Timing Window Wiper : A New Scheme for Reducing Refresh Power of DRAM

Ho Hyun Shin, Hyeokjun Seo, Byunghoon Lee, Jeongbin Kim and Eui-Young Chung
School of Electrical and Electronic Engineering, Yonsei University
50, Yonsei-ro, Seodaemun-gu, Seoul 03722 Republic of Korea.
Tel : +82-2-2123-7826, E-mail:hhshin@dtl.yonsei.ac.kr

Abstract— DRAM refresh power, which is consumed solely to preserve data, is rapidly increasing as capacity increases. A study predicts that the power will account for up to 35% of total DRAM power in the high capacity DRAM device. While various schemes were proposed to reduce the refresh power, those could not be adopted to commercial products due to cost overhead and design complexity issues. In this paper, we propose a simple refresh power saving scheme called TWW. We implement it with a much smaller amount of register size than previous works without modification on OS and DRAM controller. It eliminates unnecessary refresh operation of pre-activated rows in a specific timing window with optimum register size. We can save the refresh power up to 16% with only 6.2 KB registers in a DRAM device. This paper explains the implementation of the proposed scheme and shows the power saving effectiveness with gem5 full system simulator.

I. Introduction

The computing system of the Von Neumann architecture consists primarily of central processing unit(CPU) and main memory. The CPU has been developed to process complicated tasks more efficiently in terms of performance and energy. Multi-core processor is one of the powerful solutions for the high performance computing systems. With more than four or eight cores, the CPU handles several applications at the same time. However, the powerful parallel processor requires a very large amount of main memory.

Dynamic random access memory(DRAM) is the most popular main memory in modern computers due to its low cost and short latency characteristics. However, a DRAM cell should be recharged periodically and it consumes considerable power. The power is more significant in high capacity DRAM devices since all DRAM cells in a device should be recharged at least once in a fixed period. According to the recent study, DRAM sub-system in a server consumes up to 40% of total server power[1]. Furthermore, as the DRAM process scales down and the capacity increases, the refresh power portion increases continuously. A recent study addressed that refresh power accounts for up to 35% of total DRAM power in the high capacity DRAMs[2].

In many studies, various efforts to reduce the DRAM refresh power have been introduced such as selective refresh, retention aware and refresh scheduling schemes[3][4][5][2][6][7]. However, the schemes require a lot of cost overhead for the registers to store refresh related information such as row tags. Moreover, some schemes must modify the DRAM controller and the operating system(OS). Although the refresh related techniques were quite efficient in terms of power, they could not be applied to the products due to the design complexity.

This paper proposes a simple refresh power reduction scheme which is highly compatible with the modern computers. The proposed scheme can reduce the refresh power up to 16% with only 6.2 KB registers in a DRAM chip. Furthermore, the scheme only modifies the DRAM device instead of the DRAM controller or OS, and can be applied to the modern DRAM controller which adopts cas before ras(CBR) refresh policy.

II. Related works

Previous refresh power reduction techniques can be classified into two categories. The first is the selective refresh scheme and the second is leveraging a wide range of cell rentiontion time with OS management. The selective refresh scheme includes selective refresh architecture(SRA), Smart Refresh and ESKIMO[3][4][5]. The key idea of the works is that the DRAM controller has a tag table for all row addresses and prevents pre-activated or unnecessary row from refresh. While the idea is very simple, it suffers from very large cost overhead due to big register size which account for dozens KB. Moreover, the schemes should use ras only refresh(ROR) option which is not adopted in modern DRAM controllers. The ROR option is not adopted to the modern synchronous DRAMs since it complicate the DRAM controller and increase the

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

TABLE I
COMPARISON OF VARIOUS REFRESH POWER REDUCTION SCHEMES

Category	Scheme	Modifications		
		Device	Controller	O/S
Row selective	SRA[3]	O	O	X
	Smart Ref.[4]	O	O	X
	ESKIMO[5]	O	O	O
Retention aware	RAIDR[2]	X	O	X
	VRA[3]	O	O	X
	Flikker[6]	X	O	O
	RAPID[7]	X	X	O

interface power between the controller and the devices.

The second category is systematic approach[3][6][7]. The main idea of the schemes is to adjust refresh period of each row based upon its retention characteristics with OS management. While these techniques are very power efficient, the storage for keeping retention time increases design complexity as well as the cost. Table 1 shows the brief comparison among the various schemes.

III. MOTIVATION

In modern computing systems adopting multi-core architectures, several independent processes are running simultaneously, and they compete each other to access the main memory. As a consequence, the memory access pattern shows weak temporal locality. More precisely, the memory access pattern randomly encompasses wide address space. Under the circumstances, if the timing distance between an activation and a refresh for a row is very short, the refresh operation is able to be skipped. If the row addresses are distributed widely, the possibility that the timing distance is shortening increases. These mechanisms give us the chance to reduce refresh power.

IV. DETAILS OF TIMING WINDOW WIPER

A. The Rationale of the Proposed Technique Stems from the Window Wiper

Window wipers in a car sweep rain periodically and consistently. The wipers operate more effectively when it rains a lot. Understanding the operation of the wipers, we devise a new refresh power reduction technique called timing window wiper(TWW). It captures the row and bank addresses of active commands within a timing window and eliminates the refresh operation of the captured rows within the pre-defined timing window.

B. Overview

In DRAM operation, there are two types of activations. One is typical activation which is issued with a bank address and a row address for normal read and write operation. The other is refresh activation to restore DRAM cells,, which is triggered by a refresh command. If the timing distance between the activation and the refresh of a row is shorter than retention time, the later refresh operation can be skipped since the DRAM cells on the row are restored automatically in the previous typical activation. Fig. 1 depicts the concept diagram of TWW. A typical industrial DRAM device has 64 ms refresh period time at room temperature. It means all cells of a DRAM device should be refreshed in 64 ms with 8192(8K) refresh commands.[1] While row and bank addresses of a typical activation are issued from a DRAM controller, those of a refresh activation are generated from a refresh counter in a DRAM device. The arrow and triangle in the Fig. 1 indicate an active and a refresh command respectively. In addition, the numbers below triangle and above arrow indicate the row addresses. As we mentioned in section 3, TWW operates based on a sliding window wiper which periodically captures typical activations. In the Fig 1, the rectangles indicate the moving timing windows, and the window size is set beforehand. For instance, the size in Fig.1 is 4 ms which is same to 512 refresh command but the size can be variable according to the design concept and retention ability of DRAM cells.[2] The window moves toward next refresh range whenever a refresh command is issued. When a typical activation which has a row address belonging to the timing window is issued the row and bank addresses are stored into a register. And at the next refresh operation of the stored row address, TWW masks the refresh operation of the stored row and bank address. For example, when the window range is set from

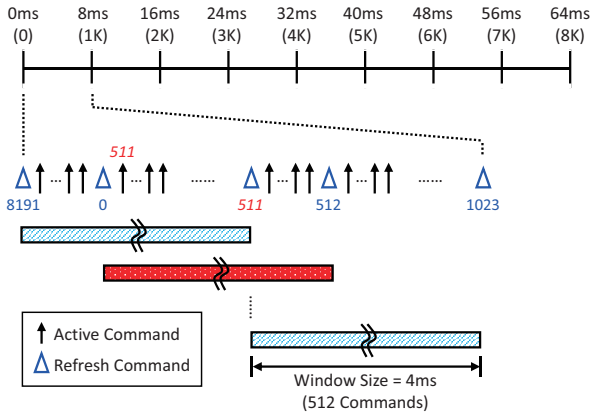

Fig. 1. Concept diagram of TWW. The window moves to next timing period whenever a refresh command is issued. The row address belonging to the timing window does not need to be refreshed at the next refresh time after the activation.

[1]In this paper, we assume that 8192(8K) refresh commands are issued in 64 ms retention time like typical DDR3 DRAM devices.

[2]In this paper, the window size is expressed as time or refresh counts.

0 to 512 and the #511 row address is activated in the window range the address is stored in the register. When the refresh time of the #511 is coming, TWW masks the #511 to be activated and thus refresh power is saved.

Algorithm 1 shows the pseudo code of TWW scheme. The algorithm executes whenever the active or refresh command is issued. It stores the issued row addresses, which are included in the window range, into a register table whenever an active command is issued. Fig. 2 shows the table structure called row address register table(RART) and the details of RART is explained in subsection D. If the row address of a typical activation is already stored in RART, RART updates only bank address bit. In contrast, if there is no matching address, it selects a new entry, sets the valid bit to 1 and finally stores the row and bank address. When a refresh command is issued, the table finds an entry corresponding to the row address generated from refresh counter. And it prevents the refresh operation of the banks matched to the bank address bits of the row address.

Algorithm 1 Pseudo code of TWW scheme

INPUT : CMD, BA, RA
// Issued command, bank address and row address

```
1: WS ← 4096 // Window size (for 32 ms)
2: RC ← 0 // Refresh address generated by the counter
3: EntryPt ← 0 // Entry pointer of the register table
4: while CMD do
5:    if (CMD = "ACT") then
6:       // When active command is issued
7:       if (RA > RC and RA ≤ RC + WS) then
8:          // If the RA is belonging to the window range
9:          EntryPt ← SearchRA(RA)
10:         if (EntryPt.ValidBit = 1) then
11:            UpdateBA(EntryPt, BA)
12:         else
13:            EntryPt ← LowestVacantEntry()
14:            StoreRA(EntryPt, BA)
15:            UpdateBA(EntryPt, BA)
16:            EntryPt.ValidBit = 1
17:         end if
18:      end if
19:   else if CMD = "REF") then
20:      // When refresh command is issued
21:      EntryPt ← SearchRA(RA)
22:      // Find the RA corresponding to the RC
23:      if (EntryPt ≠ NONE) then
24:         MaskedBank ← GenerateMaskSignal(EntryPt)
25:         ClearEntry(EntryPt)
26:         RC++
27:      end if
28:   end if
29: end while
```

C. TWW Architecture

Fig. 3 shows the overall architecture of TWW in a DRAM device. When a refresh command is issued, the command decoder generates REF signal for the refresh operation and the auto refresh counter generates a refresh address with the REF signal. The window range

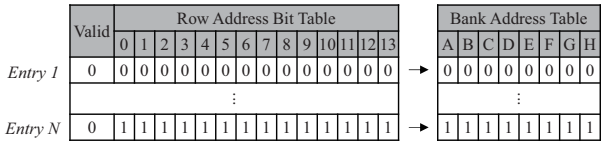

Fig. 2. Structure of RART(for a 512 MB DRAM device). The RART consists of one row and one bank address bit table, which are matched each other.

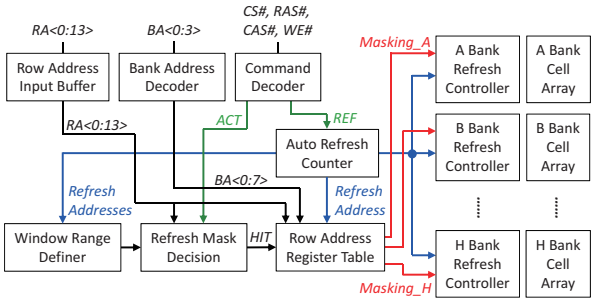

Fig. 3. Overall architecture of TWW. RART stores the row addresses not to be refreshed and it is most important for low cost overhead.

definer sets an address range with the start address and the end address which are calculated with start address and window size.

When an active command is issued with a bank and row address, the ACT signal is generated from the command decoder and goes to the refresh mask decision circuit. If the row address is within the window range, the refresh mask decision circuit stores the bank and row address into RART.

The last step of TWW is to generate the masking signals to prevent unnecessary refresh. If a refresh command is issued from the DRAM controller, the RART searches the stored row address which is same to the refresh address. If there is a matching row address in the table, the RART generates masking signals corresponding to the stored bank address bits.

RART directly affects on the cost overhead. In addition, whenever an active or a refresh command is issued, the table should be touched. Therefore, it is very important to make the table compact and power efficient. In the next subsection, we optimize the RART on circuit level and introduce its operation.

D. Implementation of RART

The RART consists of two parts(Fig. 2). One is for storing the row addresses and the other is for the bank addresses. An entry of RART has a valid bit, row address bits and bank information bits. The number of rows in a bank, which must be refreshed in a refresh command,

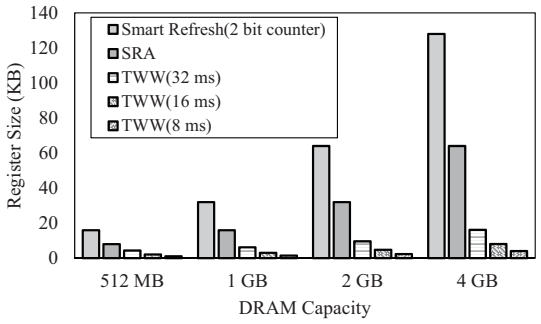

Fig. 4. Register size comparison of the previous and proposed schemes. The 32, 16 and 8 ms indicates windows sizes of the TWW.

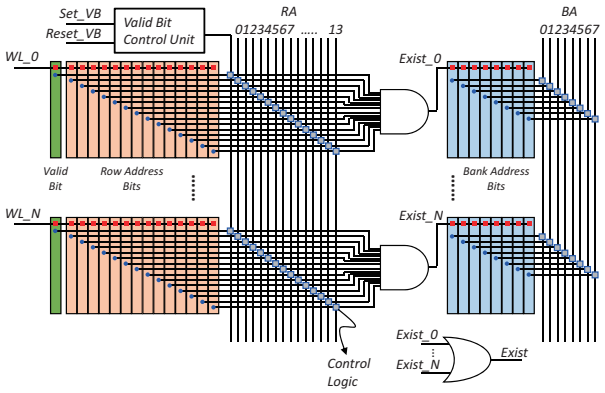

Fig. 5. A partial circuit diagram of the RART.

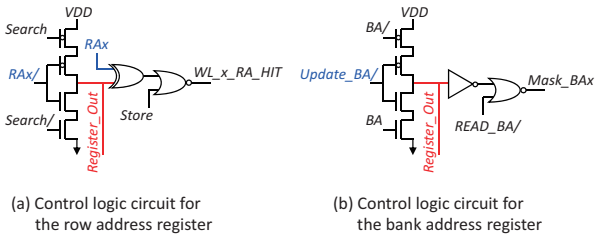

Fig. 6. Control logic circuits for row and bank addresses included in RART.

is only 1 for 512 MB devices. However, it increases to 2, 4 and 8 for 1 GB, 2 GB and 4 GB respectively. Therefore, the bank address bits should be increased to 16, 32 and 64 bits for each 1 GB, 2 GB and 4 GB device. The total register size can be calculated by *(1 + the number of row address bits + the number of banks * the number of refreshed rows per bank) * N*, where N is the number of entries. The N is ideally same to the number of the refreshed row addresses within the window size. For example, if the window size is 8 ms, the 1K refresh commands are asserted, and thus N is 1K. However, based on the above calculation the register size increases up to 40.5 KB with 32 ms window size and 4 GB density. In the calculation, we assume that all register entries are fully utilized, however we experimentally explore the necessary size in section 5 and find that only 40% of the size is enough to maximize the TWW performance. With the considerations, we compare the register sizes of proposed and other schemes. Fig. 4 shows the register size should be installed on each scheme. For 4 GB DRAM devices, our proposed scheme needs only 13% of smart refresh register size and 25% of SRA.

Fig. 5 depicts a part of the register table circuit which operates in fully associative manner. The word line(*WL*) signal gates the register cell and an *WL* is corresponding to an entry. In search mode, the RART enables all *WL*s to compare the input row address with the stored row address. Otherwise, in store mode it stores the received row address into a register entry, and only a single *WL* is enabled. The control logic circuit for the row address register which is indicated as gray box in Fig. 5. It is composed of a clocked inverter, an XOR gate and a NOR gate(Fig. 6(a)). In search mode the output of the clocked inverter is floated and *RAx*(row address) and *Register_Out* signals are compared to define in which entry has the same row address with the received *RA*. Otherwise, in store mode input *RA* signals are transmitted to the registers and the output signal of the control logic *WL_x_RA_HIT* is always maintained to low.

The bank register table uses the *Exist_x* signal instead of the *WL*. This is because the bank address register entry is only activated when the entry corresponding to the activated row entry is activated. The control logic of the bank address register is similar to the one of row one(Fig 6(b)). When updates the bank addresses, the only register cell corresponding to the activated bank address is updated to 1. Otherwise, in masking mode the *Register_Out* signals are transmitted to the *Mask_BAx* signals.

V. Experiment Results

A. Experiment Environment

We implement a simulation platform based on gem5 by plugging DRAMSim2 in it.[8][9]. The configurations of the default system and DRAM is shown in table 2. We experiment with PARSEC benchmark applications which are designed to be representative of next-generation shared-memory programs for chip-multiprocessors[10].

B. Sensitive Study on Register Size

In section 4, we calculated the size of RART in the worst case assumed that the all entries of RART are utilized. However, the accessed rows of real system are not matched to the window range perfectly, and thus it gives

TABLE II
SYSTEM CONFIGURATIONS FOR EXPERIMENTS

CPU Type/Frequency	Alpha/2 GHz
DRAM Type/Capacity/Frequency	DDR3/1 GB/667 MHz
Number of Banks/Ranks/Rows/Columns	8/1/16384/8102
Data Width	64 bits
Pre-charge Policy	Close Page
Refresh Interval(tREF)	64 ms

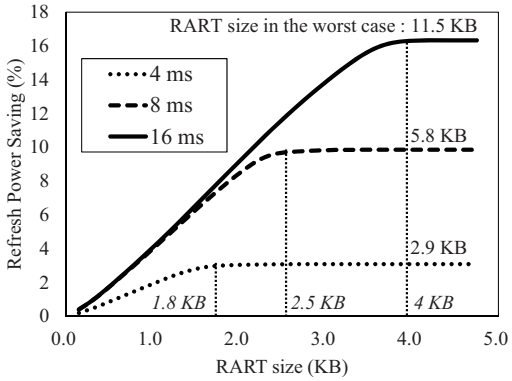

Fig. 7. Average refresh power saving with increasing RART size. Even though the ideal RART size is defined with DRAM capacity and window size, effective size is can be experimentally inferred.

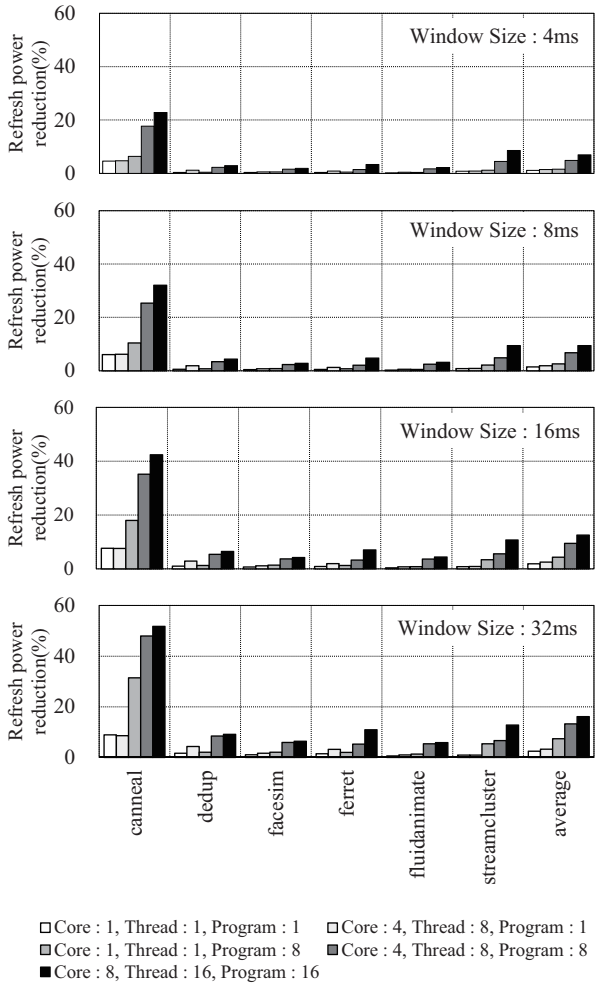

Fig. 8. Refresh power reduction under various system configurations and application environments. When the window size increases refresh power saving is improved.

us possibility to reduce the RART size. We experimentally explore the necessary RART size of practical system with PARSEC benchmark. Fig. 7 shows the average refresh power saving effect with increasing RART size for various window sizes. It depicts that the refresh power is saturated at any point and only 40% of RART size is enough to operate TWW even in the worst case. Based on the results, we perform the rest of the experiments with the 40% reduction on the size of the RART.

C. Refresh Power Reduction with Various Window Sizes and Configurations

In order to evaluate the power saving under the various configurations, we simulate TWW for multi-core, multi-threading and multi-programming configurations. Under the situations of multi-core and multi-threading, more active commands are issued in a timing period, and the probability that the active row addresses are stored in the RART also increases. Furthermore, the multi-programming needs wider physical address range than that of a single one. This wide DRAM access also increases the probability that the issued active row addresses are within the window range.

Fig. 8 depicts the refresh power reduction according to various configurations. It shows that as the number of cores and threads increases TWW reduces the refresh power more effectively. It is due to the incoming active row addresses are more widely distributed under multi threaded environment. These results imply that TWW can further improve the refresh power of the future computing system that will handle big data.

The other important factor in the performance of TWW is the window size. If the window size is enlarged, more row addresses are captured by TWW and the refresh power is also improved. Fig. 8 shows the results of the refresh power saving in accordance with the window size changes of various applications. The experiment results depicts that the average 16% refresh power is reduced with 32 ms of window size under 8 cores, 16 threads and 16 programs. Especially, in the case of canneal application, the refresh power reduction reaches to 52%. However, even though the expansion of the window size is helpful for TWW, it necessarily increases the retention time of

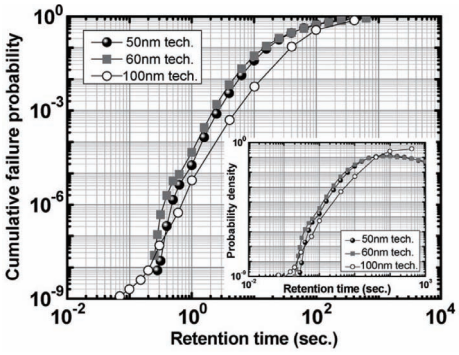

Fig. 9. Cumulative probability curve of data retention failures for three technology generations (100, 60, and 50 nm) of 1 Gb DRAM chips. The inset shows the probability distribution of incremental failures against increased pause time after data writing[11].

DRAM. In order to manage the risk, we propose a simple method using e-fuse in section 6.

VI. Risk Management

Our scheme necessarily increases the DRAM retention time 64 ms plus window size. This is a potential risk which is able to fail the DRAM operation. In order to manage the risk, we proposed a simple method using a small amount of e-fuse. Even though DRAM vendors usually screen the DRAM cells with a retention time of 64 ms, most of the cells can endure over 64 ms. This means that even if we increase the refresh time over 64 ms, the amount of failed cells can be very small. From the recent research results(Fig. 9), we analyzed the number of cells which has retention time under 100 ms, and found only dozens of bits for a 1 GB device[11]. These risky bits can be detected during the cell test process and those are stored in the risky row address table of refresh controller using e-fuse. Therefore, whenever the masking signal is arrived to the refresh controller, it defines whether the row address of the masking signal exists on the table. If there is a matched row address, the masking signal is ignored and the refresh operation is running normally.

VII. Conclusion

With deep explorations on various previous refresh power reduction schemes, we introduce a simple scheme which is highly compatible with current DRAM subsystem. We could reduce the average refresh power up to 16% with multi-threading and multi-programming applications. In addition, for specific application the improvement increases to 52%. Most importantly, it could be achieved with only logic additions on DRAM device without modifications on a DRAM controller or OS.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2016R1A2B4011799), by the IDEC and by Samsung Electronics.

References

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, "Energy management for commercial servers," *Computer*, vol. 36, pp. 39–48, Dec. 2003.

[2] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, June 2012.

[3] T. Ohsawa, K. Kai, and K. Murakami, "Optimizing the DRAM refresh count for merged DRAM/logic LSIs," in *1998 International Symposium on Low Power Electronics and Design, 1998. Proceedings*, pp. 82–87, Aug. 1998.

[4] M. Ghosh and H.-H. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3d Die-Stacked DRAMs," in *40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007. MICRO 2007*, pp. 134–145, Dec. 2007.

[5] C. Isen and L. John, "ESKIMO - energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42*, pp. 337–346, Dec. 2009.

[6] 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*, ASPLOS XVI, (New York, NY, USA), pp. 213–224, ACM, 2011.

[7] R. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*, pp. 155–165, Feb. 2006.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[9] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, vol. 10, pp. 16–19, Jan. 2011.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, (New York, NY, USA), pp. 72–81, ACM, 2008.

[11] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *IEEE Electron Device Letters*, vol. 30, pp. 846–848, Aug. 2009.