Restore Truncation for Performance Improvement in Future DRAM Systems

Xianwei Zhang† Youtao Zhang† Bruce R. Childers† Jun Yang‡

† Computer Science Department
University of Pittsburgh, PA, USA
† {xianeizhang,zhangyt,childers}@cs.pitt.edu

‡ Electrical and Computer Engineering Department
University of Pittsburgh, PA, USA
‡ juy9@pitt.edu

ABSTRACT

Scaling DRAM below 20nm has become a major challenge due to intrinsic limitations in the structure of a bit cell. Future DRAM chips are likely to suffer from significant variations and degraded timings, such as taking much more time to restore cell data after read and write access.

In this paper, we propose restore truncation (RT), a low-cost restore strategy to improve performance of DRAM modules that adopt relaxed restore timing. After an access, RT restores a bit cell's voltage only to the level required to persist data to the next scheduled refresh rather than to the default full voltage. Because restore time is shortened, the performance of the cell is improved under process variations. We devise two schemes to balance performance, energy consumption, and hardware overhead. We simulate our proposed RT schemes and compare them with the state of the art. Experimental results show that, on average, RT improves performance by 19.5% and reduces energy consumption by 17%.

1. INTRODUCTION

Modern computer systems have increasing demand for more memory capacity due to the proliferation of chip multiprocessors for high performance, graphic and cloud applications. DRAM scaling, although was the primary driver behind capacity improvement in past decades, has now become a major obstacle for the semiconductor industry as technology node sizes reach 20nm and below [47, 41, 43].

As DRAM is scaled down, the capacitor used in a bit cell becomes smaller and holds less charge [40, 18]. Similarly, the access transistor decreases in size with weaker drivability. In addition, scaling DRAM in sub-micron regimes suffers from significant process variations [38, 6, 26], i.e., cell behaviors will be more statistical instead of deterministic [13]. Cells with less stored charge and induced drain leakage [43] causes more leaky cells in one chip which needs frequent refreshes to prevent data loss. With lower transistor drivability, it takes more time to charge a DRAM cell [40, 48]. The timing of these operations, referred to as *refresh* and *restore*, respectively, in the literature, are expected to degrade in order to maintain high chip yield [26, 56].

Due to its importance, DRAM scaling has been a focus of many recent studies. Nair *et al.* proposed ArchShield that is capable of rescuing 100× more weak cells in future DRAM chips [43]. One application of ArchShield is to reduce chip refresh rate for energy saving. For DRAM restore, Kang *et al.* identified a significant performance impact of degraded restore timing [26]. Zhang *et al.* constructed fast logic rows by remapping row segments in different chips on a DIMM [56]. MCR [14] is a recently proposed scheme that shares similarity with the schemes proposed in this paper. MCR combines several rows to achieve timing benefits, with a tradeoff of significant capacity reduction. We compare to these schemes in the experiment section. In addition, schemes have also been proposed to reduce sensing time [15, 49, 55, 14, 46, 34, 9] and refresh overhead [17, 50, 42, 10, 40, 36, 6, 8].

In this paper, we describe schemes to improve performance of DRAM which has longer restore timing due to further scaling. Based on the observation that a cell leaks charge monotonically, it is often unnecessary to fully charge a row after a read or write operation. Instead, it is safe to terminate the restore operation once the cells in this row have more charge than what they have under natural decay, i.e., when the row is not accessed between two refresh commands sent to the row. We call this process *restore truncation (RT)*. Using this observation, we present two RT schemes that make tradeoffs in performance, energy, and hardware overhead.

The contributions of this paper are:

1. We observe that it is beneficial to truncate restore for performance improvement. We first propose RT-next that truncates a restore based on the time distance to the next refresh. RT-next is conservative for the worst case scenario and compatible with DRAM modules that use multi-rate refresh.
2. We propose RT-select to better integrate refresh and restore. By increasing the refresh rate of recently accessed rows, RT-select exposes more truncation opportunities while minimizing performance and energy overheads of extra refresh operations.
3. We evaluate RT-next and RT-select and compare them to the state of the art. The results show that on average, RT improves performance by 19.5% and reduces energy consumption by 17%.

The rest of the paper is organized as follows: Section 2 discusses DRAM background. Section 3 motivates the RT design and explains the two RT schemes. Section 4 presents the models used in this work. Sections 5 to 7 present the experimental methodology, analyze the results and perform sensitivity study, respectively. We discuss related work in Section 8 and conclude the paper in Section 9.

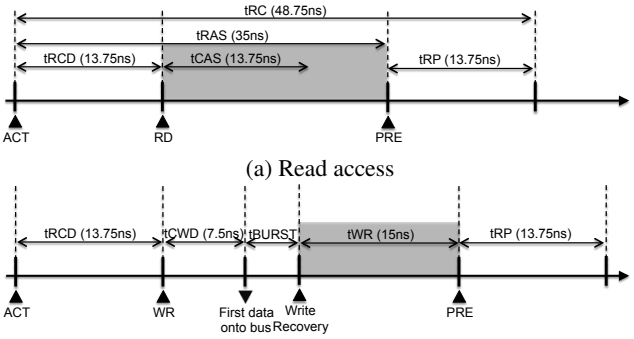
978-1-4673-9211-2/16/$31.00 ©2016 IEEE

2. BACKGROUND

2.1 DRAM Basics

DRAM has been widely adopted to construct main memory for decades. A DRAM cell consists of one capacitor and one access transistor. The cell represents bit '1' or '0' depending on if the capacitor is fully charged [1] or discharged.

DRAM supports three types of accesses — *read*, *write*, and *refresh*. An on-chip memory controller (MC) decomposes each access into a series of commands sent to DRAM modules, such as `ACT` (*Activate*), `RD` (*Read*), `WR` (*Write*) and `PRE` (*Precharge*). A DRAM module responds passively to commands, e.g., `ACT` destructively latches the specified row into the row buffer through charge sharing, and then restores [2] the charge in each bit cell of the row; `WR` overwrites data in the row buffer and then updates (restores) the values into a row's cells. All commands are sent to the device following predefined timing constraints in the DDRx standard, such as `tRCD`, `tRAS` and `tWR` [21, 22]. Figure 1 shows the commands and their typical timing parameter values [22, 5].



(a) Read access

(b) Write access

Figure 1: Commands involved in DRAM accesses.

2.2 DRAM Restore and Refresh

DRAM Restore. Restore operations are needed to service either read or write requests, as shown by the shaded portions in Figure 1. For reads, a restore reinstates the charge destroyed by accessing a row. For writes, a restore updates a row with new data values.

DRAM Refresh. DRAM needs to be refreshed periodically to prevent data loss. According to JEDEC [22], 8K all-bank auto-refresh (`REF`) commands are sent to all DRAM devices in a rank within one retention time interval (`Tret`), also called as one refresh window (`tREFW`) [7, 40, 10], typically 64ms for DDR*x*. The gap between two `REF` commands is termed as refresh interval (`tREFI`), whose typical value is 7.8μs, i.e. 64ms/8K. If a DRAM device has more than 8K rows, rows are grouped into 8K **refresh bins**. One `REF` command is used to refresh multiple rows in a bin. An internal counter in each DRAM device tracks the designated rows to be refreshed upon receiving `REF`. The refresh operation takes `tRFC` to complete, which proportionally depends on the number of rows in the bin.

[1]In this paper, a cell is considered as fully charged if its voltage reaches 0.975V_{dd} [16]. Our proposed schemes are applicable if a cell needs to reach V_{dd} to be fully charged.

[2]Restore is performed concurrently wth data read [21].

The refresh rate of one bin is determined by the leakiest cell in the bin. Liu *et al.* [36] reported that fewer than 1000 cells require a refresh window shorter than 256ms in a 32GB DRAM main memory. Given that the majority of rows have retention time longer than 64ms, it is beneficial to enable multi-rate refresh, i.e., different bins are refreshed at different rates. For discussion purpose, a DRAM cell/row/bin that is refreshed at 256ms is referred to as a 256ms-cell/row/bin, respectively.

We adopt the *flexible auto-refresh* mechanism from [8] to support multi-rate refresh, i.e., 8K refresh commands are sent every 64ms — one for each bin. If a bin needs to be refreshed every 256ms, *flexible auto-refresh* sends four `REF` commands in 256ms to this bin. However, only one is a real refresh while the other three are dummy ones that only increment the refresh counter. We assume that the memory controller knows the mapping between bin address and row address, the same as that in [8], and similar to [31].

3. RESTORE TRUNCATION

In this section, we first motivate why it is useful to partially charge (restore) a cell by truncating restore operations. We then describe design details of two restore truncation schemes: `RT-next` and `RT-select`.

3.1 Motivation

Scaling DRAM to 20nm and below faces significant manufacturing difficulties: cells become slow and leaky [48, 53] and exhibit a larger range of behavior due to process variation (i.e., there is a lengthening of the tail portion of the distribution of cell timing and leakage) [26, 56, 40].

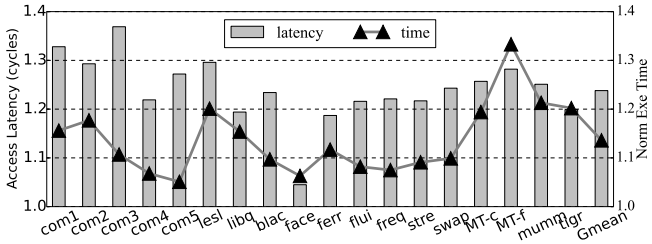


Figure 2: Access latency and execution time increase due to relaxed restore timing. `Baseline` adopts standard timing constraints in specifications.

As bit cell size is reduced, the supply voltage V_{dd} also reduces, causing cells to be leakier and store less charge [40]. For instance, DDR3 commonly uses 1.5V V_{dd}, while DDR4 at 20nm uses 1.2V [2, 40]. Performance oriented DRAM enhancements, such as high-aspect ratio cell capacitors [26, 40], often worsen the situation. DRAM scaling also increases noise along bitline and sensing amplifier [40, 45, 33], which leads to longer sensing time. Scaling also degrades DRAM restore operation due to smaller transistor size, lower drivability and larger resistance [26, 56, 48, 53].

The growing number of slow and leaky cells has a large impact on system performance. There are three general strategies to address this challenge:

- The first choice is to keep conventional hard timing constraints for DRAM, which makes it challenging to

Table 1: Adjusted restore timing values in `RT-next` (using the model in Section 4)

sub-window	Distance to next refresh			Target restore voltage (V_{dd})	tRAS	tWR	tRCD
	64ms-row	128ms-row	256 ms-row		(DRAM cycles)		
1st	[64ms, 48ms)	[128ms, 96ms)	[256ms, 192ms)	0.975	42	25	15
2nd	[48ms, 32ms)	[96ms, 64ms)	[192ms, 128ms)	0.92	27	18	15
3rd	[32ms, 16ms)	[64ms, 32ms)	[128ms, 64ms)	0.86	21	14	15
4th	[16ms, 0ms)	[32ms, 0ms)	[64ms, 0ms)	0.80	18	11	15
No Truncation				0.975	42	25	15

handle slow and leaky cells. Cells that fall outside of guardbands could be filtered (not used). With scaling, however, this approach can incur worse chip yield and higher manufacturing cost. Because the DRAM industry operates in an environment of exceedingly tight profit margins, reducing chip yield for commodity devices is unlikely to be preferred.

- A second choice is to expose weak cells, falling outside guardbands, and integrate strong yet complex error correction schemes, e.g., `ArchShield` [43]. Due to the large number of cells that violate conventional timing constraints such as `tRCD`, `tWR`, significant space and performance overheads are expected.
- A third choice is to relax timing constraints [26, 56]. This approach is compelling because it can maintain high chip yield at extreme technology sizes. However, relaxing timing, without careful management, can cause large performance penalties, as shown in Figure 2 (see Section 5 for experimental details). On average, we observed 23.8% longer memory access latency and 13.6% longer execution time.

Because the third choice is compatible with the need for high chip density and yield, we adopt it in this paper. We relax restore timing and strive to mitigate associated performance degradation. Our design principle is also applicable to the second strategy if exposed errors can be well managed. We leave this possibility to future work.

Partial restore. Due to intrinsic leakage, the voltage level of a DRAM cell reduces monotonically after a full restore. The solid curve in Figure 3 illustrates the voltage decay of an untouched cell (i.e., not accessed) within one refresh window. Stored data is safe as long as the voltage remains above V_{min} ($0.73V_{dd}$ here, discussed in Section 4) before the next refresh. If a read or write access occurs, the post-access restore operation fully charges the cell by default, as shown in the figure. However, the full charge is often unnecessary if the access is *close in time* to the next refresh, which will fully restore the cell anyway.

Based on this observation, we propose that post-access restore *partially charges* a cell's voltage to the level that the cell would have if the cell had been untouched in one refresh window. The restore operation is truncated when this target voltage level is reached.

The cell charging curve starts with a deep slope and flattens when approaching V_{full} [34, 16], as shown in SPICE modeling and simulation results in Section 4. Hence, reducing target voltage can drastically shorten restore time. For example, SPICE modeling indicates that restoring a cell's charge to $0.89V_{dd}$ rather than $0.975V_{dd}$ (fully charged) reduces `tWR` from 25 to 15 DRAM cycles, i.e., a 40% reduction.

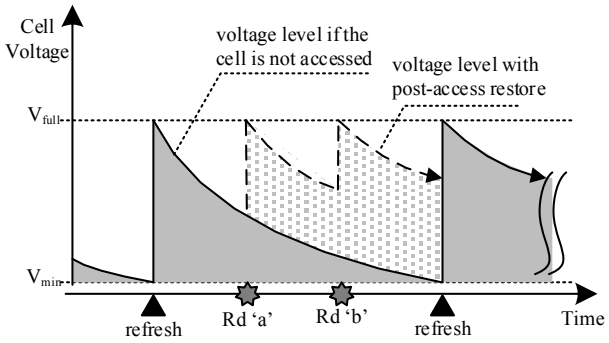


Figure 3: DRAM cell voltage is fully restored by either refresh commands or memory accesses. (V_{full} indicates fully charged; V_{min} is the minimal voltage to avoid data loss).

We next describe two schemes, `RT-next` and `RT-select`, to enable partial restore. These schemes are applied by the memory controller.

3.2 RT-next: Refresh-aware Truncation

`RT-next` truncates a long restore operation according to the time distance to its next refresh. The sooner the next refresh is, the less charge the cells in the row need, and the earlier the restore operation can be terminated.

`RT-next` works as follows. We partition one refresh window into multiple sub-windows. While accesses falling in different sub-windows use different sets of timing parameter values, those falling in the same sub-window use the same set of values. In the following, we use four sub-windows to discuss our proposed schemes — Table 1 lists the adjusted timing values for the device that we model in this paper. The smaller the timing values are, the larger truncation opportunity the truncation has. While distinguishing more sub-windows provides finer-grained control and the potential to exploit more truncation benefits, it complicates the control and provide little further benefits as shown in our experiments.

When servicing a read or write access, `RT-next` uses the following formula to calculate the time distance to the next refresh command and determine the sub-window that the access falls in. It then truncates its restore operation using the adjusted timing parameters, e.g., the right most columns in Table 1.

$$Distance = ((8192 + Bin_c - Bin_a)\%8192 + 1) \times \frac{64ms}{8192} \quad (1)$$

where $\texttt{Bin}_c$ is the last bin that was refreshed; $\texttt{Bin}_a$ is the refresh bin to which the row being accessed belongs. In multi-rate scenario, the calculation is adjusted to include the further 64ms refresh rounds, which will be discussed later.

The above calculation needs the mapping from row address to bin address. While the bin counter is maintained in the memory controller and incremented sequentially, the actual row addresses (responding to each bin-refresh command) are generated internally inside DDR*x* devices [22, 23]. This mapping may be non-linear because of vendor's full flexibility to implement the refresh. Recent studies [8, 31] assume this mapping can be made known to the memory controller. We make the same assumption in this paper.

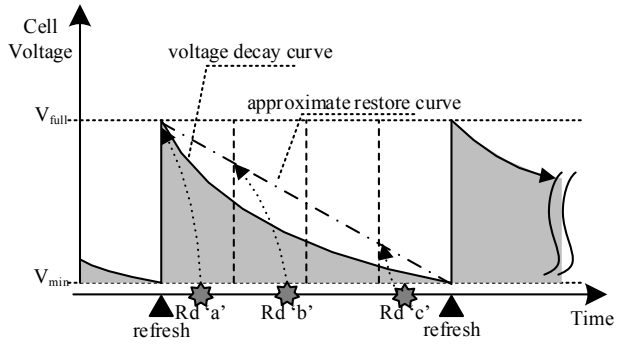


Figure 4: `RT-next` safely truncates restore operation.

The memory controller also needs to consider the page policy (open or close). A restore is truncated by a `PRE` command from the memory controller. For a closed-page policy, every access can potentially benefit from restore truncation. For an open-page policy, truncating restore of a preceding access may not beneficial if its following access is a row buffer hit. We evaluate both policies in the experiments.

To adapt to cell variations within a DRAM row, `RT-next` takes a conservative approach, as illustrated in Figure 4. In the example, reads `'a'`, `'b'`, and `'c'` are serviced in the first, the second, and the fourth sub-windows, respectively.

- `RT-next` assumes the worst case scenario, i.e., the currently accessed row has weak cells that barely meet timing constraints and these weak cells are leaky enough that their voltage levels are reduced to V_{min} before the next refresh. The weak cells are difficult to restore because fully charging them requires long latency. The adjusted restore timings in Table 1 ensure that slow and leaky cells can accumulate charge more than what they have under natural decay, i.e., they are not accessed in one refresh window.
- `RT-next` restores to the target level at the beginning of each sub-window. In particular, while it is possible to partially restore an accessed row in the first sub-window, e.g., read `'a'` in Figure 4, RT-next conservatively fully restores the row, i.e., with no truncation.
- Due to a slightly faster rate of leakage at higher voltage (as shown in Section 4), a DRAM cell has an exponential decay curve that is close but always below the linear line between V_{full} and V_{min} in Figure 4. This curve varies from row to row, which implies that different restore timing values are needed. To simplify the control in memory controller, `RT-next` conservatively sets up the voltage restore targets, at the beginning of each sub-window, as the voltage levels on the linear line, rather than on the curve. This allows `RT-next` to use the same timing parameters for all rows.

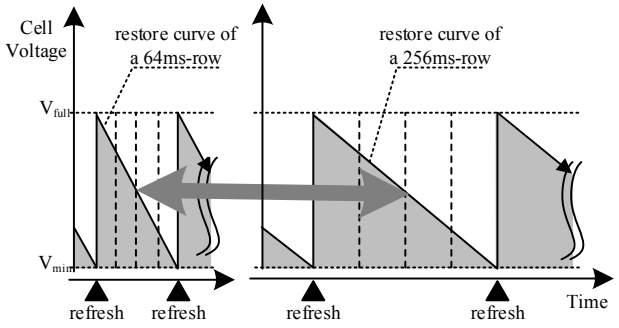


Figure 5: Restoring voltage according to linear line simplifies timing control in multi-rate refresh — a 64ms-row and a 256ms-row share the same timing values in each correspond sub-window.

`RT-next` **in multi-rate refresh.** Applying `RT-next` in a multi-rate refresh environment works similar to the case that has only one rate. To calculate the distance between a memory access and the next refresh to its bin, `RT-next` uses the same formula except adding the extra refresh rounds for low rate, i.e., 128/256ms, bins. Here we assume the underlying multi-rate refresh scheme has profiled and tagged each bin with its best refresh rate, e.g., 64ms, 128ms, or 256ms.

As shown in Figure 5, it simplifies the timing control in memory controller by restoring a cell's post-access voltage according to the linear line between V_{full} and V_{min} (rather than the exponential decay curve). Given a 64ms-row and a 256ms-row, accesses falling in the same corresponding sub-window can use the same timing values in Table 1.

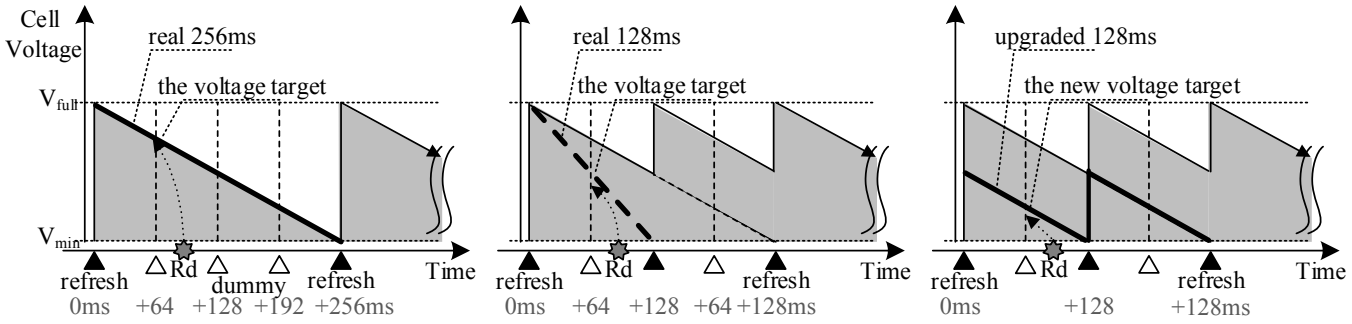
3.3 RT-select: Proactive Refresh Rate Upgrade

We next present `RT-select`, a scheme that upgrades refresh rate for more truncation opportunities. Refresh and restore are two correlated operations that determine the charge in a cell. Less frequently refreshed bins can be exploited to further shorten the post-access restore time.

Figure 6 illustrates the benefit of refreshing a 256ms-row (in multi-rate refresh) at 128ms rate. Given that this row is a 256ms-row, its voltage level decreases to V_{min} after 256ms. As shown in Figure 6(a), the refresh commands sent at +64ms, +128ms, and +192ms marks are dummy ones. The access "Rd" appears in the 2nd sub-window; it has a distance within [192ms, 128ms) to the next refresh command. According to `RT-next`, the restore can be truncated after reaching $0.92V_{dd}$ (using the 256ms-row column in Table 1).

Now, suppose the dummy refresh at +128ms is converted to a real refresh, i.e., the row is "upgraded" to a 128ms-row. With this earlier refresh, the access "Rd" is at most 64ms away from the next refresh. Using the 128ms-row column in the timing adjustment table, `RT-next` can truncate the restore after it reaches $0.86V_{dd}$, achieving significant timing reduction for the restore operation (Figure 6(b)).

Refreshing a 256ms-row at 128ms rate exposes more truncation benefits, as shown in Figure 6(c). For access "Rd", it is sufficient to restore the voltage to $0.80V_{dd}$ rather than $0.86V_{dd}$ in above discussion. This is because a 256ms-row,



(a) the voltage target of a 256ms-row (b) the voltage target of a 128ms-row (c) the voltage target Decreases when a higher refresh rate (128ms) is employed

Figure 6: The voltage target can be reduced if a strong row is refreshed at higher rate.

even if being refreshed at 128ms rate, leaks slower than a real 128ms-row. We can adjust the timing values by following the solid thick line in 6(c), rather than the dashed thick line from a real 128ms-row, as shown in 6(b). In particular, as summarized in Table 2, a row access, even if it is 128ms away from the next refresh to the row, just needs to restore the row to $0.86V_{dd}$, rather than V_{full} (=$0.975V_{dd}$) for a real 128ms-row.

Table 2: Adjusted restore timing values in `RT-select`

Upgrade	Distance to Next refresh	Target restore voltage (V_{dd})	tRAS (DRAM cycles)	tWR (DRAM cycles)	tRCD (DRAM cycles)
256ms->128ms	[128ms, 64ms)	0.86	21	14	15
	[64ms, 0ms)	0.80	18	11	15
256ms->64ms	[64ms, 0ms)	0.80	18	11	15
128ms->64ms	[64ms, 32ms)	0.86	21	14	15
	[32ms, 0ms)	0.80	18	11	15

RT-select scheme. While upgrading refresh rate reduces restore time, it generates more real refresh commands, which not only prolongs memory unavailable period but also consumes more refresh energy. Previous work shows that refresh may consume over 20% of the total memory energy for a 32Gb DRAM device [7, 36]. Blindly upgrading the refresh rate of all rows is thus not desirable.

`RT-select` upgrades the refresh rate of *selected bins*, those were touched, for *one refresh window*. It works as follows. A 3-bit rate flag is attached to each refresh bin to support multi-rate refresh. When there is a need to upgrade, e.g., from 256ms to 128ms rate, `RT-select` updates the rate flag as shown in Section 3.5, which converts the dummy refresh at +128ms in Figure 6. A real refresh command rolls the rate back to its original rate, i.e., `RT-select` only upgrades the touched bin for one refresh window, which incurs modest refreshing overhead to the system.

3.4 Discussion

Restore truncation (`RT`) is an orthogonal design to the state of the art DRAM enhancements. We have discussed how to support multi-rate refresh [8, 36]. As another example, `NUAT` [46] reduces `tRCD/tRAS` timings by exploiting the potential between V_{min} and the voltage of a cell under natural decay. `RT` restores a cell's post-access voltage to no less than this level and thus is fully compatible with `NUAT`.

`RT` also works with strong error correction (e.g., `ArchShield` [43]) and the recently proposed restore time mitigation scheme [56]. As shown in experiments, these schemes improve performance close to the one with no timing degradation. `RT` can be integrated with these designs.

`RT` does not take advantage of thermal impact on restore timing. The timing parameters used in the paper ensure reliable operation in the chip's recommended temperature range. Additional truncation opportunities may be exploited if thermal behavior is considered.

Recent studies revealed the complication in profiling the retention time of DRAM modules, which comes from two phenomena: data pattern dependence (DPD) and variable retention time (VRT) [35, 40]. DPD can be alleviated by repeated testing and ECC protection. VRT can be alleviated by enhancing profiling through ECC and guardbanding techniques [29, 44]. Our study works in conjunction with existing profiling techniques on these issues.

3.5 Architecture Enhancements

To enable `RT-next` and `RT-select`, we enhance the memory controller, as shown in Figure 7. `RT` adds a truncation controller, to adjust the timing for read, write, and refresh accesses. This control is similar to past schemes that change timings [15, 49, 46]. The memory controller has a register that records the next bin to be refreshed, referred to as $\texttt{Bin}_c$, which rolls over every 64ms. It can also infer the mapping from row address to refresh bin, the same as that in [8, 31].

Table 3: Refresh rate adjustment table

Profiled refresh rate	Rate flag	Flag after rate upgrade
64ms	000	n/a
128ms	01*A*	128→64ms: 010
256ms	1*BC*	256→128ms: 1BC⊕0B0 256→64ms: 100

To support multi-rate refresh, the memory controller keeps a small table that uses 3 bits to record the refresh rate of each refresh bin, similar to that in [36, 8]. As shown in Table 3, a 64ms-/128ms-/256ms- bin is set as '000'/'01A'/'1BC', respectively. Here 'A' and 'BC' are initialized to ones and decrement every 64ms. While the refresh bin counter increments every in 7.8μs(=64ms/8K), a real `REF` command is sent to refresh the corresponding bin only if its bin flag is '000', '010', or '100'. 'A' and 'BC' are changed back to '1' and '11' afterwards, respectively.

When upgrading the refresh rate of a refresh bin, we update the rate flag according to the last column in Table 3. For example, when upgrading a 128ms-bin to 64ms rate, we

set the rate flag as '010', which triggers the refresh in the next 64ms duration and roll back to '011' afterwards. This effectively upgrades for one round. Upgrading 256ms-row to 128ms rate sets the flag as '1BC⊕0B0', which always sets the middle bit to zero to ensure that the refresh distance is never beyond 128ms, and thus the sub-window can only be `3rd` and `4th` referring to Table 1. In general, the distance calculation in `RT-select` is adjusted by adding value in Equation (1) with the further refresh rounds indicated by the two least significant bits (LSB) of rate flag.

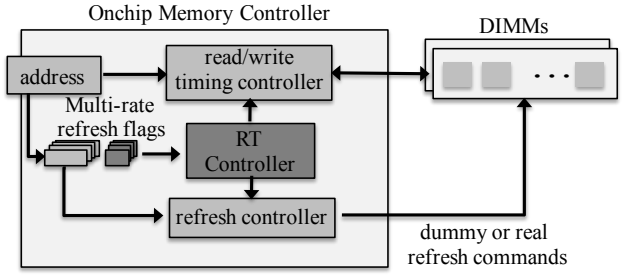


Figure 7: The RT architecture (the shaded boxes are added components).

To enable multi-rate refresh, the rate table is accessed before each refresh to determine if a real or dummy command should be sent. To enable `RT-select`, the rate table is also accessed before each memory access to decide the refresh distance, and then to complete the upgrade after the access. The extra energy and latency overheads are minimal, as shown in Section 6.4.

4. MODELING DETAILS

In this section, we present the details of modeling DRAM that are the underpinning of our `RT` schemes, including sensing delay, restore time, and retention time.

4.1 Voltage Drop

The stored charge in a DRAM cell capacitor leaks over time through its access transistor. The leakage current I_{leak} is mainly sub-threshold leakage [12, 24], and it is exponentially relates to V_{cell}, which indicates that the cell voltage drops following an exponential curve.

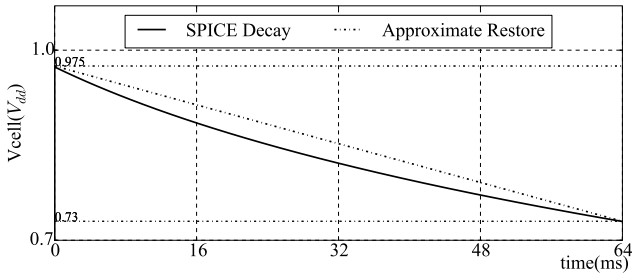


Figure 8: SPICE modeling of cell voltage drop. `SPICE Decay` is the exponential curve from SPICE simulation; `Approximate Restore` is a linear line from V_{full} to V_{min}, which is exploited to set up restore voltage targets in each refresh sub-windows.

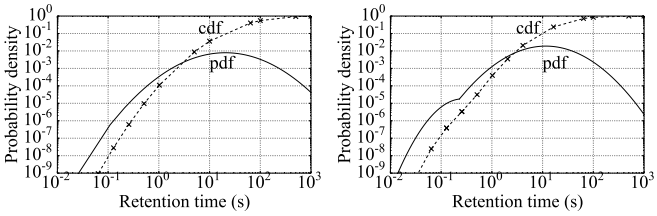
We further built a SPICE model and reported the cell voltage drop within a normal refresh window in Figure 8, which confirms the exponential decay.

4.2 Retention Time and Refresh

The amount of time that a DRAM cell can safely retain data is defined as *retention time*, T_{ret}, which is determined by the magnitude of the leakage current and the total charge that is allowed to lose [19, 32, 51]. Following previous work [6, 32], we define T_{ret} as the time until the capacitor charge/voltage leaks to the minimum sustainable value (i.e., $(0.975-0.73)V_{dd}$, which is more conservative than the 60% maximum lose used in [6, 32]). T_{ret} can be denoted as

$$T_{ret} = \frac{(V_{cell} - V_f) \times C_{cell}}{I_{leak}} = \frac{(V_{cell} - 0.73V_{dd}) \times C_{cell}}{I_{leak}} \quad (2)$$

where, I_{leak} is the leakage current, and V_f is the minimum readable stored voltage, which is $0.73V_{dd}$.



(a) Current technology (b) Future technology

Figure 9: T_{ret} trend as DRAM scales down. [30, 36]

We modeled DRAM retention time distribution based on [30, 36, 43], and reported the the results of current and future technologies as shown in Figure 9.

Similar to prior works [36, 8, 54], leaky cells are randomly distributed throughout DRAM. We derive the weak row distribution by converting the weak cell failure probability into failure row probability.

Our modeling results show that (i) while cells are becoming more leaky, the number of cells and rows that have less than 64ms retention time is still very small, which can be corrected by enhanced error rescue schemes, like *ArchShield* [43]. Hence, refreshing all DRAM chip rows per 64ms is sufficient to prevent data loss. (ii) The retention timing for cells in current commodity DRAM chips is conservative, which inspired designs to tighten timing for performance improvement [9, 34]. The opportunity is diminishing in future chips as more cells become leaky.

4.3 Sensing and Restoring Time

DRAM scaling has negative impact on sensing and restore time. DRAM cell is read out by sensing the voltage difference on bit line after charge sharing. The difference is given by the expression[28, 25, 33]:

$$\Delta V_{BL} = \frac{(V_{cell} - V_{BL}) \times C_{cell}}{C_{BL} + C_{cell}} \quad (3)$$

where ΔV_{BL} is the small voltage increase on bitline, and C_{BL} (V_{BL}) and C_{cell} (V_{cell}) are the capacitance (voltage) of bitline and cell, respectively. However, offset noise [40, 35] weakens ΔV_{BL} [18], which might lead to read failure. To correctly read the cell content, the effective signal is required to be

larger than zero:

$$\Delta V_{effective} = \Delta V_{BL} - \Delta V_{noise} > 0 \quad (4)$$

The noise voltage of existing DRAM [46, 34, 52] is conservatively set to 25mV as shown in [18]. For further scaling, a smaller V_{dd} of 1.0V is used referring to [47, 20]. In this paper, we make a conservative assumption that doubles the existing offset value of 25mV to 50mV, which indicates a $0.73V_{dd}$ minimum cell voltage following Equations (3) and (4).

DRAM restore time is degraded due to deceasing transistor drivability. While tWR has been kept at 15ns across generations, it is challenging to continue this value for sub-20nm technologies [26]. We followed [56] to obtain the distribution of tRAS and tWR, where extremely slow cells/rows are rescued by existing redundancy techniques.

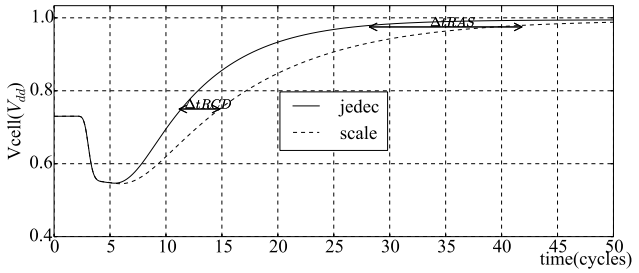


Figure 10: SPICE modeling on tRCD and tRAS. jedec follows the JEDEC specification and scale meets the projected values. tRCD ends at $0.75V_{dd}$ [46, 34], and tRAS completes at $0.975V_{dd}$.

We built core circuits for a DRAM array, including cell, sense amplifier, write driver and column mux, etc, and simulated them in SPICE. The circuits have generic topologies [21] and their transistor parameters were taken (and projected) from a DRAM modeling tool [52]. The obtained circuit curve is as shown in Figure 10. From the figure, we can see that both tRCD and tRAS are increased in future DRAM.

In this paper, we focus on the relationship between restore and retention. Consequently, unrelated timing values, such as tRCD, are unchanged [3].

5. EXPERIMENTAL METHODOLOGY

5.1 System Configuration

To evaluate the effectiveness of our proposed designs, we performed the simulation using the memory system simulator USIMM [11], which simulates DRAM system with detailed timing constraints. USIMM was modified to conduct a detailed study of refresh and restore operations.

We simulated a quad-core system with settings listed in Table 4, similar to those in [42, 46]. The DRAM timing constraints follow Micron DDR3 SDRAM data sheet [5]. By default, DRAM devices are refreshed with 8K REF within 64ms, and tRFC is 208 DRAM cycles, which translates into a tREFI of 7.8 μs (i.e., 6240 DRAM cycles). As [42], the baseline adopts closed page policy, which is preferred in multicore systems [37].

[3]Whereas this paper places a particular focus on restoring time study, the proposed schemes are orthogonal to tRCD reduction designs.

Table 4: System Configuration

Processor	four 3.2Ghz cores; 128 ROB size Fetch width: 4, Retire width: 2, Pipeline depth: 10
Memory Controller	Bus frequency: 800 MHz Write queue capacity: 64 Write queue high watermark: 40 Write queue low watermark: 20 Address mapping: rw:cl:rk:bk:ch:offset Page management policy: closed-page with FRFCFS
DRAM	2channels, 1rank/channel, 8banks/rank, 64K rows/bank, 8KB/row, 64B cache line tCK=1.25ns, width: x8 tCAS(CL): 13.75ns, tRCD: 13.75ns, tRC: 48.75ns tCWD: 6.25ns (5 cycles), tBURST: 5.0ns (4 cycles) tRAS: 35ns, tRP: 13.75ns, tFAW: 24 cycles, tRRD: 5 cycles, tRFC: 208nCK, tREFI: 7.8μs

5.2 Workloads

Table 5 lists the workloads for evaluation. They are from the Memory Scheduling Championship [1], and cover a wide variety of benchmarks, including five commercial applications *comm1* to *comm5*, nine benchmarks from PARSEC suite and two benchmarks each from the SPEC suite and the Biobench suite. Among them, *MT-fluid* and *MT-canneal* are two multithreaded workloads. As [42], the benchmarks are executed in rate mode, and the time to finish the last benchmark is computed as the execution time.

Table 5: Workloads

COMMERCIAL	comm1, comm2, comm3, comm4, comm5
PARSEC	Black, face, ferret, fluid, freq, stream, swapt, MT-canneal, MT-fluid
SPEC	leslie, libq
BIOBENCH	mummer, tigr

6. RESULTS AND ANALYSIS

6.1 Schemes to Study

To evaluate the effectiveness of RT schemes, we studied the following schemes:

— Baseline. This scheme adopts the projected relaxed timing (tRCD=15, tRAS=42, and tWR=25) in future DRAM chips. The timing is applied to all rows and chips, and fits the worst-case.

— ConvTm. This scheme assumes the conventional timings (tRCD=11, tRAS=28, and tWR=12) for future DRAM chips. This is an ideal scheme as it is unclear how to efficiently handle the large number of weak cells that cannot meet these timings.

— NoRefresh. This scheme assumes no refresh activity in Baseline, which eliminates its impact on performance as well as energy consumption. It marks the performance upper bound of multi-rate refresh and other enhancement designs, including RAIDR[36], RefreshPausing [42] and ArchShield [43].

— RT-next-f64/-var. This scheme is built on top of Baseline, and truncates a long DRAM restore operation based on its distance to the next refresh event.

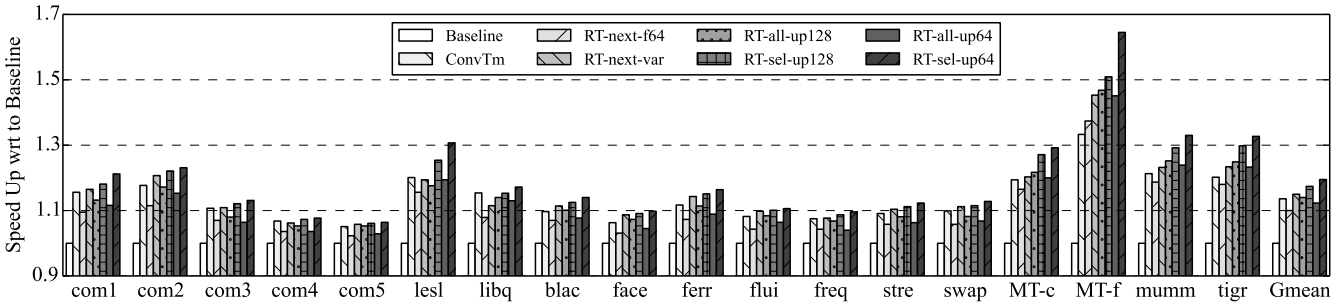


Figure 11: Performance comparison of different schemes.

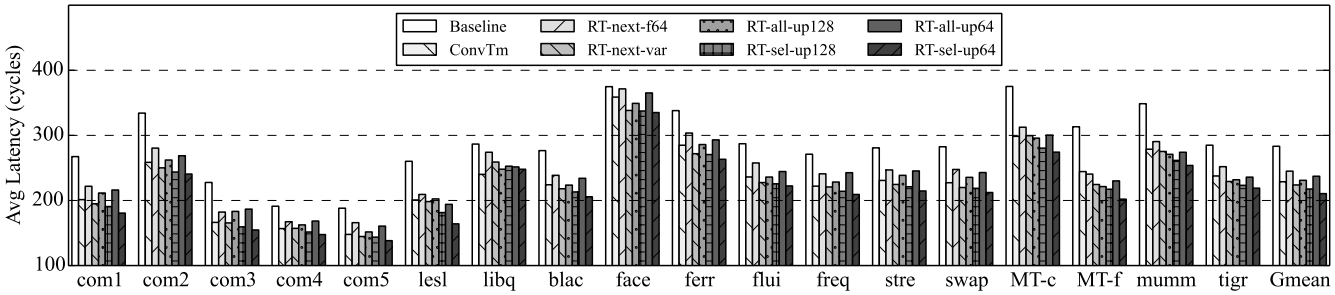


Figure 12: Access latency comparison of different schemes.

Whereas `RT-next-f64` assumes that all rows adopt 64ms refresh rate, `RT-next-var` explores the retention time variation to truncate refresh operations.

— `RT-all-up128/-up64`. This scheme is to trade refresh for restore truncation benefits by converting dummy refresh commands to real ones. It upgrades the refresh bins that have lower than 128/64ms rate to use 128/64ms rate.

— `RT-sel-up128/-up64`. This scheme is similar to `RT-all-`. The difference is that it does upgrade only for the touched bins.

Next, we compared the schemes on system performance, memory access latency and energy, and then studied their sensitivities to different configurations. To study different aspects of RT, we analyze different set of schemes in each part.

6.2 Impact on Performance

Figure 11 compares the execution time of different schemes. The results are normalized to `Baseline`. In the figure, `Gmean` is the geometric mean of the results of all workloads.

On average, `RT-next-f64` achieves 10% improvement over `Baseline` by truncating restore time. `RT-next-var` identifies more truncation opportunities in multi-rate refresh DRAM modules and achieves better, i.e., 15%, improvement. While `RT-all-up128` truncates more restore time through refresh rate upgrade, it introduces extra refresh overhead and thus is slightly worse than `RT-next-var`. `RT-sel-up128` achieves 2.4% improvement over `RT-next-var` by balancing refresh operations and restore benefits. The performance gap between upgrading all rows and selective upgrading is even larger when we aggressively upgrade refresh rate to 64ms. `RT-sel-up64` achieves the best performance — it is 19.5% speedup over `Baseline`, or 4.5% better than `RT-next-var`. The performance trend across the schemes demonstrates that our restoring schemes achieves a good balance between refresh and restore.

Generally, memory access intensive workloads such as *com1*, *libq* and *mumm* benefit most from the reduced restore timing. Particularly, *MT-f* obtains the largest performance improvement because of the parallel access patterns and relatively tight gaps between accesses, which greatly enlarges the effect of shortened `RAS` and `WR`.

6.3 Impact on Access Latency

Figure 12 compares the memory access latencies using different schemes. The average access latency of `Baseline` is 283 DRAM cycles. Restore time reduction effectively reduces the latency for all workloads. `RT-all-up64` is worse than `RT-all-up128` due to more real refresh operations slowing down normal memory accesses. `RT-sel-up64` reduces the average latency to 210 DRAM cycles, indicating a 25.8% reduction over `Baseline`.

6.4 Energy Consumption

Figure 13 compares the energy consumption of different schemes. We reported the energy consumption breakdown — background (`bg`), activate/precharge (`act/pre`), read/write (`rd/wr`) and refresh (`ref`). We summarized the results according to benchmark suites, where results are averaged over workloads within each suite. We used the Micron power equations [39], and the parameters from vendor data sheets [5] with scaling.

To enable truncation in multi-rate refresh DRAM modules, we need to query the refresh rate for each access. The refresh rates for 8K bins are organized as 3KB direct mapped cache with 8B line size. We used CACTI5.3 [3] to model the cache with 32nm technology — it requires 0.22ns access time, occupies 0.02mm^2 area, consumes 1.47mW standby leakage power, and spends 3.33pJ energy per access. The extra energy is trivial (less than 0.5%) and is reported to-

gether with bg.

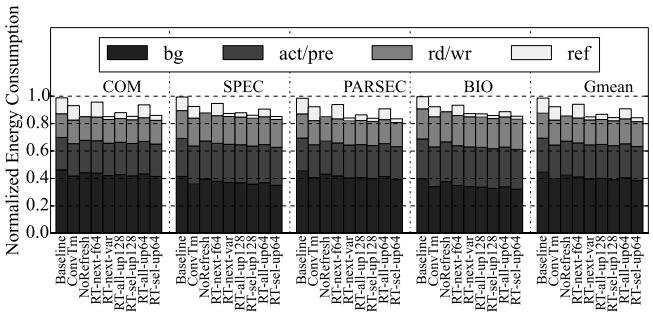


Figure 13: Comparison of memory system energy.

From the figure, we observed that the device refresh energy for 4Gb chips is small. Due to increased refresh operations, RT-all-up128/-up64 consume more refresh energy than RT-sel-up128/-up64, respectively. RT-sel-up64 saves 17% energy compared to Baseline, and consumes slightly lower energy than NoRefresh due to decreased execution time. And, as expected, RT-sel- refresh schemes is more energy efficient than RT-all- refresh peers.

6.5 Comparison against the State-of-the-art

Figure 14 compares RT with three related schemes in the literature.

- Archshield+ implements a scheme that treats all the cells with long restore latency as failures and adopts Archshield [43] to rescue them.
- MCR is the recently proposed scheme that trade DRAM capacity for better timing parameters [14]. 2x MCR and 4x MCR are the two options that reduce DRAM capacity to 50% and 25% of the original, respectively.
- ChunkRemap implements the scheme that differentiates chunk level restore difference and constructs fast logic chunks through chunk remapping [56].

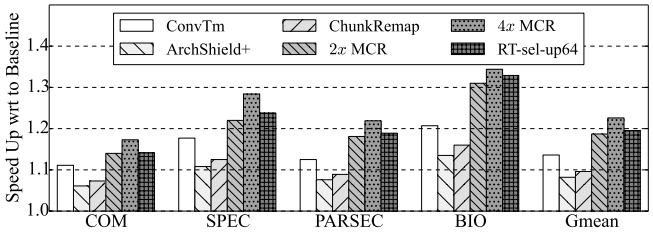


Figure 14: Comparison against the state-of-the-art.

The figure shows that Archshield+ and ChunkRemap are approaching ConvTm while RT-sel-up64 is 5.2% better than ConvTm, exploiting more benefits from reduced restore time.

MCR shares similarity with RT-select, i.e., we share the observation that a line that is refreshed more frequently can be restored to a storage level lower than V_{full}. MCR exploits this with significant DRAM capacity reduction while RT-select takes a light weight design that upgrades used bins only for one refresh window, and leaves all the other bins being refreshed at original rates.

From the figure, 4x MCR outperforms RT-sel-up64 by a modest percentage. This is because MCR improves the baseline by reducing not only restore time but also sensing time while RT-sel-up64 focuses only restore time. RT-sel-up64 works better than 2x MCR because it upgrades the refresh rate of a bin for one refresh window at a time, which significantly reduces refresh overhead (as shown with the difference from RT-all-up64).

Table 6: Comparing EDP between RT and MCR (lower is better).

Cases	ConvTm	RT-sel-up64	2x MCR	4x MCR-4
Same Chip	1.0×	0.715×	0.753×	0.713×
Same Capacity	1.0×	0.715×	0.918×	1.068×

Given that MCR improves performance at a significant capacity reduction. We next comparing the energy-delay-product (EDP) — "Same Chip" is optimistic assumption as 4x MCR has only 25% available capacity, which is likely to have more page faults in practice. "Same capacity" enlarges the raw chip in MCR by two/four times, which introduces more background power. RT-sel-up64 shows good potential as its EDP closely matches that of MCR under "Same chip" setting, and is much better under "Same capacity" setting.

6.6 Comparison against the Ideal

Table 7: Bound schemes to study.

Schemes	tRAS	tWR	tRCD	Refresh rate
Baseline	42	25	15	64ms
NoRefresh	42	25	15	-
ConvTm	28	12	11	64ms
BestInterval	18	11	15	64ms
BestIntNoRef	18	11	15	-

To further evaluate the effectiveness of RT, Figure 15 compares the proposed RT schemes against several *ideal* schemes. These schemes are *ideal* because they are infeasible in practice — Table 7 summarizes their timing values. NoRefresh eliminates all refresh operations in Baseline to set the performance upper bound of refresh enhancement schemes; ConvTm adopts conventional timings. BestInterval uses the the best timings (tRAS=21, tWR=11 and tRCD=15) reported in Table 1; BestIntNoRef further eliminates refresh operations in BestInterval.

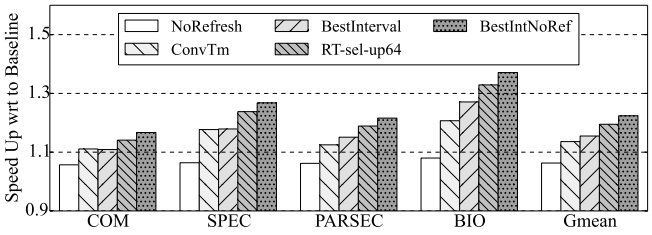


Figure 15: Comparison of RT-sel-up64 to candidate ideal schemes.

From the figure, NoRefresh and ConvTm are 6.3% and 13.6% better than Baseline, respectively, showing the large performance impacts from refresh activities and degraded timing. RT-sel-up64 is 12.4% better than NoRefresh, indicating that improving restore timing is more valuable than

reducing refresh operations. RT-sel-up64 beats both ConvTm and BestInterval. This is promising because, even though matching the conventional timing becomes challenging in future DRAM, more benefits can be gained by exposing and exploiting restore time differences. For most benchmarks, the gap to BestIntNoRef is less than 3%.

7. SENSITIVITY STUDIES

Next, we evaluated the performance sensitivity by varying configurations including chip density, refresh granularity, refresh sub-window division and page management policy.

7.1 Chip Density

Given that one refresh command is sent to refresh all rows in one refresh bin, the larger the chip capacity, the more rows the command needs to refresh, and further the larger tRFC is. [4, 40, 42] show that tRFC may grow from 260ns for 4Gb chips to 640ns for 32Gb chips.

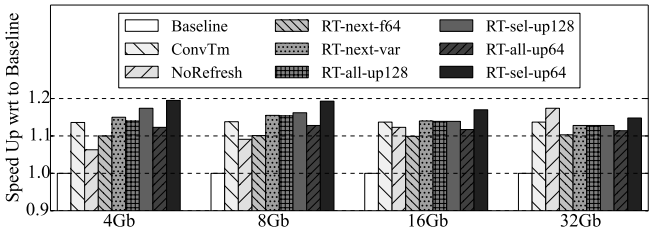


Figure 16: Impact of chip size on performance.

Figure 16 compares the results under different chip densities. The results show that refresh penalties go up as chip size increases. RT-sel-up64, while outperforming NoRefresh and ConvTm at 4Gb chip, shows decreasing benefits in larger chip sizes, and is beaten by NoRefresh and is only slightly better than the ideal ConvTm at 32Gb chip size. In addition, we observed that RT-all-up128 and RT-sel-up128 are the same as RT-next-var at 16Gb and 32Gb. The reason is that as bin size increases, from 64 to 256 and then to 512 rows, it is hard to find a bin that can be refreshed at 256ms or lower rate. This leaves no difference for RT-all-up128 and RT-sel-up128. While the improvement of RT-sel-up64 diminishes as chip size increases, it still achieves the best result among schemes except the ideal NoRefresh. And hence, RT-sel-up64 still serves as an effective scheme to mitigate restoring issues in the long future.

7.2 Refresh Granularity

DDR4 standard starts to support fine-grained refresh (FGR) modes [23, 40, 8], that is, by lowering refresh interval (tREFI) by a factor of 2× or 4× to reduce tRFC, it can send 2x (i.e., 16K) or 4x (i.e., 32K) refresh commands, instead of the 8K commands in the normal setting.

Figure 17 compares the performance of schemes using different FGR modes (X-axis). 1x/2x/4x schemes maintain 8K/16K/32K refresh bins, respectively. In Row scheme, the number of bins equals the number of rows in one bank.

Whereas RT-all- schemes observe degraded performance, all other schemes achieve better results. This is because less number of rows per REF in FGR modes help to expose more non-leaky bins, which can be further utilized to shorten the restoring timings; to the contrary, blindly refresh rate upgrading in RT-all- introduces more refresh operations, and thus lead to performance degradation. And, as the figure shows, the performance gap between RT-sel- and RT-all- schemes becomes larger at finer mode. Particularly, RT-sel-up64 achieves 32.9% performance improvement over Baseline at row granularity.

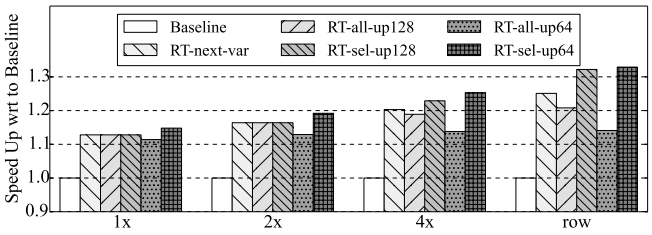


Figure 17: Sensitivity of refresh granularity on 32Gb chip (using multi-rate refresh).

7.3 Sub-window Division

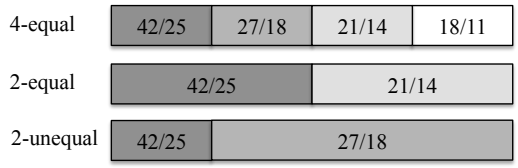


Figure 18: Using different sub-windows. Timings values are denoted as tRAS/tWR in each grid.

RT adopts sub-window based timing adjustment. It becomes more complicated for the memory controller to schedule memory requests if the number of sub-window is large. We next study the impact of the number of sub-windows on performance.

Due to the exponential (close to linear) voltage drop curve and the long-tail charge restore curve, the timing difference of the sub-window in the second half of refresh window becomes small. And Figure 15 has shown that our 4-sub-windows division is very close to the best case BestInterval. As such, it is often not necessary to differentiate more windows. Figure 18 shows the settings with two or four equal/non-equal sub-windows. The adjusted tRAS/tWR timing parameters are also listed in each sub-window.

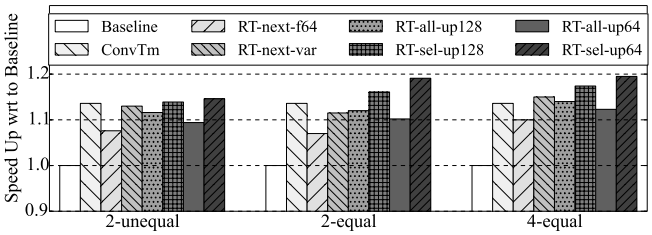


Figure 19: Comparing different numbers of sub-windows.

Figure 19 compares the performance improvements with different sub-window settings. There is a large gap between 2- and 4-sub-window designs for most schemes. And, in general, adopting four sub-window, i.e., 4-equal, achieves better performance. RT-sel-up64 is less sensitive to a small number of sub-window because RT-select charge the voltage of an upgraded row to a level much lower than V_{full},

which exploits most performance benefits.

7.4 Page Management Policy

By default, RT schemes adopt closed-page policy. We next evaluated its integration with open page-policy. We followed the recent adjustment on open-page policy [27, 8].

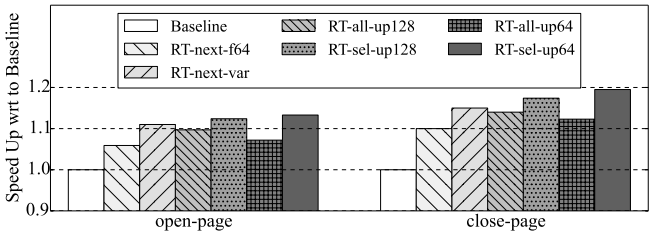


Figure 20: Performance comparison of 4Gb chip under open- and closed-page policies.

Figure 20 compares the results using different policies at 4Gb chip size. In open-page policy, the access hits in row buffer are not constrained by tRAS/tWR. Comparing to closed-page, not every access in open-page can benefit from restore truncation. Therefore, lower performance (13.3% open-page v.s. 19.5% closed-page for RT-sel-up64) improvement is expected, as shown in the figure. Similar results were observed on 8/16/32Gb chips.

8. RELATED WORK

Reducing refresh activities. As DRAM device capacity increases, refresh is expected to introduce larger performance and energy overheads [36, 42, 7]. To address the issue, a bunch of schemes have been proposed from different directions, with representative ones like Smart Refresh [17], Elastic Refresh [50], and Refresh Pausing [42], etc.

While weak cells require frequent refreshes, the majority of the cells on a DRAM chip can hold the data for a much longer time, which makes it viable to adopt multi-rate refreshes [30, 32, 6, 36, 54, 8]. Particularly, Liu *et al.* [36] proposed RAIDR to group DRAM rows into several partitions and enable different refresh rates for different partitions. Wang *et al.* [54] and Bhati *et al* [8] optimized RAIDR to make it compatible with modern DRAM standards. ArchShield was designed to tolerate high error rate in future DRAMs. It can be utilized to cover leaky cells and reduce refresh rate.

The proposed RT schemes in this work targets at the correlation of restore and refresh, and it can be integrated with the existing refresh innovations to find a better tradeoff between overall performance improvement and refresh penalty. We adopt REFLEX [8] to implement multi-rate refresh. And, the experimental results show that our proposed RT schemes are capable to win over NoRefresh, whose performance represents the upper bound of all refresh reduction designs.

Timing reduction. Reducing timing constraint values can significantly improve memory performance. TL-DRAM [15] creates row segments with low ACT and PRE latencies. CHARM [49] reduces sensing time by attaching fewer cells to each bitline. MCR [14] is a recent work that reduces both sensing time and restore time. We compare RT with MCR in Section 6.5. While above schemes are designed mainly for existing commodity DRAM, RT targets at the restoring issues in deep sub-micron DRAMs, which strive to keep high yield. Moreover, the RT schemes proposed in this paper do not need to modify the memory internal structures.

To achieve high yield and reliability, timing constraint values are set with excessive margins, which reflect the worst-case. Most DRAM chips can perform with smaller timing constraint values. Chandrasekar *et al.* [9] proposed to identify the excess in process-margins for DRAM devices at runtime. AL-DRAM [34] analyzes the timing reduction opportunities and exploits the large margin of DRAM timing parameters to improve performance. NUAT [46] exploits the electric charge variation caused by leakage to design a non-uniform access time memory controller. RT is orthogonal to AL-DRAM and NUAT, as discussed in Section 3.4. The proposed RT schemes do not depend on the excessive margins of timing parameters and thus have good compatibility.

DRAM scaling. DRAM scaling becomes a major challenge due to increased manufacturing complexity/cost, reduced cell reliability, and potentially increased cell leakage [41]. DRAM scaling leads to issues on refresh, write recovery time (tWR) [26] and variable retention time (VRT). The VRT phenomenon was studied in [29, 44]. Restore time degradation was recently studied in [26, 56]. Comparing to ChunkRemap design [56], restore truncation is an orthogonal design that has lower implementation complexity and storage overhead, while achieving much better performance improvement over the baseline.

9. CONCLUSION

In this paper, we studied the restoring issues in further scaling DRAM, identified partial restore opportunity and proposed two restore truncation (RT) schemes to exploit the opportunities with different tradeoffs. Our experimental results showed that, on average, RT improves performance by 19.5% and reduces energy consumption by 17%.

10. ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments and suggestions. This research is supported by National Science Foundation grants CCF-1422331, CCF-1535755, CNS-1305220, and CNS-1012070.

11. REFERENCES

[1] (2012) memory scheduling championship (msc). http://www.cs.utah.edu/ rajeev/jwac12/.

[2] 20nm 8Gb DDR4 DRAM. http://www.samsung.com/global/business/semiconductor/minisite/Greenmemory/news-event/in-the-news/detail?contentsSeq=13782&contsClassCd=G.

[3] CACTI 5.3: An integrated cache timing, power and area model. http://www.hpl.hp.com/research/cacti/.

[4] DDR3 SDRAM. Micron Technology.

[5] 4Gb DDR3 SDRAM - MT41J512M8. Micron Technology, 2009.

[6] A. Agrawal, A. Ansari, and J. Torrellas. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip edram modules. In *HPCA*, 2014.

[7] I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, and B. Jacob. DRAM refresh mechanisms, penalties, and trade-offs. *TC*, 64, 2015.

[8] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *ISCA*, 2015.

[9] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens. Exploiting expendable process-margins in DRAMs for run-time performance optimization. In *DATE*, 2014.

[10] K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. Improving DRAM performance by parallelizing refreshes with accesses. In *HPCA*, 2014.

[11] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, M. Awathi, and Z. Chishti. USIMM: the utah simulated memory module. Technical report, Univ of Utah, 2012.

[12] X. Chen and L.-S. Peh. Leakage power modeling and optimization in interconnection networks. In *ISLPED*, 2003.

[13] B. R. Childers, J. Yang, and Y. Zhang. Achieving yield, density and performance effective DRAM at extreme technology sizes. In *MEMSYS*, pages 78–84, 2015.

[14] J. Choi, W. Shin, J. Jang, J. Suh, Y. Kwon, Y. Moon, and L.-S. Kim. Multiple close row DRAM: A low latency and area optimized DRAM. In *ISCA*, 2015.

[15] Y. Kim D. Lee, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*, 2013.

[16] P. Demone. High speed DRAM architecture with uniform access latency. U.S. Patent.

[17] M. Ghosh and H.-S. S. Lee. Smart refresh: An enahnced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *MICRO*, 2010.

[18] S. Hong, S. Kim, J.-K. Wee, and S. Lee. Low-voltage DRAM sensing scheme with offset-cancellation sense amplifier. *JSSC*, 37(10), 2002.

[19] C. C. Hu. *Modern semiconductor devices for integrated circuits*. Prentic Hall, 2009.

[20] H. Iwai. Roadmap for 22nm and beyond. *Microelectronic Engineering*, 86(7-9):1520–1528, 2009.

[21] B. Jacob, S. Ng, and D. T. Wang. *Memory systems: Cache, DRAM, disk*. Morgan Kaufmann, 2007.

[22] JEDEC. DDR3 SDRAM Specification, 2010.

[23] JEDEC. DDR4 SDRAM Specification, 2012.

[24] H. Jeon, Y.-B. Kim, and M. Choi. Standby leakage power reduction technique for nanoscale CMOS VLSI systems. *TIM*, 59(5), 2010.

[25] B. Jiang, C. Sudhama, R. Khamankar, J. Kim, and J. C. Lee. Effects of nonlinear storage capacitor on DRAM read/write. *EDL*, 15(4), 1994.

[26] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi. Co-architecting controllers and DRAM to enhance DRAM process scaling. In *The memory forum*, 2014.

[27] D. Kaseridis, J. Stuecheli, and L. K. John. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. In *MICRO*, 2011.

[28] B. Keeth, R. J. Baker, B. Johnson, and F. Lin. *DRAM circuit design: fundamental and high-speed topics*. Wiley-IEEE Press, 2007.

[29] 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 *SIGMETRICS*, 2014.

[30] K. Kim and J. Lee. A new investigation of data retention time in truly nanoscaled DRAMs. In *EDL*, 2009.

[31] 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 disburbance errors. In *ISCA*, 2014.

[32] W. Kong, P. C. Parries, G. Wang, and S. S. Iyer. Analysis of retention time distribution of embedded DRAM - a new method to characterize across-chip threshold voltage variation. In *ITC*, 2008.

[33] S. K. Kurinec and K. Iniewski. *Nanoscale semiconductor memories: Technology and applications (devices, circuits, and systems)*. CRC Press, 2013.

[34] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timings for the common-case. In *HPCA*, 2015.

[35] 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 *ISCA*, 2013.

[36] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. Raidr: Retention-aware intelligent DRAM refresh. In *ISCA*, 2012.

[37] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level inerference in multicore systems. In *PACT*, 2012.

[38] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM Journal of Research and Development*, 46(2/3):181–212, 2002.

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

[40] J. Mukundan, H. Hunter, K.-H. Kim, J. Stuecheli, and J. F. Martinez. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. In *ISCA*, 2013.

[41] O. Mutlu. Memory scaling: A systems architecture perspective. In *IMW*, 2013.

[42] P. Nair, C.-C. Chou, and M. K. Qureshi. A case for refresh pausing in DRAM memory systems. In *HPCA*, 2013.

[43] P. J. Nair, D.-H. Kim, and M. K. Qureshi. Archshield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In *ISCA*, 2013.

[44] 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 *DSN*, 2015.

[45] J. F. Ryan and B. H. Calhoun. Minimizing offset for latching voltage-mode sense amplifiers for sub-threshold operation. In *ISQED*, 2008.

[46] W. Shin, J. Yang, J. Choi, and L.-S. Kim. NUAT: A non-uniform access time memory controller. In *HPCA*, 2014.

[47] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Can. Exploiting sub-20nm FinFET design with predictive technology models. In *DAC*, 2012.

[48] J. Son, U. Kang, C. Park, and S. Seo. Memory device with relaxed timing parameter according to temperature, operating method thereof, and memory controller and memory system using the memory device. United States Patent, 12 2014.

[49] Y. H. Son, S. O, Y. Ro, J. W. Lee, and J. H. Ahn. Reducing memory access latency with asymmetric DRAM bank organizations. In *ISCA*, 2013.

[50] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John. Elastic refresh: Techniques to mitigate refresh penalties in high density memory. In *MICRO*, 2010.

[51] J. P. Uyemura. *CMOS logic circuit design*. Kluwer Academic Publisher, 2002.

[52] T. Vogeslang. Understanding the energy consumption of dynamic random access memories. In *MICRO*, 2010.

[53] D. Wang. Backward compatible dynamic random access memory device and method of testing therefor. United States Patent, 9 2015.

[54] J. Wang, X. Dong, and Y. Xie. Proactive DRAM: A DRAM-initiated retention management scheme. In *ICCD*, 2014.

[55] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie. Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. In *ISCA*, 2014.

[56] X. Zhang, Y. Zhang, B. R. Childers, and J. Yang. Exploiting DRAM restore time variations in deep sub-micron scaling. In *DATE*, 2015.