CAR: A Compression-Aware Refresh Approach to Improve Memory Performance and Energy Efficiency

Article *in* ACM SIGMETRICS Performance Evaluation Review · June 2016

DOI: 10.1145/2964791.2901498

Wenjie Liu
Temple University
9 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Ping Huang
Huazhong University of Science and Technology
51 PUBLICATIONS **268** CITATIONS
SEE PROFILE

Ke Zhou
Huazhong University of Science and Technology
146 PUBLICATIONS **980** CITATIONS
SEE PROFILE

CAR: A Compression-Aware Refresh Approach to Improve Memory Performance and Energy Efficiency

Wenjie Liu¶, Ping Huang§, Kun Tang§ , Ke Zhou¶*, and Xubin He§
¶ Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology
§ Department of Electrical and Computer Engineering, Virginia Commonwealth University

ABSTRACT

DRAM memory is suffering increasingly aggravating refresh penalty, which no longer causes trivial performance degradation and power consumption. As memory capacity increases, refresh penalty has become increasingly worse as more rows have to be refreshed. In this work, we propose a simple, practical, and effective refresh approach called *CAR (Compression-Aware Refresh)* to efficiently mitigate refresh overheads. We apply data compression technique to store data in compressed format so that data blocks which are originally distributed across all the constituent chips of a rank only need to be stored in a subset of those chips, leaving banks in the remaining chips not fully occupied. As a result, the memory controller can safely skip refreshing memory rows which contain no useful data without compromising data integrity. Such a compression-aware refresh scheme can result in significant refresh savings and thus improve overall memory performance and energy efficiency. Moreover, to further take advantage of data compression, we adopt the rank subsetting technique to enable accesses to only those occupied chips for memory requests accessing compressed data blocks. Evaluations using benchmarks from SPEC CPU 2006 and the *PARSEC 3.0* on the recent DDR4 memory systems have shown that CAR can achieve up to 1.66 $\times$ performance improvement (11.7% on average).

1. INTRODUCTION

Memory performance is crucial to the overall system performance as it provides an intermediate stage to bridge the huge gap between the performance anticipated by processors and the stagnant operations of disk system or flash storage. As data-intensive workloads and big-data applications become more and more prevalent, the requirement for high performance and large capacity from the memory system has steadily increased. Furthermore, memory subsystem contributes a significant percentage of the overall system energy consumption.

Modern DRAM memory technology is primarily built with capacitive cells, each of which contains one capacitor and one transistor. Information in a cell is represented by the charges trapped in the capacitor. Since trapped charges leak over time by its very nature, DRAM cells must be *refreshed* periodically to guarantee data integrity [1]. The basic requirement is that every memory cell must be refreshed at least once within its so-called *retention time* window [2] even when the whole memory system is in idle state, otherwise data loss might occur. However, refresh operation brings about significant performance and energy overheads, causing "*Refresh Wall*" problem [1]. Even worse, as memory density increases, the refresh overheads grow substantially since there are more rows that need to be refreshed and the device unavailable time increases commensurately. It is projected that in future 64Gb devices, refresh operations would account for 50% of energy consumption and degrade 50% memory throughput at the same time [1]. High refresh overheads have become a critically detrimental factor to memory performance and energy efficiency and it is extremely beneficial to mitigate refresh overheads.

In this paper, we propose a low-complexity, practical, and straightforward approach to mitigate refresh overheads in respects of both performance and energy. Many previous works [3] have revealed that most cache lines are reasonably compressible, implying there exist a good opportunity to achieve refresh savings via data compression. We employ the well-known low-complexity, low-latency base-delta-immediate (BDI) [3] compression algorithm as our compression engine to minimize introduced overheads. With compression, a 64-byte data block only requires a portion of the original space, resulting in *sparse* banks which are eligible for not being refreshed. In order to take advantage of the compressed data, *Rank Subsetting* [4] is ultilized to partially access a rank (i.e., access the chips containing post-compression content) and obtain reduced memory traffic and access energy.

*Corresponding author.

SIGMETRICS '16 June 14-18, 2016, Antibes Juan-Les-Pins, France
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4266-7/16/06.
DOI: http://dx.doi.org/10.1145/2896377.2901498

2. COMPRESSION-AWARE REFRESH

2.1 Overall Architecture

Figure 1 shows a simplified schematic view of our proposed CAR architecture. As shown in the figure, we add a compression engine in the memory controller. The compression engine performs compression and decompression for cache lines. Cache lines required by the processor are first brought into the memory controller for decompression and then returned to the processor. New cache lines coming from the processor are compressed and then stored in the memory banks. The number of banks (in different chips) that a cache line actually occupies depends on the compressed size. Depending on the compression ratio, cache lines might be compressed into variable sizes. In the figure, we assume a memory system comprising 8 chips, each of which is responsible for 8 bytes of a 64-byte cache line. Due to compression, banks in some of those chips may contain much less data than when cache lines are stored in uncompressed format. Those banks are called *sparse* banks. Memory rows in sparse banks may not need to be refreshed, as they contain invalidate data, reducing the amount of rows that need to be

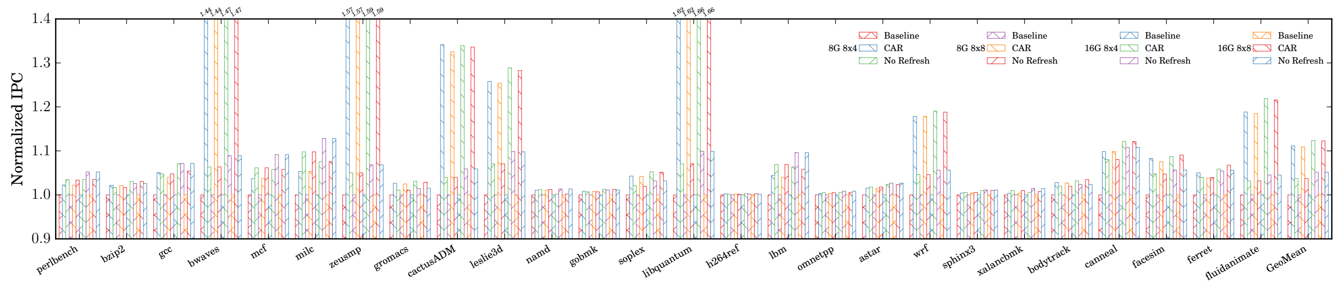

Figure 3: Performance comparison of our approach with the baseline and no refresh systems. Each benchmark has four sets of results and the middle legend in each set represents CAR and labels the memory configuration for the set. CAR can improve performance as much as 1.66× for *libquantum*, with an average of 11.7% (It should be noted that the maximum and average refresh performance penalty is 12.8% and 4.4%, respectively).

refreshed. We also make necessary changes to the refresh mechanism to skip refreshing memory rows containing invalidate data.

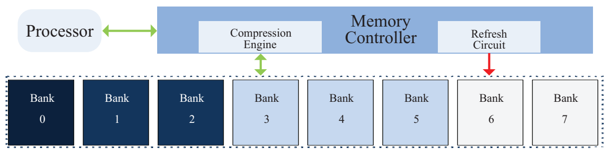

Figure 1: A simplified schematic overview of CAR architecture. Cache line compression results in *sparse* banks in which memory rows may not need refresh. The different colors represent different degrees of sparseness of the bank.

2.2 Refresh Reduction

In order to safely skip refreshing invalid cache lines in a *sparse* bank, we need to track which rows in the bank are valid. To fulfill this purpose, we devise a two-level vector data structure. As it is shown in Figure 2, the first-level is called *refresh indicator vector*, an n-bit vector. Each bit of this vector corresponds to a row in the bank and when the bit is set, the corresponding row is indicated as valid and needs refresh. The second-level is called *utilization vector*, an $(n*m)$-bit vector. Every m-bit acts as a counter to remember the number of valid cache lines the corresponding row currently contains. Whenever a compressed cache line takes space from a bank, the corresponding row counter of the bank is incremented by one. On the other hand, whenever a cache line becomes short (due to overwriting) and needs less space, then the counter corresponding to the released row is decremented by one. When a counter decreases to zero, meaning the corresponding row contains no valid cache lines, the corresponding row bit in the first-level vector is reset to 0, indicating to the memory controller to skip refreshing the corresponding row. The changes in compressed size cause counter values to be changed. Row granularity refresh controlling is possible as retention time based refresh approaches rely on similar row granularity refresh [2, 5]. When a bank receives a refresh command to refresh a set of rows, it checks the first-level vector to see which rows do not need to be refreshed. For those rows, it simply increments the refresh counter and does not perform refresh, as what a "dummy refresh" [5] does.

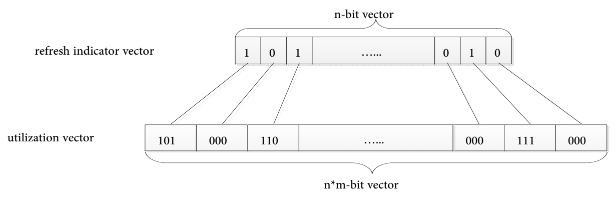

Figure 2: A two-level data structure used to efficiently track rows that do not need refresh. The first-level vector indicates which rows need refresh. The second-level vector tracks the row utilizations.

Assume an 8×8 memory system and the row size is 4KB. Then a 64-byte cache line is evenly distributed among eight banks in the eight different chips. Each bank takes 8 bytes of the cache line. Therefore, a 4KB row in a bank contains data from $\frac{4KB}{8B} = 512$ cache lines, resulting in each row counter in the *utilization vector* requiring $\log_2 512 = 9$ bits (i.e., $m = 9$). Besides, it requires one more bit for each bank row in the *refresh indicator vector*. In total, this two-level vector data structure incurs 10 bits overhead for each 4KB bank row.

3. PRELIMINARY RESULTS

We compare our approach with a no-compression, no rank subsetting, auto refresh (1x FGR mode) baseline memory system and a no-compression, no rank subsetting, no-refresh ideal memory system. Instruction Per Cycle (IPC) metric is employed to compare the performance of our approach *CAR* with the baseline and no refresh systems. Figure 3 demonstrates the normalized IPC comparison results. As it is shown in the figure, CAR is able to improve performance as much as 1.66× for *libquantum*, with an average of 11.7%. Our approach can efficiently alleviate refresh performance penalty. Generally speaking, non-intensive applications suffer from refresh penalty less than intensive applications, as fewer requests are blocked. More compressible and intensive applications benefit more from CAR.

4. ACKNOWLEDGMENTS

The research is supported in part by the National Natural Science Foundation of China (NSFC) under grant No.61232004, 61502189, and 61572012, the Education Ministry of Hubei Province of China under No. B2015054 and U.S. NSF under grants CNS1218960 and CCF1547804.

5. REFERENCES

[1] M. K. Qureshi, D. H. Kim, S. Khan, P. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[2] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[3] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," in *PACT*, 2012.
[4] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency," in *MICRO*, 2008.
[5] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob, "Flexible Auto-Refresh: Enabling Scalable and Energy-Efficient DRAM Refresh Reductions," in *ISCA*, 2015.