Towards Row Sensitive DRAM Refresh through Retention Awareness

Tanmay Goel[†], Divyansh Maura[†], Kaustav Goswami[†], Shirshendu Das[‡] and Dip Sankar Banerjee[§]
[†]Indian Institute of Information Technology Guwahati, Assam, India
[‡]Indian Institute of Technology Ropar, Punjab, India
[§]Indian Institute of Technology Jodhpur, Rajasthan, India
{tanmaygoel972,mauradivyansh,kaustavgoswami.2013}@gmail.com, shirshendu@iitrpr.ac.in, dipsankarb@iitj.ac.in

***Abstract*—Dynamic Random Access Memory (DRAM) is the *de-facto* choice for main memories in modern day computing systems. It is based on capacitor technology, which is volatile in nature. Hence, these memories require periodic refreshing, usually at 64 ms, in order to ensure data persistence. Refreshing results in blocking of the memory device for performing normal read or write operations. However, it has been found that not all cells of the device requires uniform refreshing at 64 ms. Due to shrinking of technologies, deviations are observed in nominal parameters which causes variations in retention and restoration time.**

In this paper, we propose a *retention aware* DRAM refreshing model, which is operated in auto-refresh (AR) mode of a DRAM device. We call the proposed model Lightweight Retention Time Aware Refreshing, or simply LRAR, which can be operated either in a deterministic or an approximate mode while consuming a constant amount of hardware space. The former ensures consumption of least possible area in comparison to previously proposed works. While the latter is aimed to incorporate periodic refreshing for a newly emerged DRAM phenomenon called *Variable Retention Time*, or, *VRT*, which uses the basics of approximation. After extensive evaluation, we find that our proposed model reduces execution time of programs up to 11% (9.4% on average). The memory system's energy consumption is also reduced by an average of 11.5%, and refresh energy by an average of 73.6%. We achieve the aforementioned gains at a modest area overhead of 7,240μm^2 (0.0018% of a 400mm^2 die) and storage overhead.

***Index Terms*—DRAM, retention time, DRAM refreshing**

I. Introduction

In current computing systems, DRAM technology has been the primary selection for main memories due to its cost effectiveness. Newer memory technologies have been proposed with promising results, however, in terms of cost effectiveness and longevity, DRAM devices still remain as the ideal choice for the same [1]. However, DRAM devices are also responsible for a significant portion of the system's total energy [2]. According to Cheng *et al.* [2], the DRAM alone responsible for consuming up to 40% of the system's energy. One of the driving factors for cost effectiveness of these devices is the fact that this is a *capacitor based* technology. Information in the form of bits is stored as charge in a DRAM capacitor or cell. Naturally, in order to maintain the data integrity of the device, the capacitors are periodically recharged, which is also known as DRAM *refreshing*. It is predicted that due to DRAM scaling, this energy consumption figure is likely to rise up to 50% [3]. Therefore it has now become a priority for DRAM manufacturers and researchers to optimize energy consumption in a DRAM, especially in terms of refresh energy.

There have been several research works on the aforementioned issue of optimizing DRAM refreshes [3]–[5]. Two of the major directions where there have been significant contributions include *access aware* refreshing and *retention aware* refreshing. The key observation in the former technique is that a recently accessed row would not require refreshing in the near future as its cells were recharged while accessing the data [6]. The latter exploits a key observation that not all cells of a DRAM device need refreshing at a uniform interval as defined by manufacturers. This time, called *tREFW*, is usually 64 ms [7] which accounts for the retention time of the weakest DRAM cell. This fact has been exploited by Liu *et al.* [4], where the authors proposed a refreshing mechanism which refreshes a DRAM row only at its required refreshing interval. Information of retention times of all rows is kept via the use of *bloom filters*. Hassan *et al.* [5] kept a separate space to remap weak rows in order to increase the refresh time of the DRAM device by 2x-4x.

One of the key observations that needs consideration is the fact that based on retention time profiling, we can mathematically calculate out an approximate number of rows, which requires refreshing at an interval lower than 256 ms, which we call *weak rows*. This has been previously pointed out in works like [4], [5], [8]. The number of weak rows is unlikely to change during the lifetime of a DRAM device [5], [8]. DRAM manufacturers provide an estimation of weak or variation affected rows and an equivalent number rows are present in the device in order to maintain the access latency of the device [9]. Hence, if we take this fact into consideration, we see that there is no need to keep information regarding strong rows as the set of weak rows are unlikely to change in a *commodity* DRAM device. As *commodity* DRAM devices are cost-sensitive, we believe that allocation of space done in a deterministic manner to keep information of weak rows would be sufficient to (a) reduce unnecessary refreshes, and, (b) become cost-effective.

In another direction, a new challenge for DRAM manufacturers has emerged. Known widely as *variable retention time*, or, simply *VRT*, this phenomenon shows random retention time behaviour of DRAM rows. Qureshi *et al.* [10] have shown that the conventional Error Correcting Codes (ECC) present in a DRAM device can correct most of these errors, however, cannot correct all of these errors. The authors have further shown that the set of VRT affected rows can consume a size of up to 31,798 unique rows over a period of 7 days, changing as frequently as in 15 minutes. This issue has imposed a heavy overhead on a large number of profiling based retention time works, previously proposed, due to its frequent profiling requirements.

In this work, we propose *Lightweight Retention Time Aware Refreshing*, or simply *LRAR*, a one-time profiling based retention time aware DRAM refreshing technique with minimal space and area requirements. LRAR has two primary operating modes, which covers both simple retention time based refreshing, as well as a technique to cover a large number of VRT affected rows. It also supports temperature and time scaling, which ensures its effectiveness for a longer period of usage. Following are concrete contributions of the work:

1) Deterministic Area and Storage Requirements for Retention Time Aware Refreshing Model: LRAR uses a deterministic storage structure only to store the information of weak rows. It is operated either in a deterministic manner, which does not consider VRT rows, or, in an approximate mode, where LRAR includes VRT rows alongside weak rows. Either way, the amount of space remains constant.

2) Minimizing Blocking Time of the DRAM Device by Reducing Refresh Instruction Count: This is a direct consequence of the previously mentioned contribution. Since unnecessary refreshes are reduced in the proposed model, the memory system now can serve other memory requests thus allowing programs to complete faster and consume less energy.

This paper is organized into 7 sections including introduction. Section II discusses the basics of a DRAM device. The working of LRAR is discussed in Section III. Section IV discusses the evaluation platform used and the details of the experiments performed. Results are analyzed in Section IV. Section VI highlights related works to the proposed topic. Finally, Section VII concludes this works with scope for further improvements.

II. Background

A. DRAM Organization and Working

A modern day double data rate (DDR) DRAM device is organized into six hierarchical levels. At the top, a memory channel is placed which can be operated concurrently with other memory 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. A rank is further divided into chips, which in turn are further divided into independent banks that provide the lowest level of independent operation. Each bank has its own row decoder. Banks are further divided into rows and then columns.

The master operation of a DRAM device is controlled by clock enable (CKE) [7] which must be high in order 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 which retain the data until a Precharge (PRE) command to the same bank is issued. The equivalent time required is called row cycle time (tRC), which can be written as *tRC* = *tRAS* + *tRP*. *tRAS* is called *row access strobe*, which is the time interval between row access command and data restoration in an DRAM array. *tRP* is called row precharge time, which is the time interval that it takes for a DRAM array to be precharged for another row access. 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. DRAM uses capacitors to store information as bits, which requires periodic recharging. This is known as refreshing. A refresh window (tREFW) is usually of 64 ms. A refresh instruction (tREFI) is issued in a smaller intervals of 7.8 μs. In one tREFI, a few rows are scheduled for refreshing, which consumes tRFC time. If a refresh instruction (tREFI) is sent to a row at time $t = 0$, the same row is refreshed again after tREFW time.

B. Retention Time and Variations

The refresh window (tREFW) of a modern-day DRAM is set at 64 ms in normal temperature and 32 ms in case of higher operating temperature ($\geq 85°C$) [7]. This defines the worst-case value for the entire DRAM chip. However, not all DRAM cells require recharging at 64 ms [4], [5] as there only exist a few rows which have retention time less than equal to 64 ms. There have been several works previously done in order to identify such weak rows [8], [11]. Most of these profiling techniques would pass a stream of data patterns within a wide range of operating temperatures to pinpoint DRAM cells failing to meet a set retention time threshold. Liu et. al [8] concluded that there exist approximately 1000 weak rows in a 32 GB DRAM device. Yet, even for this small fraction of rows, a DDR4 DRAM device is uniformly refreshed at 64 ms in normal working temperature. Hassan *et al.* [5] calculated that the bit error rate (BER) of the 4X refresh window DRAM device (tREFW = 256 ms) is 4×10^{-9}. Further, the authors gave the probability of the presence of a weak cell in a row as:

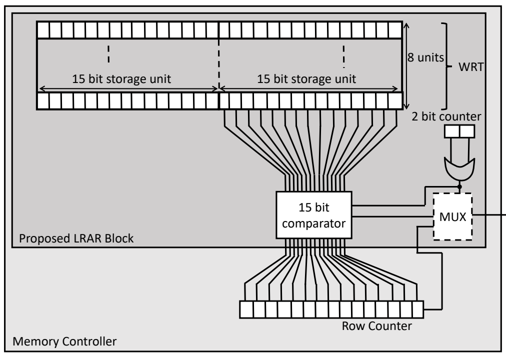
$$P_{weak\_row} = 1 - (1 - BER)^{N_{cells\_per\_row}} \tag{1}$$

Here, $N_{cells\_per\_row}$ is the number of weak cells per row. Clearly, the number of weak rows present in a DRAM bank would be very small.

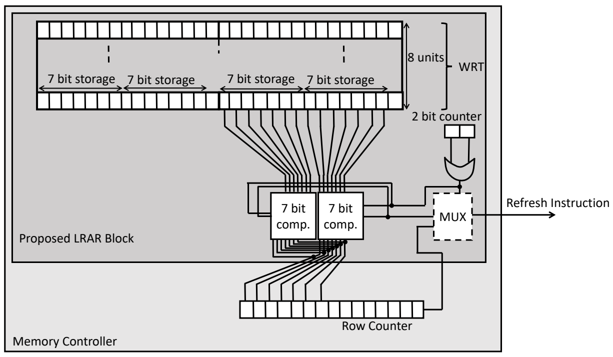
III. Methodology

A. Overview

LRAR is a hardware based technique. It has three distinct operating modes. The first is a deterministic mode (DM),



(a) Deterministic Mode



(b) Approximate Mode

Fig. 1: Proposed Operating Modes of LRAR

which is depicted in Figure 1. In DM, we keep a space equivalent to store addresses of a fixed number of weak rows. We call this storage space as weak row table (WRT). The probability of such weak rows are minimal, and, are unlikely to change [4], [5]. In order to include addresses of VRT affected rows, we use LRAR's approximate mode. In this, we cluster VRT affected rows into contiguous sets of 256 rows. To accommodate space to store these sets, we only keep 7 MSBs of the starting address of the set. Rest of the rows of the set are computed by the memory controller. Thus, storing only 7 bits, a large number of rows are included within the set. The final operation mode is for temperature scaling. We either store these rows completely in the fixed size space we have, or, we use approximation to those such rows as well.

All the proposed modifications are done in the memory controller (MC). Alongside WRT, which is maintained per bank, we also maintain a 2-bit global flag on the MC. The number of WRTs and 2-bit flags is equal to the number of ranks. VRT affected rows are simulated according to a random distribution. We also require comparators to assert whether an incoming row refresh request lies in a WRT. The number of comparators required is equal to the number of weak rows maintained per WRT. The overall hardware area, power and delay overheads incurred due to these changes are discussed in detail in Section V-D.

In this section, we explain each of these operating modes along with its core components.

B. Deterministic Mode (DM)

The WRT's size is the key in DM. The probability of presence of weak rows is 2.3×10^{-4}, which is calculated from previous findings described in Section II-B. Using this, we find that roughly 7.8125 rows are weak in a DRAM bank, which we round off to 8. We keep twice this size for (a) temperature scaling, and, (b) ensuring a longer lifetime of the device. Wang *et al.* [3] has shown that by making the refresh counter in the MC transparent would allow a refreshing mechanism based on retention time which also makes it compatible with JEDEC standards. Hence, we use those previously proposed concepts, including skip refreshing. However, the implementation of skip refresh in LRAR is different, in which, the same accounts for the time taken for checking the presence of a weak row address in WRT. Figure 1(a) represents DM of LRAR. The entire proposed module is implemented within the memory controller. As mentioned earlier, we maintain a 2 bit global flag which increments after every 64 ms and resets at an interval of 256 ms. A complete refresh would be performed when the flag bits are 00, whose state is selected using a multiplexer. We consider any row whose retention time is less than 256 ms as a weak row, and such rows are refreshed in 64 ms. Its row address (using 15 bits) is kept in the WRT. While incrementing the MC's row counter, if a weak row is encountered and the 2-bit counter state is not 00, a refresh for that particular row is issued. A 15 bit comparator is present for each weak row address in the WRT, totalling up to 16 such comparators. For all other cases, we skip its scheduled refresh. The time imposed by the comparator circuit is equivalent to skip refresh time.

C. Approximate Mode (AM)

In order to include VRT affected rows, it is essential to find an average placements of VRT rows. We have considered random distributions of VRT rows in a DRAM device for 10^3 cases. The average placement of VRT rows is considered for LRAR's AM mode. AM uses the basics of approximate storage, where, it only stores the 7 MSB of the initial address of a cluster. A cluster in this case is a set of contiguous row addresses. The key change from DM mode is that the comparator length is halved and such number of comparators are increased by two. If MSB of the row counter of the MC matches with the partially stored row address in the WRT, a refresh is issued. Figure 1(b) depicts the proposed AM mode.

The drawback of AM mode is the inclusion of false positives. However, considering VRT, which is completely random [10], a deterministic mode for the same will pose an enormous overhead. The set of VRT rows changes in intervals as small as 15 minutes [10]. Profiling the device after every 15 minutes will impose a substantial time penalty for a program. Works like AVATAR claimed that the conventional ECC present in a DRAM device has the capability to correct such errors, but, not all errors can be corrected by the same. Hence LRAR, if operated in AM mode, can correct most of such VRT affected rows. Since AM is probabilistic, the model also relies on the conventional ECC to correct rows not included under AM refreshing.

It has been mentioned in prior works [4], [5] that it is unlikely that the set of permanent weak rows will change throughout the lifetime of a DRAM device. Even in this unlikely scenario, AM can ensure age scaling of the device with both performance and energy benefits.

D. Temperature Scaling

Previously, we have mentioned that we have maintained twice the number of calculated weak rows in order to accommodate space for newly emerged weak rows due to temperature scaling. Retention time of a DRAM device is affected by temperature [4]. Temperature scaling works in a manner similar to DM due to having additional space. However, if the number of weak rows under temperature scaling becomes more than the amount of available space, the refresh model resorts to AM, ensuring coverage of all such affected rows.

In all aforementioned modes of LRAR, the key observation is the number of refresh operations saved. This minimizes the blocking time previously imposed on the system due to refreshing. The device is blocked for tRFC time when one tREFI command is issued. Now, the amount of time spent in one tREFI reduces, as the number of refreshes reduces. This provides a direct benefit of performance gain. Moreover, due to a reduced number of refreshes, we further obtain 11.5% benefit in terms of energy consumption as discussed in detail in Section V-B.

IV. Evaluation

In this section, we briefly discuss the evaluation platform used and the workloads that we have considered to evaluate our proposed refreshing model. We have used DRAMsim3 [12], a cycle accurate DRAM simulator. Major modifications are done in the memory controller of the simulator reflecting our proposal. The specification of the DDR4 memory that we simulated are given in Table I. For evaluating our results, we have selected a total of 23 benchmarks of varying workload to run our experiments. We used 15 benchmarks from SPEC CPU® 2006 benchmark suite [13] and 8 benchmarks from the PARSEC benchmark suite [14] for our experimentation.

A. Experiment Details

1) Experiments on Analyzing Timing Parameters: Minimizing the blocking time owing to the refreshing of the DRAM device will allow the DRAM device to operate in an unhindered manner for a longer period of time. Execution times of programs are reduced due to minimizing the blocking time of the DRAM device. We have made a comparative analysis of both the modes of LRAR with other refreshing techniques including the conventional DRAM device (*baseline case*), RAIDR [4] and an ideal *no-refreshing* DRAM device.

Simulation Parameter	**Specification (DDR4)**
Total Capacity	8 GB
Number of Ranks	2
Number of Bankgroups per Rank	4
Number of Banks per Bankgroup	4
Number of Rows/Banks	65536
Wordwidth	16
Memory Frequency	3200Mhz
tREFI	12480 ns
tRFC	560 ns

TABLE I: DDR4 Simulation Parameters

2) Experiments on Analyzing Energy Consumption: This is a straightforward experiment in which we aim at collecting the amount of energy savings that we obtain. We collected the required energy statistics using the 'drampower' component of the simulator DRAMsim3.

3) Overhead: We classify the overheads of the proposed technique into area, storage, power and delay. We have used Synopsys Design Compiler with 45 nm technology node for analyzing area, delay and power on a global operating voltage of 1.1 V. The model was described using Verilog HDL.

V. Experimental Results and Analysis

A. Timing Analysis

The reduced number of refreshes results in a performance gain. This is particularly due to the effective blocking time getting reduced. The blocking time gets reduced as at each instance of a refresh being skipped, we added the lookup time for determining a weak row, *i.e.* 0.36 ns, to the blocking time instead of an approximate 22 ns. This gives us a saving of 21.64 ns per row refresh skipped. This is also reflected in LRAR-AM, where there are more number of rows to refresh than DM, but there also exists a significant set of rows to be skipped. Hence, as the number of skipped rows is more than refreshed rows in both DM and AM, overall, we gain a positive time savings during execution. Overall, on average, this has resulted in 9.4% decrease in the total blocking time during the simulation, when LRAR is used in DM mode. Figure 2 represents the normalized execution time of a program with respect to LRAR's DM mode. The other cases for comparison includes a conventional DDR4 DRAM device (DDR4), LRAR-AM, RAIDR and no refresh (NR) cases. LRAR has a lower execution time than RAIDR by 3% on average. In the memory intensive benchmark bwaves, bzip2, leslie3d, libquantum, mcf and zeusmp, we observed the maximum decrease of 9.5% in terms of execution time. In case of AM, which shows a positive gain of 5% on average, over the conventional DDR4 DRAM device's case while including most of the VRT affected rows. Since LRAR-AM operates on a larger set of rows with the objective to include as many VRT affected rows as possible, the execution time of the same is higher than RAIDR in most cases. We also have to note the fact that RAIDR do not include any mechanism to incorporate VRT affected rows.

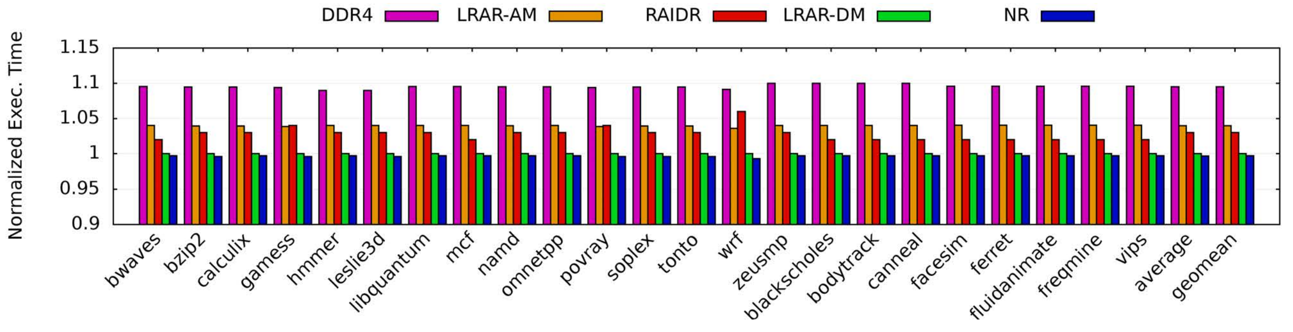


Fig. 2: Normalized Execution Time of Programs

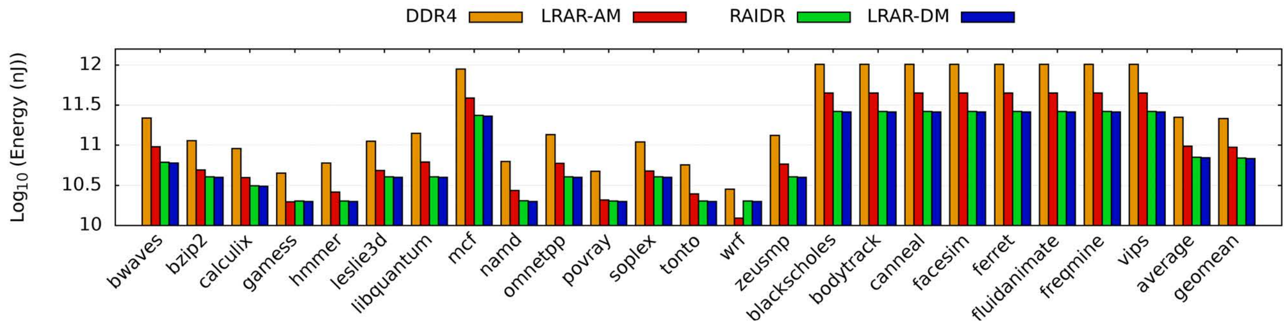


Fig. 3: Refresh Energy Consumption

B. Energy Analysis

As our proposed method, DM, significantly decreases the number of rows refreshes required, consequently refresh energy also gets reduced. For the simulated DDR4 memory in DM mode, we obtain an average of 73.6% savings. In some of the memory-intensive benchmarks like bwaves, bzip2, leslie3d, libquantum, mcf and zeusmp, our proposed model can reduce refresh energy of 69.6% on average. Compute intensive benchmarks from SPEC CPU2006 suite like omnetpp achieves up to 70.6% refresh energy savings. PARSEC benchmarks yield an average refresh energy savings of 74.7%. The reduction in total energy consumption of the system averages at 11.5%. Figure 3 represents refresh energy consumption of the memory system in 4 cases, including conventional (DDR4), LRAR-AM, RAIDR and LRAR-DM cases. LRAR achieves a higher percentage in energy savings due to the exclusion of false positive cases, unlike in RAIDR. The average energy savings is limited to a difference of 0.47% in comparison to RAIDR. The total amount of energy consumed by the system is also affected, as the number of refresh instructions are lesser. On average, LRAR saves 11.5% of the total DRAM's energy. On the other hand, LRAR's AM mode, which includes most VRT affected rows, can save refresh energy of the system by an average of 53.6%, while saving the memory system's total energy by 8.7% on average.

C. Analysis on AM

While both DM and AM can give benefits in terms of execution time and energy savings, however, our main objective in AM is to include as many VRT rows as possible. Inclusion of a modern day DRAM device is likely to be better reflected using AM. This is because of VRT affected rows. Our gains in terms of time and energy in case of AM is lower than DM and RAIDR due to the fact that AM attempts to refresh a large number of rows compared to the other two. At the same time, the other two aforementioned techniques, however, do not include VRT affected rows. A few of such rows, included in AM, may be false positives. However, at the same time, this is done in order to ensure that most VRT affected rows are included for refreshing in order to prevent any data loss. Simple profiling techniques are renders ineffective for profiling VRT rows as the set of such rows can change frequently, imposing a hefty penalty on the profiling mechanism.

The 7 MSBs of the initial row address acts like a cluster as the rest 8 bits are computed within the MC's row counter. One such set contains 256 rows. We have considered random distribution to analyze the extent of VRT affected rows. Figure 4 shows the average positions of VRT affected rows in 10^3 cases. We use this information to find 32 such clusters as our space limitation restricts us to store up to 32 approximate clusters. We have used K-Means and simple density based clustering. We have considered 25% of rows in a DRAM bank to be VRT affected. In this experiment, we have tested for another 10^3 cases and found that on average, 48.2% of VRT affected rows (12.05% of total rows of the DRAM bank) are included in those clusters when clusters are computed using density based clustering. K-Means clustering, on the other hand, shows an inclusion probability of 34%. As mentioned before, the

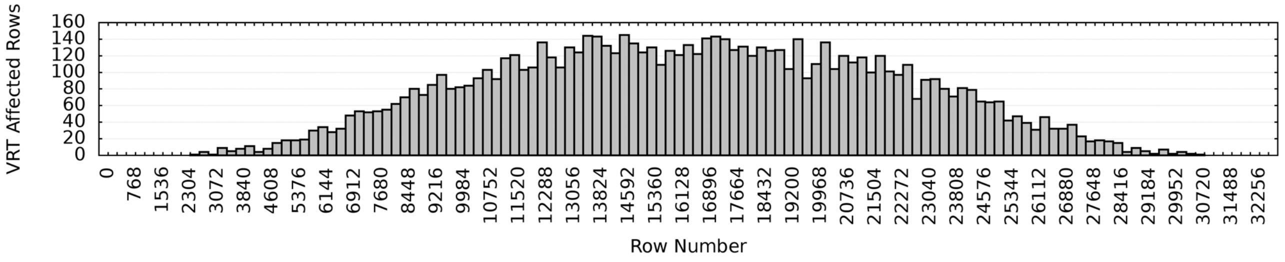


Fig. 4: Distribution of VRT rows

correctness of other rows is relied upon the conventional ECC. Furthermore, AVATAR has shown that approximately 31,798 unique rows are affected by VRT in a DRAM rank. In a typical scenario, a DDR4 DRAM device contains 8 banks in a DRAM rank, and, each DRAM bank contains 32,768 rows. This roughly translates to the fact that around 12.1% of rows in a DRAM bank are affected by VRT. Our AM model marginally includes a similar fraction of such rows, as approximately 27 additional rows rely on the conventional ECC.

D. Overhead

We have further divided overhead into 4 categories for a better understanding of the proposed model. In this subsection, we explain each of them.

1) Area: The basic design, which includes a set of 16 weak row addresses, refreshing logic and additional wire requires a total area of 7,421.28 μm^2. In other words, for one DRAM bank which contains 32,768 rows, we require an additional area of 7,421.28 μm^2. This roughly translates to the fact that LRAR only consumes 0.0018% of a 400 mm^2 die, which is negligible.

2) Storage: Storage overhead is considered in storing the weak rows in the DRAM device. One row address requires 15 bits to store. We are maintaining a WRT of a size equivalent to keep 16 such row addresses. Alongside, there is also a requirement of additional 2 bits. Therefore, our model requires 30.25 bytes per DRAM bank. Our storage requirements are merely 1.8×10^{-5} % of a rank. Likewise, for a 32 GB DRAM device, our model requires 3.75 KB bytes only.

3) Delay: We have already explained in Section III that the delay time incurred in comparison is included in skip refresh timings. Our synthesis results gave a time requirement of 0.36 ns. Note that in all our simulation, discussed in Section V-A and Section V-B, we have rounded the number to 1 ns. Savings in terms of blocking time, faster execution of programs and energy requirements is primarily reflected due to skipping of refreshes.

4) Power: There is a requirement of combinational power for the proposed model which is 0.98 mW of power.

VI. Related Works

DRAM devices are a major power consumer in the system stack [2]. Hence, there has been a large number of research works done toward reducing memory power consumption [2], [4], [5], [15]. Among this, refresh power is slowly becoming a major power consumer since the density of DRAM devices is increasing. It is predicted that refreshes will soon consume up to 40% of DRAM's total power in the near future [3]. One of the key observations behind retention aware refreshing was proposed by Liu *et al.* [4], where the authors stated that not all DRAM rows require refreshing at 64 ms. The tREFW time, *i.e.* 64 ms, is defined as a standard as it covers the worst case condition. The same authors proposed one of the best known retention aware refreshing technique called RAIDR, where the authors used bloom filters to store information of retention times of all rows, and, refresh a row at its required interval only. Hassan *et al.* [5] proposed a DRAM substrate, which acts as a cache to the main DRAM device. They identified that there exist approximately 1000 weak rows in a 32 GB DRAM device which has retention time less than 256 ms. Hence, for each DRAM sub-array, the authors added 8 additional row space, where weak rows can be mapped and the sub-array can be refreshed in 256 ms. A refresh mechanism based on timing window was proposed by Shin *et al.* [16], where it eliminates a DRAM refresh operations of *captured* rows in a pre-defined timing window. Wang *et al.* [15] extended RAIDR's design to make it compatible with JEDEC's auto refreshing standard. Clustering based DRAM profiling has been proposed by Sharifi *et al.* [17].

Approximation on DRAM storage and refreshing has emerged as a new technique for saving energy in DRAM devices. While allocating memory space, techniques like Flikker [18] allows the programmer to specify critical and non-critical data. While refreshing, the former segment is refreshed at regular intervals, however, the latter is not. This technique has been extended by Lucas *et al.* [19], where the authors utilized a non-uniform refresh of multiple DRAM chips.

VII. Conclusion

In this work, we have proposed a lightweight retention aware refresh (LRAR) mechanism for DRAM devices. LRAR is able to significantly reduce the number of refresh operations by identifying the weak rows deterministically and skipping the additional refreshes based on the different retention time of rows. Its implementation requires very minuscule modification in the memory controller without any major changes in the DRAM operation itself.

There are two operating modes of LRAR: DM and AM. In majority of cases, both of these operating modes ensure gains in terms of time and energy irrespective of the application being either compute or memory intensive. In DM, results in energy savings of 11.5% by reducing refresh energy. Along with that, it also gives us a performance gain of 9.4% by reducing the effective blocking time. All these gains are achieved with overheads which are minimal as compared to other state of the art methods as pointed out in Section V-D. LRAR, when operated in AM, gives a probability of 48.2% of including a VRT affected row, when 25% of total rows of the DRAM are assumed to be VRT affected.

We conclude that LRAR can effectively reduce energy consumption along with giving performance gains with minimal overhead in current and future DRAM systems. In the future, we plan on investigating better clustering methods for AM in order to include a larger section of VRT affected rows while minimizing false positives.

References

[1] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging NVM: A Survey on Architectural Integration and Research Challenges," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 2, Nov. 2017. [Online]. Available: https://doi.org/10.1145/3131848

[2] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, 2017.

[3] I. Bhati, Z. Chishti, S. Lu, and B. Jacob, "Flexible auto-refresh: Enabling scalable and energy-efficient dram refresh reductions," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 235–246.

[4] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *39th IEEE ISCA*, 2012.

[5] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu, "Crow: A low-cost substrate for improving dram performance, energy efficiency, and reliability," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 129–142.

[6] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. USA: IEEE Computer Society, 2007, p. 134–145.

[7] Micron Technology,, "DDR4 SDRAM," Tech. Rep. MT40A2G4, MT40A1G8, MT40A512M16, 2015.

[8] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," *SIGARCH Comput. Archit. News*, 2013.

[9] 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, "Defect Analysis and Cost-Effective Resilience Architecture for Future DRAM Devices," in *IEEE HPCA*, 2017, pp. 61–72.

[10] M. K. Qureshi, D. Kim, S. Khan, P. J. Nair, and O. Mutlu, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 427–437.

[11] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for dram retention failures: A comparative experimental study," *SIGMETRICS Perform. Eval. Rev.*, 2014.

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

[13] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, p. 1–17, Sep. 2006. [Online]. Available: https://doi.org/10.1145/1186736.1186737

[14] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[15] J. Wang, X. Dong, and Y. Xie, "Proactivedram: A dram-initiated retention management scheme," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 22–27.

[16] H. H. Shin, H. Seo, B. Lee, J. Kim, and E. Chung, "Timing window wiper: A new scheme for reducing refresh power of dram," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 133–138.

[17] R. Sharifi and Z. Navabi, "Online profiling for cluster-specific variable rate refreshing in high-density dram systems," in *2017 22nd IEEE European Test Symposium (ETS)*, 2017, pp. 1–6.

[18] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: Association for Computing Machinery, 2011, p. 213–224. [Online]. Available: https://doi.org/10.1145/1950365.1950391

[19] J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink, "Sparkk : Quality-scalable approximate storage in dram," 2014.