Content Aware Refresh: Exploiting the Asymmetry of DRAM Retention Errors to Reduce the Refresh Frequency of Less Vulnerable Data

Shibo Wang, Mahdi Nazm Bojnordi, Xiaochen Guo, and Engin Ipek.

Abstract—DRAM refresh is responsible for significant performance and energy overheads in a wide range of computer systems, from mobile platforms to datacenters [1]. With the growing demand for DRAM capacity and the worsening retention time characteristics of deeply scaled DRAM, refresh is expected to become an even more pronounced problem in future technology generations [2].
This paper examines *content aware refresh*, a new technique that reduces the refresh frequency by exploiting the unidirectional nature of DRAM retention errors: assuming that a logical **1** and **0** respectively are represented by the presence and absence of charge, **1**-to-**0** failures are much more likely than **0**-to-**1** failures. As a result, in a DRAM system that uses a block error correcting code (ECC) to protect memory, blocks with fewer **1**s can attain a specified reliability target (*i.e.*, mean time to failure) with a refresh rate lower than that which is required for a block with all **1**s. Leveraging this key insight, and without compromising memory reliability, the proposed content aware refresh mechanism refreshes memory blocks with fewer **1**s less frequently. To keep the overhead of tracking multiple refresh rates manageable, refresh groups—groups of DRAM rows refreshed together—are dynamically arranged into one of a predefined number of refresh bins and refreshed at the rate determined by the ECC block with the greatest number of **1**s in that bin. By tailoring the refresh rate to the actual content of a memory block rather than assuming a worst case data pattern, content aware refresh respectively outperforms DRAM systems that employ RAS-only Refresh, all-bank Auto Refresh, and per-bank Auto Refresh mechanisms by 12%, 8%, and 13%. It also reduces DRAM system energy by 15%, 13%, and 16% as compared to these systems.

Index Terms—Computer architecture, memory systems, DRAM.

1 Introduction

A DRAM cell encodes information by the presence or absence of electrical charge on a storage capacitor. The data stored in a cell may be lost because the charge on the capacitor leaks over time. To prevent such *retention errors*, a DRAM cell must be periodically refreshed by sensing and replenishing the stored charge. The refresh operations introduce energy and performance overheads to the DRAM system, which are expected to increase in future high-capacity main memories [1].

In a typical DRAM system, Auto Refresh commands are sent from the memory controller to the DRAM chips. During an Auto Refresh operation, part of the DRAM system is unable to serve memory requests, which increases average memory access latency and degrades memory throughput. An Auto Refresh operation can also interfere with row buffer locality. Figure 1 shows the energy and bandwidth overheads due to refresh on a set of twelve memory intensive applications using 32Gb chips. The results indicate that refresh consumes 33% of DRAM system energy and degrades sustained memory bandwidth by 20%.[1] In a future DRAM system comprising 64Gb chips, refresh operations are expected to reduce memory throughput by 50% and consume 50% of the memory power [2].

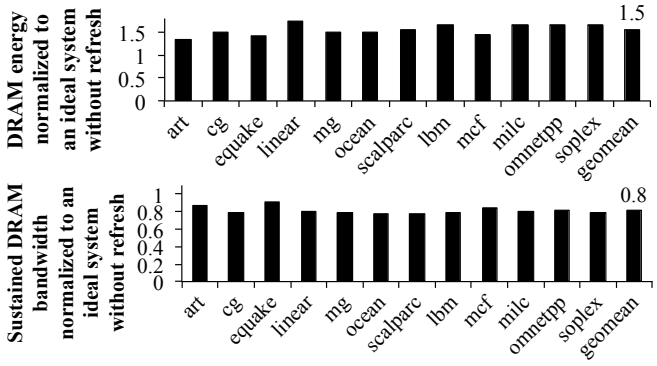


Fig. 1: Effects of refresh operations.

This paper proposes *content aware refresh*, a new technique that reduces the number of refresh operations by leveraging the fact that retention errors are unidirectional. In a DRAM system that uses a block error correcting code (ECC) to protect memory from errors, blocks with fewer **1**s exhibit a lower probability of encountering an error. As a result, to attain a target uncorrectable error probability, an ECC block with fewer **1**s can be refreshed less frequently than a block with more **1**s. Moreover, the lowest refresh rate that can achieve the desired level of reliability can be calculated based on the number of **1**s within the block.

A refresh operation typically is performed simultaneously

- *S. Wang is with Google.*
- *M. N. Bojnordi is with the University of Utah.*
- *X. Guo is with Lehigh University.*
- *E. Ipek is with the University of Rochester.*

1. The experimental setup is explained in Section 6.

on a group of DRAM rows (*i.e.*, on a *refresh group*), which consists of multiple ECC blocks. In the proposed content aware refresh mechanism, the refresh rate of a refresh group is decided based on the most vulnerable ECC block in that group, which is the block with the greatest number of **1**s. Naïvely tracking and enforcing the refresh rate for each refresh group, however, would result in exorbitant energy and storage overheads. Instead, content aware refresh assigns each refresh group into one of a small number of *refresh bins*. Each bin has a single refresh rate, and the assignment of a refresh group to a bin is determined by whether the required refresh rate of the group can be satisfied by the refresh rate of the bin. To reduce the number of refresh operations, both the refresh rates of the bins and the refresh group-to-bin assignments are adaptively changed at runtime.

By avoiding unnecessarily refreshing blocks that are dominated by **0**s, content aware refresh improves both the performance and the energy efficiency. Two types of Auto Refresh are evaluated: *all-bank Auto Refresh* and *per-bank Auto Refresh*. **Content aware refresh improves end-to-end system performance by 8% over all-bank Auto Refresh, and by 13% over per-bank Auto Refresh, with corresponding reductions in DRAM energy of 13% and 16%**. It is also possible to apply the proposed technique to RAS-only Refresh, in which the memory controller refreshes a single DRAM row at a time by issuing a precharge command followed by an activate. When applied to a high capacity DRAM chip, RAS-only Refresh typically exhibits worse performance and energy as compared to Auto Refresh, because RAS-only Refresh does not exploit the internal subarray level parallelism within the DRAM chips. **When content aware refresh is applied on top of RAS-only Refresh, however, RAS-only Refresh achieves 12% higher performance and 15% lower energy as compared to a baseline system that uses all-bank Auto Refresh**.

RAIDR [2] is a recently proposed technique that reduces DRAM refresh operations by exploiting cell-to-cell variations in retention time. When applied to the evaluated baseline system, RAIDR improves system performance by 8%, and saves 11% of the memory energy as compared to all-bank Auto Refresh. Without relying on profiling the retention time of each physical cell as RAIDR does, content aware refresh outperforms all-bank Auto Refresh by 20%, with 20% memory energy savings. Furthermore, content aware refresh is orthogonal to RAIDR, and the two techniques can be combined to achieve even greater benefits. **When content aware refresh is added on top of RAIDR, the combined system achieves 13% higher system performance and 16% lower memory energy as compared to RAIDR alone**.

2 Background and Motivation

This section reviews the basics of DRAM refresh, DRAM errors, and error protection techniques.

2.1 DRAM Refresh

DRAM devices require periodic refresh to preserve data integrity. A typical refresh interval—*i.e.*, time between successive refresh operations to a given cell—is 64ms below 85^{o}C or 32ms for 85-95^{o}C [2]. In a modern DDRx DRAM system, the memory controller spreads out the refresh operations to different DRAM rows evenly over a refresh interval. In *Auto Refresh*, the DRAM device maintains an internal address counter that points to the next group of rows to be refreshed; the memory controller initiates a refresh operation by issuing an Auto Refresh command every $tREFI$ μSec.

Auto Refresh. Auto Refresh can be issued in either an *all-bank* or a *per-bank* fashion.[2] In all-bank Auto Refresh, all of the banks are refreshed during a refresh operation, and the entire rank is unavailable for $tRFC_{ab}$ μSec. Per-bank Auto Refresh refreshes only those rows in the addressed bank and occupies the bank for $tRFC_{pb}$ μSec. By allowing the banks that are not being refreshed to service memory requests, per-bank Auto Refresh offers greater bank-level parallelism.

RAS-only Refresh. RAS-only Refresh is a technique that was supported by earlier asynchronous DRAM devices [5]. Under RAS-only Refresh, the memory controller explicitly issues refresh commands to each DRAM row. Although the DDR standard no longer explicitly supports RAS-only Refresh, RAS-only Refresh can still be implemented by sending a precharge command followed by an activate command to the row that needs to be refreshed. Compared to Auto Refresh, the fine-grained control provided by RAS-only Refresh offers greater flexibility.

2.2 DRAM Retention Errors

A DRAM retention error occurs when the stored data is lost due to charge leakage. The retention time, which is the time between when the data is stored and when a retention error occurs, varies among cells within a device due to process variability. Previous studies [6], [7] show that the retention time of the cells is characterized by a bimodal distribution: 1) over 99% of the cells exhibit a long retention time characterized by a *main distribution*, and 2) the remaining, leaky cells exhibit a much shorter retention time characterized by a *tail distribution*. Those cells that are characterized by the tail distribution determine the DRAM reliability because these include the weakest functional cells in the device. The tail distribution, which governs the relationship between the retention error probability and the refresh interval, has been shown to fit an exponential distribution [7].

2.2.1 Asymmetry of Retention Errors

Notably, DRAM retention errors are asymmetric: assuming that a logical **1** is represented by a charged cell, the probability of a **1**-to-**0** retention error is significantly higher than that of a **0**-to-**1** retention error [8], [9]. Figure 2 illustrates three of the possible leakage paths in a DRAM cell. Subthreshold leakage current can flow into or out of the cell depending on whether the cell and the bitline are charged or discharged (Figure 2 (a)). However, both the gate-induced drain leakage (GIDL) current and the junction leakage current flow from the storage node into the substrate, which discharges the cell. The asymmetry of the DRAM retention errors is due to the fact that the GIDL current (Figure 2 (b)) constitutes the

2. Although current DDRx devices support only all-bank Auto Refresh, LPDDRx type per-bank Auto Refresh support should be non-intrusive because it requires only simple changes, as suggested and evaluated in recent studies [3], [4].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2868338, IEEE Transactions on Computers

dominant leakage mechanism for the weakest DRAM cells (DRAM cells that follow the tail distribution) [10], [11]. For DRAM cells that follow the main distribution, the junction leakage (Figure 2 (c)) constitutes the main leakage path [10], [11].

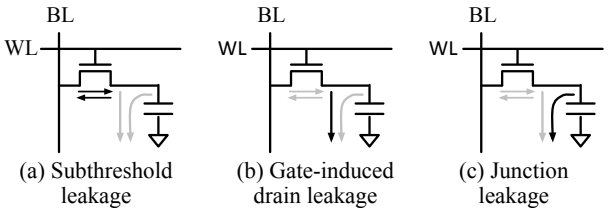


Fig. 2: Leakage paths within a DRAM cell.

Patel *et al*. [9] leverage this unidirectional feature of the DRAM retention errors to avoid refreshing row or column segments that contain only zeroes in DRAM arrays. This approach adds special circuitry to DRAM arrays to prevent blocks with only zeros from consuming energy during refresh. However, it cannot improve system performance since the number of issued refresh operations is not reduced. In addition, it requires nontrivial DRAM array modifications. In contrast, by modeling the refresh rate as a function of the data content, the proposed content aware refresh approach can improve both the system performance and energy, without changing the DRAM array structure.

Emma *et al*. [8] note the unidirectional nature of retention errors and use the Berger code [12] to detect the errors in eDRAM caches with low overhead. However, the Berger code cannot correct unidirectional errors, and this refresh reduction scheme cannot be applied to conventional DRAM systems.

2.2.2 True- and Anti-cells

In DRAM circuit design, a *true-cell* and an *anti-cell* respectively refer to DRAM cells that represent a logical **1** with the charged and discharged states [13]. Naturally, in DRAM ICs that comprise a mixture of true- and anti-cells, the asymmetry between the **1**-to-**0** and **0**-to-**1** retention error probability may be absent or diminished [14]. For these memory ICs, content aware refresh can still be applied to true- and anti-cell regions individually. For inverted bits stored in anti-cells, **0**s and **1**s will be treated respectively as **1**s and **0**s while counting the number of **1**s in the stored data blocks [3]. For convenience and without loss of generality, the following discussion and the evaluation are based on the true-cell regions.

2.2.3 Retention Time Variation

Previous studies exploit retention time variation among different DRAM cells to reduce the number of refresh operations. Similarly, RAIDR [2] groups DRAM rows into different bins based on the weakest cell in each row, and refresh each bin at a different rate. **Unlike these proposals, content aware refresh conservatively assumes the worst case retention time (*i.e.*, that of the weakest DRAM cell) for all of the DRAM cells.** Prior work that exploits retention time variations can be combined with content aware refresh to reduce the number of refresh operations further. A detailed comparison of content aware refresh to RAIDR can be found in Section 7.5.

2.2.4 Effect of Temperature

By accounting for the temperature dependence of DRAM retention times [6], some mobile DRAMs provide temperature compensated self refresh mechanisms that reduce the refresh overheads [15]. These schemes are orthogonal to the proposed approach and can be combined with it. Specifically, content aware refresh is able to set the default refresh rate based on the temperature sensor within a DRAM module, and scales the refresh rate of each bin accordingly (Sections 3 and 4).

2.3 DRAM Non-retention Errors

DRAM reliability is also affected by other types of errors, which are considered by content aware refresh. A radiation induced soft error [16] can occur in a DRAM cell. When alpha particles and cosmic rays penetrate the silicon substrate, they can generate electron-hole pairs along the path that they traverse. The generated electrons, when collected by drift, can discharge a cell capacitor and turn a previously stored **1** into a **0** [16]. The radiation induced soft errors can also occur due to other failure mechanisms such as the ALPEN effect [17], bitline strikes, and strikes at DRAM peripheral circuits (such as sense amplifiers and registers), which may result in either a **1**-to-**0** or a **0**-to-**1** error.

DRAM interference errors caused by noise coupling have been extensively studied [18]. Recent studies [13] find that DRAM disturbance errors can occur when a nearby row is repeatedly activated. The cumulative effect of repeated interference from a wordline can accelerate charge leakage from the nearby cells due to coupling effects. The error probability increases with an increased row activation rate. MEMCON [19] proposed techniques to detect and mitigate errors caused by coupling noise based on online testing. Techniques for addressing DRAM coupling noise-induced errors [19], [20] and the DRAM "row hammer" problem [13] are orthogonal to content aware refresh, and can be combined with it in designing energy-efficient, reliable DRAM systems.[4]

Hard errors also contribute a large fraction of the DRAM errors observed in the field [21]. These errors are caused by circuit defects or wearout, such as metal electromigration, metal stress migration, time dependent dielectric breakdown, and thermal cycling, and are independent of DRAM retention errors.

2.4 DRAM Error Protection

ECC-enabled commodity DRAM modules are employed to maintain system reliability. Typically, the (72, 64) Single-Error-Correction, Double-Error-Detection (SECDED) ECC is used to correct up to one error and detect up to two errors in each 72-bit word. More complex ECCs (*e.g.*, the Reed-Solomon codes [22]) can also be used to provide chipkill

3. The internal architecture of true- and anti-cells is vendor specific and usually unknown as a proprietary of each DRAM vendor. Further analysis and research will leave as future work.

4. If the default refresh rate increases due to DRAM disturbance errors (as suggested by solutions in [20]), the refresh rates in the proposed approach must be increased accordingly.

protection, tolerating an entire memory chip failure [23] without data loss. Wilkerson *et al*. [24] use ECC to reduce the refresh rate in eDRAM caches. However, the employed ECC is much stronger than the SECDED ECC, and is applied to large codewords (1KB) to amortize the cost. The proposed content aware refresh approach can work with existing (72, 64) SECDED ECC, which is widely adopted by modern DRAM DIMMs.

3 Key Insight: Sparse Blocks Can Be Refreshed Less Frequently

We define an ECC block with a large Hamming weight (*i.e.*, with more ones) as a *dense block*, and one with a small Hamming weight as a *sparse block*. Since a sparse block contains fewer vulnerable bits than a dense one, it is possible to refresh a sparse block less frequently than a dense block while achieving the same block error rate.

To model the reliability of a data block protected by SECDED ECC, the DRAM errors are categorized as retention errors, and non-retention errors that are independent of any retention issues (*e.g.*, soft errors induced by alpha particles and cosmic rays, and different types of hard errors caused by defects or wearout). With SECDED ECC, the reliability of an ECC block, $R(h, p)$, can be expressed as the probability of observing no more than one error:

$$\begin{aligned} R(h,p) =& (1-p)^h(1-p_{\text{non-ret}})^n + \\ & hp(1-p)^{h-1}(1-p_{\text{non-ret}})^n + \\ & hp(1-p)^{h-1}p_{\text{non-ret}}(1-p_{\text{non-ret}})^{n-1} + \\ & np_{\text{non-ret}}(1-p_{\text{non-ret}})^{n-1}(1-p)^h, \end{aligned} \tag{1}$$

where p is the probability that a retention error occurs at a bit position, $p_{\text{non-ret}}$ is the probability that a non-retention error occurs at a bit position, n is the number of bits within the ECC block, and h is the Hamming weight of the ECC block. Of the four terms that are summed, the first term represents the probability that the block has no error, which requires each bit within the block to be free of errors (both retention and non-retention ones). The second, third, and fourth terms of the sum together represent the probability that the block has a single error: the second term gives the probability that the block has a single retention error but no non-retention error; the third term represents the probability that a retention error and a non-retention error overlap on the same bit [5]; the fourth term represents the probability that the block has a single non-retention error but no retention error. Since only those bit positions that contain ones are vulnerable to the unidirectional retention errors, the Hamming weight (h) of the block is used instead of the total number of bits (n) in a block when calculating the error probabilities involving a retention error. In a conventional system, the refresh rate to achieve a specified reliability target is designed for the worst case, in which all of the bits in an ECC block are assumed to store **1**s. This refresh rate, however, is overly conservative for the blocks that contain fewer **1**s. Under a constant refresh rate, the reliability achieved by each block depends on its content. It is possible to achieve the same reliability for every ECC block by satisfying the following equation:

$$R(h, p_h) = R(n, p_n), \tag{2}$$

where p_n is the required retention error probability for an ECC block with all **1**s, and p_h is the required retention error probability for an ECC block with h **1**s. When the Hamming weight is reduced from n to h, we can achieve a higher tolerable retention error probability ($p_h \geq p_n$), and therefore a lower refresh rate. Figure 3 illustrates the relationship between the uncorrectable error probability and refresh rate for ECC blocks with different Hamming weights. The uncorrectable error probability is equal to one minus the probability of observing a SECDED ECC block with no more than one error. For a fixed reliability target [6], the ECC block with the lower Hamming weight requires a lower refresh rate.

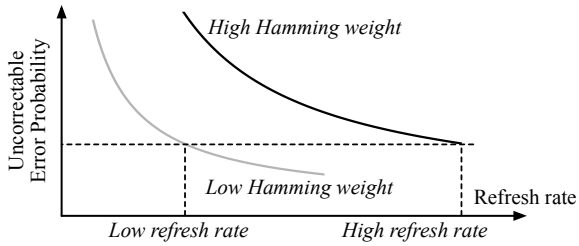


Fig. 3: The uncorrectable error probability of an ECC block vs. refresh rate.

Since the probability of a DRAM retention error is a function of the refresh rate (Section 2.2), the explicit relationship between the required refresh rate and the Hamming weight of an ECC block can be obtained by solving Equation (2). The (72, 64) SECDED ECC, which is employed in both the conventional and the proposed DRAM systems, was used to generate the data plotted in Figure 4. The retention error probability per bit for the conventional system, p_n, is assumed to be 10^{-12} based on published results [2]. $p_{\text{non-ret}}$ is set to 5×10^{-8}, which is obtained from published data gathered from DRAM field studies [21]. As can be seen in Figure 4, the required refresh rate is an approximately linear function of the Hamming weight, which provides an opportunity to reduce the DRAM refresh overheads by making refresh content-aware.

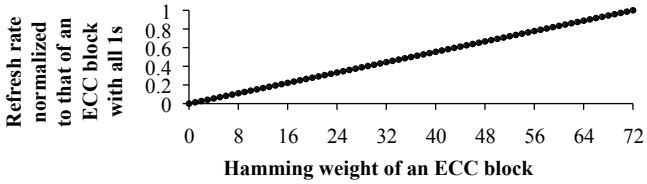


Fig. 4: Refresh rate vs. Hamming weight.

5. We assume that there is a single error when a non-retention error occurs at the same bit as a retention error.

6. Content aware refresh uses the uncorrectable error probability as the reliability target, and achieves the same target as a conventional system by satisfying Equation (2). This requirement allows the proposed approach to provide the same detectable error probability as a conventional system. If the reliability target changes, the $R(h, p)$ function in Equation (1) can be changed accordingly.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2868338, IEEE Transactions on Computers

4 Overview

In content aware refresh, DRAM is organized into multiple refresh groups, each of which is refreshed at one of multiple refresh rates. A refresh group contains a single row for RAS-only Refresh, and multiple rows for Auto Refresh. Figure 5 shows an example DRAM system with two rows, each of which contains data blocks with different values. The dense block requires the default, highest refresh rate, whereas the sparse block needs to be refreshed at only one fourth of that rate. The figure illustrates how conventional refresh and content aware refresh are used by the DRAM system.

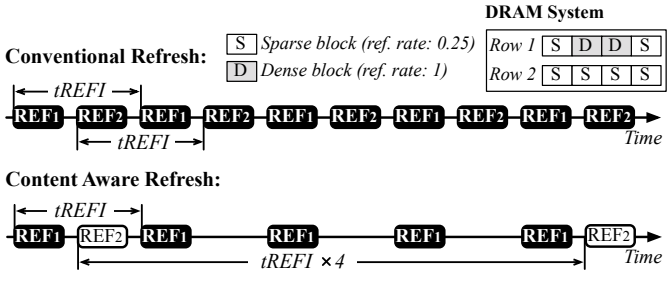


Fig. 5: Illustrative example of the key idea.

Conventional refresh is oblivious to the contents of the memory blocks; as a result, it is equivalent to a DRAM system in which all of the rows are refreshed using the default, shortest refresh interval ($tREFI$). Content aware refresh maintains two refresh rates: DRAM rows that contain a dense block need to be refreshed at the highest default rate, whereas rows that contain only sparse blocks can be refreshed at one fourth of the default refresh rate. Because the densest block within row 1 requires the highest refresh rate, under content aware refresh, row 1 requires the same refresh interval ($tREFI$) as it does under conventional refresh. However, the densest block in row 2 is a sparse block; consequently, row 2 can be refreshed using an interval of $tREFI \times 4$.

The densest ECC block within each refresh group varies during the execution. To achieve refresh reduction without compromising reliability, each refresh group must be refreshed based on the Hamming weight of its densest block. However, tracking and enforcing the refresh rate for every refresh group individually would result in significant storage and energy overheads. Instead, content aware refresh further arranges the refresh groups into different refresh bins based on the Hamming weight of the densest block within a refresh group, and refreshes each bin at its own rate as determined by the Hamming weight of the densest block in that bin.

Figure 6 illustrates an example system using the proposed architecture. A content aware refresh unit, comprising a metadata cache, a metadata update unit, a binning threshold selection unit, and a refresh scheduler, determines the refresh rate of each bin and schedules refresh requests. The metadata cache contains a small portion of the metadata that is stored in a dedicated region of the main memory. The metadata update unit works with the metadata cache to keep track of the densest block within each refresh group. The Hamming weight of the densest block is used to determine the bin to which a refresh group belongs. The binning threshold selection unit periodically recomputes the binning thresholds to adapt to program behavior as data are written into the memory. The refresh scheduler is responsible for generating and scheduling the refresh requests for each bin based on the assigned refresh rate.

When a write request is received, in addition to inserting the request into the request queue, the memory controller sends the address of the request to the metadata cache to obtain the metadata. The controller also sends the new contents (which are to be written to memory) to the metadata update unit (step ① in Figure 6). The metadata returned by the metadata cache is sent to the metadata update unit, which checks whether the new contents require a metadata update (step ② in Figure 6). If so, the updated metadata is written back to the metadata cache, and is also sent to the binning threshold selection unit to keep up-to-date information there (step ③ in Figure 6). To adapt to the changing contents of main memory and to find the optimal binning thresholds over time, the binning threshold selection unit periodically recomputes the thresholds for each bin, and sends them to the refresh scheduler (step ④ in Figure 6). Given the assigned binning thresholds and the up-to-date metadata, the refresh scheduler dynamically determines the refresh bin that the next refresh group belongs to, generates the refresh requests at the appropriate times, and inserts them into the refresh request queue (step ⑤ in Figure 6).

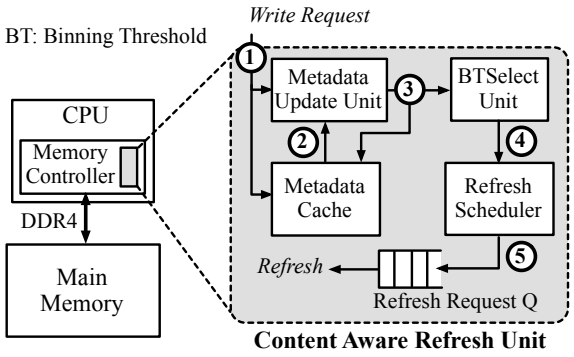


Fig. 6: An example system using the proposed content aware refresh mechanism.

5 Content Aware Refresh

Content aware refresh maintains metadata to track the dense blocks. Based on the metadata and the binning thresholds, refresh requests for each bin are dynamically generated and scheduled. To permit the binning process to adapt to the changes in the contents of the DRAM system over time, the proposed approach periodically reselects the binning thresholds.

5.1 Tracking the Dense Blocks

Tracking the densest block in each refresh group during program execution is critical to deciding the refresh rate of that group, since the Hamming weight of the densest block determines the lowest refresh rate that satisfies the reliability requirements.

5.1.1 Metadata Organization

An efficient metadata organization should maintain a minimum amount of data to provide accurate information for binning threshold selection and refresh group assignment. The proposed metadata organization is designed to track only the densest block within each refresh group. The metadata of each refresh group consists of the Hamming weight of the densest ECC block within that group (DB), the location of the *weakest line* (*i.e.*, a cache line sized region of data that contains the densest ECC block) within the refresh group, and a valid bit indicating whether the metadata is up to date. Initially (at boot time), there is no useful data stored in the refresh groups, and all of the DB values are invalid. Consequently, all of the refresh groups are initialized at the lowest refresh rate. The metadata are updated as the DRAM pages are written.

The metadata storage capacity depends on the number of refresh groups in a DRAM device. We evaluated DRAM chips from 8Gb to 32Gb. The storage overhead of metadata is less than 7MB for a 72GB memory system (less than 0.01%).

5.1.2 Updating the Metadata

Figure 7 shows how the metadata of a refresh group is updated in response to a write request. If the new data contains an ECC block that is denser than the current densest block within the corresponding refresh group, the update procedure is simple: 1) the Hamming weight of the densest block and the location of the weakest line are overwritten by the new values, and 2) the valid bit is set. This mechanism guarantees that the increase in the refresh rate of a refresh group is accurately reflected in the metadata.

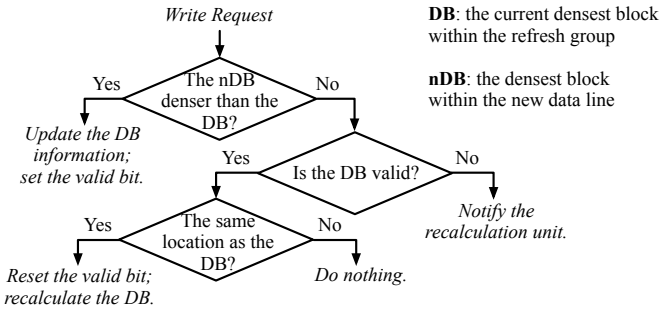


Fig. 7: Procedure for updating the metadata.

Decreasing the Hamming weight of the densest block within a refresh group requires *weight recalculation*, which involves reading out the entire refresh group to identify the new densest block within the refresh group.[7] To avoid unnecessary weight recalculations, the valid bit and the location of the densest block are checked. If the valid bit is not set, indicating that the densest block recalculation for the refresh group may already be in progress, the new data must be included in the ongoing weight recalculation. If the metadata is valid, the location of the new data line within the refresh group is compared to that of the current densest block. If the locations do not match, the metadata does not need an update, because the new data does not affect the current densest block within the refresh group. Otherwise, the valid bit is reset to indicate the need for a recalculation. While the recalculation is in progress, the location of the densest block is kept at its original value to ensure correctness. To reduce the performance impact of weight recalculation, the memory requests issued for weight recalculation wait in a queue and are opportunistically scheduled at times when the memory bus is idle. Simulation results on twelve memory intensive benchmarks indicate that the additional DRAM accesses increase total memory traffic by less than 1%. Moreover, the number of DRAM cycles consumed by the additional memory requests is $5000\times$ smaller than the number of DRAM cycles saved by reducing the refresh rate.

5.1.3 The Metadata Cache

A small, dedicated region of main memory is allocated for the metadata by setting the kernel parameter *memmap* in the boot loader's configuration file. Accessing the metadata region in main memory on every write request would significantly degrade performance and energy. A metadata cache is therefore proposed and placed inside the memory controller, which is 4-way set associative, write-allocate, managed with an LRU replacement policy, and less than 7KB in all of the evaluated configurations. Each metadata read brings in a block of data that contains metadata for 18 to 36 refresh groups.[8] Due to the high spatial and temporal locality in metadata accesses, the metadata cache has a high hit rate (from 94% to 99% depending on the application), which significantly reduces the extra memory requests caused by metadata accesses and the associated overheads. A sensitivity analysis on the size of the metadata cache is presented in Section 7.6.1.

5.2 Refresh Scheduling

Each refresh bin has a single refresh rate, and is controlled by a dedicated refresh scheduler (Figure 8). Each refresh scheduler needs to identify the refresh groups that belong to the corresponding bin, and refresh them at a specified rate. Tracking the addresses of all of the relevant refresh groups is impractical since the refresh groups that belong to a particular bin can reside in arbitrary locations within the memory. Instead, each refresh scheduler independently traverses the entire memory space at a pace determined by the refresh rate, and refreshes only the groups that are assigned to its bin.

A refresh scheduler comprises a *timer*, an *address counter*, a *densest block Hamming weight buffer*, and a *refresh request generator* (Figure 8). The timer controls the refresh rate. The address counter sequentially traverses the refresh groups in memory, pointing to the address of the refresh group that is going to be refreshed next if it is assigned to this bin. The densest block Hamming weight buffer stores the Hamming weights of the densest blocks within the refresh groups, which are filled by sequentially prefetching the metadata. The prefetching overhead is amortized over multiple refresh operations (*e.g.*, one 64B prefetch operation is required every 85 all-bank Auto Refresh operations). The refresh generator compares the binning thresholds with the Hamming weight

7. Notably, the bit position of a 1 does not affect the error probability of the ECC block.

8. The precise number of refresh groups that can be covered by a fixed size metadata block depends on the size of the refresh group.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2868338, IEEE Transactions on Computers

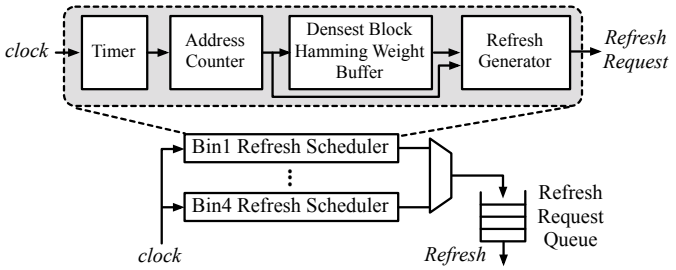


Fig. 8: Illustrative example of the refresh scheduler with four bins.

of the densest block within a refresh group to decide whether the candidate refresh group is assigned to this bin, and thus should be refreshed. If so, a refresh request is generated and sent to the refresh request queue, which employs an FCFS policy.

5.3 Selecting the Binning Thresholds

Selecting the right binning thresholds can significantly increase the benefits obtained from content aware refresh. The evaluation indicates that compared to simple fixed binning with evenly distributed thresholds, the optimal binning scheme proposed here can reduce the number of refresh operations by 12-28% (Section 7.6.3). Content aware refresh employs dynamic programming to generate these optimal binning thresholds.

5.3.1 Selection Algorithm

The goal of the binning threshold selection is to minimize the total number of refresh operations generated by each refresh bin. Because the refresh rate is a linear function of the Hamming weight (Section 3), this goal can be reformulated as the problem of finding all of the binning thresholds by minimizing

$$\sum_{i=0}^{N-1}(y(t_{i+1})-y(t_i))t_{i+1}, \quad (3)$$

where N is the number of refresh bins, t_i is the i^{th} binning threshold, and $y(t_i)$ is the total number of DRAM refresh groups whose densest block has a Hamming weight less than or equal to t_i. Since the number of bits in an ECC block, t_N, and the total number of refresh groups in the DRAM system, $y(t_N)$, are both constants, this optimization problem can in turn be expressed as the problem of maximizing the *left Riemann sum* [25].

We propose a simple dynamic programming algorithm to solve the problem. The proposed algorithm derives the maximum left Riemann sum with N thresholds by examining the solutions to a set of subproblems. The maximum left Riemann sums with one, two, ... , N thresholds are computed sequentially, by memoizing and using the results obtained in a given step in the next step of the algorithm. Figure 9 shows how the proposed algorithm finds the optimum binning thresholds for a simple example. Three binning thresholds (t_1-t_3) need to be selected in this example (t_3 is forced to take on the value of the largest Hamming weight, 5). Each node $A[h,i]$ represents the maximum left Riemann sum computed for t_0-t_i, and h represents the Hamming weight threshold selected for t_i. The proposed algorithm comprises a *forward* step, and a *backtracking* step. In the forward step, the left-most column of nodes is initialized; each remaining column is then computed sequentially, based on the previous column (Figure 9-a). After computing the column t_3, the best binning thresholds are chosen in the backtracking step (Figure 9-b). In this example, Hamming weights 5, 4, and 1 are identified as the optimal binning thresholds for t_3, t_2, and t_1.

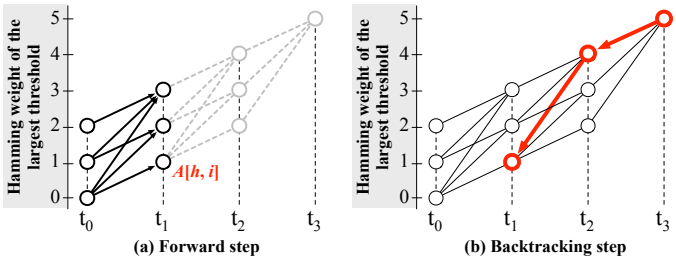


Fig. 9: Illustrative example of the proposed dynamic programming algorithm.

5.3.2 Hardware Implementation

The proposed binning threshold selection is performed in hardware at the memory controller, as shown in Figure 10. The hardware implementation relies on a cumulative histogram table y, a maximum value table A, and a tracking table T that stores the thresholds that can achieve the optimum value for each subproblem. To generate the optimal solution to each subproblem $A[h,i]$, the binning threshold selection unit sequentially reads out the result of each relevant subproblem, performs the computation, and sends the result to the max A update unit (step ①), which keeps track of the maximum result and the corresponding Hamming weight (step ②). Table A and T are updated based on the values stored in max A and the best Hamming weight registers (step ③). When all of the subproblems have been solved, the tracking table is used to trace back the series of binning thresholds that lead to the optimum solution. The proposed binning threshold selection unit requires less than 1KB of SRAM. The energy, latency, and area overheads are evaluated by synthesizing an RTL model.

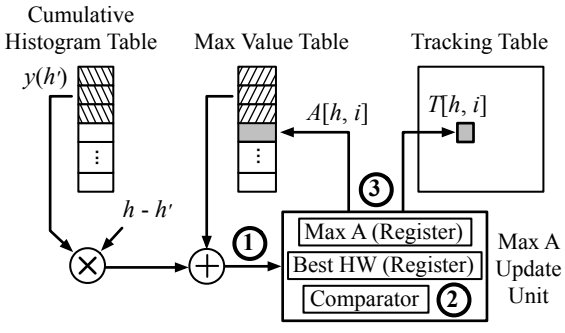


Fig. 10: Binning threshold selection.

5.3.3 Periodic Adjustment of the Binning Thresholds

To maximize the benefits, the binning thresholds should be adjusted at runtime based on the phase behavior of

the applications. A set of counters is used to track a histogram recording the Hamming weight of the densest block in each refresh group. To amortize the overhead of the binning threshold selection and assignment, the binning thresholds are re-calculated periodically. At the end of each execution interval, the binning threshold selection unit fills the cumulative histogram table (Figure 10) based on the latest values of the histogram counters, calculates a new set of optimum binning thresholds in the background, and sends the new binning thresholds to the refresh scheduler [9].

We conduct a thorough sensitivity study to find an appropriate execution interval for each application. Notably, the best execution interval varies among different applications. For simplicity, we choose a single interval (512 million cycles) that results in the lowest DRAM energy across all of the evaluated benchmarks. The evaluation indicates that the area, latency, and energy consumption overheads of periodically adjusting the thresholds are not significant.

5.4 Applying Content Aware Refresh on Top of RAS-only Refresh and Auto Refresh

Content aware refresh can be combined with either RAS-only Refresh or Auto Refresh. For RAS-only Refresh, the refresh group contains a single DRAM row. Because the refresh group of Auto Refresh contains multiple rows, Auto Refresh is more likely to encounter a dense block in a refresh group as compared to RAS-only Refresh. However, Auto Refresh exhibits significantly lower metadata overhead than RAS-only Refresh. In addition, recent studies [4] show that as DRAM density increases, RAS-only Refresh exhibits lower performance and consumes more energy as compared to Auto Refresh. This is because RAS-only Refresh cannot simultaneously refresh multiple rows that reside in different subarrays as Auto Refresh does, and consumes greater command bus bandwidth.

In order to apply the proposed approach to Auto Refresh, the memory controller must tell the DRAM system where to start refreshing. Since the address bus is not used by the Auto Refresh command, the memory controller can use it to write the start address of each Auto Refresh into an internal address counter, and the internal counter can automatically point to the rest of the rows that need to be refreshed as part of the refresh operation (which is a built-in feature of Auto Refresh).

5.5 Applicability to Memory Systems with Data Scrambling

Some memory controllers [26] incorporate *data scrambling* to reduce the power supply noise due to successive **1**s and **0**s on the data bus [27]. **Content aware refresh is compatible with data scrambling**. Given a system with data scrambling, content aware refresh relies on the Hamming weight of each scrambled data block (rather than the original one) to determine the refresh requirements. Notably, content aware refresh is able to reduce the number of refresh operations as long as the data are not all **1**s. Since data scrambling tries to balance the number of **1**s and **0**s, it leaves significant room for the proposed approach to achieve performance and energy benefits. A detailed evaluation of content aware refresh in the presence of data scrambling is presented in Section 7.4.

9. Before setting the new binning thresholds, each bin needs to be refreshed at the default refresh rate for one iteration to make sure that refresh group meets the upcoming deadline.

6 Experimental Setup

Analyzing the performance, energy, and the area of content aware refresh requires both architectural and circuit-level design and evaluation.

6.1 Architecture

We use a heavily-modified version of the SESC simulator [28] with a cycle accurate DRAM model for evaluation. The configuration parameters are shown in Table 1. The following systems are modeled and evaluated: 1) a baseline system with the all-bank Auto Refresh (AB); 2) a baseline system with the per-bank Auto Refresh (PB); 3) content aware refresh applied on top of the all-bank Auto Refresh (CAAB), the per-bank Auto Refresh (CAPB), and the RAS-only Refresh (CAROR); 4) an ideal system that eliminates refresh (IDEAL); 5) RAIDR; and 6) content aware refresh applied on top of RAIDR (CAAB + RAIDR, CAPB + RAIDR, and CAROR + RAIDR).

TABLE 1: Evaluated system configuration.

Core	8 out-of-order cores, 3.2GHz, fetch/issue/commit width 4/4/4
IL1 cache	32KB, direct-mapped, 64B line
DL1 cache	32KB, 4-way, 64B line
Coherence	Snoopy bus with MESI protocol
L2 cache	4MB, 8-way, 8 banks, 64B line
Memory controller	FR-FCFS scheduling, open-page policy, page-interleaved address mapping, content aware refresh for the proposed system, all-bank Auto Refresh and per-bank Auto Refresh for baseline systems
DRAM (Timing parameters are in DRAM cycles)	DDR4-1600, 2 channel, 2 ranks/channel, 16 banks/rank, 8KB rows, CL = 10, WL = 9, $tRCD$ = 10, tRP = 10, $tRAS$ = 28, tRC = 38, $tFAW$ = 20, BL = 8, $tREFI$ = 3120, $tRFC_{ab}$ = 280/424/712 (8/16/32Gb DRAM chips), $tRFC_{pb}$ = 70/106/178 (8/16/32Gb DRAM chips)

All of the systems are evaluated with shared DRAM systems that employ the widely used (72, 64) SECDED ECC with 64B last level cache blocks (which matches the DRAM data access granularity). A 32ms retention time is used for the evaluation, which is a typical setting for servers [3]. Because commodity DDR4 DRAM does not support per-bank Auto Refresh, we estimate the $tRFC_{pb}$ values based on prior work [3]. Other timing parameters are also in line with prior work [1], [3], [4]. McPAT 1.0 [29] is used to estimate the processor energy at the 22nm technology node. DRAM system energy is calculated following the methodology described in [30], with parameters taken from [31].

6.2 Synthesis

The metadata update unit, the refresh scheduler, and the binning threshold selection unit are designed and verified in Verilog RTL, and synthesized with the FreePDK 45nm library [32] using the Synopsys Design Compiler [33]. The area, timing and power numbers are scaled from 45nm to 22nm process technology using the scaling parameters reported in prior work [34], [35]. The metadata cache is modeled by CACTI 6.0 [36]. FabMem [37] is used to model the buffers within the refresh scheduler.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2868338, IEEE Transactions on Computers

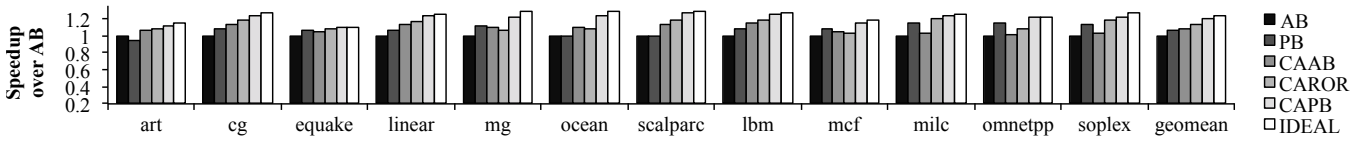


Fig. 11: Performance of the evaluated systems.

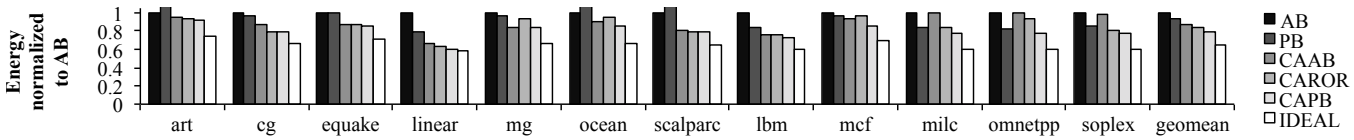


Fig. 12: Energy consumption of the evaluated memory systems.

6.3 Applications

We evaluate twelve applications that are readily portable to our simulator from the NuMineBench [38], Phoenix [39], NAS [40], SPLASH-2 [41], SPEC OMP [42], and SPEC 2006 [43], as shown in Table 2.

The memory footprint of the evaluated applications varies from 20MB to 460MB. Pages with all values equal to zero constitute 13% of the total memory space on average, which is similar to the results shown in prior work [44].[10] We use SimPoint [45] to find a representative 1 billion instruction region from each SPEC2006 benchmark to reduce the simulation time. The memory is warmed up prior to simulation, and the initial binning thresholds are computed based on the data stored in the memory at the end of the warm-up interval.

TABLE 2: Applications and data sets.

Benchmarks	Suite	Input
ScalParc	NuMineBench	A32-D125K
Linear Regression	Phoenix	100MB key file
Ocean	SPLASH-2	514×514 ocean
CG	NAS OpenMP	Class A
MG	NAS OpenMP	Class A
Art-Omp	SPEC OpenMP	MinneSpec-Large
Equake-Omp	SPEC OpenMP	MinneSpec-Large
Lbm	SPEC 2006	Reference
Mcf	SPEC 2006	Reference
Milc	SPEC 2006	Reference
Omnetpp	SPEC 2006	Reference
Soplex	SPEC 2006	Reference

7 Evaluation

In this section, we compare the performance and energy of the proposed architecture to the baseline systems. We also analyze the latency, area, and power overheads of the components in the content aware refresh unit.

7.1 Performance

Figure 11 presents the system performance of all of the evaluated systems with 32Gb DRAM devices. On average, CAAB, CAROR and CAPB respectively outperform the AB baseline by 8%, 12%, and 20%. Most of the performance improvement comes from the reduction of refresh operations (Figure 13). However, as CAPB is built on top of per-bank Auto Refresh, it also benefits from the increased bank level parallelism. Since PB has a 7% performance improvement over AB, the actual speedup achieved by content aware refresh for the CAPB system is 13%.

Among the three systems that use content aware refresh, CAPB performs the best, which exhibits less than 5% performance degradation as compared to the IDEAL system. When the underlying refresh mechanism is changed from RAS-only Refresh to per-bank Auto Refresh and all-bank Auto Refresh, the granularity of the refresh group becomes coarser, which increases the probability of finding a dense block within each refresh group. Since the required refresh rate of each group is determined based on the Hamming weight of the densest block within the group, the likelihood of reducing the refresh rate is lower for a larger refresh group. In Figure 13, we see that applications such as `mcf` and `soplex` exhibit this behavior, especially with the CAAB system.

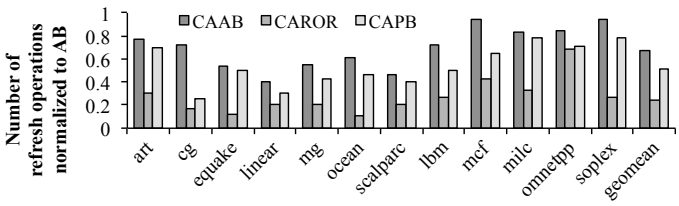


Fig. 13: Total number of refreshes.

7.2 Energy

The energy consumption of the memory system (DRAM plus the memory controller) is shown in Figure 12. CAAB, CAROR, and CAPB respectively reduce the energy consumption of AB by 13%, 15%, and 20%. The energy reduction comes from both the performance improvement and the refresh reduction. The total energy overhead of the additional hardware and the extra memory requests generated by content aware refresh represent less than 1% of the memory system energy for CAAB, CAROR, and CAPB.

Figure 14 illustrates the total system energy consumption broken down between the processor (which includes the cores and the shared L2 cache) and the memory system. CAAB, CAROR and CAPB respectively reduce system energy of AB by 6%, 9%, and 12%. In general, the energy reduction due to the memory system is greater than that achieved on the processor.

10. We assume that the data residing in physical pages that are unused by an application has the same Hamming weight distribution as that of the physical pages that are used by the application.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2868338, IEEE Transactions on Computers

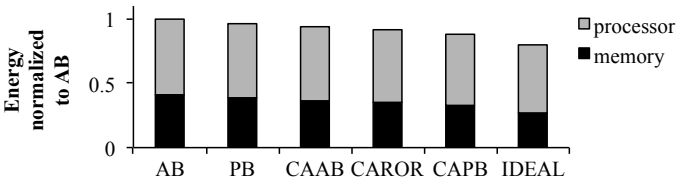


Fig. 14: Total system energy consumption.

7.3 Delay, Area, and Power Overheads

Table 3 lists the latency, area, and power overheads of different components within the content aware refresh unit. The respective sizes of the metadata cache are 0.75KB, 1.25KB, and 7KB for CAAB, CAPB, and CAROR. All of the data shown in the table is based on a configuration with 16 bins. As the number of bins decreases, the overhead decreases as well. Since CAAB, CAPB, and CAROR support different refresh group granularities, the overhead of the metadata cache and the refresh scheduler differ significantly. Although the latency of the binning threshold selection unit is large, this latency is not on the critical path.

TABLE 3: Power, area, and latency at 22nm.

	Power (mW)	Area (mm^2)	Latency (ns)
Metadata update unit	0.5	0.0004	0.5
Metadata cache (CAAB)	0.4	0.002	0.25
Metadata cache (CAPB)	1.1	0.003	0.25
Metadata cache (CAROR)	6.5	0.015	0.25
Binning thresholds selection unit	4.6	0.01	4822
Refresh scheduler (CAAB)	1.2	0.006	1.5
Refresh scheduler (CAPB)	1.2	0.006	1.5
Refresh scheduler (CAROR)	2.2	0.012	1.75

7.4 Effect of Data Scrambling

We evaluate content aware refresh on systems that use data scrambling to reduce the electrical resonance on the DRAM data bus [26], [27]. Three content aware refresh mechanisms (CAAB, CAROR, and CAPB) are evaluated, and the results are shown in Figure 15. Among the three mechanisms, CAROR is affected most significantly by scrambling: the performance improvement over AB falls from 12% to 9%, and the memory energy improvement from 15% to 12%. Due to the fine granularity of the refresh group in the original CAROR, the Hamming weight of the densest block can be very small for some refresh groups, resulting in significant refresh reduction. With data scrambling, the data are randomized, which decreases the probability of finding light refresh groups. Unlike CAROR, CAAB benefits from data scrambling. As discussed in Section 7.1, CAAB has a greater chance of finding a dense block within each refresh group due to the coarser granularity. Consequently, it achieves only a modest reduction in refresh operations. Data scrambling reduces not only those data blocks with many **0**s, but also those with many **1**s. As a result, the densest block within each refresh group tends to become lighter, which results in fewer refresh operations. Since the original refresh reduction of CAPB is close to 50% (Figure 13), the effect of data scrambling on CAPB is small.

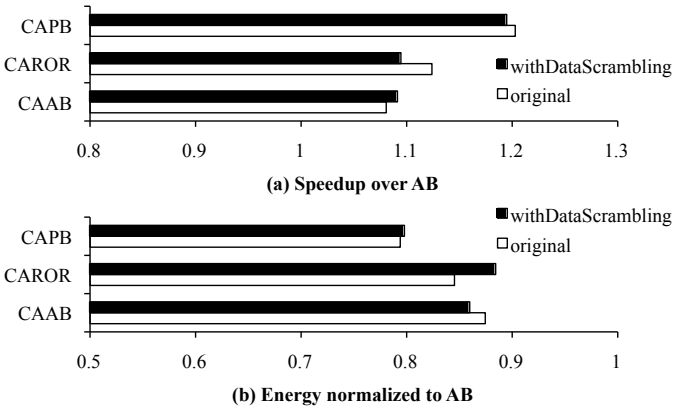


Fig. 15: The effect of data scrambling.

7.5 Comparison to RAIDR

Content aware refresh does not rely on the variability in retention times due to the physical cell characteristics. Figure 16 shows the comparison to prior work, RAIDR [2], which exploits such cell-to-cell variations. Without profiling the retention time of each physical cell, CAAB achieves performance and energy results that are similar to RAIDR. CAROR and CAPB respectively improve the system performance by 4% and 12%, and reduce the memory energy consumption by 5% and 12% over RAIDR. Since the proposed scheme is orthogonal to RAIDR, it can also be combined with RAIDR to achieve greater benefits. Indeed, CAPB + RAIDR outperforms RAIDR by 13%, and reduces memory energy consumption by 16% as compared to RAIDR.

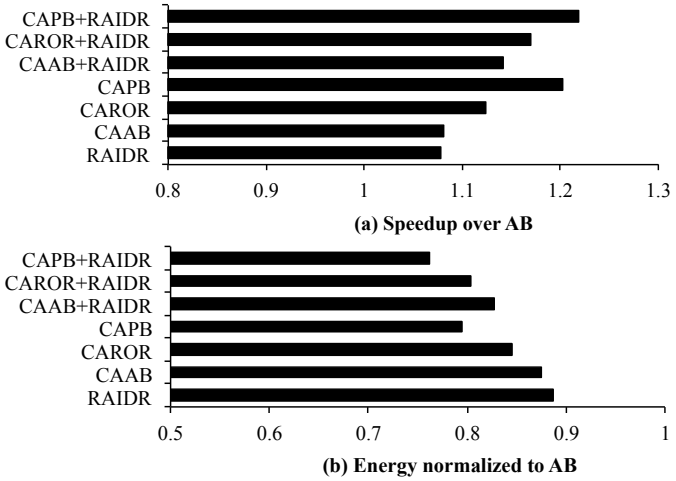


Fig. 16: Comparison to RAIDR.

7.6 Sensitivity Analyses

Sensitivity to important system parameters are studied in this section.

7.6.1 Metadata Cache Size

Figures 17 and 18 show the effect of the metadata cache size on the performance and energy consumption of the proposed system. Since the metadata cache size is already small for CAAB (0.75KB) and CAPB (1.25KB), the analysis is presented for CAROR only. In the figure, the metadata

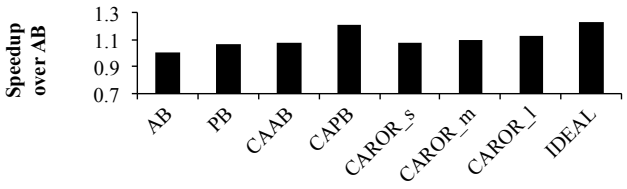


Fig. 17: Performance vs. metadata cache size.

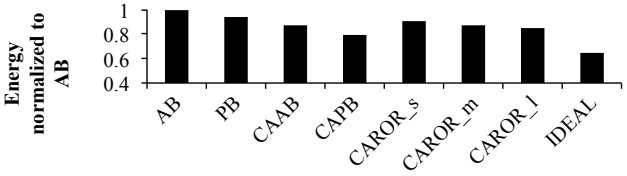


Fig. 18: Energy vs. metadata cache size.

cache size is 1.75KB, 3.5KB, and 7KB for CAROR small, CAROR medium, and CAROR large, respectively. Because CAROR has a relatively small refresh group, it requires a larger metadata cache. With a small metadata cache, both the performance and the energy consumption of CAROR are worse than those of CAPB and CAAB.

7.6.2 Number of Bins

The number of refresh bins is a design choice that directly affects the duration of the required refresh intervals. We examined three configurations with the performance and energy results shown in Figures 19 and 20. The system performance degrades slightly with fewer refresh bins. The memory energy consumption is more sensitive to the number of bins than system performance. Although a larger number of bins requires a larger refresh scheduler, the energy benefit of having fewer refresh operations outweighs the overhead of the additional hardware.

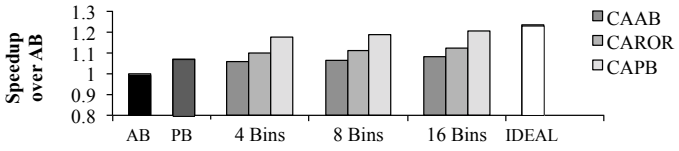


Fig. 19: Performance of systems with different configurations.

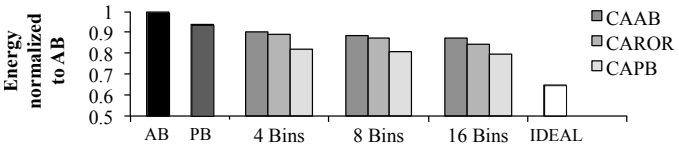


Fig. 20: Energy consumption of memory systems with different configurations.

7.6.3 Binning Thresholds

In addition to the proposed dynamic programming approach, we also evaluate content aware refresh with 16 evenly distributed binning thresholds. As compared to this straightforward method that uses fixed binning thresholds, the dynamic threshold selection algorithm respectively reduces the number of refresh operations by 12%, 28% and 22% for CAAB, CAROR, and CAPB (Figure 21). The superior results come from (1) using the optimal binning thresholds rather than suboptimal ones; (2) adaptation to the changes in the contents of main memory over time.

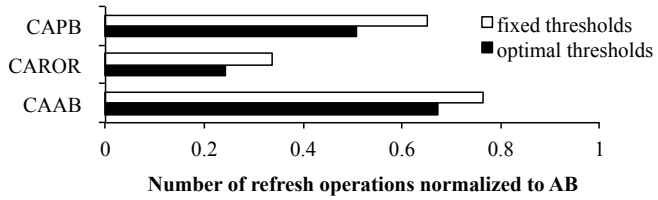


Fig. 21: The effect of binning thresholds.

7.6.4 DRAM Density

Figures 22 and 23 show how the performance and energy consumption of the baseline and the proposed systems scale with DRAM chip capacity. As the device capacity increases from 8Gb to 32Gb, all of the evaluated systems suffer from degraded performance and energy consumption. However, the systems that use content aware refresh are less sensitive to device capacity than the baseline systems that employ Auto Refresh. CAPB scales better than CAROR and CAAB because 1) CAROR is built on top of RAS-only Refresh, which exhibits poor scalability as compared to Auto Refresh; and 2) the coarser granularity of the refresh group used by CAAB increases the probability of finding a dense block within a refresh group when the device capacity is increased.

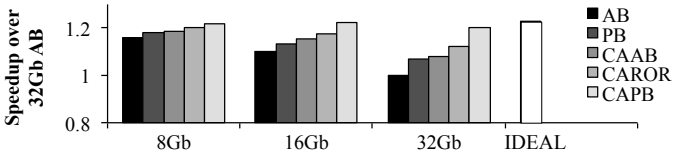


Fig. 22: Performance vs. device capacity.

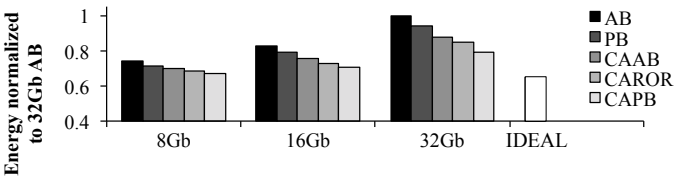


Fig. 23: Energy vs. device capacity.

8 Conclusions

The unidirectional nature of DRAM retention errors allows data blocks with fewer **1**s to be refreshed less frequently as compared to blocks with a greater number of **1**s. Rather than refreshing each DRAM row at the default refresh rate, a system that employs content aware refresh groups DRAM rows into different bins, and refreshes each bin at a rate determined by the Hamming weight of the densest block in that bin. A system that uses both content aware refresh and per-bank Auto Refresh achieves the best performance with the least energy consumption among all of the evaluated systems. Overall, this system achieves a 13% performance improvement and a 16% energy reduction over a baseline system that employs per-bank refresh. We conclude that

content aware refresh holds significant potential to improve the performance and energy efficiency of future high-capacity DRAM systems.

REFERENCES

[1] J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. F. Martínez, "Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 48–59.

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

[3] K.-W. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM performance by parallelizing refreshes with accesses," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, Feb 2014, pp. 356–367.

[4] I. S. Bhati, "Scalable and energy efficient DRAM refresh techniques," Ph.D. dissertation, University of Maryland, College Park, 2014.

[5] Micron Technology, "Various methods of DRAM refresh," 1995.

[6] T. Hamamoto, S. Sugiura, and S. Sawada, "On the retention time distribution of dynamic random access memory (DRAM)," *Electron Devices, IEEE Transactions on*, vol. 45, no. 6, pp. 1300–1309, Jun 1998.

[7] M. White, J. Qin, and J. Bernstein, "A study of scaling effects on DRAM reliability," in *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*, Jan 2011, pp. 1–6.

[8] P. Emma, W. Reohr, and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications," *Micro, IEEE*, vol. 28, no. 6, pp. 47–56, Nov 2008.

[9] K. Patel, L. Benini, E. Macii, and M. Poncino, "Energy-efficient value-based selective refresh for embedded DRAMs," in *Proceedings of the 15th International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation*, ser. PATMOS'05, 2005, pp. 466–476.

[10] K. Saino, S. Horiba, S. Uchiyama, Y. Takaishi, M. Takenaka, T. Uchida, Y. Takada, K. Koyama, H. Miyake, and C. Hu, "Impact of gate-induced drain leakage current on the tail distribution of DRAM data retention time," in *Electron Devices Meeting, 2000. IEDM '00. Technical Digest. International*, Dec 2000, pp. 837–840.

[11] S. Jin, J.-H. Yi, J. H. Choi, D. G. Kang, Y. J. Park, and H. S. Min, "Prediction of data retention time distribution of DRAM by physics-based statistical simulation," *Electron Devices, IEEE Transactions on*, vol. 52, no. 11, pp. 2422–2429, Nov 2005.

[12] J. Berger, "A note on error detection codes for asymmetric channels," *Information and Control*, vol. 4, no. 1, pp. 68 – 73, 1961.

[13] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14, 2014, pp. 361–372.

[14] 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," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 60–71.

[15] Micron Technology, "Mobile DRAM power-saving features and power calculations," 2005.

[16] T. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *Electron Devices, IEEE Transactions on*, vol. 26, no. 1, pp. 2–9, Jan 1979.

[17] E. Takeda, K. Takeuchi, D. Hisamoto, T. Toyabe, K. Ohshima, and B. Kiyoo Itoh, "A cross section of alpha -particle-induced soft-error phenomena in VLSIs," *Electron Devices, IEEE Transactions on*, vol. 36, no. 11, pp. 2567–2575, Nov 1989.

[18] Z. Al-Ars, S. Hamdioui, and A. J. van de Goor, "Effects of bit line coupling on the faulty behavior of drams," in *VLSI Test Symposium, 2004. Proceedings. 22nd IEEE*, 2004, pp. 117–122.

[19] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent dram failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50, 2017, pp. 27–40.

[20] T. Sekiguchi, K. Itoh, T. Takahashi, M. Sugaya, H. Fujisawa, M. Nakamura, K. Kajigaya, and K. Kimura, "A low-impedance open-bitline array for multigigabit dram," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 4, pp. 487–498, Apr 2002.

[21] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: Positional effects in DRAM and SRAM faults," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13, 2013, pp. 22:1–22:11.

[22] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. pp. 300–304, 1960.

[23] T. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," IBM Microelectronics Division, Tech. Rep., 1997.

[24] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, 2010, pp. 83–93.

[25] E. W. Swokowski, *Calculus with analytic geometry*. Taylor & Francis, 1979.

[26] I. Corporation, "Desktop 4th generation Intel Core processor family, desktop Intel Pentium processor family, and desktop Intel Celeron processor family," March 2015, datasheet, Volume 1 of 2.

[27] M. C. Falconer, C. P. Mozak, and A. J. Norman, "Suppressing power supply noise using data scrambling in double data rate memory systems," 6 2013, uS Patent App. 12/646,823.

[28] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005, http://sesc.sourceforge.net.

[29] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 469–480.

[30] Micron Technology, "Calculating memory system power for DDR3," 2007.

[31] ——, "4Gb x4, x8, x16 DDR4 SDRAM," 2014.

[32] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, June 2007, pp. 173–174.

[33] Synopsys. (2010) Synopsys design compiler. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx.

[34] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 189–200.

[35] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11, 2011, pp. 365–376.

[36] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *International Symposium on Microarchitecture*, Chicago, IL, Dec. 2007.

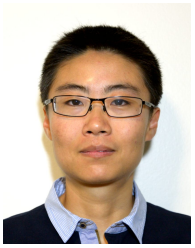
[37] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *Proceeding of the 38th annual international symposium on Computer architecture*, ser. ISCA '11, 2011, pp. 11–22.

[38] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi, "NU-MineBench 2.0," Northwestern University, Tech. Rep., August 2005, tech. Rep. CUCIS-2005-08-01.

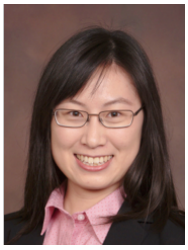
[39] R. M. Yoo, A. Romano, and C. Kozyrakis, "Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization*, 2009.

[40] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks—summary and preliminary results," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '91, 1991, pp. 158–165.

[41] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA-22*, 1995.
[42] Standard Performance Evaluation Corporation. (2001) SPEC OMP2001. https://www.spec.org/omp2001/.
[43] ——. (2006) SPEC CPU2006. https://www.spec.org/cpu2006/.
[44] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 172–184.
[45] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X, 2002, pp. 45–57.



Shibo Wang is a software engineer at Google. Dr. Wang received her M.S. (2013) and Ph.D. (2017) degrees in Computer Science from University of Rochester.



Xiaochen Guo is an assistant professor in the Department of Electrical and Computer Engineering at Lehigh University. Her research focuses on energy-efficient computer architectures, feature-rich memories, and computing platforms based on emerging technologies. Dr. Guo received her M.S. (2011) and Ph.D. (2015) degrees in Electrical and Computer Engineering from University of Rochester; and B.E. (2009) in Computer Science and Engineering from Beihang University.

Mahdi Nazm Bojnordi is an assistant professor in the School of Computing at the University of Utah. His research focuses on computer architecture, with an emphasis on energy-efficient computing. Dr. Bojnordi received a PhD in electrical engineering from the University of Rochester. His research has been recognized by two IEEE Micro Top Picks awards and an HPCA 2016 distinguished paper award.

Engin Ipek is an associate professor of Electrical and Computer Engineering and Computer Science at the University of Rochester, where he leads the Computer Systems Architecture Laboratory. His research interests are in energy-efficient architectures, high performance memory systems, and the application of emerging memory technologies to computer systems. Dr. Ipek received his BS (2003), MS (2007), and Ph.D. (2008) degrees from Cornell University, all in Electrical and Computer Engineering. Prior to joining University of Rochester, he was a researcher in the computer architecture group at Microsoft Research (2007-2009). His research has been recognized by the 2014 IEEE Computer Society TCCA Young Computer Architect Award, two IEEE Micro Top Picks awards, an invited Communications of the ACM research highlights article, an ASPLOS 2010 best paper award, and an NSF CAREER award.