Online Profiling for Cluster-Specific Variable Rate Refreshing in High-Density DRAM Systems

Rasool Sharifi, Zainalabedin Navabi
School of Electrical and Computer Engineering
University of Tehran, Iran
{ab.sharifi,navabi}@ut.ac.ir

Abstract— **Multi-rate refresh techniques are among the methods that use non-uniformity in retention time of DRAM cells to reduce the DRAM refresh overheads. Unfortunately, retention time of some DRAM cells may change unpredictably over time due to *variable retention time (VRT).* In this paper, we propose an Online Profiler that divides DRAM cells into clusters and proactively tests and measures retention time of each cluster over time. The Online Profiler decides on increasing refresh period of a cluster based on a measured retention time, where this retention time has passed all tests of different data sets. Also, for ensuring maximum data integrity, the Online Profiler reads the entire memory periodically for correction of possible errors. We show that our proposed mechanism, that uses cluster-specific variable rate refreshing, can provide reliable operation while reducing refresh overhead of the performance by 6%, 13%, and 23%, and Energy-Delay Product (EDP) by 7%, 13%, and 27% for 32GB, 64GB, and 128GB DRAM modules, respectively.**

Keywords—Dynamic Random Access Memory, Online Profiler, Variable Retention Time, Memory Scrubbing

I. Introduction

Dynamic Random Access Memories (DRAMs) are widely used in most of today modern computer systems as main memory because of their high densities. Each bit in a DRAM consists of only one access transistor and one small capacitor. Leakage currents cause the stored data to be lost over time. Therefore, DRAM cells need to be refreshed periodically to retain their data. As density of DRAM increases, more cells need to be refreshed causing significant degradation in performance and waste of energy. For example, refreshing in a 32 Gb chip could result in 25% and 30% of performance and power overheads, respectively [1].

Refresh period, the time interval in which all of the cells should be refreshed, is typically 64ms for modern DRAMs. Despite the fact that cells in a DRAM are refreshed at such a high rate, the majority of cells could retain their data for much longer times. For example, [2] has shown that 99.7% of cells could retain their data for longer than 1 second even at high temperatures. Many recent works take advantage of this fact and try to reduce refresh overheads by tracking weak cells [3]. Multi-rate refresh mechanisms are among methods which exploit this property [3, 4] by refreshing cells with different refresh rates. This is made possible by identifying rows with weaker cells and refreshing them with higher rates, and rows with stronger cells at lower rates. A common aspect of all these methods is that there is an accurate profiler that can accurately measure or profile retention time of cells in a limited time. However, profiling the retention time of cells in a short period of time could be extremely challenging, as the same cell may show different retention times over the time. This property which is called *Variable Retention Time* (VRT) causes some DRAM cells to transition between different retention times at different points in time. More importantly, as shown in [5] some VRT cells tend to stay in their high retention time state longer than the time they remain at the low state. In fact, this type of VRT cells are very hard to find even after hours of exhaustive testing. Therefore, changing refresh rates of rows without considering the effects of VRT cells could result in a large number of intermittent retention failures.

In this paper, we propose, present, and evaluate an online profiler. We show that instead of a simple offline retention time profiling, an alternative approach is to detect and mitigate effects of retention time variations in the field, during the operation of DRAM in the system. In this case, the memory controller is responsible for proactive testing of cells retention time and updating the refresh rate of clusters according to the test results over time. An online profiler, in order to be effective, needs to meet two essential requirements. First, it requires the inclusion of a kind of feedback for protecting data integrity against transition of VRT cells to lower retention time states and temporal temperature fluctuations. For this purpose, a scrubber that is playing the role of the required feedback, reads and checks the entire memory periodically for finding and correcting potential errors caused by the variable refresh rate. Second, such a system while running in the background, should be able to give an accurate profile of retention time of cells with minimum effect on the performance and power of the overall memory system. As a result, scheduling of the memory scrubbing process at high rates of scrubbing for maximum reliability is crucial.

The rest of this paper is organized as follows: Section 2 provides background information and motivation. Section 3 introduces and describes different components of our proposed system. Section 4 shows the evaluation methodology and discusses the results and Section 5 concludes.

978-1-5090-5457-2/17/$31.00 ©2017 IEEE

II. Background and Related Wrok

A. Background

1) DRAM Refresh and VRT

Ideally, a DRAM cell should retain charge on its capacitor for a long time. However, because of various sources of leakage it loses its charge over the time. To maintain data integrity, DRAM cells are refreshed, periodically. The variable retention time phenomenon is not a new issue for DRAM devices. Fluctuations in the leakage current caused by the traps results in uncertainty in the retention time of cells and cause a phenomenon called *Variable Retention Time*. Figure 1 shows cells distribution for average hold time of minimum and maximum retention time states for two DRAMs. Both cells in this figure show two categories for VRT cells that constitutes the majority of existing cells. One with high average hold-time for maximum retention state (near vertical axis), and the other with high average hold-time for minimum retention state (near horizontal axis). The cells closer to vertical axis spend a short period of time in the low retention state and a high period of time in the high retention state.

2) Temperature Effect on Retention Time and VRT

Prior works have demonstrated that the retention time of cells decrease exponentially as temperature increases. In fact, $10°C$ increase in temperature results in approximately 45% decrease in retention time of DRAM cells [6]. Besides, as shown in [7], average hold-time of minimum and maximum retention time states depend exponentially on the temperature.

3) Multi-Rate Refresh Methods

It is found that the retention time distribution consists of tail distribution and main distribution. The tail distribution constitutes only a small number of cells. Therefore, most of the cells can be refreshed at lower rates despite the fact that all rows in DRAM are refreshed every 64ms. Taking advantage of this fact, prior works, e.g., [3, 4], reduce performance and power overheads of DRAM refresh by tracking the rows with higher retention time cells and refresh them with lower rates. They assume that there is an accurate system-level profiling mechanism for measuring retention time of cells. Although, profiling the retention time of cells at system boot may help to detect many retention time failures, still as shown comprehensively in [5], even after a long period of test, e.g., 24 hours, a large fraction of VRT cells may not be detected.

B. Related Work

Several approaches have been proposed to reduce refresh overheads on the performance and power in DRAMs. One approach is to exploit non-uniformity in the retention time of DRAM cells. Multi-rate refresh mechanisms, e.g., [3, 4], group rows into different bins according to their weakest cells and apply a lower refresh rate to bins with high retention time rows. However, AVATAR [1] is the only VRT-aware multi-rate refresh scheme that considers the effect of VRT cells on the data integrity. Nonetheless, although AVATAR considers VRT, it cannot tolerates temporal temperature variations. RAPID [4] is a software approach which favor longer-retention pages over shorter-retention pages when allocating DRAM pages. Although it includes online testing, still because it does not employ any feedback, it cannot tolerates errors caused by the VRT cells. RAIDR [3] groups DRAM rows into retention time bins and applies a different refresh rate to each bin. A second approach, involves the use of error-correction codes (ECC) for tolerating errors. A third approach, relies on software hints on the susceptibility of program data to the errors and decreasing the

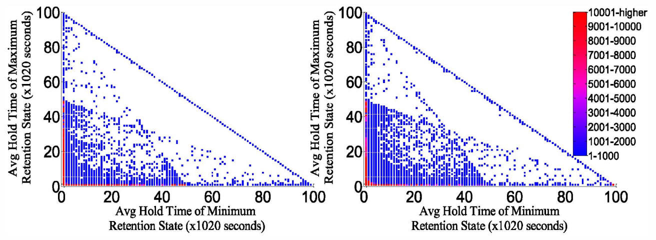

Fig. 1 Average Hold Times of Cells for two different DRAM modules [5]

refresh rate of DRAM for non-critical or invalid regions. Table 1 compares our proposed method with three other methods in different aspects.

TABLE I. Comparison between previous works and our work

Ref.	Online Testing	VRT Aware	Reliability	Scalability	Sensitive to VRT variations	Refresh Overhead Reduction
RAIDR[3]	✗	✗	Low	High	✗	High
AVATAR[1]	✗	✓	High	Low	✗	Medium
RAPID[4]	✗	✗	Medium	Medium	✗	High
Our Work	✓	✓	High	Low	✓	Medium

III. Proposed Online Profiler

An alternative approach for unreliable traditional testing at system boot-up in multi-rate refresh methods is an online profiler which is in charge of testing and measuring the retention time of cells in the background, while the system is running and the memory is in use. The online profiler should be accurate as much as possible for ensuring maximum data integrity. Also, it should have a minimum overhead on the performance.

Figure 2 illustrate the block diagram of our proposed Online Profiling system which become part of the memory controller. As shown, the components of the online profiler consisting of Tester, Scrubber, and Refresh Look up Table (RLUT) are responsible for determining the refresh rate of DRAM. The Tester monitors the request queue and new requests from Last Level Cache (LLC). Using these and other parameters, the Tester and Scrubber test the memory and update the RLUT with new refresh rates. Memory controller issues new refresh commands whenever the RLUT sends a refresh signal. The following subsections explain each component of the online profiler in details. The proposed method is implemented and evaluated in a cycle accurate memory controller simulator.

A. Online Tester

1) Cluster Testing

The online tester is in charge of testing the retention time of clusters and updating the refresh rates in the RLUT as shown in algorithm 1. The online profiler partitions the DRAM into a number of clusters determined by *N*. Each cluster contains a number of successive rows and is refreshed with respect to its refresh rate in the RLUT. Initially, all clusters are refreshed at the nominal rate, i.e., every 64ms (line 1).

At each round of test, the online tester selects a number of clusters using *ClusterSelector* based on the size of *ClustersBuffer* (BS), which is a temporary storage for the valid data of the

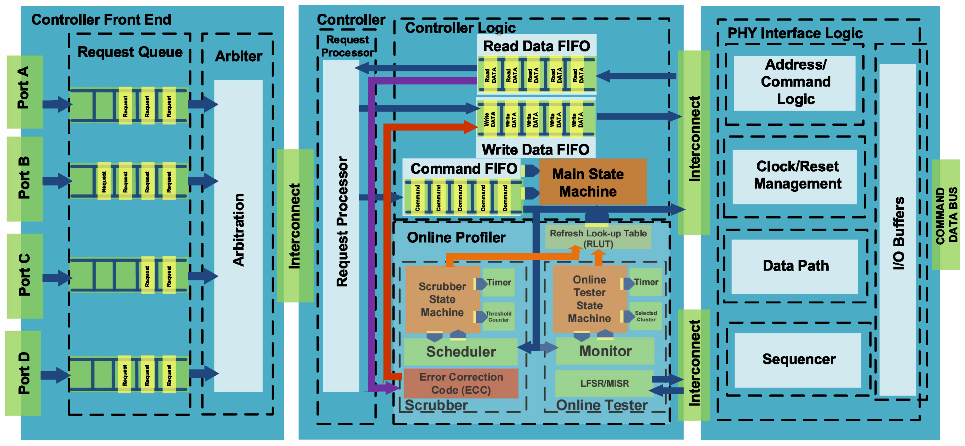

Fig. 2 Overall overview of the controller with proposed online profiler

clusters, and size of each cluster (CS). The *ClusterSelector* selects a number of candidate clusters which have the highest refresh rate in the RLUT, and monitors each cluster for two times of its current refresh period, and eventually selects the clusters for testing, which have no recent request for this period of time. This will help to minimize the effect of testing on the performance of the system. Next, the clusters which are selected by the *ClusterSelector* will be buffered in *ClustersBuffer* to be tested for twice of their current refresh periods in the RLUT (line 6, 18). After the test is done, test data are read and checked for errors and original data are recovered from *ClustersBuffer* (line 9-20).

While the selected clusters are under the test, the online tester also monitors new requests coming to the memory controller, and if there is a new request for one of the selected clusters, the online tester drops the test and recovers the original data into the DRAM array, and lets the memory controller serve the request to prevent starvation (line 4-8).

An important parameter in the proposed method is the size of each cluster, i.e., *CS*. On the one hand, as more rows are grouped into a cluster, more weak cells are included and as a result, possibility of reducing refresh rate for the cluster is reduced. Also, a larger cluster size impose more performance overhead when data should be transferred between DRAM and the *ClustersBuffer*. On the other hand, if the cluster size is chosen to be very small, hardware overhead of the RLUT and the online tester is increased and could impose high overhead on the power. As a result, a trade-off should be considered for considering the size of each cluster.

2) *Updating the RLUT*

For considering the effect of Data Pattern Dependency (DPD) and VRT, a cluster refresh period is increased and updated in the RLUT only after the cluster is tested by a number of different patterns determined by the N_{rt} and no error is reported (line 11-12). For maximum coverage, different data patterns can be used as the test data (e.g., zero, one, ten, five and random) [5]. If any error is observed with one of the test data patterns, the online tester halves the refresh period of cluster in the RLUT (line 14). Note that a cluster is tested with different data patterns at long intervals. This results in higher coverage in detection of VRT cells which stay in their high retention time state for a long period of time. The online tester repeats these steps until all clusters are tested for twice of their present refresh time in the RLUT which leads testing to be done only at granularity of 128ms, 256ms, 512ms, 1024ms, 2048ms, etc. Increasing the refresh time of clusters continues until the maximum possible refresh period for each cluster is reached. Two reasons may stop the online tester from increasing a cluster refresh time: 1) reaching the maximum static refresh time determined by the retention time of the weakest cell in the cluster and 2) encountering a VRT cell which is in its low retention time state.

3) *Advantages*

An important aspect of our proposed test mechanism is that it tries to test clusters for high refresh periods as much as possible which has three advantages. First, since we are testing the clusters for high retention times, for the multi-state VRT cells, the chance of finding cells with low retention times increases. Second, the Tester can increase refresh period of a cluster faster in the case that the cluster do not have any VRT cell. Third, because of longer intervals between the tests, the online tester has much less overheads on the system performance and power.

Another advantage of our proposed method is that it is flexible. Therefore, various test strategies can be easily implemented for different requirements. For example, one can simply enhance the data integrity by putting a guard-band on the refresh period of the clusters. This is done by testing a cluster for retention time of 256ms but increasing its refresh period to 128ms if no error is detected.

Algorithm 1 Testing Process

```
Require: required round of tests ($N_{rt}$), number of clusters (N),
ClustersBuffer size (BS), cluster size (CS)
1: Set RLUT [0: N-1] to 64ms; Set PassedTests [0: N-1] to $N_{rt}$
2: while ClustersBuffer ≠ ∅ do
3:   for j in ClustersBuffer [0: BS/CS-1] then
4:     if RequestQueue≠ ∅ and request ∈ ClustersBuffer[j] then
5:       drop the test for the ClustersBuffer[j] and recover the data;
6:       ClusterSelector() //call the for finding another candidate cluster;
7:       continue;
8:     end if;
9:     if ClustersBuffer[j] test is done = True then
10:      if #ERROR = 0 then
11:        PassedTests[n] ← PassedTests[n] -1; // n is the cluster index
12:        if PassedTests[n] = 0 then RLUT[n]← RLUT[n] × 2; end if;
13:      else then
14:        RLUT[n]← $\frac{RLUT[n]}{2}$;
15:        recover the data for the ClustersBuffer[j];
16:      end if;
17:        recover the valid data;
18:      ClusterSelector() //call the for finding another candidate cluster
19: end if; end for; end while;
```

B. Scrubber

Online testing of retention time of cells allows us to be more aware of important factors that are completely random, e.g., VRT, or are dependent on the environment conditions, e.g., temperature. Online testing of refresh time though necessary, yet is not enough for ensuring data integrity and can result in high error rate. Therefore, we need a kind of feedback for checking whether any error has occurred due to changes in the retention time of cells in the clusters. One way to implement this feedback into the system is by applying the memory scrubbing [8], which periodically traverses the entire memory and checks for potential errors. In our work, we use memory scrubbing to read rows and check them for errors using ECC. If any error has occurred, ECC corrects it, and in addition, the memory controller sets refresh period of the cluster with error to its initial value. As a result, accumulation of errors which may result in uncorrectable errors and system failure is prevented. The refresh period of clusters may be risen again only by the online tester.

1) Scheduling Scrubbing Process

An important parameter in scrubbing is the scrubbing period. Although, the scrubber needs to be as fast as possible to find any error before the accumulation of errors. Still, it should not impose an overhead on the performance and power of the system.

For scheduling purpose, we consider that many applications have compute and memory phases [9]. To reduce the impact of the scrubbing on the performance, the scrubber tries to predict presence of memory idle time (compute period) by monitoring the memory controller request queue. Algorithm 2 shows one period of our proposed scrubbing method. The first parameter is the time within which the entire memory has to be scrubbed. This time is in the order of tens of seconds, e.g., 90 seconds. The second parameter is the time it takes for one complete round of memory scrubbing. This time that is distributed in the first time period is in the order of several hundreds of milliseconds, e.g., 800ms. Both times are expressed in terms of number of memory cycles, i.e., N_{sp} (scrubbing period), and N_{rs} (round of scrubbing).

Algorithm 2 Scrubbing Process

```
Require: scrubbing period ($N_{sp}$), round of scrubbing ($N_{rs}$)
1: Set TotalElpasedCycles to 0;    Set $\alpha$ to 0;
2: while TotalElpasedCycles $\neq N_{sp}$ do
3:     TotalElpasedCycles ← TotalElpasedCycles + 1;
4:     if RequestQueue = 0 then QueueEmptyCount← QueueEmptyCount+1;
5:     else QueueEmptyCount ← 0;
6:     end if;
7:     if QueueEmptyCount is greater than Threshold then
8:         Set StartScrubFlag True;
9:         CountScrub ← CountScrub + 1;
10:    else
11:        StartScrubFlag ← False;
12:        $\beta$ = CoefCalc ($N_{sp}$,$N_{rs}$,TotalElpasedCycles,CountScrub);
13:        $\text{Threshold} = \text{Threshold} + \left(\frac{\beta-\alpha}{100}\right) \times \text{Threshold};$
14:        Replace $\alpha$ by $\beta$;
15:    end if;
16: end while;
17: function CoefCalc ($N_{sp}$, $N_{rs}$, TotalElpasedCycles, CountScrub)
18:     ElapsedTimePercent = TotalElpasedCycles / $N_{sp}$;
19:     WorkDonePercent = (CountScrub / $N_{rs}$);
20:     $\beta$ = WorkDonePercent / ElapsedTimePercent;
21:     Return $\beta$;
22: end function
```

For implementation of this algorithm, several variables are required. *TotalElpasedCycles* counts the number of elapsed cycle from the beginning of the scrub period and is set to zero when a new round of the scrub begins. The other important variable is the *Threshold* time that is the wait time for starting scrubbing before scrubbing session begins. This wait time resets to zero if there is a memory request within *Threshold*. Our simulations reveal that that after *Threshold* elapses, we have an enough amount of time to perform scrubbing before the next memory request. Third variable is *CountScrub* that keeps count of number of scrubbings. When this variable reaches N_{rs}, a scrubbing round is completed.

In the course of the execution of the algorithm 2, when the *QueueEmptyCount* counter exceeds *Threshold* (line 7), the *StartScrubFlag* is set and the *CountScrub* is increased by the number of issued scrub requests. With a new memory request to the memory controller, the scrubber stops its operation and waits for another overflowing start.

Due to dependence of the *Threshold* value on the memory demand, the scrubber tries to emend *Threshold* by calling the *CoefCalc* (line 17) function. This function returns β that is a measure of remaining scrubbing to be done to complete necessary scrubbing period. A high value for β shows that the scrubbing process is going well, and the scrubber can get relaxed about finishing the scrubbing process, thus increases the *Threshold*. Otherwise, for a low β, the scrubber finds out that the memory demand is high and *Threshold* value should be lowered to get the scrubbing job done in time. In addition, due to smaller amount of free time between requests in memory-intensive workloads, smaller number of successive scrub requests are sent to the memory for reducing the performance overhead of the scrubbing.

C. Hardware Implementation of RLUT

Figure 3 shows the overall micro-architecture of proposed RLUT which is a kind of frequency divider and generates refresh signals to be issued by the memory controller. The value of *n* is determined by the number of granularities of refresh periods. For example, eight refresh granularities (64ms, 128ms, 256ms to 8192ms) can be encoded with 3 bits (*n* = 3). The address size of the RLUT equals to the number of clusters. A system with 32K clusters and eight refresh granularities needs a 96KB memory.

The RLUT sweeps the entire clusters refresh-memory every 64ms. When a cluster refresh period is read from the memory it is decoded into an *m* bits binary number by the Look up Table and the multiplexer selects one bit out of the *m* bits binary. The memory controller issues a refresh command for the cluster if the output of the multiplexer is one. Otherwise, it bypasses the refresh command for the cluster. The value of *m* is determined by the highest refresh period. For example, a maximum refresh period of 8192ms needs 128 bits (128*64ms = 8192ms).

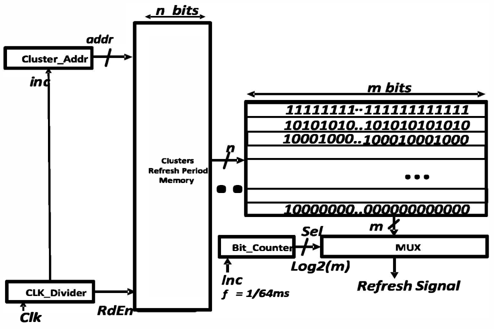

Fig. 3 Hardware implementation of the RLUT

IV. Results

A. *Simulation Setup*

For analyzing the impact of our proposed method on the performance and power of a DDR3 memory module, a similar retention time error model as in [10] is employed in DRAMSim2 [11]. First, for considering effects of systematic process variation on the retention time of neighboring cells, a multivariate normal distribution with spherical spatial correlation structure is employed for each chip in R language. In our simulations, we use Continuous Time Markov Chains (CTMCs) for modeling two and multi-state VRT cells. We form the transition rate matrix of VRT cells using values provided by [5]. In this model, each cell spends a random amount of time T, which follows an exponential distribution in each state, and then it moves to the other states. Random paths are generated at 45°C, 65°C and $85°C$ using Prism [12]. At the beginning of simulation, line address of each VRT cell is selected by generating random line addresses with a uniform distributed random number generator to consider the fact that VRT cells are randomly scattered across the chips. For evaluation of the performance and power of our proposed method, we use Gem5 [13], a cycle accurate full system simulator for the x86-64 architecture with integrated DRAMSim2 [11]. Table 2 shows configuration of our simulated system. We use multi-programmed workloads constructed from the SPEC CPU2006 benchmarks. Workloads are composed based on the memory intensity and row-buffer hit rate as determined in [14].CACTI [15] is used for calculation of the power in the RLUT and chip buffers.

TABLE II. Evaluation System Configuration

Component	Specification
Processor	X86, O3 cores, 8-core, 3 GHz
Per Core $	64 KB L1-I$ per core, 64 KB L1-D$ per core, 4MB L2 shared (8 way), LRU
Memory Controller	FR-FCFS scheduling, open-page policy, 32 entries read queue, 32 entries write queue
DRAM Organization	2 channel, 2 ranks/channel, 8 banks/rank, Timings: DDR3-1600

B. *Effective Refresh Saving*

Figure 4 shows the number of refresh commands issued by the memory controller in our proposed method, and in the normal situation for a time period of 280 minutes in 32 GB DRAM module. Each cluster contains sixteen 2KB rows, and a 64KB buffer is assumed for each DRAM chip. Therefore, two (64/32=4) clusters can be tested simultaneously for different refresh periods. Also, each cluster is tested with five different data patterns which give the maximum coverage [5]. As shown, after a period of time the system reaches to its stable state. In the stable state, the number of cells that get lower refresh rates by the online tester balances with the number of rows that get higher refresh rates by the scrubber.

The number of refresh commands that the memory controller issues in the stable state determines the total refresh power and performance overheads that can be mitigated. This settling value depends on four different factors: 1) the static retention time distribution of the cells, 2) hold-time distribution of the VRT cells in the high and low retention states, 3) temperature, and 4) the rate at which the online tester is testing the DRAM.

The point at which the online tester cannot increase the

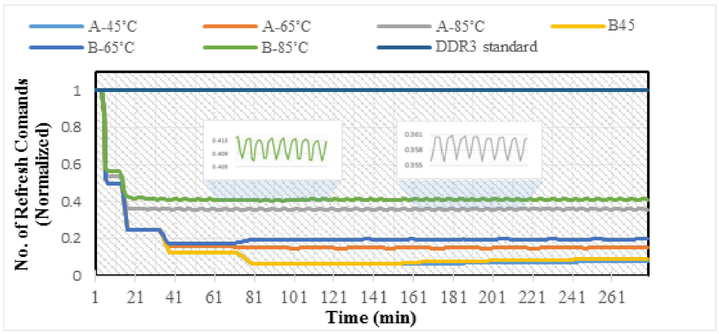

Fig. 4 Number of issued refresh commands in 280 minutes at three different temperatures for two 32 GB DRAM modules normalized to the baseline system. Scrubbing Time=90s, Number of Clusters = 32K, Cluster Size = 32KB

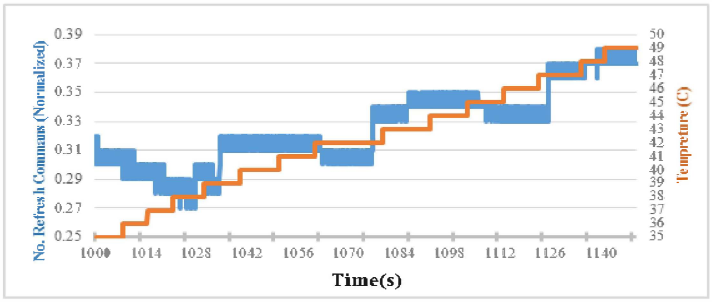

Fig. 5 Transient response of the system to the variations in the temperature for a 32 GB DRAM. Scrubbing Time=90s, Number of Clusters = 32K, Cluster Size=32KB

refresh period any further depends on the main and tail retention time distribution of the cells, and spatial correlation between the retention times of neighboring cells. As the weak cells, which belong to the tail distribution scatter more sparsely on the chip, fewer number of clusters can be refreshed with lower rates.

As shown in Fig. 4, the system reaches its stable state in a shorter time at higher temperatures. This can be explained by considering the fact that at higher temperatures, tests are done at higher rates because of lower retention time of cells and smaller intervals between tests. In addition, VRT cells have a higher rate of transition between retention time states. As a result, cells with VRT can be found more easily. Also, figure 4 shows more instability in the number of refresh commands at higher temperatures, e.g., 85 ºC. This is due to higher transition rate of the VRT cells at high temperatures.

C. *Transient Response*

Figure 5 shows transient response of the system to the fluctuations in the temperature. Lee et al. measured the DRAM ambient temperature in a server cluster and desktop system running a memory-intensive benchmark, and found that temperature never exceeds 34°C and 50°C, and never changing by more than 0.1 per second [16]. Therefore, temperature variation rate and interval of 0.1 per second and 35°C to 50°C are selected for our experiments, respectively.

As shown, the scrubber in the role of a feedback detects errors and increases refresh rate to prevent accumulation of errors which could result in uncorrectable errors and system failure. Scrubbing period determines how fast the system can respond to the transient disturbances.

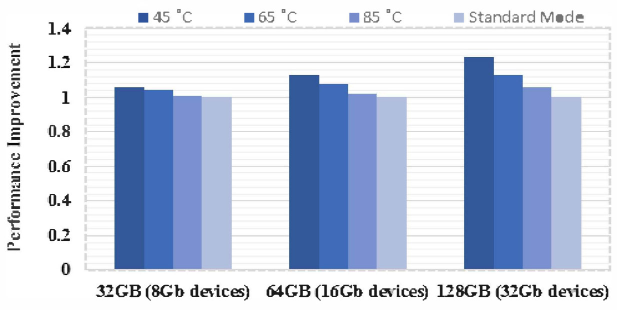

Fig. 6 Performance improvement for three different modules at 45°C, 65°C and 85°C. Scrubbing Time = 90s

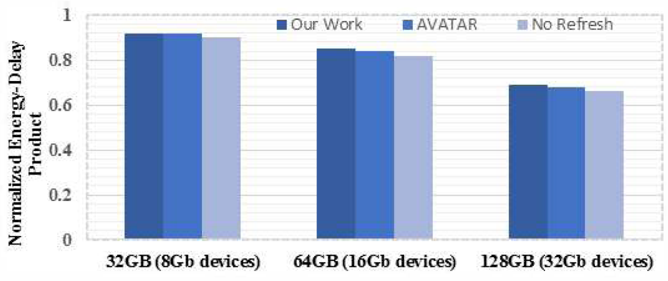

Fig. 7 The Energy-Delay Product comparison between the proposed method and the AVATAR at 45°C

D. Power and Performance Evaluation

Figure 6 shows performance improvement of our proposed method relative to the baseline system which employs JEDEC-specified 64ms refresh, and is protected by a SECDED ECC. Our proposed method improves the performance by 6%, 13%, and 23% at 45°C for 32GB, 64GB, and 128GB sizes, respectively.

The total energy for refreshing an 8GB DIMM once is approximately 1.1mJ, whereas the energy for one round of scrubbing is approximately 161mJ [1]. However, in the worst case (scrub period of 90 seconds) the scrub energy is consumed less frequently (64ms vs. 90s) than the refresh energy. Figure 7 compares the Energy-Delay Product (EDP) of our proposed method with AVATAR-1, and *No Refresh* cases. In the *No Refresh* case, EDP can be reduced by 10%, 18%, and 34% for 32GB, 64GB, and 128GB nodes, respectively. Whereas, our proposed method reduces EDP by 7%, 13%, and 27% at 32GB, 64GB, and 128GB nodes, respectively.

As expected, less power and performance overhead of refresh can be mitigated at higher temperatures due to lower retention time of the DRAM cells and higher transition rate of the VRT cells. Also, for ensuring high data integrity, scrubbing should be applied faster at higher temperatures which result in less reduction in the refresh overheads.

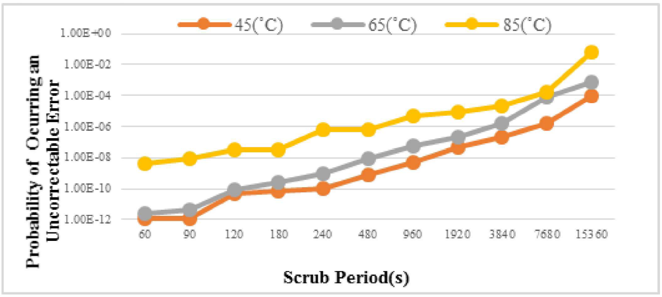

Fig. 6 Probability of occurring an uncorrectable error in a system protected by a SECDEC (8B) ECC for a 32GB DRAM

E. Failure Rate Analysis

Figure 8 shows the probability of occurring an uncorrectable errors with respect to the scrubbing period. We assume that the VRT cells are randomly scattered throughout the memory [1], and an uncorrectable error always result in system failure.

As shown, temperature can reduce reliability of the system by four orders of magnitude even at a high rate of scrubbing (i.e., 90 seconds). As temperature rises, rate of transition to lower states in cells with VRT increases and as a result, higher rate of error is observed. Therefore, for more reliable operation, higher rate of scrubbing is required.

Although, scrubbing memory at high rates helps us to increase reliability of the system, still not all the applications need this much of reliability. In fact, some dedicated applications refresh can be disabled with negligible impact on the performance [17].

V. Conclussion

In this paper, we introduced an online profiler based on the multi-rate refresh technique for reducing the refresh overhead on the performance and power in the presence of VRT cells and other environment dependent variables. We proposed a heuristic approach for reducing impact of the memory scrubbing on the performance. Our proposed online profiler consisting of the online tester and the scrubber reduces refresh overhead on performance by 6%, 13%, and 25%, and EDP by 13%, and 27% for 32GB, 64GB, and 128GB DRAM modules, respectively.

References

[1] M. K. Qureshi, et al.,"AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," *DSN*, vol. 2, no. 4Gb, p. 20, 2015.
[2] S. Baek, et al.,"Refresh Now and Then," *IEEE Trans. Comput.*, vol. PP, no. 99, p. 1, 2013.
[3] J. Liu, et al.,"RAIDR: Retention-aware intelligent DRAM refresh," in *ISCA,* 2012, pp. 1–12.
[4] R. K. Venkatesan, et al.,"Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM," in *HPCA,* 2006, pp. 155–165.
[5] S. Khan, et al.,"The efficacy of error mitigation techniques for DRAM retention failures," *SIGMETRICS*, pp. 519–532, 2014.
[6] J. Liu, et al.,"An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," *ISCA*, vol. 41, no. 3, p. 60, 2013.
[7] H. Kim, et al.,"Characterization of the variable retention time in dynamic random access memory," *IEEE Trans. Electron Devices*, vol. 58, no. 9, pp. 2952–2958, 2011.
[8] S. S. Mukherjee, et al.,"Cache scrubbing in microprocessors: Myth or necessity?," in *PRDC*, 2004, pp. 37–42.
[9] Y. K. Y. Kim, et al.,"ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," *HPCA,* 2010.
[10] C. Weis, et al.,"Retention time measurements and modelling of bit error rates of WIDE I/O DRAM in MPSoCs," in *DATE*, 2015, pp. 495–500.
[11] P. Rosenfeld, et al.,"DRAMSim2: A cycle accurate memory system simulator," *Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, 2011.
[12] M. Kwiatkowska, et al.,"PRISM: Probabilistic symbolic model checker," Springer, 2002, pp. 200–204.
[13] N. Binkert, et al.,"The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[14] M. K. Jeong, et al.,"Balancing DRAM locality and parallelism in shared memory CMP systems," *HPCA*, pp. 53–64, 2012.
[15] S. Thoziyoor, et al.,"Cacti 5.3," *HP Lab. Palo Alto, CA*, 2008.
[16] D. Lee, et al.,"Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in *HPCA, 2015*, 2015, pp. 489–501.
[17] M. Jung, et al.,"Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs," in *MEMSYS*, 2015, pp. 85–91.