MRLoc: Mitigating Row-hammering based on memory Locality

Jung Min You
Department of Electrical and Computer Engineering
Sungkyunkwan University
Samsung Electronics Co. Ltd
Suwon, Korea
yugura@g.skku.edu

Joon-Sung Yang
Department of Semiconductor System Engineering
Sungkyunkwan University
Suwon, Korea
js.yang@skku.edu

ABSTRACT

With the increasing integration of semiconductor design, many problems have emerged. Row-hammering is one of these problems. The row-hammering effect is a critical issue for reliable memory operation because it can cause some unexpected errors. Hence, it is necessary to address this problem. Mainly, there are two different methods to deal with the row-hammering problem. One is a counter based method, and the other is a probabilistic method. This paper proposes the improved version of the latter method and compares it with other probabilistic methods, PARA and PRoHIT. According to the evaluation results, comparing the proposed method with conventional ones, the proposed one has increased row-hammering reduction per refresh 1.82 and 7.78 times against PARA and PRoHIT in average, respectively.

CCS CONCEPTS

• **Security and privacy → Hardware attacks and countermeasures**; • **Hardware → Dynamic memory**;

KEYWORDS

Row Hammering, Probabilistic Method, Weight, Circular Queue

ACM Reference format:
Jung Min You and Joon-Sung Yang. 2019. MRLoc: Mitigating Row-hammering based on memory Locality. In *Proceedings of The 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA, June 2–6, 2019 (DAC '19),* 6 pages.
https://doi.org/10.1145/3316781.3317866

1 INTRODUCTION

Memory reliability is one of the most important aspects in modern computing. No matter how well a memory device is designed, its functionality is at risk if reliability is not guaranteed. Hence, it is necessary to address memory reliability issues, which the row-hammering problem is one.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317866

Row-hammering occurs when a certain row in DRAM is accessed frequently [6, 9]. When this happens, the data bits in the rows adjacent to the frequently accessed row can be flipped. This problem is due to the electromagnetic interference between the DRAM cells, which is the result of large-scale integration in state-of-the-art semiconductor design [8]. This problem is critical to memory reliability because it flips the data without actually accessing it [7]. Henceforth, the accessed row is called an aggressor row and the adjacent rows are called victim rows.

To overcome the row-hammering problem, it is necessary to additionally refresh the victim rows [13] before a certain row becomes the victim row more than the row-hammering threshold (typically 2000 [12]). Choosing the victim rows that are refreshed additionally is also an important issue. Since additional refreshes need power, it concludes to the energy consumption problem. As a result, many methods are researched to find effective algorithms to choose which rows should be refreshed properly and those are considered as main solutions against the row-hammering problem. This paper suggests a new method to overcome the limitations of conventional row-hammering solutions.

There are two main-stream methods in the row-hammering solutions. One is counter-based methods [5, 11] and the other is the probability-based methods [7, 12]. The counter-based methods perform better in terms of reliability improvement, because these are based on counting all victim rows from memory accesses. On the other hand, since these methods require a counter for each row in the memory, of the considerably increase the memory area. Despite their high reliability, they are not preferred because of their poor area efficiency.

The second type of solutions (i.e., the probability-based methods), try to address the huge area overhead of the counter-based methods. Those methods help in a considerable reduction in area overhead. The main examples of probabilistic methods are Probabilistic Adjacent Row Activation (PARA) [7] and Probabilistic Row-hammer HIstory Table (PRoHIT) [12]. This paper focuses on discussing these methods with the proposed one.

There are two major conventional probabilistic methods that deal with the row-hammering problem. The first is PARA, which adapts a simple probabilistic algorithm to solve the problem. The probability of PARA is fixed and assigned to the victim rows to decide whether the victims should be refreshed at every memory access. This method is easy and simple to implement, however, it might suffer from significant drawback. For instance, PARA can still allow the row-hammering problem to occur if the problematic victim row is not refreshed based on the probability.

The second algorithm is PRoHIT. This method is a combination of two main concepts: table management, which uses two tables

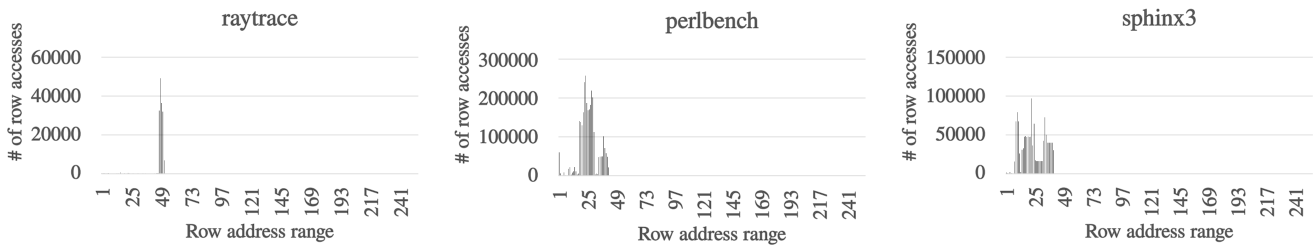

Figure 1: Locality of benchmarks

to prioritize the victim rows, and probabilistic operation. PRoHIT has been proposed to address the reliability issue of PARA. On the other hand, PRoHIT can increase the additional refresh operations despite its improved row-hammering reduction. Hence, PRoHIT may lead to high power consumption problem.

This paper tries to mitigate the reliability and power issues of the conventional methods by optimizing the refresh probability based on the memory locality. To overcome the disadvantages of the conventional methods, the victim row addresses are stored in a special queue and are used to calculate the probability for additional refreshes. Applying the stored row addresses in the queue, the proposed method calculates a *weight* that is used to obtain the optimized refresh probability. This paper suggests an equation and a calculation process of the *weight* in Section 3. The following items are the main contributions of this paper.

- This paper introduces a new method that adapts memory locality patterns to the conventional probabilistic method to increase the reliability.
- The proposed method uses a circular queue as a data structure to store the victim rows that are used to calculate the locality-based *weight*.
- The proposed method calculates the probability based on DRAM locality to prevent the excessive refreshes that occur in the conventional algorithms

The rest of the paper is organized as follows. In Section II, background knowledge about the conventional row-hammering solutions, PARA and PRoHIT, has been presented. The detailed architecture, queue management policy, and probability calculation of the proposed algorithm are described in Section III. To evaluate the benefits of the proposed method, different experiments and results are discussed in Section IV. Finally, Section V concludes the paper.

2 RELATED WORKS

The general approach for mitigating the row-hammering is by performing additional refresh operations. Since additional refreshes are a power consuming process, it is necessary to choose the rows to be refreshed efficiently. There are several methods to select the rows based on different concepts [1, 10]. This paper focuses on one of the main streams in those methods (i.e., the probability-based algorithms) that have an advantage in area overhead against the other main stream (i.e., counter-based algorithms). Below, two existing probability-based solutions are explained for the row-hammering problem.

2.1 PARA

PARA method applies a fixed probability to perform additional refreshes. The main advantage of this algorithm is the simplicity of the logic circuit implementation. Since this method only requires probability calculation circuit, its logic circuit is much more simple and compact to be implemented in memory device than other methods. However, this method may have a drawback in some cases. Because it is based on a fixed probability and ignoring DRAM characteristic, PARA can allow the row-hammering problem to occur. Incrementing the probability has been suggested internally to overcome this issue. However, this makes the refresh operation more frequent, leading to increased power consumption during the memory operations. Consequently, this might not be an efficient solution to deal with the row-hammering issue in the case of modern DRAM, which requires low power operation.

2.2 PRoHIT

Another algorithm for mitigating the row-hammering problem is PRoHIT. Addressing the low reliability concerns in PARA, this method prioritizes to the victim rows by managing two tables named as “cold table” and “hot table.”

When a certain row becomes a victim (i.e., when its adjacent row is accessed), it is inserted to the cold table. On the other hand, if the corresponding row already exists in the cold table, the controller promotes the row to the hot table. Since it is promoted from the cold table, the promoted row has lowest priority in the hot table. Subsequently, if the row becomes victim again, its priority increases inside the hot table. Consequently, the victim row which has highest priority in the hot table is refreshed at every regular row refresh period (i.e., 7.8 μs [12]). This is one of the main ideas of PRoHIT which is called as the two table management algorithm.

Additionally, PRoHIT utilizes a probability concept in the table management algorithm. For instance, when a certain row becomes a victim, PRoHIT decides whether the insertion/promotion should be executed based on a fixed probability. Thus, those operations of PRoHIT may be skipped. This enables PRoHIT to deal with the data sets whose size is bigger than the cold/hot tables capacity.

The additional refreshes of PRoHIT method are performed at every auto row refresh period which is fixed to 7.8 μs. The experimental results (details in Section 4) show that this method can result in huge number of refresh operations because of a short additional refresh period of 7.8 μs. This leads to a considerable increase in power consumption.

3 PROPOSED METHOD

Fig. 1 illustrates the locality of memory accesses from the three benchmarks used for evaluation in Section 4. The system configuration for the results in Fig. 1 is given in Table. 2. It can be seen that there are a large number of memory accesses to a few memory rows. The proposed method exploits this fact of strong memory access locality.

Fig. 2 describes the overall architecture of the proposed method. As demonstrated in Fig. 2, not only regular auto refresh command, but also additional refresh is performed based on a calculated probability. The bold path in Fig. 2 represents how the proposed method works inside the conventional memory system. The logic of proposed method decides whether the current victim row should be refreshed by using the calculated probability. If the victim row is selected to be refreshed, then its address is sent to the refresh controller which is already implemented in the memory controller. After that, the refresh controller additionally refresh the victim row to prevent the row-hammering problem.

Since all row-hammering reduction methods are a time and power consuming process, it is necessary to determine an efficient algorithm that minimizes the additional refreshes and still effectively defends DRAM against the row-hammering problem. In fact, the conventional solutions cannot guarantee to achieve both these goals at once. The conventional methods, PARA and PRoHIT, have their own drawbacks of suffering from a low reliability and a high power consumption, respectively. Hence, this paper proposes a locality based refresh algorithm to alleviate the row-hammering problem, which is called mitigating row-hammering based on memory locality (*MRLoc*). The proposed algorithm performs a refresh operation with a dynamically calculated probability considering the memory locality. As a result, it overcomes the drawbacks of the conventional row-hammering solutions (i.e., it has a higher reliability performance than PARA and a lower power consumption than PRoHIT). The following subsections describe the basic concept, queue management and probability calculation of the proposed method.

3.1 Basic Concept

Unlike conventional methods [7, 12], the proposed method calculates the refresh probability for a victim row based on previous memory access history. For instance, if a certain row has been accessed recently, a higher probability is assigned to its corresponding victim rows adjacent to the accessed one. According to the memory locality, the recently accessed rows are more likely to be visited again. Hence, the proposed method tries to avoid a DRAM hammering issue by maintaining a higher refresh probability for the victim rows adjacent to the recently accessed row. In addition, while the conventional approaches with a static probability refresh all victim rows equally including the victims adjacent to rarely visited rows, the proposed method reduces excessive or useless refresh operations by lowering the probability assigned to them.

To track the victim row addresses of the recently accessed rows, the proposed method stores them in a queue. Since a queue is used, the recently stored victim rows are located in the rear of the queue.

Fig. 3 shows an entire flow of the proposed method. When there is a row access, one of the victim rows adjacent to the accessed row is selected. After that, the proposed method obtains queue information by its queue management algorithm. Then it calculates a refresh probability using that information. The probability determines whether a refresh for the current victim row is needed. Finally, the queue is updated with the current victim row address. The same sequence (i.e., from the selection of current victim row to the queue update) is repeated twice for every memory access because a single memory access results in two victim rows.

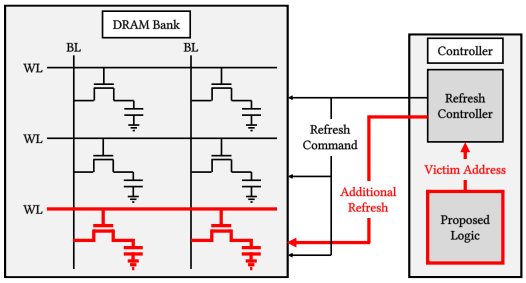

Figure 2: Overall System of MRLoc

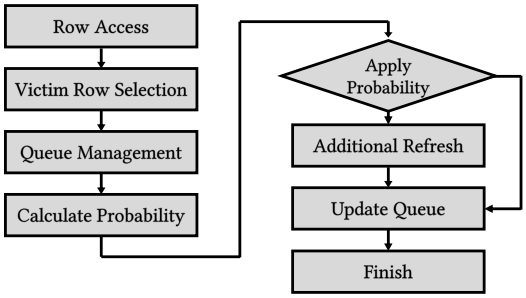

Figure 3: Flowchart of MRLoc

3.2 Queue Management

The proposed method determines the refresh probability considering the memory locality. To utilize the locality information, the proposed method stores a history of recent victim rows in a queue. The queue is managed to extract the memory locality difference among victim rows in the queue.

The proposed queue management scenarios are illustrated in Fig. 4 with a queue depth of 5. In Fig. 4(a), there are victim rows in the queue whose addresses are 0x00ab, 0x0024, 0x001c, 0x00ff and 0x0004. As can be seen, 0x00ab and 0x0004 are the least and most recently stored addresses, respectively. Assume that there is a memory access to 0x00fe. Its neighboring rows, 0x00ff and 0x00fd, become current victim rows. The proposed method first searches the addresses in the queue with the current victim row that has higher address, 0x00ff. In Fig. 4(a), 0x00ff is already stored in the queue, therefore, the search leads to a *queue hit*. As it is found at the second place from the rear, the proposed method finds the *queue hit distance* as 2. Subsequently, the queue pops the least recent victim row (i.e., 0x00ab), shifts up the remaining victim row addresses, and pushes the current victim row (i.e., 0x00ff). This updates the queue with the victim rows of 0x0024, 0x001c, 0x00ff, 0x0004, and 0x00ff. The *distance* information is used to calculate a refresh probability and this will be discussed in the next section.

Once the current victim row with higher address has been considered, the other current victim row, 0x00fd, is considered in the

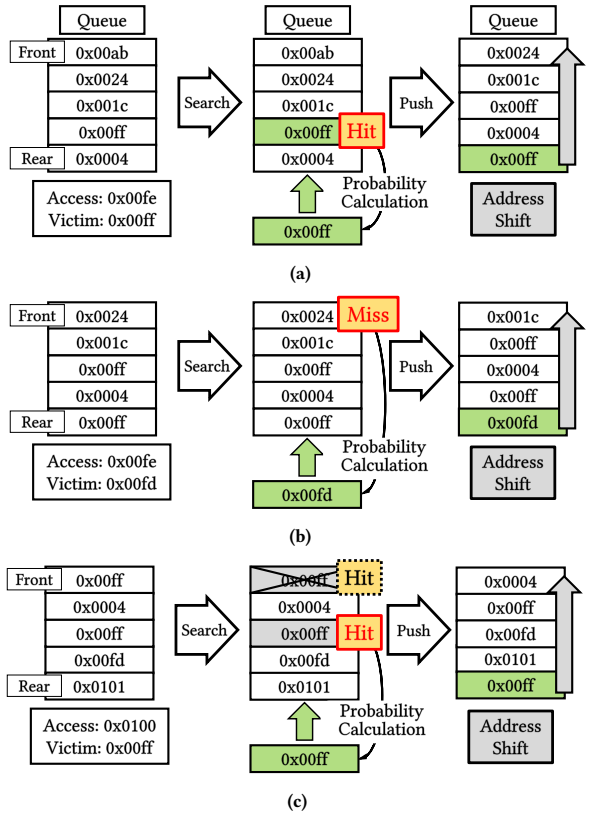

Figure 4: Queue Management (a) Queue Hit (b) Queue Miss (c) More than one corresponding victim rows exist in the queue

same fashion in Fig. 4(b). After updating the queue in Fig. 4(a), the proposed method scans the queue to see whether 0x00fd is already stored in the queue. In this scenario, no corresponding victim row exists in the queue and this results in a *queue miss*. For the *queue miss*, the proposed method assigns the queue size+1 as the distance. As the queue size is 5, all *queue misses* have the *distance* equal to the queue size+1, 6. The queue is updated with the entries 0x001c, 0x00ff, 0x0004, 0x00ff, and 0x00fd.

Assume that 0x0100 is accessed next. The current victim row with higher address, 0x0101 is considered first and this results in a *queue miss* thus giving a *distance* of 6. After the update of 0x0101, the following addresses 0x00ff, 0x0004, 0x00ff, 0x00fd and 0x0101 are stored in the queue. Then, the other victim row address, 0x00ff, needs to be taken into account. As shown in Fig. 4(c), there are two *queue hits* are found whose *distances* are 3 and 5, respectively. Because the proposed method is based on the memory locality, the *distance* with a smaller value is chosen. Similar to the other cases, the current address 0x00ff is also pushed to the queue and the least recent victim row 0x00ff is removed. Note that a circular queue can be used to make the shift operation easier.

3.3 Probability Calculation

The proposed method considers the memory locality in determining a refresh probability. As explained in the previous section, the proposed method obtains the *distance* from *queue hit* or *queue miss* based on the memory locality. Then the information is used for refresh probability calculation.

The proposed refresh probability equation is composed of two major terms, a *base probability* and a *weight*. The *base probability* denotes the minimum refresh probability and the *weight* gives a result of weight calculation which uses a *queue hit* or *miss distance* to reflect the memory locality. The following equation is proposed for refresh probability calculation:

$$p' = p + \alpha \times (L - d + 1) \tag{1}$$

p' denotes the proposed refresh probability which is referred to as a *weighted probability* in this paper. p is the *base probability*, which is obtained from conventional methods as a static probability. d is a *queue hit/miss distance* determined by the proposed queue management algorithm. L is the queue depth and α is used as a *weight constant* reflecting that how much of the memory locality is taken into consideration. In the *weight* term, 1 is added to make p' to p when *queue miss* occurs. The initial values of the equation constants are denoted in Table. 1.

Table 1: Initial Values of the Equation Constants

Parameter	Value
p [7]	0.0005
α (empirical)	0.00005
L (empirical)	15

Once the *weighted probability* is calculated, the proposed method performs the additional refresh operations based on p' which is dynamically assigned reflecting the memory locality. The victim row that has a higher locality gives a short *queue hit distance* resulting in a higher refresh probability and vice versa for the victim rows with low locality. Hence, the *weighted probability* by the proposed method tries to optimize the additional refreshes for DRAM hammering issues by reflecting the memory locality.

4 EVALUATION

To address the row-hammering problem, it is inevitable to perform the additional refresh operations. Hence, in comparing the row-hammering solutions, it is necessary to analyze how much of the row-hammering reduction is achieved (i.e., reliability) and how many additional refresh operations are needed (i.e., power consumption). This section evaluates the proposed method along with the two conventional solutions, PARA and PRoHIT.

Table 2: Simulated System Configuration

Parameter	Value
CPU clock	3.4 GHz
Number of CPUs	1
memory type	DDR3-1066
memory size	2 GB
l1i cache size	8 kB
l1d cache size	8 kB
l2 cache size	512 kB
the number of instructions	1000000000

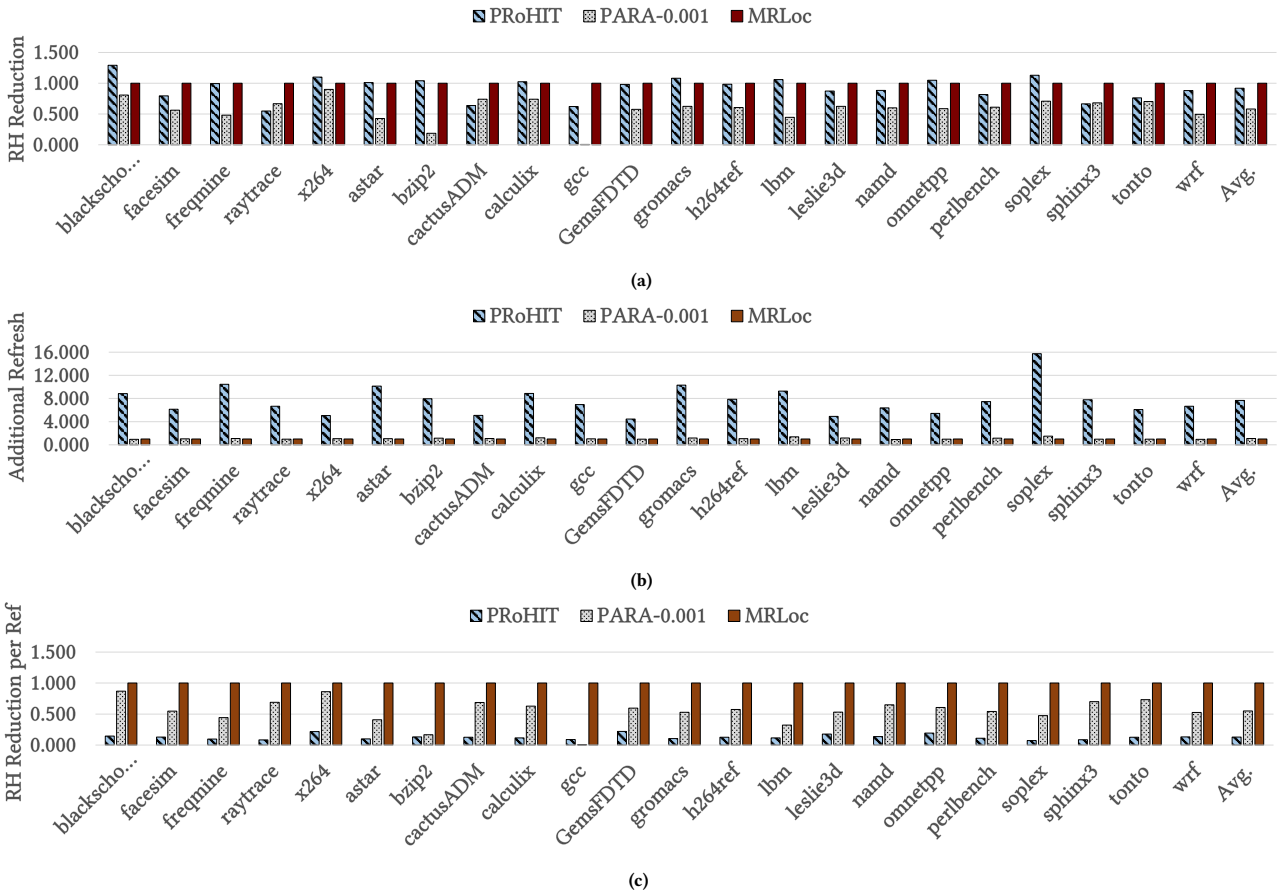

Figure 5: Performance Evaluation (a) Row Hammering Reduction (b) Additional Refresh (c) Row Hammering Reduction per Refresh

4.1 Evaluation Setup

To evaluate the performance of the proposed method, different experiments are performed with gem5 CPU Simulator [3] and 22 benchmark programs, 5 from PARSEC 3.0 [2] and 17 from SPEC CPU2006 [4]. The simulation configuration is given in Table 2.

A total of 1 billion instructions are simulated which are relatively large enough to observe the memory locality from the benchmark programs [12]. The regular refresh period is set to 7.8 μs and 64 ms for the row and total memory, respectively.

Like the most of programs in reality, it is necessary to obtain the highly memory-intensive traces. Hence, an *acceleration mode* [12] is used in the simulation. It helps a single-core system to mimic the multi-core system with a high clock frequency. Assuming that N instructions are executed in one refresh cycle, it is clear that $60N$ instructions are processed in 60 refresh cycles. Therefore, this mode reflects an extreme system which has a higher clock frequency than the simulation configuration given in Table 2.

4.2 Results

This paper compares two conventional solutions and the proposed method, to examine its advantages over the other approaches.

- PARA-X describes the PARA [7] method which uses a fixed probability as X. This paper adopts PARA-0.001. It is the same probability used in PRoHIT [12] for comparison.
- PRoHIT[12] is a conventional solution that uses the hot/cold tables and a probabilistic method.
- MRLoc is the proposed method which exploits the memory locality and *weighted probability* to optimize the additional refreshes.

In order to investigate the performance of the proposed method, the aforementioned indicators (i.e., a row-hammering reduction and additional refreshes) should be measured. The former defines the reliability of a solution, and the latter shows the power efficiency.

For the first indicator, the row-hammering reduction is defined as a difference between the frequency of row-hammering problems without solution and with solution. This paper selects the row-hammering threshold as 2000, then considers the row-hammering problem occurs if a certain row has become a victim row more than threshold value (i.e, 2000) in one refresh period. Hence, the row-hammering reduction denotes the protection strength against row-hammering that measures the reliability performance of row-hammering solution. Fig. 5(a) illustrates the row-hammering reduction simulated by 22 benchmarks. For convenience, this paper

normalizes the evaluation results of PARA and PRoHIT using MRLoc as a baseline.

According to Fig. 5(a), it is clear that the proposed method shows 1.72 times better performance on average than PARA-0.001 for all benchmarks. Although the average performance is 1.09 times better in MRLoc as compared to PRoHIT, PRoHIT shows higher reliability in nine benchmarks, such as *blackscholes*, *x264*, *astar*, *bzip2*, *calculix*, *gromacs*, *lbm*. *omnetpp*, *soplex*. Since the proposed method is based on the memory locality, it may not be much efficient for those few benchmarks that have low memory locality. On the other hand, the proposed method shows higher performance in the other benchmarks with an overall row-hammering reduction than PRoHIT.

In fact, it is not enough to define the performance of the row-hammering solution only with the row-hammering reduction. The other critical performance measuring factor is the number of additional refresh operations. In case of PRoHIT, it might look efficient in reliability performance without considering power issue for a few benchmarks. Since its reliability comes from the relatively high number of refreshes, the power aspect of PRoHIT is worse than PARA and MRLoc. Fig. 5(b) illustrates the ratio of the number of additional refreshes between PARA and PRoHIT relative to MRLoc. This result is obtained from normalizing the number of additional refreshes of PARA-0.001 and PRoHIT using MRLoc as a baseline. According to Fig. 5(b), the number of additional refreshes (i.e., excessive power consumption) of PARA and PRoHIT is 1.08 times and 7.66 times bigger than that of MRLoc, respectively. As can be seen in Fig. 5(b), PRoHIT executes much more additional refreshes than PARA and MRLoc due to the short refresh period of 7.8 μs. Thus, PRoHIT has higher chances to perform additional refreshes than PARA and MRLoc, so that it leads to a high power consumption problem.

To compare the performance properly, the previous two indicators, row-hammering reduction and the number of additional refreshes, should be considered together. Hence, this paper suggests a new indicator which is the ratio of the previous two performance evaluation factors (i.e., RH reduction per refresh). Fig. 5(c) illustrates the details of the performance between PARA, PRoHIT, and MRLoc based on the proposed indicator using MRLoc as a baseline. According to Fig. 5(c), the average performance of MRLoc is 1.82 times and 7.78 times higher than PARA and PRoHIT, respectively. It is clear that the performance of MRLoc is highly improved when the row-hammering reduction and additional refreshes are considered together. Consequently, comparing to PARA and PRoHIT, MRLoc obtains the advantage of both high row-hammering reduction (i.e., reliability enhancement) and less additional refreshes (i.e., decreased power consumption).

5 CONCLUSION

Row-hammering is a critical reliability problem in DRAM. Several methods exist to address this problem such as PARA and PRoHIT. In fact, these methods are more preferred than counter-based because of their low area overhead. However, those have some drawbacks in reliability and power consumption aspects, respectively. This paper proposes a novel method which exploits memory access locality to improve over naive probabilistic approaches. By managing a circular queue, the proposed method stores the address of previous victim rows, and then uses them to optimize the refresh probability at every *queue hit*. Based on the memory locality, MRLoc can increase row-hammering reduction by 1.72 times compared to PARA and 1.09 times compared to PRoHIT. Also, the number of additional refreshes of PARA and PRoHIT is 1.08 times and 7.66 times larger than that of MRLoc, respectively. Moreover, the performance of the proposed method is 1.82 times better than PARA and 7.78 times better than PRoHIT based on the row-hammering reduction per additional refresh. In conclusion, the proposed method achieves both high reliability and low power operation than conventional methods.

ACKNOWLEDGMENTS

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea by the Ministry of Education under Grant NRF-2018R1D1A1B07049842, in part by the Korea Institute for Advancement of Technology (KIAT) by the Korean Government (Motie:Ministry of Trade, Industry Energy, HRD Program for Software-SoC Convergence) under Grant N0001883, in part by the MOTIE (Ministry of Trade, Industry Energy (10080594) and KSRC(Korea Semiconductor Research Consortium) support program for the development of the future semiconductor device.

REFERENCES

[1] A. Amaya, H. Gomez, and E. Roa. 2017. Mitigating Row Hammer attacks based on dummy cells in DRAM. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*. 442–443.

[2] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton, NJ, USA. Advisor(s) Li, Kai.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[4] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17.

[5] D. Kim, P. J. Nair, and M. K. Qureshi. 2015. Architectural Support for Mitigating Row Hammering in DRAM Memories. *IEEE Computer Architecture Letters* 14, 1 (Jan 2015), 9–12.

[6] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2016. RowHammer: Reliability Analysis and Security Implications. *CoRR* abs/1603.00747 (2016).

[7] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372.

[8] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. *2013 5th IEEE International Memory Workshop* (2013), 21–25.

[9] Kyungbae Park, Donghyuk Yun, and Sanghyeon Baeg. 2016. Statistical distributions of row-hammering induced failures in DDR3 components. *Microelectronics Reliability* 67 (2016), 143 – 149.

[10] R. Qiao and M. Seaborn. 2016. A new approach for rowhammer attacks. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 161–166.

[11] S. M. Seyedzadeh, A. K. Jones, and R. Melhem. 2017. Counter-Based Tree Structure for Row Hammering Mitigation in DRAM. *IEEE Computer Architecture Letters* 16, 1 (Jan 2017), 18–21.

[12] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. 2017. Making DRAM Stronger Against Row Hammering. In *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17)*. New York, NY, USA, Article 55, 6 pages.

[13] K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis. 2017. Relaxing DRAM refresh rate through access pattern scheduling: A case study on stencil-based algorithms. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 45–50.