Charge-Aware DRAM Refresh Reduction with Value Transformation

Seikwon Kim
Samsung Research, Samsung Electronics
seikwon.kim@samsung.com

Wonsang Kwak
School of Computing, KAIST
wskwak@kaist.ac.kr

Changdae Kim
ETRI
cdkim@etri.re.kr

Daehyeon Baek
School of Computing, KAIST
bdh0404@kaist.ac.kr

Jaehyuk Huh
School of Computing, KAIST
jhhuh@kaist.ac.kr

Abstract*—As the memory capacity in a system has been growing, refresh operations consume increasing ratios of the total DRAM power. To reduce the power consumption of such refresh operations, this paper proposes a novel value-aware refresh reduction technique called ZERO-REFRESH which exploits zero values in memory contents. A DRAM cell can retain the discharged state without refresh operations, and ZERO-REFRESH skips refresh operations on rows with all discharged cells. For abundant unallocated memory pages in typical systems, the operating system fills them with zeros to clean the contents. For those idle pages, ZERO-REFRESH can eliminate refresh operations in an OS-transparent way without any new interface to DRAM. However, for allocated memory pages, memory contents may not have many consecutive zero values to match the refresh granularity of DRAM. To increase the frequency of zero values and to arrange them to match the refresh granularity, ZERO-REFRESH transforms the value of memory blocks to the base and delta values, inspired by the prior BDI (Base-Delta-Immediate) compression technique. Once values are converted, bits are transposed to be stored as consecutive discharged bits at the refresh granularity. Such value transformation and rearrangement can make the memory contents friendly to refresh reduction based on discharged cells. The experimental results based on simulation show that the DRAM refresh operations are reduced by 37% on average for a set of benchmark applications, if the entire memory is allocated for the applications. If the memory usage statistics collected from three data center traces are applied, the DRAM refresh operations can be reduced by 46%, 57%, and 83% respectively for the three scenarios.

I. Introduction

Refresh operations in DRAM account for a significant portion of DRAM power consumption. As the refresh must be applied to the entire DRAM capacity during a fixed time period, commonly 32ms or 64ms, their portion in the total DRAM energy grows as the capacity of the system memory increases. The recent popularity of big memory applications has been accelerating the increase of memory capacity in systems. Furthermore, system consolidation based on virtualization and container technologies also requires bigger memory to accommodate more virtual machines. Such increasing memory demands have been exacerbating the refresh energy consumption significantly.

To mitigate the excessive power consumption, the prior work proposed to selectively skip refresh operations for certain rows of DRAM [2], [5], [8], [11], [19], [26], [40]. The prior approaches exploit the variance in DRAM retention time, the recency of DRAM accesses, or OS memory utilization information. However, the prior approaches have their limitations. The retention time-based approaches exploit the skewed distribution of retention times across cells. However, the retention time changes dynamically by various factors, requiring careful checking of the status [12], [18], [33]. Smart Refresh skips refresh for recently accessed rows [8]. However, compared to the entire memory capacity, the portion of accessed rows within a fixed refresh period tend to decrease as the total memory capacity increases with much faster rates. Skipping refresh for unallocated memory pages relies on system memory utilization [2], [11]. However, a new HW interface to DRAM and non-trivial OS changes are required to support it.

One important aspect not fully investigated by the prior approaches is to exploit the value property of each cell. In DRAM, the discharged state does not require a refresh operation. If an entire row of a DRAM bank contains only discharged cells, the row can skip the refresh operation without losing the data. In the logical memory contents, the memory can potentially contain many zero values stored as discharged states in DRAM. The abundance of zero values can be exploited to reduce the refresh operation.

Based on the discharged property of DRAM cells, this paper proposes a value-based refresh reduction architecture, called ZERO-REFRESH. It consists of two components, *charge-aware refresh reduction* on the DRAM side, and *value transformation* on the CPU side. The DRAM side charge-aware refresh reduction component skips refresh operations for rows with all discharged cells. The CPU-side value transformation components transform the value of a cacheline to create consecutive discharged bits friendly to the refresh skip mechanism.

A critical consideration for the proposed design is the existence of true and anti-cells in DRAM, which store zero value in discharged and charged states respectively.

Without proper value conversion, abundant zeros are stored in charged states in anti-cells, and thus refresh cannot be skipped. Therefore, the value transformation component must be aware of the cell types of rows in DRAM. Figure 1 shows the two components. The cell-type aware value transformation generates as many discharged rows as possible, based on the cell type identification from the prior work [16], [42]. The charge-aware refresh reduction skips refresh operations for the rows with all discharged cells.

With the two components, ZERO-REFRESH can skip refreshes for 1) unallocated memory filled with zeros by the operating system and 2) allocated memory whose values are transformed to produce consecutive discharged bits. First, for the unallocated memory, in typical systems with fluctuating memory demands, a significant portion of memory is in the unallocated state. The OS cleanses the page content when a page is assigned to a process. However, if the cleaning procedure is done at the deallocation time, the idle pages contain zeros until they are allocated again.

Second, for the allocated memory, it may contain zero values abundantly at fine-grained byte granularity. However, to reduce refresh operations for rows with zeros, many consecutive zero values must exist, and they must be arranged friendly to refresh granularity and cell types. The value transformation technique of ZERO-REFRESH increases the frequency of discharged bits and transposes them to place consecutive discharged bits on each row of DRAM as much as possible. ZERO-REFRESH employs a value conversion technique originated from the BDI (Base-Delta-Immediate) compression, and the data contents are converted to the differences from the base value within a cacheline unit [30]. To arrange discharged bits in the same row, ZERO-REFRESH transposes bits, re-mapping bits to place consecutive discharged bits in each row. In addition, the mapping of cachelines to multiple chips is adjusted to allow a set of rows in different chips to have all discharged cells.

To the best of our knowledge, ZERO-REFRESH is one of the first studies to use charge-aware refresh skipping augmented by value transformation for refresh reduction. It can skip refreshes both for unallocated and allocated pages in an OS and application transparent way. The new contributions of the paper are as follows.

- The paper proposes an efficient refresh skipping mechanism for the rows with all discharged cells. It significantly reduces the extra storage to record the discharged status of rows.
- The paper proposes an OS-transparent refresh reduction for unallocated pages. As long as the OS cleanses unallocated pages for security, those pages do not require refreshes. Such refresh skipping does not require any new interface to DRAM, as they are purely value-based.
- The paper shows that the frequency of zero values can be increased significantly by employing the BDI representation within a cacheline unit of data. The value transformation also considers the existence of true and anti cells in DRAM chips. Furthermore, the paper proposes bit plane transformation and rearrangement of cachelines to DRAM chips to allow discharged bits to constellate within the unit of refresh operations.

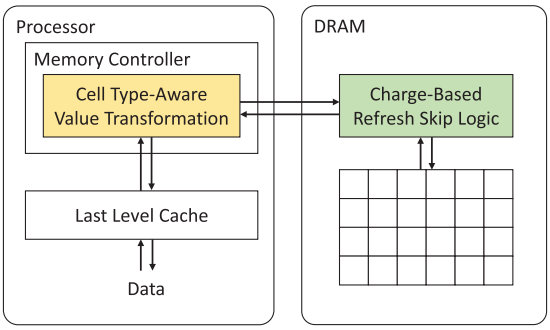


Figure 1: Value transformation on the CPU side and refresh reduction on the DRAM side

The paper evaluates the effectiveness of ZERO-REFRESH with simulation results. The results indicate that it reduces refresh operations by 37% on average from the conventional DRAM refresh, if the entire memory is allocated for applications. However, if the three memory usage statistics from real-world traces are applied, refresh operations are reduced by 46%, 57%, and 83% for the three scenarios, saving refresh power significantly.

The rest of the paper is organized as follows. Section II presents the background for DRAM refresh. Section III discusses the motivation of value-based refresh reduction. Section IV and Section V present the refresh skip and value transformation architecture of ZERO-REFRESH. Section VI evaluates the effectiveness with experimental results, and Section VII concludes the paper.

II. Background

A. DRAM Organization

Dynamic Random Access Memories (DRAM) are organized in hierarchical order into channels, ranks, banks and arrays. A DIMM consists of multiple ranks, and each rank is composed of multiple physical chips. A set of chips in a rank is operated in unison by the memory controller. Each chip contains multiple banks, typically from 4 to 16 banks. With a RAS (row address strobe) signal, the differential sense amplifier fetches an entire row in a bank. Once a row data is brought to the sense amplifier, part of the row data is read or written with a CAS (column address strobe) signal.

With the differential sense amplifier, the rows in an array are divided into two partitions. For each column, there are a pair of wires connected to the different side of the amplifier as shown in the Figure 2. The cells in a partition are connected to the wires of one side of the amplifier and the cells in the other partition are connected to the other side of the amplifier. For a read, depending on the charge state of the

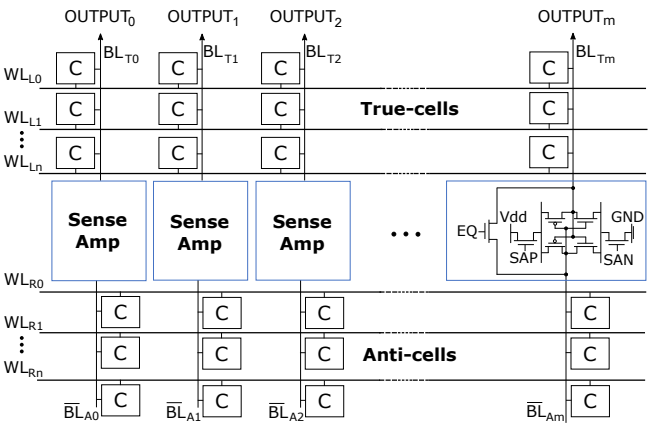


Figure 2: True and anti-cells with sense amplifiers

cells, the voltage of the wire on one side becomes slightly higher or slightly lower than the wire in the opposite side. The amplifier senses the voltage difference and drives the predefined high voltage on the wires with a slightly higher voltage and the ground voltage on the wires with a slightly lower voltage.

B. True and Anti-cell

It is important to note that the discharged state of a cell can be read as a logical 0 or 1, depending on which partition the cell is located. The wires on one side of the amplifier are used as the output wires. For the partition connected to the output wires, the charged cells drive the high voltage on the output wires and the discharged cells drive the ground voltage on the output wires. Since the high voltage on the output wire is read as 1, the charged state represents 1 and the discharged state represents 0 for cells in the partition. Such cells are called *true-cells*. For the other partition, the charged cells drive the ground voltage on the output wire since they drive the high voltage on the connected wire at the opposite side. As the charged state represents 0 and the discharged state represents 1 for such cells, they are called *anti-cells*.

Identifying cell types: Although the locations of true and anti cells are hidden in DRAM chips, the prior study showed that their row locations are regular and in each row, only one type of cells are used. [16], [42]. The prior work identified that true and anti-cell rows are interleaved by every N rows, and in common DRAMs, N is typically found to be 512. The cell type can be distinguished in a systematic way [16], [42]. After writing all zeros to a row, the refresh operation is disabled for a couple of refresh cycles. After skipping refresh, if the values are still zeros, the row consists of true cells. Otherwise, the row consists of anti cells. ZERO-REFRESH uses the systemic identification of cell types when it transforms values into as many consecutive discharged bits as possible. However, the CPU-side value transformation does not require to have 100% accurate identification of cell types, as the wrong identification incurs only the loss of refresh reduction chances.

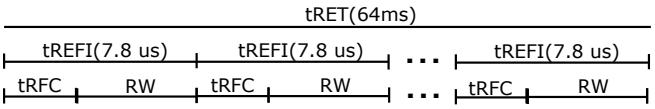


Figure 3: Refresh timing illustration

C. DRAM Refresh

DRAM is dynamic since a capacitor in a DRAM cell storing electrons is not stable. The capacitor of a cell gradually discharges and eventually the cell loses its value integrity over time. To maintain the value integrity, all capacitors in the DRAM must be recharged before they lose the values. To preserve the integrity of values, DRAMs periodically re-charge the cells to hold their values through *refresh* operations. The retention time (*tRET*) indicates the time period of a refresh to guarantee the value integrity of every cell. tRET is 64ms in general temperature, but refresh rates change depending on the temperature condition. If the temperature is beyond 85°C the retention time halves to 32ms or less.

To refresh within tRET in modern DDRx memories, a memory controller sends 8,192 times of *auto-refresh (AR)* commands to memory within tRET with the all-bank refresh policy. Therefore, the AR command is sent every 7.8us($\frac{64ms}{8192}$) and DDRx refreshes $\frac{MemorySize}{8192}$ amount of memory within 7.8us. The interval of refresh commands sent from a memory controller is tREFI and the amount of time to refresh $\frac{MemorySize}{8192}$ in memory is tRFC. The refresh timing is illustrated in Figure 3. During a refresh operation (tRFC), a whole rank or part of memory cannot be accessed depending on the refresh policy.

Upon receiving an AR command, each DRAM chip conducts refreshes for the rows indicated by the *refresh counters*, which is incremented after refreshing each row. Each DRAM chip refreshes multiple rows for an AR command. For example, a 32Gb DDR4 chip refreshes 1024 rows for each all-bank AR command. The *refresh counters* in each DRAM chip maintains the internal row address to be refreshed. In commercial DRAMs, the *refresh counter* is often initialized as a random number per device since the address rotates after reaching the end of the row address in a chip and the entire device gets refreshed after receiving 8k refresh commands regardless of an initial address setting.

The memory controller can issue refresh commands at a rank level (all-bank) or a bank level (per-bank). The all-bank refresh policy is supported in commodity DDRx, while the per-bank refresh policy is supported in mobile LPDDRx and HBMs.

All-bank refresh: In the all-bank refresh policy, refresh commands from the memory controller are operated at a rank level. Once a memory controller sends a refresh

command to a memory, a subset of rows in every bank gets refreshed. For instance, suppose 32GB memory with 8 banks and 4KB row-size receives an all-bank refresh command. For the all-bank refresh command, all 8 banks start the refresh operation, and 512KB or 128 rows (32GB/8192/8) of each bank get refreshed by a single command, and thus a total of 4MB in the memory is refreshed with the auto-refresh command. While refreshing, the entire rank is not available for read or write operation since all the banks in the rank is in the refresh mode.

Per-bank refresh: Unlike the all-bank refresh policy, the per-bank refresh policy allows the memory controller to send an auto-refresh command to an individual bank. As the refresh request is sent for each bank, the other banks not involved in the refresh are allowed to accept normal accesses. For a 32GB memory with 8 banks and 4KB row size, a single per-bank auto-refresh command refreshes 512KB of memory (128 rows). Although per-bank refresh allows selective refresh for each bank, the refresh commands are issued *numBank* times more often within tRET, compared to the all-bank refresh commands.

D. Prior Work

There are several prior studies to skip unnecessary refreshes, reducing both energy and performance overheads.

Retention Time Aware Skipping: The length of refresh period is determined by the shortest *retention time* of cells in a DRAM. In reality, a small fraction of cells ($<$1%) have a short retention time, while the majority of cells have much longer retention times [19]. VRA [26] and RAIDR [19] are HW-based techniques to exploit the retention time variance. RAPID increases the refresh period through the OS memory management by allocating memory pages by decreasing order of retention time [40]. Baek et al. proposed OS-based techniques on the Linux kernel and evaluated them on a real hardware [2].

The aforementioned techniques use the static information of retention time, however, the retention time actually changes over time and breaks the data integrity with increased refresh periods, which is called *Variable Retention Time (VRT)* [12], [18], [33]. Qureshi et al. proposed AVATAR which adaptively changes the refresh period to reduce the number of refreshes while compensating bit errors due to VRT [31]. However, AVATAR has a trade-off between the data reliability and energy/performance overhead as it periodically scrubs all memory contents to detect potential VRT.

Access Aware Skipping: Ghosh et al. proposed Smart Refresh which skips refreshes for recently accessed rows [8]. However, the benefits are limited to the accessed region between two refreshes. As DRAM capacity increases, the accessed portion between two refreshes becomes small compared to the total memory capacity.

Validity Aware Skipping: SRA [26], ESKIMO [11], and PARIS [2] skip refreshes on invalid data and unallocated memory regions. In these techniques, an OS or compiler specifies the regions which do not require refresh and HW cooperates in skipping the refreshes on them. The memory compression techniques [15], [29], [39] can potentially increase the skipped regions by reducing memory utilization, but add decompression latencies for cache misses. EAR proposed a memory compression technique for refresh reductions [10]. If the compressed space is large, it skips refreshes. Otherwise, it puts extra ECC to increase the refresh intervals.

Error Tolerable Data Aware Skipping: Flikker exploits non-critical data, such as the output buffer of video processing, to reduce DRAM refreshes [20]. To use Flikker, application programmers should annotate the non-critical data. The OS splits critical and non-critical data regions in memory, and controls HW to lower the refresh rate for non-critical data.

Value Bias Aware Skipping: Patel et al. utilized frequent zero bits in memory without any value transformation, and proposed to skip refreshes if some portion of DRAM cells consists of all zeros [27]. However, as the mechanism adds a Zero Indicator Bit for every 8$\sim$32 bits on DRAM, its area overhead is at least 1/8$\sim$1/32 of DRAM capacity.

Other related work: In addition to skipping refresh operations, there are other related studies for improving the refresh mechanism. Bhati et al. revealed that per-row refresh is not feasible for the current commodity DRAM due to its overhead, proposing a new refresh batching technique [5]. Cui et al. proposed a DRAM architecture, DTail, which uses a portion of DRAM to store the refresh information and controls refreshes based on the information [7].

Another approach is to hide the performance penalty of refresh by scheduling optimization of memory commands and parallelization of refresh with accesses. Elastic Refresh postpones the refresh if the accesses are expected [37]. Coordinated Refresh exploits the low power mode of DRAM [4]. It coordinates the refresh scheduling and transition to the low power mode to reduce refresh energy. Mukundan et al. investigated the command queue seizure problem where the command queue becomes full due to the requests blocked by the on-going refresh [23]. Besides scheduling, Nair et al. [25] added a refresh pause command to DRAM architecture and uses the command to prioritize the read requests. Other approaches parallelize the refresh with accesses. Chang et al. [6] and Zhang et al. [44] proposed DRAM architectures to exploit sub-array-level parallelism for refreshes and accesses. Kotra et al. exploited bank-level parallelism for refreshes and accesses [17].

This work extends our prior work for zero-aware refresh reduction [14]. This paper increase the scope of value-based refresh reduction to provide refresh skips for unallocated pages with a simple OS change (early zeroing of the unal-

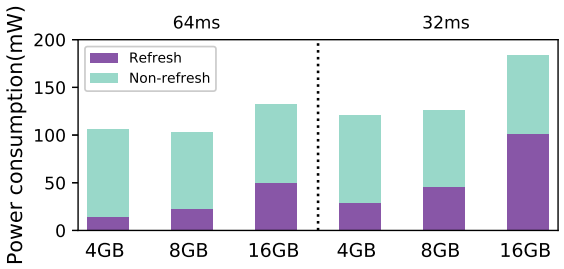


Figure 4: Refresh power consumption in normal temperature(left) and extended temperature(right) [22]

Traces	Allocated Memory
Google trace [41]	70%
Alibaba trace [9]	88%
Bitbrains trace [35]	28%

Table I: Average allocated memory of three traces

located pages). In addition, we further explore the detailed DRAM-side architectural extension for true/anti cells and efficient tracking of discharged rows.

III. Motivation

A. Overheads of DRAM Refresh

DRAM refresh operations are necessary to retain the value integrity, but the operations negatively affect both DRAM energy and performance. As the memory capacity increases, the portion of power consumption by the refresh operations increases significantly. Figure 4 illustrates the power consumption in DDR4 obtained with a Micron DDR4 calculator [22]. To calculate the power consumption, we evaluate with DDR4-2400MHz DRAM with data bus inversion (DBI) on. For the analysis, the percentage of clock cycles for reading from DRAM is set to 8%, while that for writing is set to 2%. The row buffer hit rate is set to 50%.

As the capacity increases, a larger number of cells should be refreshed in a fixed time period, *tRET*, and thus the power consumption by refresh operations increases. For 32ms retention time, required at high temperatures, 16Gb memory consumes more than half of the total power for refresh. This tendency will become much worse in the future as the memory capacity increases [19]. Furthermore, the refresh operation degrades the application performance since refresh operations also consume the memory bandwidth. While refreshing, other memory accesses are stalled, and after refreshing, the next data access is likely to have a row buffer miss even though the accesses have spatial locality.

B. Unallocated Pages in Systems

In typical systems, there are always a certain amount of unallocated memory pages. For security purpose, the operating system cleanses deallocated pages by filling with zeros, before they are re-allocated for another process. Although when the zero-filling procedure occurs may differ by operating system implementations, memory pages must be zeroed-out before they are re-used. If the OS cleanses memory pages right before re-allocation, as done by common Linux systems, the operating system can be slightly changed to execute the cleansing procedure at the deallocation time. With the zero-filling at the deallocation time, idle pages are in zero contents for longer periods of time.

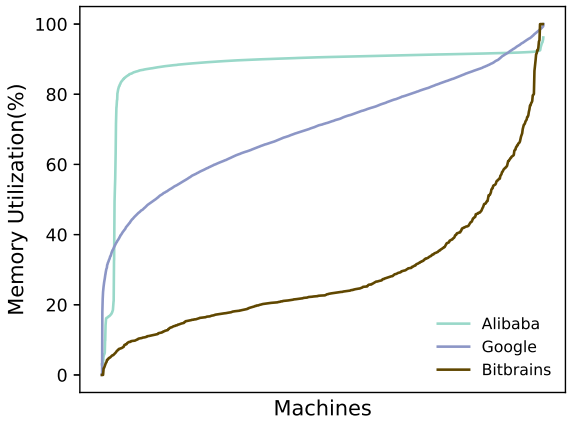


Figure 5: Cumulative distributions of memory utilization with three traces

Once deallocated memory pages are filled with zeros, the ZERO-REFRESH HW can automatically detect the zero content of the pages, and skip refreshes on them without any extra interaction between the OS and ZERO-REFRESH. Unlike ESKIMO [11], ZERO-REFRESH does not require any new HW interface to mark DRAM rows as unused. As long as the OS fills the content of memory with zeros, refresh operations can be skipped based on the value. Such a value-based refresh skip of ZERO-REFRESH simplifies the OS support significantly.

In data center systems, DRAM memory is often over-provisioned to satisfy peak demands. To show how much memory capacity is used, we analyzed three published data center traces from Google, Alibaba, and Bitbrains [9], [35], [41]. Figure 5 presents the cumulative distributions of memory utilization for the three traces, and Table I shows the average percentage of allocated memory in the system. For the three traces, the Bitbrains trace contains the virtual machine data for enterprise services. Since the trace contains the data even when VMs are idle, we used the memory utilization only when the CPU utilization is higher than 30% for conservative assessment of memory utilization for our study.

As shown by the results in Table I, systems have a significant portion of memory as unused pages. The Alibaba trace shows a relatively small 12% unused memory on average. However, the Google and Bitbrains traces show 30% and 72% of unused memory pages on average.

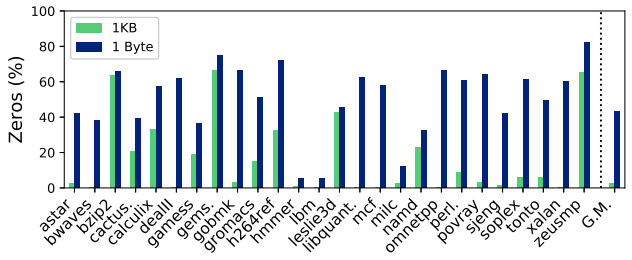


Figure 6: The portion of zeros at 1KB and 1Byte granularity

If the contents of the unused pages are filled with zeros, refreshes are eliminated by ZERO-REFRESH. However, to skip refresh operations for zero values, the values must be properly converted for true and anti-cell row. The proposed value transformation will encode zero pages to store them in discharged states for true and anti-cell rows with two different encoding schemes.

C. Zeros in Allocated Pages

In addition to unused pages, allocated pages can contain many zero bits, and ZERO-REFRESH reduces refreshes for such allocated pages by increasing occurrences of zero bits and by rearranging them for effective refresh skip. Note the zero bit can be directly represented as discharged state for true-cells, but an inversion is necessary for anti-cells, which will be addressed in our work. To exploit the discharged property of DRAM cells, there are two challenges, *frequency* and *contiguity* of zero values. First, the allocated memory contents must contain as many zero values as possible. Second, the zero values must be clustered to the unit of refresh operation (row), as a refresh operation is applied only to the entire row unit of 1KB-8KB.

Figure 6 shows the portion of zero blocks at 1KB and 1 byte granularities, in the memory contents of allocated memory pages. The plotted data are extracted from the memory dump of SPEC CPU2006 benchmarks. For conservative analysis, the results show the values only from the memory pages accessed at least once from the application execution. As shown in the figure, only an average of 2.3% of 1KB blocks consists of consecutive zeros. However, if the block size reduces to 1 byte, 43% of the memory contains zeros. Although there are significant portions of zero values at byte granularity, they are not easily exploitable with the current row-based refresh operations. Furthermore, the frequency of zero values can be potentially improved. ZERO-REFRESH will increase the frequency of zero values as well as the contiguity of them to effectively skip refreshes with the current DRAM architecture.

IV. Charge-Aware Refresh Reduction

This section presents the charge-aware refresh skip mechanism, which is added to the DRAM module to support ZERO-REFRESH. The mechanism requires additional logics to the refresh component.

A. Overview

This paper proposes a charge-aware refresh mechanism to skip refreshes for rows with all discharged cells. Such rows with all discharged cells are denoted as *discharged-rows* in the rest of the paper. The proposed refresh reduction does not require a new interface to the software stack. It detects discharged rows and skips refreshes on the discharged-rows in a software-transparent manner.

ZERO-REFRESH requires to modify the refresh logic in the DRAM module. While handling an auto-refresh (AR) command, the discharged status of each row must be looked up, and the refresh is skipped if the row is marked as discharged. The design in the rest of the paper is based on the per-bank AR, as used by REFLEX with per-bank AR [5]. Although the per-bank AR command is supported for LPDDR and HBM currently, the prior studies suggested that supporting the per-bank AR to general DDR devices requires only a minimal change [5], [6]. Alternatively supporting ZERO-REFRESH in all-bank AR is also possible, at the expense of the increased refresh logic complexity, as the discharged status of each row of multiple banks must be checked simultaneously and refreshes should be skipped selectively across different banks.

Note that due to the existence of true- and anti-cells, zero values must be properly encoded to be stored as discharged states. We will call the value of discharged state as *discharged bit*. The cell type-aware value transformation in Section V will properly encode zero values to discharged bits based on the locations of true and anti-cell rows.

A key component of the charge-aware refresh skip mechanism is to mark and track discharged-rows efficiently. The next section presents how to record the discharged status of each row without a large SRAM storage.

B. Tracking Discharged Rows

Discharged row detection: The first necessary component for charge-aware refresh reduction is to identify whether a row is a discharged-row or not based on the contents. The discharged status of each row is determined during a refresh operation, based on the charge status of all cells in the row. In the DRAM organization shown in Figure 2, the sense amplifier can detect the charge status for both true and anti-cell rows, when rows are read for refresh. The charge status of each bit is wire-ORed to determine the charged or discharged status of a row, which incurs minor area overheads. For spared rows used by row sparing for fault tolerance, skipping refresh is disabled.

Discharged status tracking: To skip refreshes on discharged-rows, the refresh logic must identify whether a row to be refreshed is a discharged-row or not. The discharged-status of all rows is stored in *discharged-status*

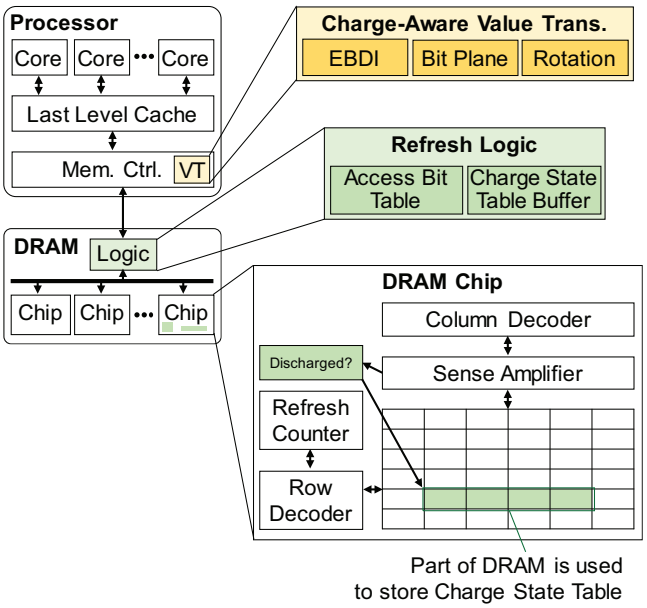


Figure 7: ZERO-REFRESH architecture

table. A naive design of the discharged-status table is to use an SRAM array in a DIMM module. In the naive design, the entry for a row is updated by checking the content, when the row is written. However, with a 4KB row size, the 32GB memory consists of more than 8.3 million rows which require a 1MB SRAM structure in the refresh logic to track the discharged status of each row. Adding such a large SRAM array on the DRAM module not only requires significant costs, but also consume non-negligible static energy.

To assess the power cost of the naive design, we evaluated the leakage power of a 1MB SRAM array with CACTI [24]. According to CACTI 6.5, the 1MB SRAM array requires 337.14mW of leakage power on 32 nm technology. The leakage power of the 1MB large SRAM-based table potentially consumes a significant energy. Therefore, reducing the extra storage overhead is critical to make ZERO-REFRESH practical.

To reduce the leakage power of the SRAM array, ZERO-REFRESH stores the discharged-status table in part of the DRAM space, instead of storing it entirely in an extra SRAM array. However, storing the table in DRAM causes a costly extra step during memory writes. For each write, the discharged-status table entry for the row may need to be updated as the row content can change, requiring an extra DRAM write. To avoid such extra writes to DRAM, our optimization employs a coarse-grained *access bit table* in an SRAM array. Each entry with a one-bit state records whether any write has been conducted for a range of rows since the last refresh cycle. For each memory write request, the corresponding bit in the access bit table is set to true. Note that one bit in the access bit table covers many rows,

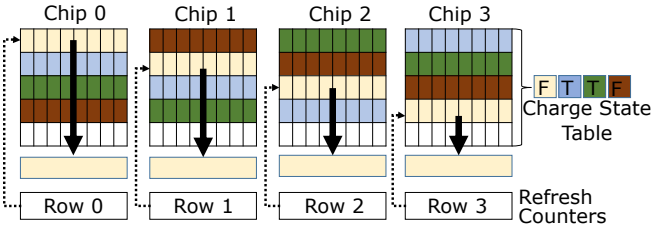


Figure 8: Refresh counter modification

and thus the SRAM array for the table is small. The range of rows corresponding to each entry is sized to match the number of rows refreshed by an auto-refresh command.

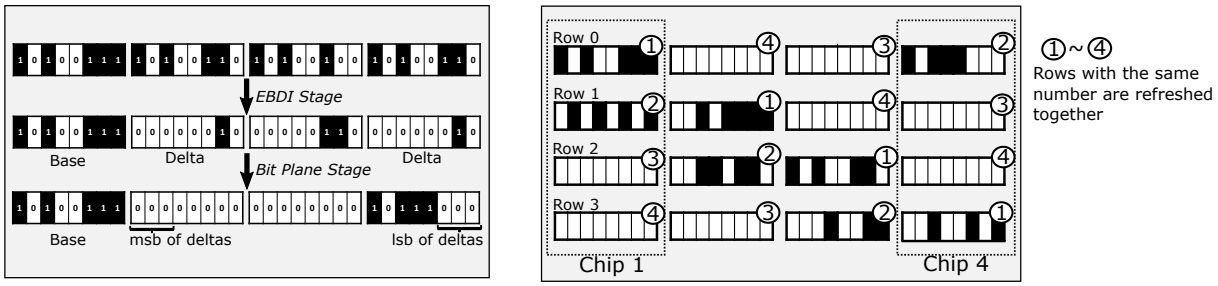
While processing an auto-refresh command, if the access bit is set in the table, the refresh operations are done normally without any skipping for the range of rows. However, during the refresh of each row, the actual discharged status of the row is checked. Since each row is already fetched for refresh, the discharged checking does not require any extra DRAM read. The renewed discharged-status bits for the range of rows is collected in a register, and they are written to the DRAM-resident discharged-status table only once for each AR command.

For an auto-refresh command, if the access bit is not set, it indicates that no data update occurred for the range of rows covered by the AR command. Therefore, the refresh operation reads the discharged-status bits for the rows from the DRAM-resident table, and uses the information to skip refresh operations depending on the discharged status. This coarse-grained access bit eliminates the need for updating the DRAM-resident table for every write request. Note that the discharged status bits of the refreshed rows for each AR command are read from the DRAM only once to process the AR command, and temporarily stored in a 128-bit register for 32Gb memory device.

Given 8K refresh operations within cell retention time (tRET) as described in Section II-C, ZERO-REFRESH partitions the memory to (8,192 * numBank) sets for per-bank refreshes. Therefore, ZERO-REFRESH manages an access bit per $\frac{MemorySize}{SetSize}$. With a 32GB and 8-bank memory, the memory is partitioned to 512KB ($\frac{32GB}{8192*8}$) with per-bank AR. In the case, the size of the SRAM for access bits is 8KB (=8192 * 8bits). The CACTI-estimated area overhead of 8KB SRAM is as small as $0.076mm^2$. The static power reduces from 337.14mW with the naive full SRAM design to 2.71mW for the 8KB SRAM. This optimized design drastically reduces the SRAM capacity and power required.

C. Refresh Counter Modification

In DRAM, a refresh counter exists per chip, which indicates the next row to be refreshed. In ZERO-REFRESH, the refresh counters are initialized in a staggered manner to refresh the rotated cachelines. This staggered refresh across chips is necessary to match the discharged value mapping used by value transformation presented in Section V. The



(a) Processing cacheline data

(b) Cachelines across multiple chips after data rotation

Figure 9: Value transformation overview

initial values of the refresh counters are chip numbers and they are increased by the following formula.

$$RefreshRow = ((initRow + n) \mod numChip) + \frac{n}{numChip})$$

initRow is the initial value of the refresh counter set to the chip number. n is the row number to be refreshed at chip 0, and *numChip* is the number of chips in memory. Figure 8 illustrates a four-chip example, with each column corresponding to a chip. In the figure, the rows in the same color across four chips are refreshed simultaneously, by setting the refresh counters in four chips in a staggered way. With such a refresh order across chips, the discharged-rows formed after the data rotation stage in Section V match the rows refreshed together in different chips.

D. Changes in DRAM modules

Figure 7 describes the modifications in the processor and DRAM modules ZERO-REFRESH requires. The cell type-aware value transformation components in the processor side is explained in the next section. The colored components in the figure show additional components for ZERO-REFRESH. The memory modification is to change the refresh mechanism to use the in-DRAM discharged-status table, and to add the coarse-grained access bit table. To support the access bit table, ZERO-REFRESH requires an 8KB SRAM buffer as described in the Section IV. In addition, as the discharged-status table is stored in DRAM, ZERO-REFRESH also adds a 16B register as a discharged-status buffer for 128 rows, which is per-bank refresh granularity for each AR command. The access bit table and discharge status buffers are located in a separate chip in a DRAM module, which can be added to the register clock driver (RCD) chip used in RDIMM or LRDIMM.

In addition to the refresh mechanism changes, the discharged checking logics must be added to DRAM. During a refresh operation, the discharged status checking is done for each refresh row. The checked state is collected in the 128-bit charge state register, and later written to the discharged-status table stored in DRAM at the end of AR.

V. Value Transformation

This section presents the value transformation architecture of ZERO-REFRESH to produce consecutive zero values. As shown in Figure 7, the value transformation is done at the CPU-side between the LLC miss handling and memory controllers. The cell types of all rows are detected by the method used in the prior work [16], [42], and data are encoded differently for true and anti-cell rows.

A. Overview

The charge-aware refresh reduction mechanism can skip refreshes for a row if the entire row contains discharged values. For unallocated memory, the contents are filled with zeros by the OS, but the zero values are converted properly considering the true and anti-cell types. In this section, we propose a value transformation to produce consecutive discharged bits both for the unallocated memory and allocated pages exploiting the low variance in values in typical memory contents.

Figure 9 illustrates the overall architecture of the value transformation in ZERO-REFRESH. The value transformation consists of two steps; a cacheline data transformation step that applies to all cachelines evicted from LLC as shown in Figure 9 (a) and a data rotation step that maps cacheline data to memory chips according to row addresses as shown in Figure 9 (b). During the two steps, three logical stages are used: ❶ In the EBDI stage, using base-delta-immediate representation, values are converted to generate zero bits from the original cacheline. ❷ In the bit transform stage, zero bits are transposed toward more significant bit positions. ❸ In the data rotation stage, values are rotated to aggregate zero-words in the same refresh row. Note that the first EBDI stage is the only stage that incurs non-negligible overheads. The second stage and the third stage are the stages with wire routing without logic overheads.

B. EBDI stage

EBDI (Encoded BDI) is originated from the BDI (Base-Delta-Immediate) compression technique [30]. Unlike the compression technique, which compresses data by taking advantage of the value locality within a cacheline, the EBDI

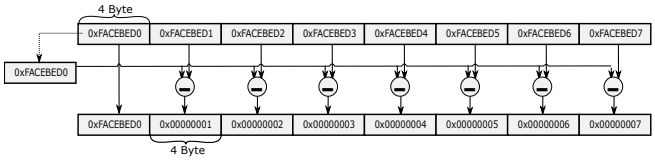


Figure 10: EBDI stage in ZERO-REFRESH

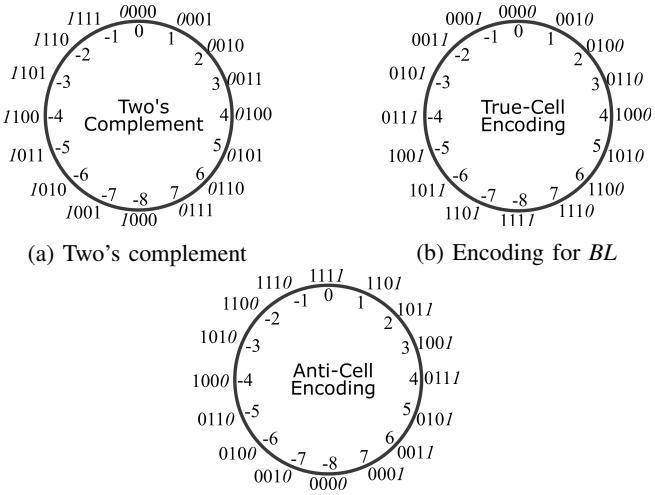
stage transforms the original cacheline into a converted cacheline containing more zero bits with the same size.

In the BDI compression, a cacheline is compressed to a base value and a sequence of delta values. The delta values are the differences between the base value and original value. Figure 10 illustrates the EBDI stage with a 32-byte cacheline divided into 8 words of 4B word size. The first word is selected as the base value, the rest of the words are represented by the delta values from the base. As the value differences tend to be small within a cacheline, the delta in each word has a small absolute value either with a negative or positive sign. Using the first 4B as the base, the rest of the 7 words are represented with the differences commonly expressed with a smaller number of bytes. Note that in the experimental configuration in the paper, the word size is fixed to 8 bytes, although the illustration in this section uses the word size of 4 bytes for simplicity.

From the original BDI compression, ZERO-REFRESH makes several modifications in EBDI. First, EBDI does not compress the data size, and the delta values are stored in the original word size. In addition, the delta value of the first word is not stored as the first word is always used as the base. As a result, the EBDI stage does not change the total cacheline data size. Second, EBDI does not require extra bits to store the sign bits of deltas. Although the BDI technique uses extra space to store sign bits for the positive and negative delta values [28], a new value encoding is devised for EBDI to represent negative deltas efficiently.

In addition, the encoding also considers the true- and anti-cells to maximize the occurrence of discharged bits in the delta representation. To generate discharged bits, EBDI provides two different encoding types for true-cells and anti-cells. Unlike two's complement representation where most of the high-order bits are 1 in negative numbers, EBDI generates zeros in the high-order bits for true-cell encoding. For anti-cells, the value 1 needs to be placed in high-order bits to maximize the occurrence of discharged bits. Thus, anti-cell encoding employs the bits reversed from the true-cell encoding.

Figure 11 shows the comparison between two's complement, true-cell and anti-cell encodings. The signed numbers inside the circles are the original delta value, and the binary numbers outside the circles are the encoded outputs. EBDI must select the encoding scheme differently for true- and anti-cells. Using the cell type identification method from the prior work [16], [42], ZERO-REFRESH applies different



(a) Two's complement (b) Encoding for *BL*

(c) Encoding for $\overline{BL}$

Figure 11: Encoding schemes for EBDI. *Italic* represents sign bit

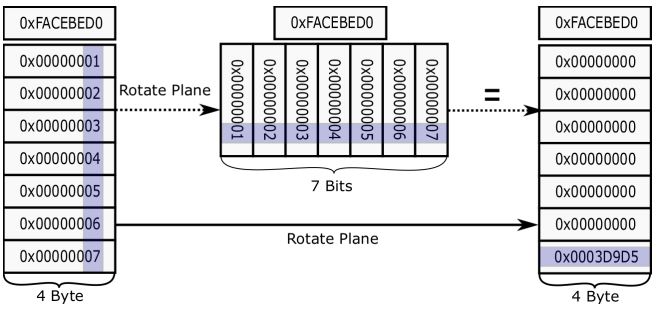


Figure 12: Bit plane transformation

encoding based on DRAM row addresses. Note that the cell type identification does not need to be 100% accurate. Even if the cell type of a row is mis-predicted, the original values are returned by the reverse value transformation for DRAM reads. Miss-prediction affects only the effectiveness of refresh reduction.

C. Bit Plane Stage

Transforming the bit-plane is motivated by BPC compression technique [13]. After the prior EBDI stage, each word within a cacheline contains a small delta value with consecutive zero bits in high order bits, but zero bits are not consecutive across words, as the least significant byte tends to be a non-zero in its original delta value. The bit plane transformation re-orders bits within a 64B unit to place zero bits consecutively.

The bit-plane transformation is demonstrated in Figure 12. In the figure, values in the shaded area show the delta values in the low-order bits of each word. Transposing the bit plane results in non-zero values concentrated in the lowest-order word in the cacheline, placing zero bytes consecutively except for the first base word and the lowest-order word.

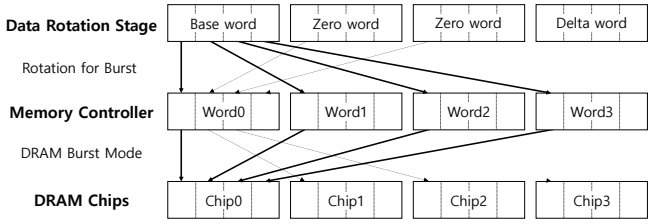


Figure 13: Writing a cacheline to multiple DRAM chips

Figure 9 (a) shows the changes of a cacheline through the EBDI and bit plane stages. The figure uses an example of a 32-bit (4B) cacheline. The word size is 8 bits. Through the two stages, non-zero bytes are concentrated in the base word and the least significant word (*delta word*) of the cacheline.

D. Data Rotation Stage

The final step of transformation is the data rotation stage to map a cacheline across multiple DRAM chips of a DIMM in the way matching the refresh unit. The actual mapping depends on the byte-to-chip mapping scheme used by the target DIMM architecture, and it should be configured to match the DRAM setup. In this section, we discuss this stage with a common DDRx mapping architecture. Figure 9 (b) presents the outcome of the data rotation stage which maps transformed cachelines into four memory chips. Each column corresponds to a chip. To simplify the figure, a row in a chip contains only one word, and four horizontal rows of four chips contains a cacheline. After the data rotation stage, the base words of consecutive cachelines are stored in different columns, mapping multiple cachelines into four chips in a rotated manner. The circled numbers represent groups of rows refreshed at the same time across multiple chips.

First, this stage must consider the byte mapping scheme for multiple DRAM chips. For example, when a 64-byte cacheline is written on a typical DDRx memory with the burst mode, the memory controller transfers a sequence of 8-byte words to 8 chips, a byte per chip in each step. The mapping of an 8-byte word across 8 chips will distribute the non-zero bits in the base and delta words in all the chips. To avoid such dispersed placement, the byte locations are rearranged to concentrate the base and delta words in their assigned chips. Figure 13 outlines the byte mapping change. It places the adjacent bytes to different words, to make the DRAM burst mode re-gather the non-zero base word to the first chip.

The second mapping consideration is to rotate the chip location of the words of a cacheline. As shown in Figure 9 (b), this rotation step allows the base words are grouped in the rows refreshed together, and the delta words are grouped together too. Therefore, discharged words are concentrated in the row groups refreshed together, without any base and delta words in the rows.

CPU Processor	4 cores, out-of-order x86 ISA, 4GHz
CPU L1-D cache	I cache 32KB, D cache 32KB, 64B cache-line, 8-way
CPU L2 cache	2 MB per core, 64B cache-line, 32-way
Memory Configuration	32GB capacity, 8 chips, 8 banks, 4KB row buffer, 1 logic component for refresh management
Timing Parameters (ns)	tRAS=28, tRCD=11, tRRD=5, tFAW=24, tRFC=28
Chip Energy Parameters (mA)	IDD0=23, IDD1=30, IDD2P=7 IDD2N=12, IDD3=8, IDD4W=58 IDD4R=60, IDD5=120, IDD6=8 IDD7=105

Table II: Simulated system configuration

With the two mapping factors, the data rotation stage remaps the byte location of a cacheline to match the refresh unit. In Figure 9 (b), the diagonally located words in four chips contain the base values. Next four words located diagonally contain non-zero delta values. The other two sets of wrapped diagonal words contain all discharged bits. The refresh mechanism will exploit the regular location of discharged words to effectively skip refresh operations.

VI. Experimental Results

A. Evaluation Methodology

Our evaluation uses a timing simulator that combines McSimA+ [1], GEMS [21], and DRAMSim2 [34]. The cores are modeled with McSimA+ with the PIN-based driver [32] and the cache hierarchy is modeled with GEMS (Ruby). The core simulator models 4-way out-of-order execution cores. The processor contains four cores, and the last-level cache capacity is 8MB, 2MB LLC size per core. For the main memory simulation, we integrate DRAMSim2 to GEMS, and configure the DRAM model to the DDR4 parameters. Table II presents the detailed configuration for the simulated system. To model an accurate value conversion, the execution-driven simulation uses the actual memory contents during the application execution.

We assume the temperature is in the extended temperature range, using 32ms as the base refresh rate. We use the timing and power parameters from the ones described in [5]. Table II lists the relevant memory timing parameters (ns) and current values (mA).

We conduct our evaluations with 17 benchmarks from SPECCPU 2006 [36], 2 benchmarks from NPB [3], and 4 benchmarks from TPC-H [38] suites as our benchmark applications. For each evaluation, the identical benchmark runs in each core. After fast-forwarding each application to the representative phase, the results are collected by executing more than 256 ms to achieve 8 refresh operations.

To reflect the memory utilization of real-world systems, we use four different memory allocation scenarios. The first scenario (`100% alloc`) does not have any unused memory page. The entire DRAM is filled with the memory page of an application. In addition, we employ three memory usage

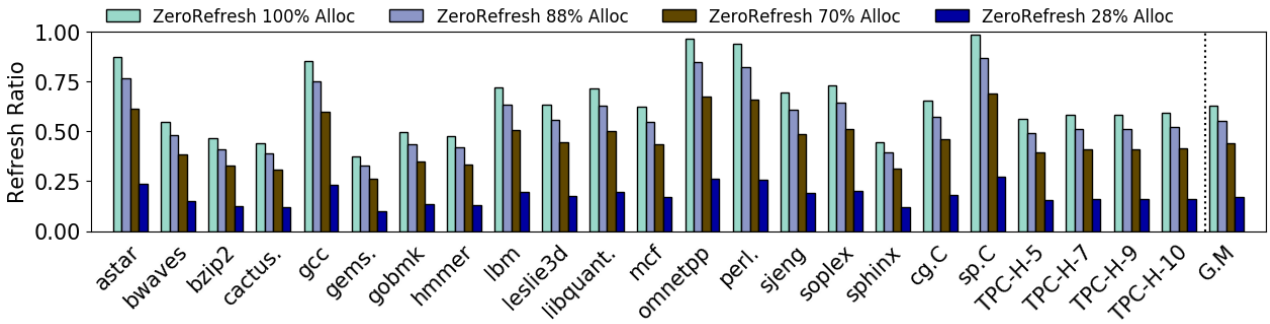


Figure 14: Normalized refresh operations for 100%, 88%, 70%, and 28% memory usage scenarios

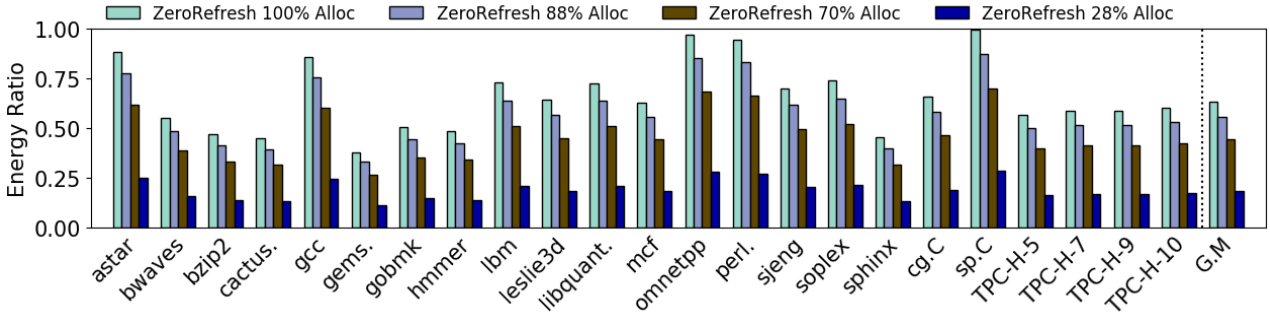


Figure 15: Normalized refresh energy consumption compared to conventional refresh (The energy overheads of ZERO-REFRESH are included.)

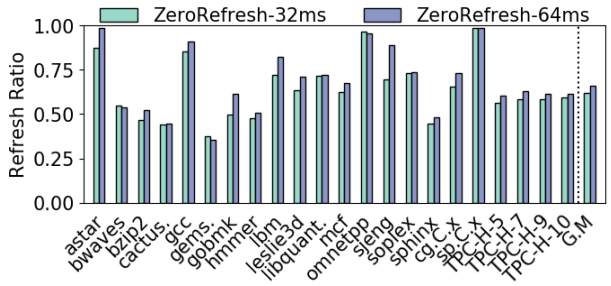


Figure 16: Normalized refresh in normal temperature and extended temperature (100% allocated)

scenarios from the three traces presented in Table I. The three scenarios (88% Alloc, 70% Alloc, and 28% Alloc) correspond to Alibaba, Google, and Bitbrains traces.

B. Results

Refresh Reduction: This section evaluates the reduction of refresh operations with ZERO-REFRESH compared to the conventional DDR under four different memory utilization scenarios. Figure 14 shows the refresh reduction normalized to the conventional DRAM refresh. As shown in the figure, ZERO-REFRESH reduces on average 37.1% of refresh operations from the baseline. Even though the row buffer size is 4KB, ZERO-REFRESH effectively gathers zero values in 37% of rows on average, even if the entire memory is used for applications without any unused pages. Since the value-based nature of ZERO-REFRESH, the effectiveness of value transformation can vary by the memory contents of diverse applications. Although `gems.` and `sphinx` have high reduction ratios, `omnetpp`, `perl.`, and `sp.C` have only small reduction ratios.

As the portion of unallocated pages increases, the refresh reduction ratios increase significantly. For the three scenarios, the refresh operations are reduced by 46%, 57%, and 83% respectively. For the most optimistic Bitbrains scenario, more than 80% of DRAM refreshes can be reduced by ZERO-REFRESH.

A comparison between normal temperature mode, which refreshes all cells within 64ms, and extended temperature mode, which refreshes all cells within 32ms, is illustrated in Figure 16. As shown in the graph, the number of write accesses within 64ms is limited compared to 32ms. The average difference is only about 4.4% less reduction for normal temperature mode compared to the extended temperature mode.

Energy Analysis: This section evaluates the energy saving by ZERO-REFRESH. To analyze energy consumption properly, we also evaluate the extra power consumption overheads by both of the EBDI module and access bit table. The EBDI module is utilized for both memory reads and writes while the access bit is utilized for write requests only. The energy of the EBDI component is modeled with the Vivado Design Suite 2017.4 over a Zynq device (xc7z020clg484-1)

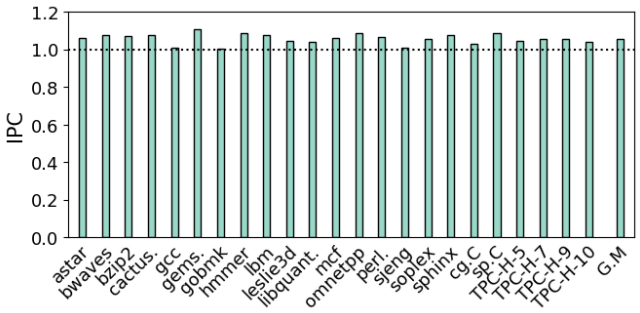


Figure 17: Normalized IPCs compared to conventional refresh

with a 1ns clock which corresponds to 1GHz [43]. According to the design suite, the EBDI component consumes 15pJ per operation.

The power consumption of the access bit table is modeled with CACTI 6.5 [24]. For the simulated system configuration, the 8KB SRAM access bit table is required, which consumes 2.71mW of standby leakage power based on 32nm technology. Both of the aforementioned extra power consumption for the ZERO-REFRESH components are included in our overall energy evaluation. We also add the amount of energy to read the zero status table from the DRAM for each refresh cycle.

Figure 15 compares the energy consumption of ZERO-REFRESH to the baseline DDR auto-refresh. ZERO-REFRESH reduces the overall energy consumption by 36.5% from the baseline. With the three idle memory scenarios, the energy consumptions are reduced by 44%, 55%, and 82%. Despite the extra components added in the proposed technique, the energy reduction by refresh reduction far outweighs the energy overheads of the extra components.

IPC Results: As a memory bank cannot serve normal requests during its refresh process, reducing refresh operations can potentially improve the performance as well energy consumption, by increasing the effective bandwidth. Figure 17 shows the normalized IPCs compared to the baseline. In terms of instructions per cycle (IPC), the average performance with the benchmark applications improves by 5.7%. The maximum performance improvement is achieved from `gemsFDTD` by 10.8%, while minimum performance improvement is with `goBMK` by 0.3%.

Row Size Sensitivity: The size of row buffer can affect the effectiveness of ZERO-REFRESH, as it must gather zero values entirely in a row to skip future refresh operations on the row. However, if the row size increases, the chance to gather zero values entirely in a row will decrease. In commodity DRAMs, row sizes of 2KB-8KB are often available. In this evaluation, we examine how different row buffer sizes affect the effectiveness of refresh reduction. We evaluate a row size from 2KB to 8KB with ZERO-REFRESH. Figure 18 presents the sensitivity of the refresh reduction

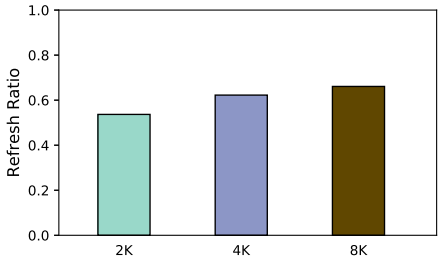


Figure 18: Normalized refresh operations with 2K, 4K, and 8K row buffer sizes (100% allocated)

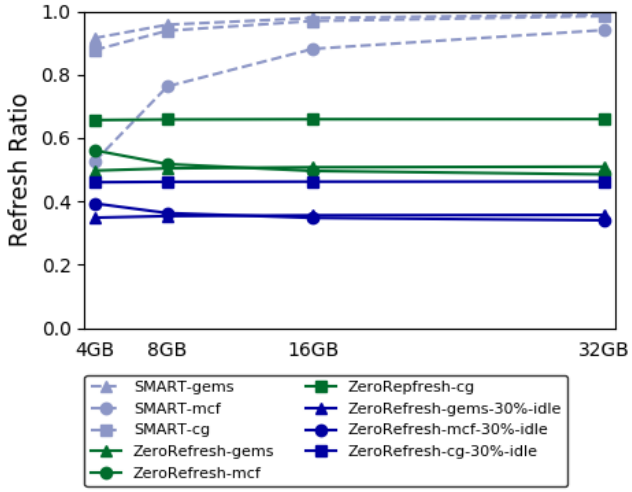


Figure 19: Scalability comparison between smart refresh and ZERO-REFRESH

on average for the identical set of benchmarks. For the figure, 100% memory is allocated without any idle pages. As shown in the figure, the 2KB row buffer reduces refreshes by 8.6% more than 4KB row buffer size, resulting 46.3% of refresh reduction from the baseline. The 8KB row buffer, on the other hand, reduces 33.9% from the baseline which is 3.8% less refresh reduction than that with the 4KB row buffer size. Although our base configuration uses the 4KB row buffer size for conservative evaluation, a smaller low buffer size will improve the effectiveness of ZERO-REFRESH significantly.

C. Comparison to Access-Aware Refresh

This section compares the refresh reduction between the prior access-based Smart refresh and ZERO-REFRESH. The access-based approach skips refreshes for recently accessed rows, and thus its effectiveness relies on how much portion of memory capacity is accessed during a refresh cycle. As a larger portion of memory is accessed, more refresh reduction is possible. The capacity of memory has been increasing precipitously to run large memory applications. However, the portion of memory accessed during a refresh cycle is more dependent on the working set of applications.

Figure 19 presents the normalized refresh operations for Smart and ZERO-REFRESH when the memory capacity increases from 4GB to 32GB. For the experiments of ZERO-REFRESH, the unused memory space is filled with the data from the benchmark application. Without the assumption, the unused memory space can be filled with zero values, which will unfairly favor the ZERO-REFRESH mechanism. Note that Smart refresh was originally proposed for a 3D stacked DRAM with a modest 64MB capacity.

The refresh increases from 52.6% in 4GB of memory to 94.1% for `mcf` with Smart refresh, while ZERO-REFRESH exhibits almost constant refresh reduction across different memory sizes. As the working sets of applications do not increase with the memory capacity increase, the effectiveness of Smart refresh is significantly reduced with large memory capacities. When 30% idle pages are added, as in the Google data center scenario, ZERO-REFRESH becomes much more effective than Smart Refresh.

VII. Conclusion

This paper proposed a novel value-based refresh reduction technique inspired by compression techniques. The proposed technique can skip refreshes for unused pages in an OS-transparent way. In addition, the paper showed that the memory data content can be transformed to contain more consecutive discharged bits by base-delta representation and transposition of bit location. The proposed technique effectively reduces 37% of refreshes on average compared to the conventional DDR memory, even if the entire memory is utilized. As the memory utilization decreases, the refresh reduction increases significantly, since it can effectively skip refreshes on unused pages.

Acknowledgement

This work was supported by National Research Foundation of Korea (NRF-2019R1A2B5B01069816) and the Institute for Information & communications Technology Promotion (IITP-2017-0-00466). Both grants are funded by the Ministry of Science and ICT, Korea. Changdae Kim was supported by Electronics and Telecommunications Research Institute(ETRI) grant funded by the Korean government (20ZS1300).

References

[1] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proc. International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[2] S. Baek, S. Cho, and R. Melhem, "Refresh now and then," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 3114–3126, Dec 2014.

[3] 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 *et al.*, "The NAS parallel benchmarks," https://www.nas.nasa.gov/-publications/npb.html.

[4] I. Bhati, Z. Chishti, and B. Jacob, "Coordinated refresh: Energy efficient techniques for DRAM refresh scheduling," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2013.

[5] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob, "Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2015.

[6] K. K. W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM performance by parallelizing refreshes with accesses," in *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[7] Z. Cui, S. A. McKee, Z. Zha, Y. Bao, and M. Chen, "DTail: A flexible approach to DRAM refresh management," in *Proc. International Conference on Supercomputing (ICS)*, 2014.

[8] M. Ghosh and H.-H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2007.

[9] J. Guo, Z. Chang, K. Wang, and G. Xu, "cluster-trace-v2018," https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018.

[10] J. Hong, H. Kim, and S. Kim, "EAR: ECC-aided refresh reduction through 2-D zero compression," in *Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018.

[11] C. Isen and L. John, "ESKIMO - energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2009.

[12] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *Proc. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2014.

[13] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2016.

[14] S. Kim, W. Kwak, C. Kim, and J. Huh, "Zebra refresh: Value transformation for zero-aware DRAM refresh reduction," *IEEE Computer Architecture Letters (CAL)*, vol. 17, no. 2, 2018.

[15] S. Kim, S. Lee, T. Kim, and J. Huh, "Transparent dual memory compression architecture," in *Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.

[16] 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 *Proc. International Symposium on Computer Architecture (ISCA)*, 2014.

[17] J. B. Kotra, N. Shahidi, Z. A. Chishti, and M. T. Kandemir, "Hardware-software co-design to mitigate DRAM refresh overheads: A case for refresh-aware process scheduling," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[18] 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 *Proc. International Symposium on Computer Architecturei (ISCA)*, 2013.

[19] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2012.

[20] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[21] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, 2005.

[22] Micron Technology Inc, "DDR4 SDRAM system-power calculator," https://www.micron.com/support/tools-and-utilities/power-calc/.

[23] 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 *Proc. International Symposium on Computer Architecture (ISCA)*, 2013.

[24] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP laboratories*, 2009.

[25] P. Nair, C. C. Chou, and M. K. Qureshi, "A case for refresh pausing in DRAM memory systems," in *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[26] T. Ohsawa, K. Kai, and K. Murakami, "Optimizing the DRAM refresh count for merged DRAM/logic LSIs," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 1998.

[27] K. Patel, L. Benini, E. Macii, and M. Poncino, "Energy-efficient value-based selective refresh for embedded DRAMs," in *Proc. International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2005.

[28] G. Pekhimenko, "Talk in Microsoft Research: Base-Delta-Immediate compression: Practical data compression for on-chip caches," https://youtu.be/t7PwCRiVcn4?t=20m20s.

[29] 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 *Proc. International symposium on Microarchitecture (MICRO)*, 2013.

[30] 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 *Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[31] M. K. Qureshi, D. H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. International Conference on Dependable Systems and Networks (DSN)*, 2015.

[32] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, "PIN: A binary instrumentation tool for computer architecture research and education," in *Proc. Workshop on Computer Architecture Education (WCAE)*, 2004.

[33] P. J. Restle, J. W. Park, and B. F. Lloyd, "DRAM variable retention time," in *International Technical Digest on Electron Devices Meeting*, Dec 1992, pp. 807–810.

[34] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters (CAL)*, vol. 10, no. 1, 2011.

[35] S. Shen, V. van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *The 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015.

[36] Standard Performance Evaluation Corporation, "SPEC CPU 2006," https://www.spec.org/cpu-2006/.

[37] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2010.

[38] TPC, "Transaction processing performance council," http://www.-tpc.org/tpch/.

[39] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, and P. M. Bland, "IBM memory expansion technology (MXT)," *IBM Journal of Research and Development*, 2001.

[40] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM," in *Proc. International Symposium on High-Performance Computer Architecture (HPCA)*, 2006.

[41] J. Wilkes, "More google cluster data," http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[42] X.-C. Wu, T. Sherwood, F. T. Chong, and Y. Li, "Protecting page tables from RowHammer attacks using monotonic pointers in DRAM true-cells," in *Proc. the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[43] Xilinx Inc, "Vivado Design Suite 2017.4," https://www.xilinx.com/support/download.html.

[44] T. Zhang, M. Poremba, C. Xu, G. Sun, and Y. Xie, "CREAM: A concurrent-refresh-aware DRAM memory architecture," in *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, 2014.