Towards Enhanced System Efficiency while Mitigating Row Hammer

KAUSTAV GOSWAMI, Indian Institute of Information Technology Guwahati, India
DIP SANKAR BANERJEE, Indian Institute of Technology Jodhpur, India
SHIRSHENDU DAS, Indian Institute of Technology Ropar, India

In recent years, DRAM-based main memories have become susceptible to the Row Hammer (RH) problem, which causes bits to flip in a row without accessing them directly. Frequent activation of a row, called an *aggressor row*, causes its adjacent rows' (*victim*) bits to flip. The state-of-the-art solution is to refresh the victim rows explicitly to prevent bit flipping. There have been several proposals made to detect RH attacks. These include both probabilistic as well as deterministic counter-based methods. The technique of handling RH attacks, however, remains the same. In this work, we propose an efficient technique for handling the RH problem. We show that the mechanism is agnostic of the detection mechanism. Our RH handling technique omits the necessity of refreshing the victim rows. Instead, we use a small non-volatile Spin-Transfer Torque Magnetic Random Access Memory (STTRAM) that ensures no unnecessary refreshes of the victim rows on the DRAM device and thus allowing more time for normal applications in the same DRAM device. Our model relies on the migration of the aggressor rows. This accounts for removing blocking of the DRAM operations due to the refreshing of victim rows incurred in the previous solution. After extensive evaluation, we found that, compared to the conventional RH mitigation techniques, our model minimizes the blocking time of the memory that is imposed due to explicit refreshing by an average of 80.72% in the worst-case scenario and provides energy savings of about 15.82% on average, across different types of RH-based workloads. A lookup table is necessary to pinpoint the location of a particular row, which, when combined with the STTMRAM, limits the storage overhead to 0.39% of a 2 GB DRAM. Our proposed model prevents repeated refreshing of the same victim rows in different refreshing windows on the DRAM device and leads to an efficient RH handling technique.

CCS Concepts: • **Hardware** → **Non-volatile memory**; **Memory and dense storage**; **Dynamic memory**; **Memory and dense storage**; **Dynamic memory**; **Non-volatile memory**; • **Security and privacy** → **Security in hardware**; **Security in hardware**

Additional Key Words and Phrases: DRAM, row hammer, memory cache, STTMRAM

ACM Reference format:
Kaustav Goswami, Dip Sankar Banerjee, and Shirshendu Das. 2021. Towards Enhanced System Efficiency while Mitigating Row Hammer. *ACM Trans. Arch. Code Optim.* 18, 4, Article 40 (July 2021), 26 pages.
https://doi.org/10.1145/3458749

This work is partially supported by Science and Engineering Research Board (SERB) grant, numbered EEQ/2019/000695, funded by the Department of Science & Technology, Government of India.

Authors' addresses: K. Goswami, Indian Institute of Information Technology Guwahati, IT Park Street, Bongora, Guwahati, Assam, India, 781015; email: kaustavgoswami.2013@gmail.com; D. Sankar Banerjee, Indian Institute of Technology Jodhpur, NH 62, Surpura Bypass Rd, Karwar, Rajasthan, India, 342037; email: dipsankarb@iitj.ac.in; S. Das, Indian Institute of Technology Ropar, Nangal Rd, Rupnagar, Punjab, India, 140001; email: shirshendu@iitrpr.ac.in.

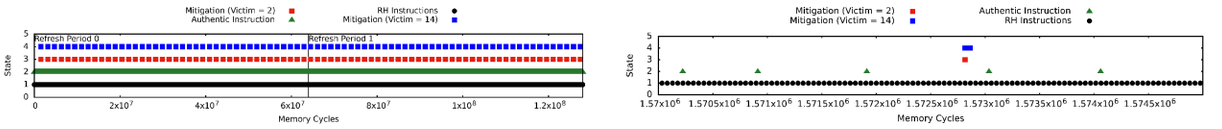
1544-3566/2021/07-ART40 $15.00
https://doi.org/10.1145/3458749

1 INTRODUCTION

The most popular technology in the domain of memories is the capacitor-based DRAM technology, which has spawned several generations (DDRx [27, 41, 44], GDDRx [36, 42], LPDDRx [43], DDRxL [26], etc.). DRAM devices provide a cheap option for memory and provide moderately good performance. To further increase performance, **Static Random Access Memory (SRAM)** was proposed, which acted as memory caches [30]. The scaling of DRAM devices over the generations has boosted DRAM performance in recent years. However, several problems such as the deviation of nominal parameters of the device (process, voltage, and temperature variation [4]) and electromagnetic interference **(Row Hammer (RH)** [18]) have emerged due to scaling. It is projected that this electromagnetic interference will become more pronounced in future generations of DRAM technology [16].

Due to the scaling of the DRAM device, electromagnetic interference between the DRAM cells is observed in recent DRAM devices [29]. This can be initiated by repeatedly accessing one particular row. This may cause bits of the adjacent rows to flip. The attacked rows are called *aggressor rows,* and the affected rows are called the *victim rows*. The victim rows are the adjacent rows on both sides of the aggressor rows. The number of rows affected by each aggressor row varies between 1 to 7 in each side [18]. Hence, the total number of victim rows for each aggressor row varies between 2 to 14. As the density of the DRAM technology is increasing, the number of DRAM cells within a bank is also increasing. Thus, the chances of a higher number of victim rows per aggressor row are consequential. The straightforward technique to handle the RH attack is to first accurately detect an attack and then selectively refresh the victim rows [18]. We call this particular refreshing, which mitigates an RH attack as *RH-Refresh*. There are many RH detection techniques already proposed. These techniques can be categorized as probabilistic [18, 38], deterministic [14, 24, 37], and remapping-based [10, 17]. Almost all the existing detection techniques have to perform the RH-Refresh after detecting an RH attack. Though the existing techniques guarantee the correctness of the victim rows, they cannot prevent the aggressive activation of the aggressor rows. Such aggressive access prevents other innocent applications to access the memory, as most of the time the DRAM is busy servicing the aggressor rows. As per our experimental analysis, the regular applications (benchmarks) can be considered as innocent applications in terms of RH attacks. The applications conducting RH attacks are specially designed to perform such attacks. We call such applications as *malicious applications*. During execution, the innocent applications should not starve because of these RH-refreshes and aggressive activation.

The RH attack has both direct and indirect impacts on DRAM. The direct impact is obviously the bit flips of the victim rows. Most of the existing RH detection mechanisms only target how to handle the direct impact of RH attack on DRAM. However, RH attacks also have indirect impacts on the DRAM in terms of latency and energy consumption. Refreshing the victim rows (RH-Refresh) and aggressive activation of the aggressor rows block the DRAM for normal read/write operations. It impacts the performance of the DRAM. Also, RH-refreshes and aggressive activation consume additional energy. We consider these impacts as indirect, because they do not create any incorrectness issues in the DRAM. This article proposes a mechanism to handle these indirect impacts of RH attacks. We propose **RH Latency Mitigation (RHLM)** to reduce the blockage time and energy overhead results due to the RH attack and RH-Refresh. The proposed mechanism can be used with most of the existing RH detection mechanisms including but not limited to References [18, 24, 38]. During the experimental evaluation, we have found that our proposed model, RHLM, can save up to 99.97% of blocking time on the DRAM device previously imposed by explicit refreshing of victim rows. In the worst-case scenario, our model still minimizes blocking time by 80.22% on average across different RH-based workloads. In addition, we have also reduced the overall memory sub-system's energy consumption by 15.82% across different workloads.



(a) Incoming Requests to the DRAM Device over 2 Different Refresh Windows

(b) Incoming Requests to the DRAM Device in a Smaller Time Window

Fig. 1. Study on RH attack pattern on the DRAM device.

The article is organized as follows: The immediately following section highlights the motivation of our work, which is followed by a brief study into the background of a DRAM device. Section 3 discusses various related works and further explains the shortcoming of those techniques. Section 4 provides an insight into the proposed technique, which is followed by the evaluation configurations. Section 6 discusses the results obtained and analysis on the same. Section 7 concludes the article.

1.1 Motivation and Contributions

RH Mitigation comprises two steps: (a) Detection of the attack and (b) Mitigation of the attack via refreshing of the victim rows. Existing works like TWICE [24] show that the detection part can be performed in the background without increasing the latency of the DRAM. However, refreshing the victim rows while mitigating the attack blocks the DRAM for normal read/write operations. We did a small study on the access pattern of RH attacks on the DRAM device. The memory trace is collected by executing a malicious application mixed with PARSEC benchmarks [2]. Figure 1 shows the distribution of row activation for the same. There are four states of the trace in Figure 1, where the DRAM device is shown to be operating: (State 1) RH activation; where a malicious READ/WRITE request is sent to the memory, (State 2) authentic accesses; may be either a READ or a WRITE (State 3 and State 4) mitigation of the RH attack with varying victim row counts. It can be observed from Figure 1(a) that, on average, 40 instances of RH-Refreshes are required in one refresh period. The number of explicit refreshes in one instance can range from 2 to 14, depending on the number of victim rows. This consumes from 0.003% to 0.04% additional time for RH-Refreshes. As the density of the DRAM device increases, this blocking time is likely to increase further. Figure 1(b) zooms out the distribution, where we can see that RH activation constantly uses the memory device at all times. This aggressive activation blocks the memory for innocent applications.

Another behavior that is observed during RH is the continuous attack on the same aggressor row across different refresh periods. Figure 1(a) shows incoming access requests to the memory system over two refresh periods. The hammered row remains the same in both periods. States 3 and 4 represent clock cycles consumed for explicit refreshing of the victim rows (RH-Refresh) in both the best and worst cases, respectively. The existing RH mitigation techniques have to perform RH-Refresh in each refresh period. This repeated access, seen in Figure 1, also compels the DRAM device to serve memory requests continuously. In Figure 1(b), we can see that there exist clear gaps between authentic accesses. In normal circumstances, if this period is determined to be larger than a set threshold, then the DRAM device is transitioned into low-power mode [27, 34, 44]. However, if there are continuous RH activations incoming to the memory, then transitioning of DRAM devices into low-power mode is restrictive. Thus, the exploitation of existing power management mechanisms fails to save any power from the DRAM device.

Considering all the aforementioned issues, an alternate technique, which can remove the necessity of RH-Refresh operations, can definitely reduce both blocking times as well as the energy consumption of the memory. This collectively leads us to propose RHLM, which uses a different

approach to handle RH attacks as compared to the existing solution. It relies on the aggressor row migrations rather than the victim row refreshing. Our design lies along the lines of next-generation hybrid memory design. Overall, our concrete contributions lie in the following:

- Minimizing the total blocking time imposed on the system for mitigating RH attacks during RH-Refresh.
- Efficiently utilizing existing power management techniques of a DRAM device by handling the RH-Refresh.
- Improving the overall throughput of the system.

2 BACKGROUND

DRAMs are the *de facto* choice for main memories due to its cheap cost. This section first highlights the basic organization and working of a DRAM that is followed by RH. The section concludes with a brief paragraph on the basics of STTMRAM.

2.1 Organization and Working of a DRAM Device

A modern-day DDRx DRAM device is organized in six hierarchical levels. The memory channel sits at the top, which can be operated concurrently with other channels. A DRAM can accommodate multiple **Dual Inline Memory Modules (DIMMs)**, which is a module containing one or more **random access memory (RAM)** chips. These are divided into ranks, which can be used to distinguish DIMM level independence and internal bank-level independence. The rank is further divided into chips and then into independent banks that provide the lowest level of independent operation. DRAM banks are further divided into rows and then columns.

The master operation of the DRAM device is controlled by **clock enable (CKE)** [27], which must be high for the DRAM to receive commands. The incoming command or address is pushed into the decoding logic of the DRAM. The first command sent to the DRAM is usually an **Activate (ACT)** command, which is responsible for selecting the appropriate bank and row address. The data stored in the corresponding DRAM cells are then transferred to the sense amplifiers that retain the data until a Precharge (PRE) command to the same bank is issued. Every ACT command has to have a PRE command associated with it. A READ or a WRITE can only be performed by the DRAM in its active state.

2.1.1 Power Components and Power Management. Power in a typical DRAM is consumed under five components: Background, Activate; Precharge, Read/Write, **On-Die Termination (ODT)**, and Refresh. CKE dictates the amount of current to be consumed in a DRAM device. In the active state, a higher amount of current is consumed by the device. If there is a prolonged period of inactivity of the device, then CKE is set to low, which consumes a lower amount of current. This management scheme is implemented in a per-rank manner in all existing DRAM devices and is usually invoked whenever the command queue of the DRAM device is empty for a longer period of time [27, 33, 34, 44]. It is also the most popular power management technique employed in DRAMs. However, irrespective of operations, the device always consumes a static amount of power, which is called the background power of a DRAM device. The fraction of power consumed under any ACTIVATE (ACT) and/or PRECHARGE (PRE) command constitutes Activation; Precharge power. A Read or Write consumes power depending upon the fraction of time the data was on the bus. The system-dependent power components, such as output driver power, constitute ODT power. Since the DRAM is a capacitor-based technology, it requires periodic recharging of the capacitors to retain the data it holds. This constitutes the Refresh power.

2.1.2 Row Hammer. Frequently accessing a particular DRAM row causes its adjacent row's bits to flip. This problem is known as the DRAM row hammer problem. It occurs due to the electromagnetic interference between the DRAM cells, which is the result of large-scale integration in state-of-the-art semiconductor design [18]. These bit flips cannot be corrected by the DRAM **Error Correcting Codes (ECC)** [17], which makes it a critical problem [18].

There are typically two approaches to detect RH attacks. The first is based on probability, where the idea is to predict if accessing a row increases the chances of row hammer on its adjacent rows. These adjacent rows are now updated with an increased probability of victim rows, and immediate refresh is issued after some threshold is reached. The other method is based on counters [14, 24, 37]. This is a deterministic method that ensures higher reliability. Counters are present for the DRAM rows that help to identify victim rows. The main issue of this method is the hardware overhead of the counters. TWICE [24] solved this problem by using a minimum number of counters for proposing a deterministic RH detection mechanism. Address remapping [10, 17] is the third approach towards solving this issue.

2.2 STTMRAM: Spin-Transfer Torque Magnetic Random Access Memory

Of late, there has been significant progress made towards designing newer memory technologies. These include **Non-Volatile Memories (NVM)** like **Spin-Transfer Torque Magnetic Random Access Memory (STTMRAM)** [40], **Phase Change Memory (PCM)** [7], Wide-IO [12], **High Bandwidth Memory (HBM)** [13], and **Hybrid Memory Cube (HMC)** [21]. These devices are in its preliminary stages but have shown promising results [20–23, 32, 46].

Spin Transfer-Torque Magnetic Random Access Memory, simply STTRAM, in recent times, has garnered significant attention as an alternate option to the DRAM-based main memories. The main reason being its negligible leakage power and high write endurance capacity (as compared to the other NVM technologies). The information carrier in these devices is the ***Magnetic Tunnel Junction* (MTJ)** instead of electric charges. It is a variation of **magnetic random access memory (MRAM)** technology. The main difference between STTMRAM and MRAM is the process of orienting the MTJ. In MRAM, the orientation is set using an external magnetic field. Whereas in STTMRAM, the same is done by using polarized electric current. STTMRAM is based on the principle of spin transfer-torque, which is an effect where the orientation of a magnetic layer in an MTJ or spin valve can be modified using a spin-polarized current [5]. Charge carriers have a property known as spin. It is a small quantity of angular momentum intrinsic to the carrier. An electric current is generally un-polarized. A spin-polarized current contains electrons with either spin. Passing a current through a thick magnetic layer produces spin-polarized current.

Worst-case Read/Write time for these devices is higher than the DRAM device [1, 3, 6, 28]. However, average case read time is less than a DRAM device, which is verified via extensive simulations using popular cycle-accurate memory simulator NVmain [31]. Writes, however, are more expensive in STTMRAM than DRAM devices. Table 1 compares the important parameters of different memory technologies. Khan et al. [15] have shown that STTMRAM devices are also susceptible to RH attacks. However, the probability of a read disturbance is 2.9×10^{-7}.

3 RELATED WORKS

There have been several proposals made to detect RH attacks [10, 14, 17, 18, 24, 37, 38]; the solution to handle the victim rows, however, remains nearly the same, which is explicit refreshing or RH-Refreshing.

Table 1. Parameters of Different Memory Technologies [3]

Parameter	SRAM	DRAM	STTMRAM	PCM
Write Endurance	10^{16}	$>10^{15}$	10^{12}–10^{15}	10^{8}–10^{9}
Read Latency	0.2 ns–2 ns	10 ns	2 ns–35 ns	20 ns–60 ns
Write Latency	0.2 ns–2 ns	10 ns	3 ns–50 ns	2 ns–150 ns
Leakage Power	High	Medium	Low	Low
Dynamic Power	Low	Medium	Low/High	Medium/High

3.1 Row Hammer Detection Mechanisms

The accurate method to detect RH attack would be to keep one counter per DRAM row for counting the total number of activation per refresh period. In a modern DDRx DRAM device, there are approximately 32768 rows per DRAM bank [27]. Hence, using separate counters for each row is not feasible to implement due to its high area requirements. Several optimizations are made on the deterministic approach of RH detection to reduce both performance and area overheads. Recently, DDR4 memories also support a **TRR mode (Target Row Refresh)** [34] that issues explicit refreshes to the victim rows. However, the detection mechanism of TRR is unclear [8].

Kim et al. [18] first proposed a probabilistic method to issue ACTs on victim rows to mitigate RH attacks. The authors would select an aggressor row and issue ACTs to its adjacent rows depending upon a probability function. The authors demonstrated that with a probability of 0.001, they were able to mitigate nearly all incoming RH attacks. Although this technique works under a considerably low area overhead, the probability factor adds to the inclusion of both false positives and true positives. PROHIT, proposed by Son et al. [38], added two tables to measure the extent of hammering. The authors first move a frequently accessed row to their proposed cold table. Further activations of the same row promote the row to the hot table. An ACT to the victim rows is issued to the aggressor row on top of the hot table. The authors minimized the count of false positives in their design. However, a probabilistic model would always tend to include some false positives and true positives in their design.

A deterministic RH detection mechanism (CBT) was proposed by Seyedzadeh et al. [37]. The authors proposed a technique where counters will be assigned dynamically to frequently activated DRAM rows. A counter-based tree is proposed in the work. This tree dynamically assigns counters to frequently accessed rows. These sets of rows are called *hot rows*. The authors first demonstrated the fact that why statically assigned counters fail to detect most RH attacks and why the same would consume a large amount of energy. The dynamic approach, which is essentially composed of a tree structure, converges to a balanced tree in case of uniform memory accesses. There are several threshold values that ultimately detect a successful RH attack. These thresholds define the splitting of the tree as well. Ultimately, a non-uniform binary tree is formed that has two distinct groups of rows: hot and cold. While the hot rows are a set of small rows, which are frequently accessed, cold rows, however, are a set of a large number of rows. The group pertaining to hot rows shrinks, ultimately pinpointing an aggressor row. Finally, refreshes are issued to a set of hot rows, and RH attacks are thus mitigated.

While CBT is one of the most efficient methods to detect and mitigate RH attacks due to its efficiency in terms of both area and energy, the algorithm can be broken via a specific synthetic memory access pattern. Memory access, not necessarily RH attacks, can be synthetically made. One such example is to fill up half of the memory device and repeatedly access the same set of rows. This will cause CBT to split every time and unnecessary refreshes would be issued. This is demonstrated by Lee et al. [24] in their work called TWICE, where the authors mathematically calculated out the number of possible RH attacks in one refresh window (tREFW) and used the

same number of counters. The authors further reduced the number of counters by pruning their model and provided an efficient method to accurately detect RH attacks. On successful detection, RH-Refresh is performed on the victim rows.

An alternate method to handle RH attacks was proposed by Hassan et al. [10] titled CROW, which employs a small DRAM cache, called CROW-cache, where DRAM rows can be copied to. The design of CROW not only limits the model to handle RH attacks but also to improve the overall system's performance via parallel execution of memory access using the CROW-cache. Additional commands to copy rows from the normal DRAM device to the CROW-cache is also proposed in their work. The authors use a new command called ACT-c to copy one row from the DRAM array to the CROW-cache in a single access. The authors modified the circuit of a normal DRAM device with connections to the rows of the CROW-cache. During ACT-c, a row is written back to the DRAM array and CROW-cache simultaneously. It takes around 18% more time to complete one ACT-c command.

CROW's RH handling scheme is to copy the victim rows to the CROW-cache and continue the execution of the program. This opens a window for optimization, as it does not solve the issue of blocking operations imposed on the DRAM device. Moreover, the material used as the CROW-cache is DRAM itself. It carries over the RH threat in the design as well.

A fundamentally different approach to handle RH attacks was proposed by Kim et al. [17] from the conventional probabilistic and deterministic approaches. This technique uses the concept of remapping of DRAM addresses. The objective is to scramble addresses and make RH attacks difficult. The authors used a remapping matrix and a two-level remapping scheme. Addresses incoming to the memory device are first remapped in the chip level to make the address of each chip different. The second level remaps an address inside a chip. This makes every bit in a word to be originated from a different address. RH attacks are evenly distributed in this scheme over multiple words. This allows the ECC, present in the DRAM, to correct the errors caused due to RH attacks.

3.2 Limitations of the Existing Solutions

Although there have been several RH detection mechanisms proposed, the mitigation technique, however, remains nearly the same, i.e., refresh the victim rows before the bits flip. This is achieved by activating the victim rows explicitly, which we call RH-Refresh. In DDR3 memory devices, this problem of RH first appeared on a large scale where bit flips occurred in approximately 139,000 ACT on the aggressor row [18]. The density of these devices is increasing. This means that the effect of RH will increase for DDR4 DRAM modules. This is evident in a recent study by Kim et al. [16], where bit flips for DDR4 memory is observed in 45,000 ACTs only. The authors also state the fact that the state-of-the-art RH mitigation mechanism is additional refreshes. Frigo et al. [8] state that refreshing the DRAM in 2x of its initial refresh interval is also a popular RH mitigating technique. However, the number of victim rows is likely to increase. As per the aforementioned study by Kim et al. [18], the number of victim rows ranged from 2 to 14 in the year 2014. The higher the number of victim rows, the more is the number of RH-Refreshes that are required to mitigate the same. Moreover, the energy associated with mitigation is also higher. The same row can be hammered approximately 40 times in one refresh window (tREFW). Although techniques like TWICE can detect the attack accurately, the number of explicit activates issued to its adjacent rows will remain the same. For each attack, 2 to 14 numbers of RH-Refreshes are required to be issued. This blocks the DRAM device for serving normal DRAM requests. Also, the attack on one single row can continue. This will cause the device to block for RH-Refresh on the same set of victim rows repeatedly.

Another important point to note is the access pattern in RH attack traces. Code Listing 1 depicts a typical memory trace of an RH attack in closed page policy.

Listing 1. RH Trace for Closed Page Row Buffer Policy

```
//ADDRESS  TYPE  CLOCK
0xDEADBEEF  READ  48
0XDEADBEEF  READ  96
0xDEADBEEF  READ  144
```

RH attacks need to appear in quick successions for it to cause bit flips in their adjacent rows. However, they must be separated by at least tRC time, as these addresses pertain to one particular bank. This makes the DRAM rank busy, as the command queue is constantly receiving incoming addresses. This does not allow the device to transition into low-power mode, as discussed in Section 2.1.1. None of the existing techniques exploit this fact to extract power savings in their respective proposed methods.

Our proposal is aimed at addressing the aforementioned shortcomings of the existing RH mitigation mechanism. We propose migration of aggressor rows, instead of refreshing of victim rows, to a technology less affected by RH attacks. In this context, we select STTMRAM technology, which is fundamentally different than DRAM. Although, these devices are also affected by electromagnetic interference, which essentially is responsible for causing bit flips, however, the extent of such attacks are far less than DRAM devices [15]. Moreover, a recent work by Jang et al. [11] has proposed several effective techniques to mitigate such bit flipping attacks on STTMRAM devices.

Migrations will remove most hammered rows, which will allow the DRAM device to serve other requests and also be transitioned into low-power mode. Moreover, after migration, there is no requirement of recurrently refreshing victim rows, as the threat of RH is eliminated in a once-and-for-all manner. However, the one-time cost of migrating the row is added. It becomes a challenge if the memory space of the STTMRAM runs out. However, in Section 4.2.2, we show that, theoretically, it is not possible for the conventional RH mitigating technique to achieve better results than the proposed method. Further, in Section 6, we show that we are able to save nearly all of the blocking time imposed by RH attacks compared to the conventional case.

4 PROPOSED METHODOLOGY

The primary motive of this article is to reduce the latency overhead of the RH attack. The latency overhead of the RH attack is initially discussed in Section 1.1. In DRAM-based main memories, each rank has multiple chips and each chip has multiple banks. RHLM follows a hybrid memory structure. Previously, such hybrid memory structures using PCM and DRAM memories have been proposed by Pourshirazi et al. [32] and Lee et al. [25]. DRAM and STTMRAM as a hybrid memory is also studied and efficient LLC design has been previously proposed by Hameed et al. [9]. The architecture of RHLM is inspired by such designs. To implement RHLM, each DRAM bank is associated with a small-sized STTMRAM-based buffer called **Rh-buf**. The row size of DRAM bank and the Rh-buf is the same, i.e., each row of Rh-buf can store one row of the DRAM. The maximum number of rows that can be stored in the Rh-buf is called the order of the Rh-buf. Both the DRAM bank and the STTMRAM-based Rh-buf have their respective **row buffers (RB)**. For distinction, the row buffer in DRAM bank is termed as RB-bank and the row buffer in Rh-buf is termed as RB-buf. In this article, we propose a novel latency mitigation technique for RH attacks. We call this technique **RH Latency Mitigation (RHLM)**. RHLM can be combined with existing RH detection techniques. Figure 2 shows a DRAM Rank with all the components required for RHLM. All the additional components in this diagram are discussed next.

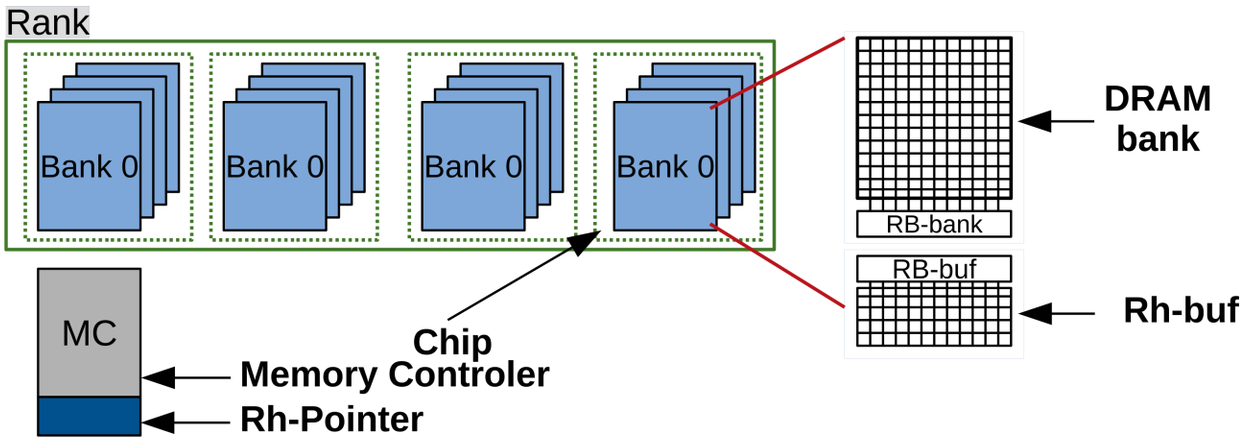


Fig. 2. DRAM rank with RHLM.

4.1 How RHLM Works

On detecting a possibility of RH attack, the aggressor row is migrated to the Rh-buf and the next aggressive activation (for Read/Write) to the row is performed in Rh-buf. Hence, the subsequent aggressive activation of the row has a very low chance to flip the bits of the victim rows. The solution replaces RH-Refreshes with RH-Migrations, where it migrates aggressor rows instead of refreshing the victim rows. The advantages of RHLM are already discussed in Section 1.1. As shown in Figure 2, a small table called Rh-Pointer is maintained by the **memory controller (MC)**. This is because MC has address decoding information that makes it easier to maintain a table with addresses. Each entry in the table has a mapping between the row number of the bank and the corresponding row in the Rh-buf. The total number of entries in this table is in the order of Rh-buf. The complete process of RHLM is discussed under the following headings:

4.1.1 Row Migration. After detecting (or predicting) an RH attack on, say, row X (the aggressor row), the row has to be moved from the DRAM bank to the Rh-buf. To perform the proposed migration, the MC issues an explicit migration command. We call this command either **ROW-MGR** or **ROW-MGRX**, depending upon the direction where the row is migrated to. The functionality of migration is discussed in Section 4.2.2. After moving the row to Rh-buf, the Rh-Pointer is updated accordingly. The consecutive Read/Write requests to row X are served from Rh-buf until it is migrated back to the DRAM device again.

4.1.2 Performing Read/Write Operations. The Read/Write requests to both DRAM or Rh-buf need a row activation (ACT) if the row is not open (not currently in the row buffer). Before issuing any ACT command, the MC first checks the Rh-Pointer. If the row has an entry in Rh-Pointer, then the ACT needs to be performed in Rh-buf, otherwise in the DRAM bank. An entry in Rh-Pointer means the row must be accessed from Rh-buf. If the activation is performed in Rh-buf, then a flag bit **B** is set to 1. Otherwise, it is set to 0. Now the row buffer waits for the column Read/Write command.

The column read command is served either by the RB-bank or the RB-buf depending on the value of the flag bit B. If B is 1, then it means that the data must be read from RB-buf. The procedure of column read is the same in both RB-bank and RB-buf. The time required to complete a read operation (starting from activation) is called **read cycle time (tRC)**. Same as reading, writing also depends on the value of the flag bit B. The procedure of writing in Rh-buf, however, must be handled efficiently, as a write in STTMRAM cell in the worst case is higher than to DRAM cell. This is further discussed in Section 4.2.3.

4.1.3 Benefits of RHLM. Once the aggressor row is migrated to the Rh-buf, it cannot perform the aggressive activation in DRAM. Hence, any chance of bit-flip in victim rows is eliminated. The aggressive access of the aggressor rows is managed by the Rh-buf, which is designed with STTMRAM, hence a very low chance of RH attack on Rh-buf. Since both Rh-buf and the DRAM banks are independent entities, they can be accessed in parallel. Such parallel execution allows the MC to send multiple commands to be performed on both DRAM banks as well as on Rh-buf in parallel. Migration of aggressor rows to the Rh-buf and parallel execution makes the DRAM available for innocent applications without blocking for Rh-Refresh and aggressive activation. As a consequence of the same, an RH attack is isolated within the memory device. In previous methods, blocking time is imposed on the execution time of the innocent applications, as the DRAM device is blocked while mitigating such RH attacks. In RHLM, however, once the aggressors are moved to the Rh-buf, the DRAM device remains exclusive for innocent applications. Though both DRAM and Rh-buf can execute in parallel, it may be possible that due to shifting the aggressive activation into Rh-buf, the DRAM may have some idle intervals. As mentioned in Section 3.2 DRAM can be transitioned into low-power mode in these intervals. All the existing RH detection techniques, which handle the attack within the DRAM bank, cannot hide the blockage time that occurred due to the RH-refresh and the aggressive row activation. Section 1.1 already shows the effect of RH mitigation overhead in terms of latency, which is further explained in Section 3.2. The proposed mechanism, however, has some challenges to implement, which are discussed in the next section.

4.2 Implementation Challenges

The challenges of implementing RHLM are mentioned below. All these challenges are discussed in this section.

- Overhead of Rh-Pointer.
- Overhead of row migration.
- Overhead of write operation on STTMRAM-based Rh-buf.

4.2.1 Overhead of Rh-Pointer. Rh-Pointer is essentially a table that maps rows of the DRAM memory to the STTMRAM (Rh-buf). Each DRAM bank has its own RH-Pointer stored in the MC. With the typical values of tRC, tRAS, and tRP for a DDR4 memory device, we have calculated that we can hammer approximately up to 40 rows in one refresh period (*tREFW*) without pruning [24]. For RHLM, we use the size of the Rh-Pointer as 128, i.e., more than twice the number of rows that can be hammered in one refresh window, and, in an exact power of 2. This value is found after performing extensive experimentation as a balance point for additional storage overhead and desired performance. A value less than 128 may cause constant migrations to and from the Rh-buf and would leave under-utilized bits required for maintaining the RH-pointer. The entries in the table are composed of DRAM rows (16 bits), Rh-buf row (6 bits), flag bit (1 bit), and a dirty bit (1 bit) = 24 bits per entry. These 128 entries are required for each DRAM bank separately. A DRAM rank usually has 8 banks. Hence, there would be 8 Rh-Pointers. A total of $\frac{24\times128}{8} = 384$ bytes of memory will be consumed in the proposed design.

4.2.2 Overhead of Row Migration. As mentioned above, on the detection or prediction of an RH attack on row X, RHLM needs to move the row from the DRAM bank to Rh-buf. The complete procedure of row migration is shown in Figure 3. Row migration has two major modules: **Back-Migration** (issued via **ROW-MGRX** command) and **Migration** (issued via **ROW-MGR** command). Back-Migration is only required when Rh-buf is full. In that case, an existing row from Rh-buf must be migrated back to the DRAM to make a room for X. Figure 3 shows both Back-Migration and Migration as separate rectangles. Each of these rectangles is partitioned vertically.

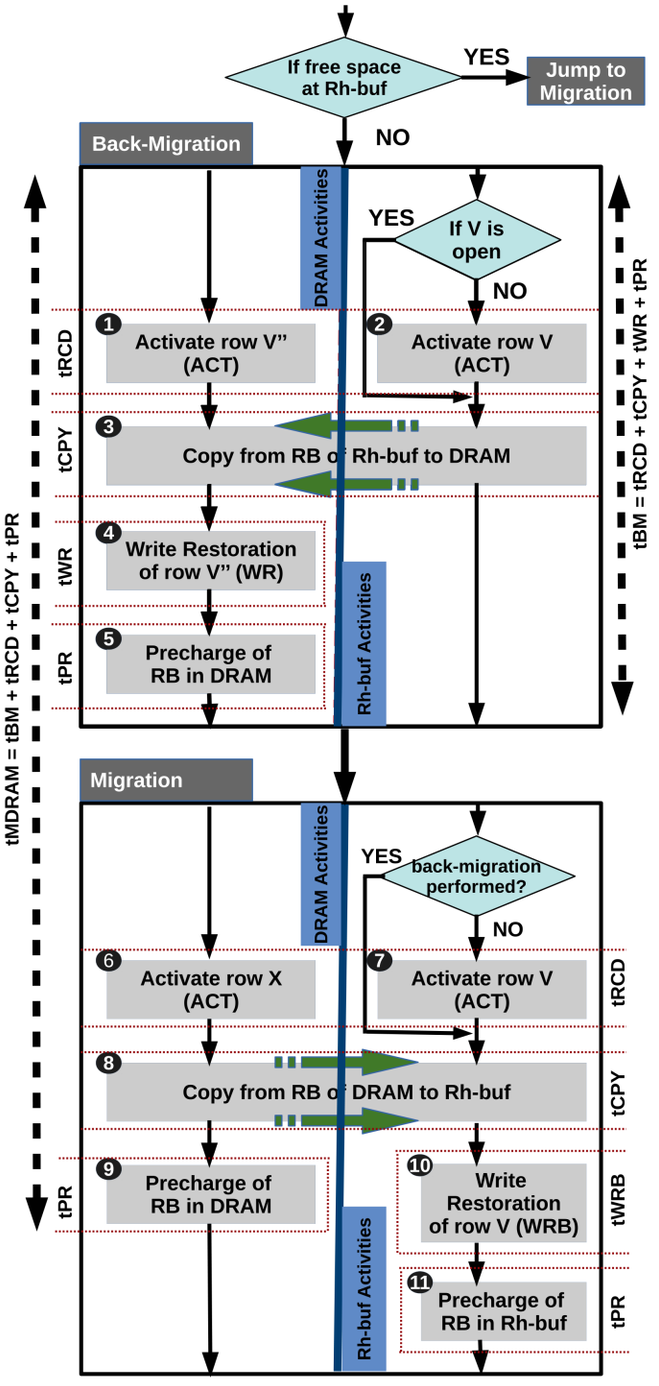


Fig. 3. Row migration.

The left part shows the actions performed on the DRAM bank and the right part shows the actions of Rh-buf. All the internal rectangles marked with a number in the upper left corner represent an action. Any two actions placed horizontally linear means they can be executed in parallel.

To migrate, say, a row X from DRAM to Rh-buf, the Rh-buf must have a free space to allocate X. If there is no free space in Rh-buf, then the Back-Migration module first back-migrates a row V from Rh-buf to the DRAM bank via a ROW-MGRX command. RHLM can accommodate several types of eviction policies. For this demonstration, assume that the **least recently used row (LRU)** is the row to be evicted. The row V is the LRU row of the Rh-buf. Consider the corresponding row of V in the DRAM bank to be V''. To back-migrate V, both V (in Rh-buf) and V'' (in DRAM bank)

need to be activated. This action copies the content of V at the row buffer, RB-buf, and the row V'' (in DRAM) is open for write. After completing actions ❶ and ❷, the data from the RB-buf is copied from RB-buf to RB-bank using the write driver (Action ❸). After the copy, the DRAM needs some time (tWR) to restore the data into the row V'' (Action ❹). In the end, a precharge is required in DRAM, as shown in action ❺. Note that the precharge is not required at Rh-buf to make the row V open to overwrite with the content of the migrated row X.

The module "Migration" in Figure 3 shows the actions required for migrating row X from DRAM to Rh-buf. An activation is required in DRAM (action ❻) to copy the content of row X to the row buffer RB-bank. At the same time, an activation is required in Rh-buf if back-migration is not performed (action ❼). After executing these two actions in parallel, the content of RB-bank is copied into RB-buf using the action ❽. The row V, on which the migrated data needs to be stored in Rh-buf, is already opened. Restoring data to the row of Rh-buf needs some time (action ❿), which can be performed in parallel with the precharge of DRAM bank (action ❾). Finally, a precharge operation is required at Rh-buf to complete the row migration. After this migration, the corresponding entry in the Rh-Pointer needs to be updated.

Time required to migrate: To measure the total time required for row migration, we have used some additional timing constraints in addition to the existing timing constraints of DRAM. The additional timing constraints are tCPY, tWRB, tMM, tMDRAM, and so on. All these parameters are discussed in this section. Actions ❶ and ❷ execute in parallel and take tRCD time. Action ❸ needs an additional tCPY time. We assume $tCPY \leq tBL$. The proposed RHLM architecture has connected bit lines between both the buffers. Each bit line can copy bits equivalent to the column width. There is one bit line after each column width totaling up to 1,024 bit lines. A normal column access requires tBL time. Since we are performing copy operation in parallel, the tCPY time will not exceed tBL time. Actions ❹ and ❺ need tWR and tPR times, respectively. Hence, the total time required for back-migration is $tBM = tRDC + tCPY + tWR + tPR$. In the migration module the time required for actions ❻ and ❼ is tRCD. Action ❽ also needs tCPY time. The time required for action ❿ is the write restoration time at Rh-buf. We consider it as tWRB. Actions ❾ and ❿ start concurrently but ❾ finishes earlier. Action ⓫ needs tPR times. The total blockage time for DRAM and Rh-buf is different. Also, the blockage time is contingent upon the execution of the back-Migration module. Hence, The maximum blockage time for DRAM is $tMDRAM = tBM + tRCD + tCPY + tPR$, which is less than the time of two write cycles (tWC). The blockage time is only $tRCD + tCPY + tPR < tWC$ when the back-migration is not required. The blockage time of Rh-buf is slightly more than DRAM because of the longer write retention time of STTMRAM.

Assigning the usual values for a typical DDR4 DRAM [27] device and STTMRAM device [31, 39], we get tMDRAM = 27 ns, provided that back-migration is not required. However, if there is a requirement of back-migration, then this value goes up to 87 ns. In either case, the total time required for our method is still less than the best case of the current RH mitigating techniques, i.e., RH-Refresh, which consumes at least 96 ns when the victim row count is 2 (minimum). Hence, for either of the cases of row-migration operation, the blockage time of DRAM is less than the blockage time required to refresh two victim rows (less in absence of back-migration). However, the parallel execution, as discussed in Section 4.2.3, hides this overhead to some extent. Also, for consecutive refresh period attacks (cf. Section 1.1), the row migration is only required during the first refresh period. It can also be noted that because of the increasing density of DRAM the number of victim rows per RH attack is increasing. As mentioned in Section 3, the maximum number of victim rows per RH attack can be up to 14. Hence, the proposed RHLM improves the overall latency

even after an additional latency of row migration. Section 6 shows the experimental analysis of the additional DRAM latency required for row migration over the total reduction of the DRAM blockage time using RHLM.

4.2.3 How to Handle Write Operations. Table 1 shows the READ/WRITE latencies of DRAM and STTMRAM. During simulations, we verified this, as STTMRAM READs are usually faster than DRAM devices, whereas WRITEs consume nearly equivalent time in STTMRAM and DRAM, with instances of the former being slower than the latter by a few clock cycles at times. Hence, the write time in the STTMRAM cell may be slower than the DRAM cell. Extra care shall be taken to hide this additional time requirement. As mentioned in Section 4.1.2, a column write can be performed either in DRAM bank or Rh-buf, depending on the flag bit B (the flag bit is set during the ACT command required to execute before column write). Since the writing in Rh-buf can be performed in parallel with other DRAM operations, the DRAM is not blocked for Rh-buf writes. Though there may be instances where accessing data in Rh-buf may take a longer time than DRAM, the Rh-buf is mostly accessed by the malicious applications. In Section 4.1.3, we have already mentioned that an RH attack is isolated with all of its active aggressor rows migrated to the Rh-buf. As a consequence of the same, innocent applications will have less blocking time on the DRAM device, and the only program that will be affected is the RH attack. This is evident in Section 6.1.1. Figure 10 shows the distribution Rh-buf access by the malicious applications vs. the innocent applications.

Another problem of STTMRAM devices pertains to the fact that these devices have a lower endurance than DRAM devices [3]. But we still have considered STTMRAM for designing the Rh-buf, because the instances of row hammer attack are limited throughout the lifetime of the main memory. Most of the innocent applications will be served by the DRAM and only the aggressor rows of the attacker application will run on Rh-buf. Since, in the long run, the number of accesses on Rh-buf will be much lesser than DRAM, using STTMRAM to design Rh-buf may not be an issue. Even in worst-case scenario, existing endurance enhancement techniques like Reference [35] can be used.

4.2.4 RH Attack on Rh-buf. As mentioned in Section 2.2, Khan et al. [15] have shown that RH attacks can also be carried out on STTMRAM. The authors further reported that the worst-case probability of a read disturbance is 2.9×10^{-7}. As the probability suggests, the occurrence of such errors in the STTMRAM is rare, in comparison to the DRAM device. In previous work by Jang et al. [11], the authors discussed several forms of attack on STTMRAM and its subsequent mitigation techniques. Given the rare occurrence of RH-directed bit flips in the STTMRAM, we believe that stronger on-device **Error Correcting Code (ECC)**, previously proposed by Jang et al. [11], will be sufficient to efficiently mitigate such attacks on the STTMRAM.

4.3 Implication of RHLM

Kim et al. [18] first demonstrated RH attack via the assembly code segment depicted in Code Listing 2. Here, the authors read memory addresses (MEM_ADDR_1 and MEM_ADDR_2) of the same bank and ensure that the request falls into two different rows and also into the memory subsystem every time by flushing the cache. Unlike issuing RH-Refreshing to the adjacent rows of both *MEM_ADDR_1* and *MEM_ADDR_2* in the code listing, RHLM migrates both *MEM_ADDR_1* and *MEM_ADDR_2* from DRAM memory to a location, say, *MEM_ADDR′* and *MEM_ADDR′′* in the STTMRAM. This ensures no further blocking time post-migration, which was previously required every time RH detection detects the attack. Note that there are two aggressor rows in this attack. Since RHLM migrates both of these aggressor rows, Code Listing 2 is rendered null and void for subsequent iterations. RH detection mechanisms cannot differentiate between attack types. Hence, we ensure RHLM, as an RH mitigation technique, to differentiate and mitigate RH attacks.

Listing 2. Demonstration of a Typical RH Attack [18]

```
loop1:
 mov (MEM_ADDR_1), %eax
 mov (MEM_ADDR_2), %ebx
 cflush (MEM_ADDR_1)
 cflush (MEM_ADDR_2)
 mfence
 j loop1
```

The implication of energy usage is previously mentioned in Section 3.2. In conventional systems with RH-Refreshing as the mitigation solution, Code Listing 2 ensures both *MEM_ADDR_1* and *MEM_ADDR_2* are read from the DRAM memory at each iteration. This compels the DRAM device to operate at all times. As RHLM is based on the migration of the aggressor row, the DRAM device can be clearly transitioned into low-power mode. Note that the subsequent request will be served from the STTMRAM, which will consume energy as well. In Section 6.2, we show that a combination of both the device's energy consumption still yields energy savings.

4.4 Hardware Overheads

We have already discussed the requirements of Rh-Pointer in Section 4.2.1. There are usually 8 banks in a DDR4 DRAM device. Typical row size is 8 KB. Since each row of the DRAM bank and Rh-buf are equal in size, therefore there would be $\frac{8\times128\times8}{1024} = 8.0$ MB of additional storage required in our proposed design. Compared with a DRAM device of size 2 GB (1 rank, 1 chip, 8 banks), the storage overhead of our proposed design including both the table and the pointer is 0.39%, which is negligible. According to Chi et al. [6], the maximum ratio between an STTMRAM cell to a DRAM cell is 8.3. Therefore, our proposed model would pose an approximate area overhead of 3.23%. In addition, considering the fact that Rh-buf would have to replace rows when filled, it, therefore, possesses some additional overhead on the overall design. We have considered several cache replacement policies for RHLM, which includes **pseudo least recently used (pLRU)**, **most recently used (MRU)**, and **counter-based row replacement (CBR)** policy.

Addresses in a DRAM device are decoded via the Address decoder that performs shift operations on a given address based on a masking value to identify its corresponding channel, rank, chip, bank, row, and column. Each of these steps requires 1 clock cycle with a total of 6 clock cycles. We use a dual-edged flip-flop [45] where both the rising and the falling edge are used as triggers.

The table index uses a combination of DRAM rank, bank, and row values. As we already have their values decoded, we append bank and row on the rank value and create a key for the table. This process requires 2 clock cycles. We then retrieve one entry from the table in 1 clock cycle using the key generated previously. Then, to decode the retrieved address with masking bits of 6 (corresponding Rh-buf row), 1 (flag bit), and 1 (dirty bit), we require additional 3 clock cycles. In total, 12 clock cycles are used, which are then compressed into 6 clock cycles via the use of dual-edged flip-flops. In an overall scenario, the lookup time never lies outside the critical path of the DRAM device. Updating the Rh-Pointer can be done in parallel to the working of both the memory devices.

5 EVALUATION

We have considered several sets of distinct workloads based on PARSEC benchmark suite [2] to evaluate our proposed method. The base sets include normally executed PARSEC benchmarks (W1) and a set of synthetic RH workloads (W6). There are eight benchmarks selected from the PARSEC benchmark suite. Note that, ideally, W1 should not demonstrate any RH attack, as these are

Table 2. Description of Workloads

Idx	Benchmark	Desc.	Desc.	Desc.	Desc.	Idx	Trace Desc.
B1	blackscholes	B1 + S1	B1 + S3	B1 + S4	B1 + S5	**S1**	Single Row Attack (Closed Page Attack)
B2	bodytrack	B2 + S1	B2 + S3	B2 + S4	B2 + S5	**S2**	Double Row Attack (Opened Page Attack)
B3	canneal	B3 + S1	B3 + S3	B3 + S4	B3 + S5	**S3**	Limiting the Migration Table (Random RH)
B4	facesim	B4 + S1	B4 + S3	B4 + S4	B4 + S5	**S4**	Mixed RH Attack I
B5	ferret	B5 + S1	B5 + S3	B5 + S4	B5 + S5	**S5**	Mixed-based RH Attack II
B6	fluidanimate	B6 + S1	B6 + S3	B6 + S4	B6 + S5		
B7	freqmine	B7 + S1	B7 + S3	B7 + S4	B7 + S5		
B8	vips	B8 + S1	B8 + S3	B8 + S4	B8 + S5		
W1		**W2**	**W3**	**W4**	**W5**	**W6**	

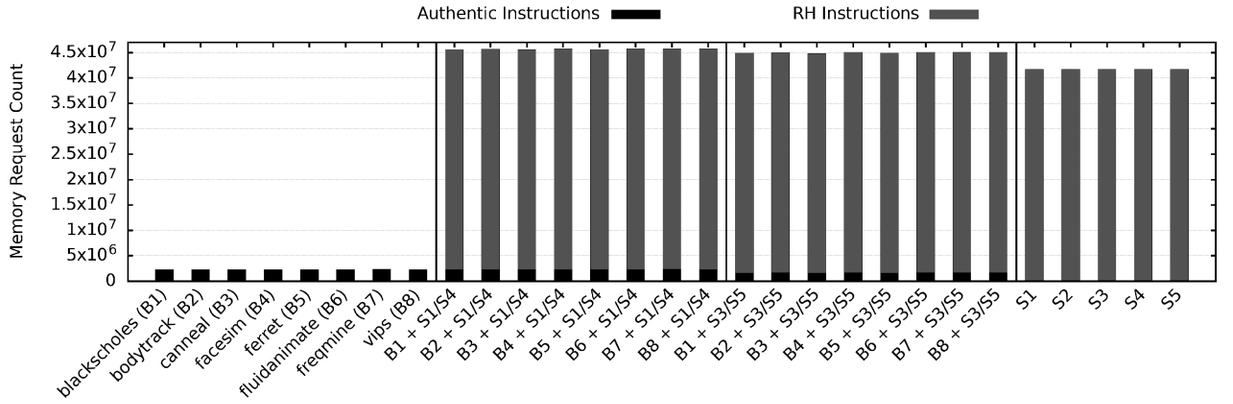


Fig. 4. READ/WRITE request count for all the workloads. On the left, the request count of PARSEC benchmarks is shown, which is followed by the request count of W2/W4. The second-to-last segment shows READ-/WRITE request for W3/W5, which is finally followed by the five synthetic traces.

genuine benchmarks. Since we are demonstrating RH attacks detected in a deterministic manner, we mix W1 with synthetic traces (W6) and create four distinct test sets. The test sets or workloads are W2, W3, W4, and W5. These workloads also demonstrate how a typical RH attack is carried out. W2 is a workload set created by mixing W1 and S1 from W6. S1 represents a read-based RH attack where there is only one aggressor row. The address is first written to and then repeatedly read from. W3 is created by combining W1 with space limiting RH attack trace S3 from W6. W3 demonstrates the bottleneck effect of rapid migrations. S3 contains 160 random addresses hammered repeatedly and forces RHLM to migrate rows to the Rh-buf constantly. This thus allows the least possible Rh-buf hits. It is ensured that there is a gap of at least 15 rows between two aggressor rows in the same trace. All these 160 addresses are first written in the memory and then are read repeatedly, similar to S1. Workloads W4 and W5 are created with S4 and S5, respectively. These contain a fair proportion of writes alongside reads. The number of memory accesses in a memory trace remains constant both in its primary and its respective extended workload, i.e., W2 and W4, and W3 and W5. A collective description of all the workloads are depicted in Table 2 and in Figure 4. The traces are primarily executed on a modified version of cycle-accurate DRAM simulator called DRAMSim2 [33]. All statistics pertaining to the DRAM device are gathered via DRAMSim2. The timing and power analysis and, in addition, the verification of the STTMRAM, are done using NVmain [31] and Ramulator [19], respectively. To demonstrate RH attacks more efficiently, we use closed page policy in the DRAM simulator.

5.1 Experiment Details

The deterministic counter-based RH detection technique with RH-Refreshing-based mitigation is considered as the baseline design for this experiment. As mentioned previously, RH-Refresh-based mitigation is considered the state-of-the-art RH mitigation technique. Based on the number of victim rows per RH attack, the baseline is further divided into two cases: (a) **Base-VRC2**: where the victim row count is 2, and (b) **Base-VRC14:** where the victim row count is 14 (worst-case condition). The deterministic RH detection mechanism is used in the baseline to provide an accurate number of RH counts. As mentioned in Section 3.1, such detection mechanisms require a separate counter for each row. The hardware overhead of the detection mechanism is ignored in experiments, as it is not a part of RHLM. The proposed RHLM technique can be used with any RH detection mechanism, as it works on the mitigation part of RH-attack. We initially assume the fact that if RHLM can provide better results with the ideal detection mechanism, then it can provide better results with all other detection mechanisms [18, 24, 38]. We verify this claim in Section 6.3. Blocking time imposed in 2x refreshing, previously mentioned in Section 3.2, is also evaluated during experimentation. In addition, we have also considered other recent row hammer mitigation techniques, including CROW [10] and remapping-based mitigation technique [17]. CROW is mentioned as CROW-x, where x is the number of victim rows migrated from the DRAM array to the CROW-cache. RHLM is analyzed using several row replacement policies on the Rh-buf toward finding the most efficient version of the same. These replacement policies are previously discussed in Section 4.4. These include *pLRU*, *MRU*, and *CBR*. Henceforth, the replacement technique used while evaluating RHLM is denoted as RHLM-t, where t is the replacement policy used for Rh-buf.

The evaluation platform DRAMSim2 simulator is modified such that respective traces for Ramulator and NVmain are generated dynamically. Timing is kept consistent to avoid any breaking of dependencies among memory requests. Generated traces for Ramulator and NVmain are executed in a normal manner. This outputs the extra time required for simulating the STTMRAM and its power consumption, respectively. We report the final results as a combination of all the values. Our motive lies in analyzing both timing and energy consumption of the memory system. Hence, we highlight each of them in the following subsections.

5.1.1 Timing Analysis. We perform all the experiments as described above. We use the initial values obtained via DRAMSim2 to define our baseline cases. Baseline values include the ideal running time of the simulation and the extra time for mitigating RH attacks. In the case of RHLM, we combine the simulation time of both the DRAM and the STTMRAM (Rh-buf). Since these devices operate in parallel, we consider the worst-case runtime as the total runtime of time of the slower device. Though both DRAM and Rh-buf can execute in parallel, the type of benchmarks used for this experiment makes idle intervals in DRAM after shifting aggressive activation to Rh-buf (cf. Section 4.1.3). Hence, the extent of parallelization is limited, but because of migrating the aggressor rows to Rh-buf, the DRAM can be transitioned into low-power mode to save energy. The details about energy saving are discussed next. For the existing techniques, the time required for refreshing victim rows in the DRAM device is considered as the blocking time. In the case of RHLM, we consider the migration time as the equivalent blocking time of the system.

5.1.2 Energy Consumption Analysis. We simulate all the experiments in a DRAM device that is capable of transitioning into low-power mode. DRAMSim2 uses a heuristics-based approach to transition DRAM ranks into low-power mode if the command queue is empty for a long period of time. As discussed in Section 2.1.1, a lower current value is consumed during low-power mode. This helps in reducing the total energy consumption of the DRAM device. Both the baseline as well as the proposed models are simulated in low-power mode. However, the baseline model is

incapable of fully utilizing this feature because of the aggressive activation of RH attacks. The proposed model's energy is the combination of both DRAM energy as well as STTMRAM energy.

6 EXPERIMENTAL ANALYSIS

In this section, we report our findings in terms of time and energy consumption of the memory sub-system having RHLM. Timing analysis is further divided into latency time improvement, analysis on the Rh-buf, and analysis on overall throughput of the system.

6.1 Timing Analysis of the Memory System

6.1.1 Latency of the Memory System. In our experiments, latency is measured in terms of total blocking time imposed on the DRAM sub-system because of RH mitigation. The basic motivation of RHLM is to minimize the blocking time of the DRAM while mitigating RH attacks. Here, we verify our claim via simulating RHLM on the workloads W2, W3, W4, and W5, defined in Table 2. Figure 5 and Figure 6 compare the amount of time the DRAM device is blocked for mitigating RH attacks in various techniques. The blocking time for mitigating RH attacks in RHLM is determined by the number of migrations, both forward and backward, the memory sub-system makes. Figure 5 shows results for the set of primary workloads W2 and W3. Figure 6 shows the result for the set of extended workloads W4 and W5. In both the figures, the techniques are organized in ascending order of blocking time for workloads W3 and W5. The order is *RHLM-CBR, RHLM-MRU, RHLM-pLRU, VRC-2, CROW-2, VRC-14, CROW-14, remapping-based,* and *2x refresh.* All the techniques used here for analysis are already discussed in Section 5.1.

CROW works on the principle of migration of rows, similar to RHLM. However, unlike RHLM, CROW migrates victim rows to its subarray. One such copy instruction takes 18% more tRAS time than its nominal counterpart. However, if an address is present in both the array and in the CROW subarray simultaneously, then there is a reduction in tRAS time by 7%. However, when CROW mitigates RH attacks, it moves victim rows from the DRAM array to the CROW-cache instead of copying it to maintain the integrity of the victim rows. In the context of mitigation of row hammer attacks, CROW requires 8 rows per DRAM subarray. Each subarray contains 512 rows, and one DRAM bank typically contains 64 subarrays. This totals to an additional 512 rows per bank. In remapping-based RH mitigation technique [17], each incoming address is remapped for the ECC to be able to correct row hammer–oriented errors. We have considered the additional time in the *critical path* while decoding an address, due to the addition of local and global decoders while evaluating.

In the case of W2 and W4, the total blocking time of RHLM, irrespective of the replacement policy, on the DRAM device is nearly non-existent, since there is only one row that is being hammered over consecutive refreshing windows. Once this row is migrated to the Rh-buf, there is no additional blocking time imposed on the DRAM device. Though all the RHLM variants show improvement over the existing technique, RHLM-CBR performs the best. A detailed comparison of the various RHLM variants is discussed in Section 6.1.2. In this section, we have discussed the performance of RHLM-CBR over the other existing techniques. Compared to Base-VRC2, RHLM successfully reduces the total blocking time of the system by 99.97%. This also suggests that our model deviates from an ideal DRAM device by a factor of 1.2×10^{-6}%. For the Base-VRC14 case, the blocking time reduction further increases by up to 99.99%. Hence, in RHLM, the threat of recurrent single-row attacks across consecutive refresh periods is removed. In comparison with CROW-2, which is limited to two victim rows migration, RHLM, irrespective of any replacement policy, performs 74.97% better on average across all memory traces in workloads W2 and W4. However, in the CROW-14 victim rows case, RHLM minimizes blocking time by 94.42% on average. For

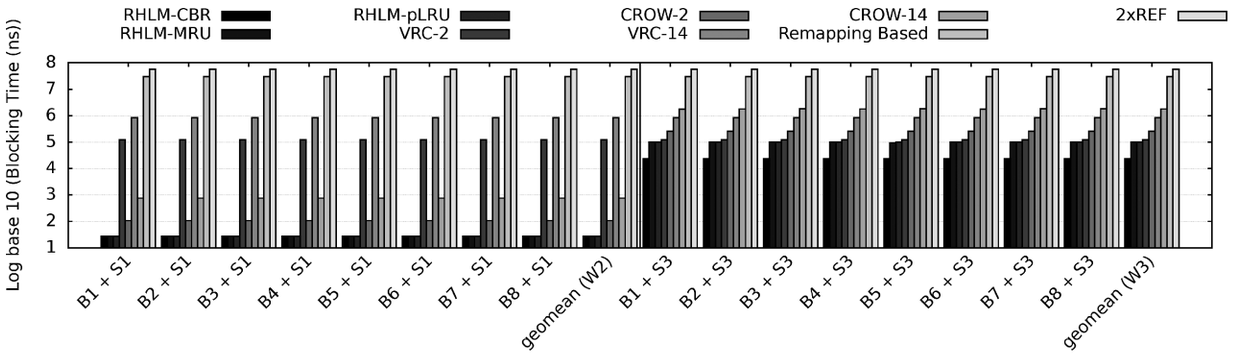


Fig. 5. Blocking time on the DRAM sub-system for the primary set of workloads.

the remaining mitigation techniques, i.e., remapping-based mitigation and 2x refresh, RHLM saves more than 99.99% of blocking time for workloads W2 and W4.

W3 and W5 represent the worst-case scenario for RHLM, in general, using an ideal RH detection mechanism. This is because RH attacks on the device are done with random addresses ensuring complete saturation of RH-pointer. This causes rows to migrate frequently rows to and from the Rh-buf. Note that for the baseline case, the blocking time remains the same with W2. This is because the same number of RH attacks are being hammered (1,250). Here, only the addresses are different to initiate constant migrations. Even in this case, All the variants of RHLM perform better than all other existing RH mitigation techniques. RHLM-CBR performs the best by saving 80.22% of the blocking time on average compared to the Base-VRC2 case. Further, in the case of Base-VRC14, RHLM-CBR saves 97.17% blocking time on average. In comparison with CROW-14, RHLM-CBR minimizes 89.67% of blocking time. Finally, for remapping-based mitigation and 2x refresh cases, RHLM saves blocking time in either of the workloads by more than 99.9% and 99.92%, respectively.

The motivation behind using a mixed RH attack, with a fair proportion of writes, is to study the additional time required to complete the RH program. After migration of active aggressor rows, genuine programs in all mixed traces, W4 and W5 exclusively get the DRAM device to perform its read and write operations. Hence, the blocking time for the genuine applications is minimized and is equal to the amount of blocking time savings as reported for W2 and W3, respectively. The synthetic traces S4 and S5, mixed within W4 and W5, respectively, however, is executed on the Rh-buf during the majority of the execution. Writes in an STTMRAM cell is expensive in terms of both time and energy, which is evident from previous works by Boukhobza et al. [3], Mittal et al. [28], Chi et al. [6], and Asifuzzaman et al. [1]. From Table 1, we can see that the write latency of STTMRAM is worse than its DRAM counterpart. However, total energy consumption is more efficient in RHLM due to the transitioning of the DRAM device into low-power mode, which hides the high energy requirements of writing into an STTMRAM device. Also, considering the fact that RHLM can segregate an RH attack program, therefore, only a write-based RH program is likely to take more time in RHLM while allowing the normal applications to be executed in the DRAM device with minimal blocking time.

6.1.2 Analysis on Blocking Time while Using Different Row Replacement Policies in RHLM. As mentioned in the previous section, all variants of RHLM yield less blocking time than all other existing RH mitigation techniques. This can be seen in workloads W3 and W5, where there are constant migrations to and from the Rh-buf. This is clearly evident in Figure 5 and Figure 6. Statistics show that RHLM-CBR performs the best in the lot. RHLM-CBR minimizes 75.85% blocking time on average than RHLM-MRU. Furthermore, in comparison with RHLM-pLRU, RHLM-CBR minimizes 76% of blocking time on average. This is due to less number of forward and backward

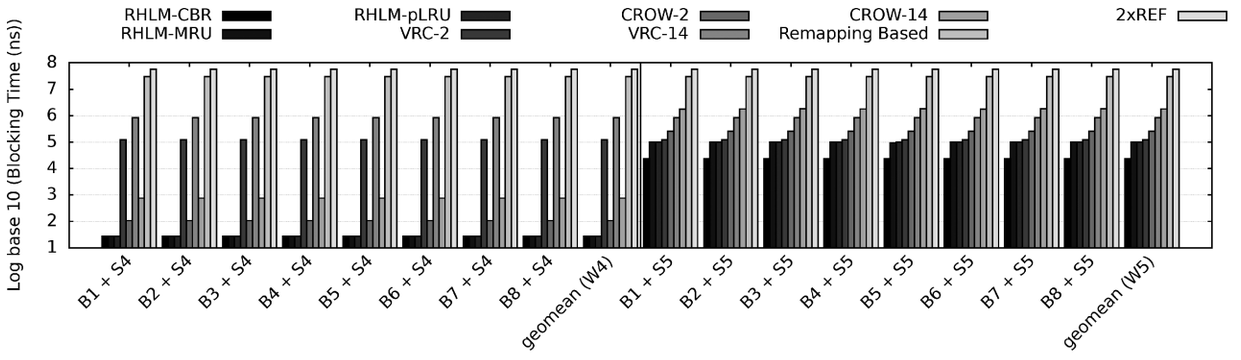


Fig. 6. Blocking time on the DRAM sub-system for the extended set of workloads.

migration count. Further, in Section 6.1.3, we have highlighted the number of migrations encountered in each of these variants.

RHLM-pLRU, which performs the worst among all other RHLM variants, still shows a positive gain on blocking time savings over VRC-2 case. On average, RHLM-pLRU minimizes 17.6% of blocking time on the DRAM device over VRC-2 case. The most hardware-efficient replacement policy, i.e., RHLM-MRU, manages to minimizes 18.08% of blocking time on average.

6.1.3 Analysis on Rh-buf. When average timings are considered, typical read access of STTM-RAM requires fairly similar time as compared to a DRAM device, while writes require more time in Rh-buf than DRAM. This device operates in parallel and serves addresses simultaneously to the DRAM device. Moreover, in the following sections, we show improvements in terms of both throughput and energy due to the inclusion of Rh-buf in the device.

RH accesses constitute a large number of memory accesses in a given mixed workload (Figure 4). Hence, a row, if migrated to the Rh-buf, enables the same to be further read or written from the Rh-buf. The DRAM device can now serve other requests. In Section 4.4, we mentioned that RHLM can accommodate several eviction policies. For Rh-buf hit evaluation, we have considered **pseudo least recently used (pLRU)**, **most recently used (MRU)**, and **counter-based replacement (CBR)** policy. In addition, we have also compared the aforementioned eviction policies for *CROW-cache* hits of CROW. Figure 7 plots the same where additional requests served in the case of W2. Here, none of the row replacement policies are invoked, as there is only one aggressor row being hammered every time. Rh-buf provides hit rates of about 95.38% in the case of W2, irrespective of the replacement policy. Figure 8 shows Rh-buf hits for W3. W3 forces migration in all of the aforementioned replacement policies, as the number of unique aggressor rows is 160, greater than the size of the Rh-buf. From the figure, we can see that MRU and CBR policies perform near equivalent in terms of RhH-buf hits followed by p-LRU. The reason why recency-friendly replacement policies have a better outcome is due to the structure of the RH traces. In Section 5, we have explained the structure of these RH traces. Overall, p-LRU achieves an Rh-buf hit rate of 32.45%, MRU achieves 67.2%, and CBR achieves 67.22%. In comparison to similar migration-based RH mitigation works like CROW, our model performs better in terms of Rh-buf hits primarily due to the fact that RHLM is based on the principle of migrating the aggressor row. CROW migrates victim rows to avoid bit flips in the DRAM array. W3/W5, when evaluated for CROW, averages 5,515.75 CROW-cache hits, whereas RHLM, used with CBR, averages about 29,133,217 Rh-buf hits.

Alongside Rh-buf hits, we also have to consider the number of migrations, both forward and backward, for each of these replacement policies. While workloads W2 and W4 have only one forward migration, the same is not true for workloads W3 and W5, which limit the Rh-buf. Figure 9 shows the number of migrations encountered in different replacement policies for workloads W3

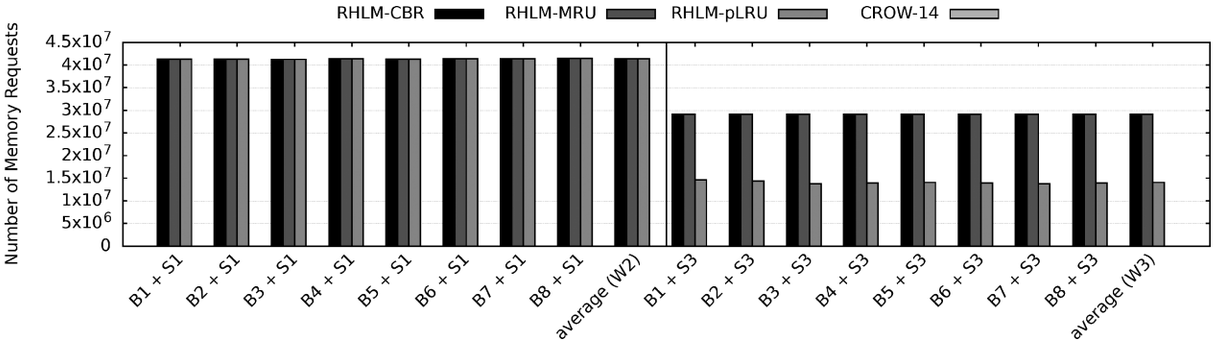


Fig. 7. Rh-buf hits/CROW-cache for the primary set of workloads.

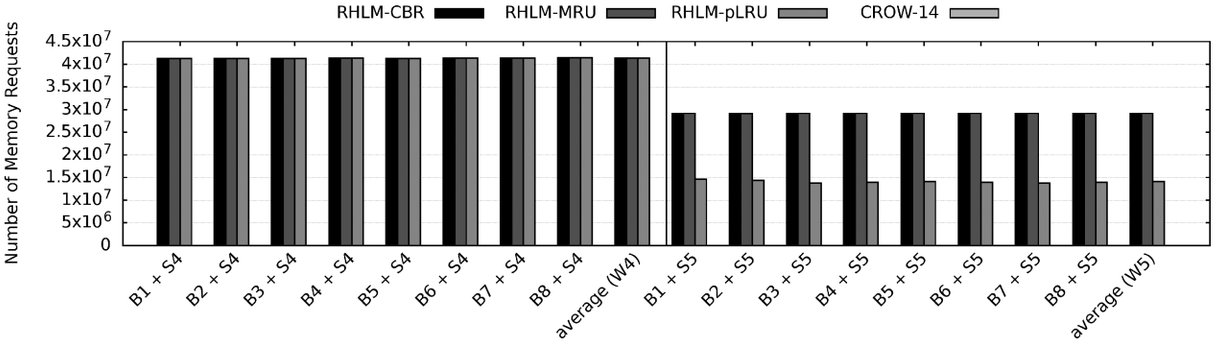


Fig. 8. Rh-buf hits/CROW-cache for the extended set of workloads.

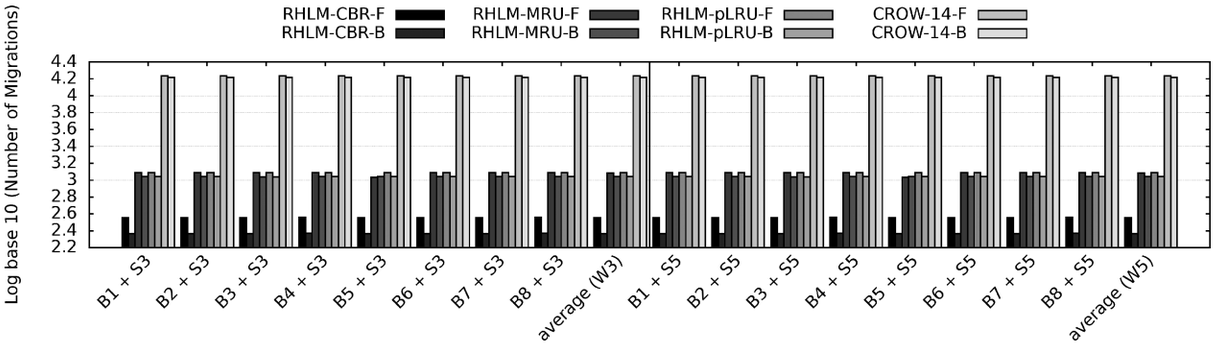
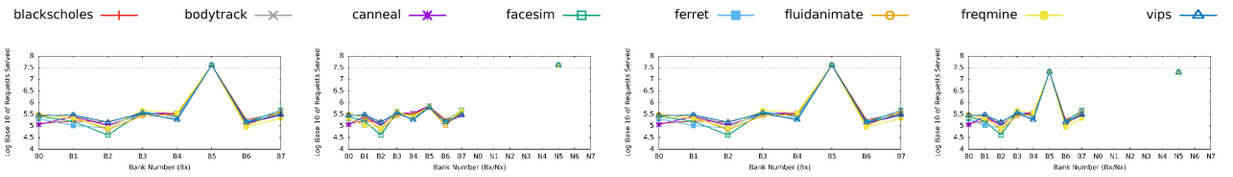


Fig. 9. Number of migrations (including **F**orward and **B**ackward migration count).

and W5. A tag *F* is used for an index represents forward migration count; i.e., DRAM to Rh-buf, and a tag *B* similarly represents a backward migration. CBR policy is the most efficient replacement policy in the context of the RH workloads selected. Its implication can be directly seen in the blocking time encountered by the DRAM device in Figure 5 and Figure 6. On average, there are a total of 594 migrations for CBR policy. MRU and pLRU average at 2,301.75 and 2,319.3 migration count, respectively. CROW, which migrates victim rows, has an average migration count of 33,693.5. Given the two-way benefit of CBR policy over other replacement policies with the expanse of higher area requirements, we select CBR policy as the standard replacement policy for RHLM.

Bank utilization for both baseline and proposed cases are depicted in Figure 10 for both W2/W4 and W3/W5. Banks for the DRAM device are denoted by B*i*, and its corresponding Rh-buf is denoted by N*i*. DRAM banks are operated independently (Section 2), hence worst-case execution time for the DRAM device can be estimated with the slowest bank. We concentrate all RH attacks



(a) Request Distribution for the Baseline in W2/W4 (b) Request Distribution for RHLM in W2/W4 (c) Request Distribution for the Baseline in W3/W5 (d) Request Distribution for RHLM in W3/W5

Fig. 10. Memory request distribution among banks in W2/W4 and W3/W5.

only on bank 5 (B5) of the DRAM device. This causes the same to have a higher number of memory accesses than all other banks in order of 10^2. The average number of memory accesses served by the DRAM device, in this case, is 5,417,439.6. The baseline mitigation technique issues additional ACTIVATES to mitigate RH attacks. In the proposed method, we migrate the hammered row to the corresponding bank of the Rh-buf. This enables other requests to be served from the DRAM while RH rows are served from the Rh-buf. This reduces the load on the DRAM device to some extent, as the average memory accesses for the DRAM device drops down to 295,740.1. Since there is only one row that is being hammered in W2/W4, the load distribution after row migration results in a nominal condition in the DRAM device (Figure 10(b)). Likewise, for W3/W5, our proposed technique does help in achieving Rh-buf hits. However, the rate for the same is low. This is because of constant row migrations to and from the Rh-buf device. Figure 10(d) represents the same.

6.1.4 Improvement on Throughput. Since the Rh-buf acts as a DRAM cache, it is expected that RHLM provides throughput improvement for the memory sub-system. However, this would be limited due to the following reasons:

- Although there are two devices serving memory requests, the incoming time for given memory access remains constant.
- If a copy of the same memory request is present in both DRAM and Rh-buf, then priority is assigned to Rh-buf. This ensures (a) consistency and (b) idle period for the DRAM device.

The above points restricts throughput improvement. In the case of W2 (Figure 11), we see that seven out of eight benchmarks demonstrate a marginal increment in the overall throughput of the memory device. The memory system is now able to serve 6,927.125 additional requests on average for W2. In the case of B7 + S1 benchmark, most authentic requests fall into the same bank where addresses are being hammered (B5/N5). This inversely affects the overall runtime in the proposed model. Here, a deviation of −0.01% is observed from the baseline case, which remains the only case where our model fails to provide better results than the baseline case. For W3, however, this particular case is absent. This allows us to achieve improvement for all benchmarks, serving an additional 10,258.25 requests on average, with the highest additional request increment observed for B1 + S3.

For mixed workloads W4 and W5, we gain additional memory request counts averaging at 42,813.04 and 186,982.31, respectively. Figure 12 represents additional requests served by the memory device for the set of extended workloads. The highest increment in throughput for W4 is observed for B3 + S4 trace, averaging at 52,185 additional memory accesses. However, B2 + S5 trace gains the highest increment in terms of throughput for workload W5.

6.2 Energy Consumption

RHLM exploits existing energy-saving mechanisms present in present-day DRAM generations. All of our simulations are executed in low-power mode [27, 33, 34, 44], which is an inbuilt feature of

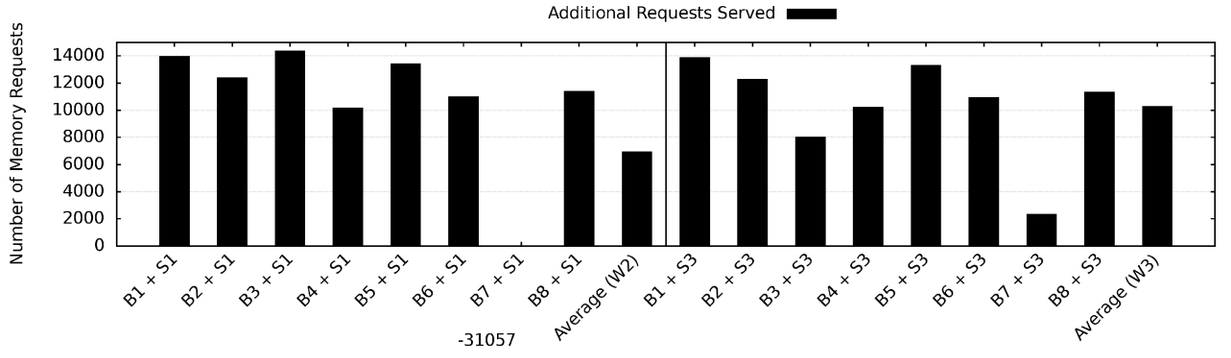


Fig. 11. Additional output of the memory sub-system for the primary set of workloads.

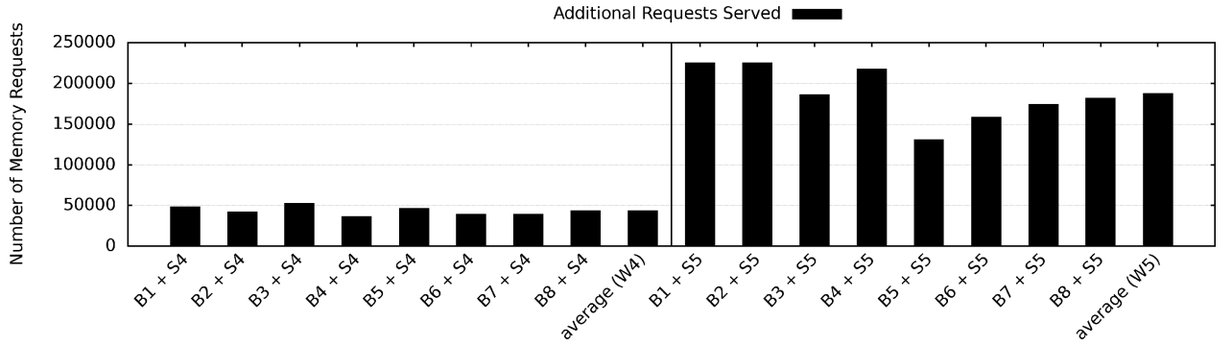


Fig. 12. Additional output of the memory sub-system for the extended set of workloads.

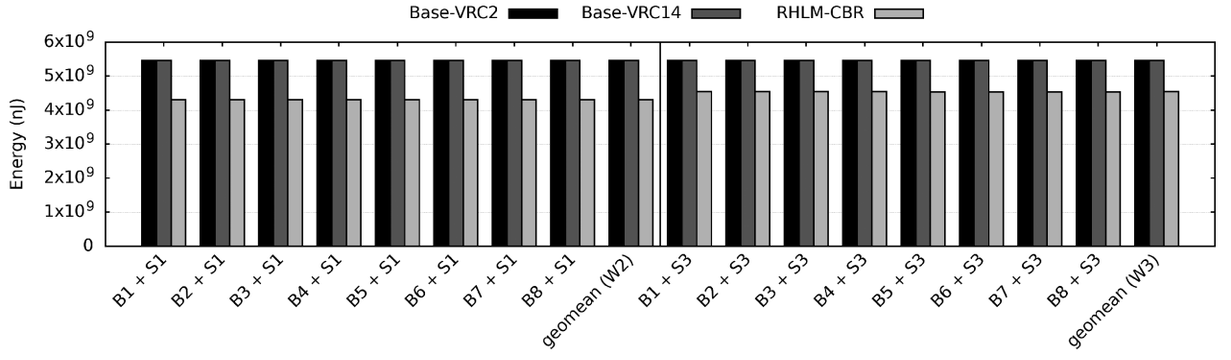


Fig. 13. Energy consumption of the memory sub-system for workloads W2 and W3.

all modern DRAM devices. If the command queue of the DRAM device is idle for a prolonged period of time, then the DRAM ranks are automatically transitioned into low-power mode. This is previously explained in Section 2.1.1. Figure 13 and Figure 14 represent the energy consumed during the simulation of both the DRAM and Rh-buf. Best- and worst-case baseline conditions (Base-VRCx) are depicted in the first two plots. It is followed by the total energy consumed in RHLM.

From our experiments, we have seen that background energy constitutes a significant portion of the total energy consumption of the DRAM device. If the device is transitioned into idle mode, then background energy reduces. Since RHLM generates such windows where transitioning is made possible, we obtain energy savings for the proposed model. Operations like coherency are absent in this model, which benefits the device's overall energy consumption at the cost of higher performance improvement. Nevertheless, we obtain both energy savings and performance improvement for the proposed model.

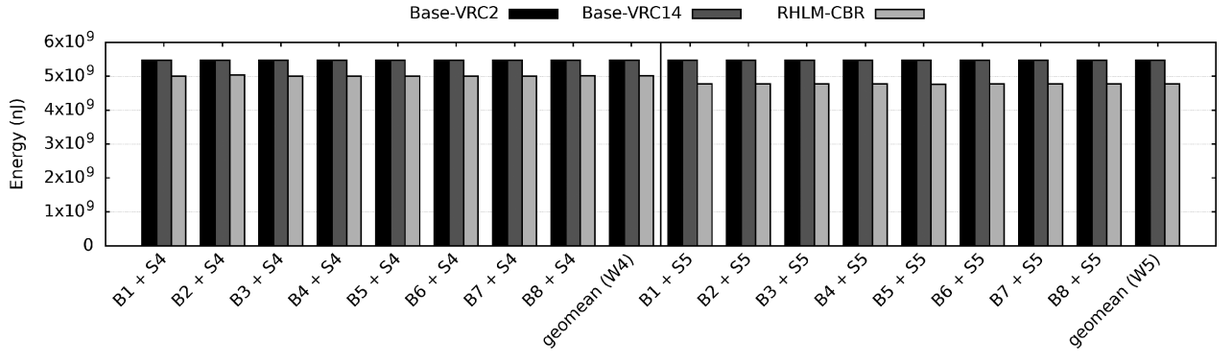


Fig. 14. Energy consumption of the memory sub-system for workloads W4 and W5.

For the set of W2, we achieve average power savings of 21.03% and 21.04% compared to Base-VRC2 and Base-VRC14, respectively. The small difference is because the extra number of ACTIVATES does not directly have a significant impact on total power consumption. For W3, the generated window for transitioning DRAM ranks is small. This restricts the total energy savings to 11.24% on average for both VRC cases for W3. Maximum power savings are obtained for the benchmark B8 + S1 in W2 with 21.09% of energy savings. In case of workload W3, B5 + S3 shows the maximum energy savings with 16.9%. However, for mixed traces W4 and W5, we have considered energy including the extra time the Rh-buf remains active. Reads are more energy-efficient in STTMRAM, and therefore, we achieve higher energy savings in the aforementioned cases. However, as W4 and W5 are mixed traces with a fair proportion of writes present, the energy savings scenario reverts. W4, which has only one aggressor row, repeatedly accesses the Rh-buf. In terms of energy associated with writes, this step is expensive. Hence, we only see an average of 8.47% energy savings for this particular workload. However, the number of Rh-buf accesses drastically drops in W5. Although, even after the period of low-power mode transition has shortened, due to the inherent characteristics of an STTMRAM device, this leads to a slightly higher energy savings of 12.74%. Overall, across all test workloads W2, W3, W4, and W5, RHLM, when using with CBR policy, achieves an average energy savings of 15.82%.

6.3 Performance of RHLM with Other RH Detection Techniques

We have also incorporated RHLM with other RH detection mechanisms including PARA [18] and TWICE [24] to evaluate on W3. PARA detects an RH attack with some probability that we have assumed to be 0.001. PARA-0.001 incurs a larger overhead for migration, as the RH attack count will be higher than the deterministic method, as it includes both true positive as well as a large number of false negatives. Nevertheless, RHLM is successfully able to save blocking time by 9.01% and 87% on average in Base-VRC2 and Base-VRC14 cases, respectively. It also acts as a caching device with Rh-buf hit rates of up to 99.3% and saves energy from the experiments with an average of 4.5%. For the latter case, i.e., TWICE, we achieve results along with all the parameters nearly equivalent to the deterministic case. While it is true that the amount of blocking time savings in the case of VRC2 in PARA-0.001 is low, the important point to note here is the fact that RHLM provides blocking time savings in this case as well. Implications of RH mitigation time is directly impacted by the efficiency of the Rh-detection technique used. Hence, RHLM, when combined with existing and popular Rh-detection mechanisms in the worst-case scenario (using workload W3), is still able to provide better results when compared to the widely accepted RH mitigation technique for all cases.

7 CONCLUSION

In this work, we proposed a novel RH mitigating technique that migrates aggressor rows from a DRAM device to an STTMRAM device. Our focus is primarily on the aggressor row; unlike other works, which largely focus on victim rows. The motivation behind it is to reduce the amount of time spent mitigating RH attacks. Our technique improves upon this blocking time on the DRAM device up to 99.97% in a general RH attack compared to the best-case scenario of the currently accepted RH mitigating technique where the victim row count is 2. Savings drop down to 80.22% in the worst-case scenario, i.e., workloads W3 and W5, when RHLM is evaluated using CBR policy. In either case, our method provides results better than the previously proposed solutions.

Transitioning the DRAM device into low-power mode requires the device to be idle for a period of time. Migration of aggressor rows makes the device idle for a brief period of time, as addresses to the memory in case of normal applications are unlikely to appear in quick succession. This is evident in the case of PARSEC benchmarks, which we have simulated. Our model improves upon the memory system's energy consumption by 15.82% on average across all different RH-based workloads, compared to the best case of the existing RH mitigation technique. The storage overhead is limited to 0.39% of a 2 GB DRAM device. Given the amount of savings obtained under blocking time savings, throughput increment, and energy savings, this figure seems marginal.

The concept of hybrid memories is evolving. It is predicted that soon hybrid memories will be used in any general computing scenario. The model that we have proposed is built on top of the concept of hybrid memories, as we require a small RH proof NVM device called Rh-buf. The ideal choice for us was STTMRAM due to its limited READ and WRITE latencies, which are comparable to a DRAM device. We also plan on investigating the possibilities of further reducing the latency of STTMRAM devices for a hybrid memory design. During our evaluations, we found that a particular aspect of RH mitigation is still overlooked, i.e., the interpretation of whether a given victim row holds any critical information or not. Although this would require a context-sensitive analysis of the contents of the DRAM device, the predicted benefits, in terms of both performance and power, would be immense. The reason is that in this particular case, we can simply omit RH-Refreshes of a large number of victim rows that are devoid of any sensitive data.

In the future, we plan to look into these aforementioned issues.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Kazi Asifuzzaman, Rommel Sánchez Verdejo, and Petar Radojković. 2017. Enabling a reliable STT-MRAM main memory simulation. In *Proceedings of the International Symposium on Memory Systems (MEMSYS'17)*. Association for Computing Machinery, New York, NY, 283–292.

[2] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT'08)*. Association for Computing Machinery, New York, NY, 72–81.

[3] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. 2017. Emerging NVM: A survey on architectural integration and research challenges. *ACM Trans. Des. Autom. Electron. Syst.* 23, 2 (Nov. 2017).

[4] S. Cha, O. Seongil, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. H. Ahn, and N. S. Kim. 2017. Defect analysis and cost-effective resilience architecture for future DRAM devices. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 61–72.

[5] E. Chen, D. Lottis, A. Driskill-Smith, D. Druist, V. Nikitin, S. Watts, X. Tang, and D. Apalkov. 2010. Non-volatile spin-transfer torque RAM (STT-RAM). In *Proceedings of the 68th Device Research Conference*. IEEE, 249–252.

[6] P. Chi, S. Li, Yuanqing Cheng, Yu Lu, S. H. Kang, and Y. Xie. 2016. Architecture design with STT-RAM: Opportunities and challenges. In *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC'16)*. 109–114.

[7] Y. Choi, I. Song, M. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y. Lee, J. Yoo, and G. Jeong. 2012. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *Proceedings of the IEEE International Solid-state Circuits Conference*. IEEE, 46–48.

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

[9] F. Hameed and J. Castrillon. 2019. A novel hybrid DRAM/STT-RAM last-level-cache architecture for performance, energy, and endurance enhancement. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.* 27, 10 (2019), 2375–2386. DOI: https://doi.org/10.1109/TVLSI.2019.2918385

[10] Hasan Hassan, Minesh Patel, Jeremie S. Kim, A. Giray Yaglikci, Nandita Vijaykumar, Nika Mansouri Ghiasi, Saugata Ghose, and Onur Mutlu. 2019. CROW: A low-cost substrate for improving dram performance, energy efficiency, and reliability. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*. Association for Computing Machinery, New York, NY, 129–142.

[11] Jae-Won Jang and Swaroop Ghosh. 2016. Performance impact of magnetic and thermal attack on STTRAM and low-overhead mitigation techniques. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'16)*. Association for Computing Machinery, New York, NY, 136–141. DOI: https://doi.org/10.1145/2934583.2934614

[12] JEDEC. 2014. *Wide I/O 2 (WideIO2)*. Technical Report JESD229-2.

[13] JEDEC. 2020. *High Bandwidth Memory (HBM) DRAM*. Technical Report JESD235C.

[14] I. Kang, E. Lee, and J. H. Ahn. 2020. CAT-TWO: Counter-based adaptive tree, time window optimized for DRAM row-hammer prevention. *IEEE Access* 8 (2020), 17366–17377.

[15] M. N. I. Khan and S. Ghosh. 2018. Analysis of row hammer attack on STTRAM. In *Proceedings of the IEEE 36th International Conference on Computer Design (ICCD'18)*. 75–82. DOI: https://doi.org/10.1109/ICCD.2018.00021

[16] Jeremie S. Kim, Minesh Patel, A. Giray Yaglikci, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. arXiv:cs.AR/2005.13121 (2020).

[17] M. Kim, J. Choi, H. Kim, and H. Lee. 2019. An effective DRAM address remapping for mitigating rowhammer errors. *IEEE Trans. Comput.* 68, 10 (2019), 1428–1441.

[18] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceeding of the 41st International Symposium on Computer Architecture (ISCA'14)*. IEEE Press, 361–372.

[19] Y. Kim, W. Yang, and O. Mutlu. 2016. Ramulator: A fast and extensible DRAM simulator. *IEEE Comput. Archit. Lett.* 15, 1 (2016), 45–49.

[20] M. P. Komalan, C. Tenllado, J. I. G. Pérez, F. T. Fernández, and F. Catthoor. 2015. System level exploration of a STT-MRAM based level 1 data-cache. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'15)*. 1311–1316.

[21] K. Kondo, K. Takahashi, and Morihiro Kada. 2015. *Three-dimensional Integration of Semiconductors: Processing, Materials, and Applications*.

[22] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. 256–267.

[23] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. *SIGARCH Comput. Archit. News* 37, 3 (June 2009), 2–13.

[24] Eojin Lee, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. 2019. TWiCe: Preventing row-hammering by exploiting time window counters. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*. Association for Computing Machinery, New York, NY, 385–396.

[25] S. Lee, H. Bahn, and S. H. Noh. 2011. Characterizing memory write references for efficient management of hybrid PCM and DRAM memory. In *Proceedings of the IEEE 19th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. 168–175. DOI: https://doi.org/10.1109/MASCOTS.2011.68

[26] Micron Technology. 2011. *1.35V DDR3L SDRAM SODIMM*. Technical Report MT16KTF51264HZ, MT16KTF1G64HZ.

[27] Micron Technology,. 2015. *DDR4 SDRAM*. Technical Report MT40A2G4, MT40A1G8, MT40A512M16.

[28] S. Mittal, J. S. Vetter, and D. Li. 2015. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *IEEE Trans. Parallel Distrib. Syst.* 26, 6 (2015), 1524–1537.

[29] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *Proceedings of the 5th IEEE International Memory Workshop (IMW'13)*. 21–25.

[30] David A. Patterson and John L. Hennessy. 1990. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

[31] M. Poremba and Y. Xie. 2012. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *Proceedings of the IEEE Computer Society Symposium on VLSI*. IEEE, 392–397.
[32] B. Pourshirazi and Z. Zhu. 2016. Refree: A refresh-free hybrid DRAM/PCM main memory system. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. IEEE, 566–575.
[33] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.* 10, 1 (2011), 16–19.
[34] Samsung Electronics. 2014. *DDR4 SDRAM*. Technical Report. Retrieved from: https://www.samsung.com/semiconductor/global.semi/file/resource/2017/11/DDR4_Device_Operations_Rev11_Oct_14-0.pdf.
[35] Puneet Saraf and Madhu Mutyam. 2019. Endurance enhancement of write-optimized STT-RAM caches. In *Proceedings of the International Symposium on Memory Systems (MEMSYS'19)*. Association for Computing Machinery, New York, NY, 101–113. DOI: https://doi.org/10.1145/3357526.3357538
[36] Hynix Semiconductor. 2009. *Datasheet for 1Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR*. Technical Report H5GQ1H24AFR.
[37] S. M. Seyedzadeh, A. K. Jones, and R. Melhem. 2017. Counter-based tree structure for row hammering mitigation in DRAM. *IEEE Comput. Archit. Lett.* 16, 1 (2017), 18–21. DOI: https://doi.org/10.1109/LCA.2016.2614497
[38] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Making DRAM stronger against row hammering. In *Proceedings of the 54th Design Automation Conference (DAC'17)*. Association for Computing Machinery, New York, NY.
[39] Everspin Technologies. 2018. *256K x 16 MRAM Memory*. Technical Report MR2A16A.
[40] Everspin Technologies. 2018. *EMD3D256M - 256Mb Spin-transfer Torque MRAM*. Technical Report EMD3D256M08BS1,EMD3D256M16BS1.
[41] MICRON Technologies. 2006. *DDR2 SDRAM* . Technical Report MT47H512M4,MT47H256M8,MT47H128M16.
[42] MICRON Technologies. 2017. *TN-ED-03: GDDR6: The Next-Generation Graphics DRAM* . Technical Report TN-ED-03: GDDR6.
[43] Micron Technology. 2005. *TN-46-12: Mobile DRAM Power-Saving Features and Power Calculations*. Technical Report TN46_12.
[44] Micron Technology. 2007. *TN-41-01: Calculating Memory System Power for DDR3*. Technical Report TN41_01DDR3.
[45] S. H. Unger. 1981. Double-edge-triggered flip-flops. *IEEE Trans. Comput.* C-30, 6 (1981), 447–451.
[46] Cong Xu, Dimin Niu, Xiaochun Zhu, Seung H. Kang, Matt Nowak, and Yuan Xie. 2011. Device-architecture co-optimization of STT-RAM based memory for low power embedded systems. In *Proceedings of the International Conference on Computer-aided Design (ICCAD'11)*. IEEE Press, 463–470.

Received July 2020; revised December 2020; accepted March 2021