-interleaved fashion. Each CBF takes turns to count ACTs. Hence, the total memory size used by the mechanism for one rank with an implementation counting at the bank level is 16 banks $\times$ 2 CBFs $\times$ 1024 counters $\times$ 13 bits $=$ 52KiB. Instead, for our rank-level approach the total memory size is 2 CBFs $\times$ 8192 counters $\times$ 13 bits $=$ 26KiB, which is 50% lower than the bank-level implementation.

Table 3 summarises the memory usage for both Graphene [7] and BlockHammer [8] considering a bank- and a rank-level protection in a DDR4 memory. We observe that both RH mitigation techniques significantly reduce their storage overheads when changing the granularity of the RH detection to the rank level.

TABLE 3
Comparison of the implementation of Graphene and BlockHammer at bank level and rank level for DDR4

Mitigation	Bank level	Rank level	Storage reduction
Graphene	9.61KiB	5.79KiB	40%
BlockHammer	52.00KiB	26.00KiB	50%

Counting at a rank granularity implies coping with a higher ACT frequency. BlockHammer introduces a $< 1ns$ latency to check the safety of every memory access, and can therefore withstand the minimum period between two activations at rank level $t_{RRD\_S} > 3ns$. While the authors of Graphene do not specify the processing time, they suggests to use a recent CAM design [20], which is capable of searching words at 370MHz (2.7ns). This is higher than the minimum frequency to meet the 21.67ns t_{FAW} for the DDR4, or 200MHz for the DDR5. If timing becomes an issue, multi-banking solutions can be explored to parallelize multiple access to the counters.

5 Conclusion

The most efficient Rowhammer countermeasures rely on counters to detect the attack. Most of them use a separate set of counters for each bank. In this work, we have shown that by using a common set of counters for a rank, we can decrease the memory usage of the Graphene [7] and BlockHammer [8] mitigation techniques by approximately 50% for typical DDR4 DRAM memories. We show that this reduction will be more important with the newer DDR5 standard due to the increase in banks per rank. Furthermore, the RH corruption threshold continues to decrease in newer technology nodes [14], which leads to growing storage requirements in RH mitigation techniques. Thus, we believe that future mitigation proposals should consider the implementation at rank-level rather than bank-level, and that previously-published counter-based mitigations could be improved by moving the implementation to rank-level.

Acknowledgments

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-19-CE39-0008 (project ARCHI-SEC).

References

[1] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ISCA'14*.
[2] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, 2015.
[3] A. Kwong *et al.*, "Rambleed: Reading bits in memory without accessing them," in *SP*, 2020.
[4] D. Gruss *et al.*, "Rowhammer.js: A remote software-induced fault attack in javascript," in *DIMVA'16*.
[5] F. de Ridder *et al.*, "SMASH: Synchronized many-sided rowhammer attacks from javascript," in *USENIX Security'21*.
[6] M. Lipp *et al.*, "Nethammer: Inducing rowhammer faults through network requests," in *EuroS&PW'20*.
[7] Y. Park *et al.*, "Graphene: Strong yet lightweight row hammer protection," in *MICRO'20*.
[8] A. G. Yağlikçi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *HPCA'21*.
[9] G. Saileshwar *et al.*, "Randomized row-swap: Mitigating row hammer by breaking spatial correlation between aggressor and victim rows," in *ASPLOS'22*.
[10] M. Son *et al.*, "Making DRAM stronger against row hammering," in *DAC'17*.
[11] J. M. You and J.-S. Yang, "MRLoc: Mitigating row-hammering based on memory locality," in *DAC'19*.
[12] S. M. Seyedzadeh *et al.*, "Mitigating wordline crosstalk using adaptive trees of counters," in *ISCA'18*.
[13] I. Kang *et al.*, "CAT-TWO: Counter-based adaptive tree, time window optimized for DRAM row-hammer prevention," *IEEE Access*, 2020.
[14] J. S. Kim *et al.*, "Revisiting rowhammer: An experimental analysis of modern DRAM devices and mitigation techniques," in *ISCA'20*.
[15] JEDEC, "JESD79-3 DDR3 SDRAM," 2013.
[16] ——, "JESD79-4 DDR4 SDRAM," 2021.
[17] Micron, "DDR5 SDRAM product core datasheet," 2021.
[18] ——, "Micron DDR5 SDRAM: New features," 2021.
[19] S. Tarkoma *et al.*, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, 2011.
[20] S. Jeloka *et al.*, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory," *JSSC*, 2016.