VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency

Anup Das
Drexel University
Philadelphia, PA, USA
anup.das@drexel.edu

Hasan Hassan
ETH Zürich
Zürich, Switzerland
hhasan@ethz.ch

Onur Mutlu
ETH Zürich
Zürich, Switzerland
omutlu@gmail.com

ABSTRACT

A DRAM chip requires periodic refresh operations to prevent data loss due to charge leakage in DRAM cells. Refresh operations incur significant performance overhead as a DRAM bank/rank becomes unavailable to service access requests while being refreshed. In this work, our goal is to reduce the performance overhead of DRAM refresh by reducing the *latency* of a refresh operation. We observe that a significant number of DRAM cells can retain their data for longer than the worst-case refresh period of 64*ms*. Such cells do not always need to be *fully* refreshed; a low-latency *partial* refresh is sufficient for them.

We propose Variable Refresh Latency DRAM (VRL-DRAM), a mechanism that fully refreshes a DRAM cell *only when necessary*, and otherwise ensures data integrity by issuing low-latency partial refresh operations. We develop a new detailed analytical model to estimate the minimum latency of a refresh operation that ensures data integrity of a cell with a given retention time profile. We evaluate VRL-DRAM with memory traces from real workloads, and show that it reduces the average refresh performance overhead by 34% compared to the state-of-the-art approach.

CCS CONCEPTS

• **Hardware → Dynamic memory**; *Modeling and parameter extraction*; Hardware-software codesign;

KEYWORDS

DRAM, refresh cycle time (tRFC), DRAM circuit modeling

ACM Reference Format:
Anup Das, Hasan Hassan, and Onur Mutlu. 2018. VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency. In *DAC '18: The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196136

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196136

1 INTRODUCTION

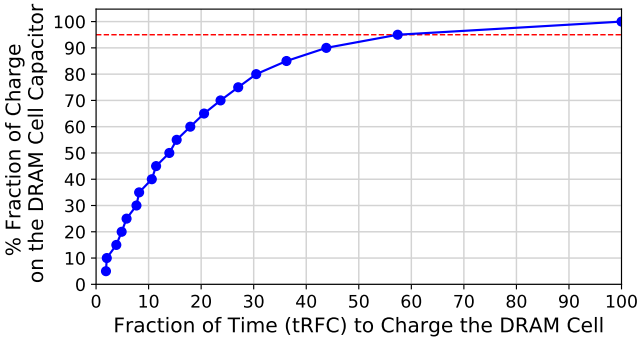
Dynamic Random Access Memory (DRAM) is the prevalent main memory technology in modern computing platforms. A DRAM cell, which stores a single bit of data, consists of a capacitor and an access transistor. Unfortunately, charge leaks from the capacitor over time. To ensure that a DRAM cell does not lose its data, the cell needs to be periodically refreshed, typically with a *refresh period* of 64*ms* [11, 27]). To refresh all cells in a DRAM chip, the memory controller issues a refresh command once every $7.8\mu s$,[1] which is known as the *refresh interval*, *tREFI*. Each refresh operation completes within a time interval *tRFC*, known as the *refresh cycle time*. A DRAM bank is *unavailable* to service any access requests during the *tRFC* portion of each *tREFI* interval. Hence, the refresh performance overhead (i.e., data throughput loss due to refresh) of a bank is $\frac{tRFC}{tREFI}$. The refresh overhead is significant in current DRAM devices, and it is expected to become even more critical in the future as DRAM chip capacity increases [1, 4, 13, 17, 25, 27–33].

Although a DRAM chip is typically refreshed every 64*ms*, most of the DRAM cells can retain their charge for a much longer time [9, 17, 27, 28, 32, 33]. If we know the retention time profile of DRAM cells, we can reduce the refresh overhead by increasing *tREFI* [7, 22, 27, 32, 33, 36] for those cells that can retain data longer than *tREFI*. Liu et al. propose RAIDR [27] to do exactly so. RAIDR relies on retention time profiling [32] data of a DRAM chip to classify the DRAM rows based on the retention time of the weakest cell in each row. Instead of refreshing each row at the fixed 64*ms* interval, rows that can retain their data for a longer time are refreshed less frequently.

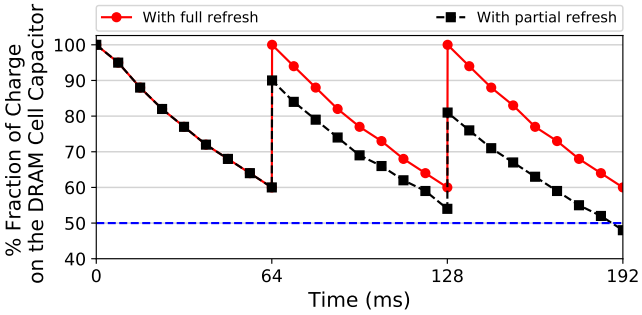
In this work, we show that the refresh overhead can be further reduced (beyond RAIDR) by exploiting DRAM circuit characteristics to reduce *tRFC*, i.e., the latency of each refresh operation. To this end, we make the following two key observations.

Observation 1. During a refresh operation, more than half of the refresh cycle time is spent injecting the last 5% of the charge of a fully-charged DRAM cell. We validate this observation using detailed SPICE simulations. In Figure 1a, we plot the fraction of *tRFC* required to restore different fractions of charge of a fully-charged DRAM cell. We see that approximately 60% of *tRFC* is spent charging the cell to 95% of its capacity. During the remaining 40% of the time, only the last 5% of charge is injected to the capacitor. These simulation results are consistent with those reported in prior work [5, 14, 23–25, 32]. Our motivation is that, if a refresh operation is truncated at 95% of a cell's charge capacity (i.e., if we perform a partial refresh), the refresh latency (*tRFC*) can theoretically be reduced by up to 40%. *tRFC* is influenced by several factors such as

[1] During a 64*ms* refresh period, the memory controller issues a total of 8192 refresh commands to a DRAM chip.



(a) The charge restoration status of a DRAM cell during a refresh operation.



(b) Refreshing a DRAM cell with full and partial refresh operations.

Figure 1: Two key observations on DRAM refresh.

data pattern dependence [15, 16, 28], sneak paths leaking charge over time [27, 28, 32, 33], and bitline/wordline parasitic capacitance coupling effects [15, 26]. To estimate the practically possible *tRFC* reduction for partial refresh operations, we can perform detailed circuit-level SPICE simulations. However, such simulations are very time consuming. As a faster alternative, we develop a new analytical model for refresh operations based on a rigorous analysis of the DRAM circuit characteristics. Our model can accurately estimate *tRFC* required to restore a DRAM cell to any given fraction of the cell's full charge capacity.

Observation 2. We observe that a DRAM cell that has a retention time higher than the refresh period, once fully restored, can sustain *partial refreshes* without sacrificing data integrity. To validate this, in Figure 1b, we analyze the charge leakage and restoration behavior of an example DRAM cell that is refreshed every 64*ms*. We plot two cases where the cell is refreshed using (1) a large *tRFC* value (i.e., full refresh) and (2) a small *tRFC* value (i.e., partial refresh). In both cases, the cell is initially charged to its full capacity. When we use full refresh, each refresh operation restores the cell to its full capacity. Thus, the cell correctly retains its data value with full refresh, but each refresh operation incurs high latency. In contrast, when we use partial refresh, we see that the cell can still retain its data value when a full refresh is followed by a partial refresh, yet the partial refresh has low latency. This is because the DRAM cell in the example has a retention time higher than the refresh period of 64*ms*. However, the same cell *cannot* sustain two back-to-back partial refreshes as the charge level drops below the 50% threshold, and the cell loses its data value. Hence, to operate correctly, the cell needs a full refresh during the next refresh period after it is partially refreshed. Due to the large amount of heterogeneity (variance) in the retention time distribution [12, 17, 24, 25, 27, 28, 32, 33] of real DRAM chips, different cells can sustain different numbers of partial refresh operations[2] between two consecutive full refreshes.

Based on our two observations, we propose Variable Refresh Latency DRAM (VRL-DRAM), a mechanism that (1) accurately estimates the number of partial refreshes that a DRAM cell can reliably sustain, and (2) whenever possible, issues partial refreshes to reduce the refresh overhead. To this end, we develop a new circuit-level analytical model. Using our analytical model and the retention time profile data of a DRAM chip, our mechanism determines the number of partial refreshes for each DRAM row. Our evaluations show that VRL-DRAM (1) reduces the average refresh performance overhead for real workloads by 34% on top of RAIDR [27] and (2) requires *less than* 2% of the area of a DRAM bank at the 90*nm* technology node.

We make the following major contributions:

- We develop a new detailed circuit-level analytical model for DRAM refresh that takes into account factors such as data pattern dependence, sneak paths, and bitline/wordline parasitic coupling (Sec. 2) to estimate the amount of refresh latency a DRAM cell requires. Our model is available as an open-source tool [38] to engender future research.
- We propose Variable Refresh Latency DRAM (VRL-DRAM), a new mechanism that enables partial refresh operations to reduce DRAM refresh performance overhead (Sec. 3).

2 ANALYTICAL MODELING

In this section, we describe our analytical model for DRAM refresh and charge leakage in DRAM cells. For detailed explanations of DRAM architecture and operation, we refer the reader to prior work [5, 6, 8, 9, 16–18, 20, 21, 23–25, 27, 32, 34, 35].

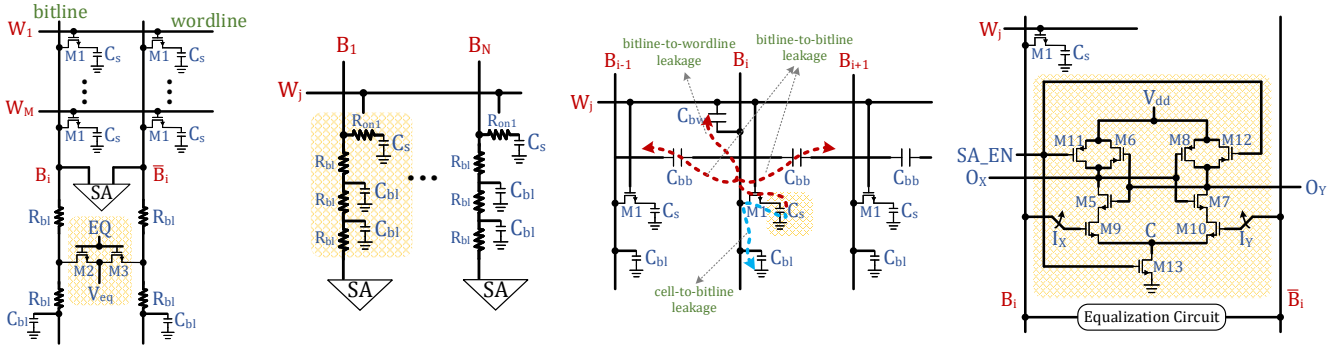
The refresh cycle time *tRFC* is composed of three main phases: (1) *equalization delay*, i.e., the time required to deassert the wordline of the currently-open row and equalize the voltages of the bitlines, (2) *pre-sensing delay*, i.e., the time required to activate the row to refresh and share the charge of the cells with the bitlines, and (3) *post-sensing delay*, i.e., the time required to sense the voltage difference on a bitline and its complement and restore the charge of the DRAM cell accordingly.

2.1 Equalization Delay

DRAM typically implements a differential voltage sense amplifier [10] that detects the voltage difference between a bitline pair, i.e., a bitline and its complement, connected to different terminals of a sense amplifier. Before activating a DRAM row, the bitline pair needs to be prepared for activation by equalizing the voltage of both bitlines to $V_{eq} = V_{dd}/2$. Figure 2a illustrates a DRAM cell array with an equalization circuit [39]. Prior to enabling the equalization circuit, since a row in the DRAM cell array is activated, the voltage of one of the bitlines is V_{dd} and the other V_{ss}. The equalization logic is connected to the bitline pair (i.e., $\mathtt{B}_i$ and $\bar{\mathtt{B}}_i$) via two NMOS transistors, M_2 and M_3. When the EQ signal is asserted, the equalization circuit drives V_{eq} into the bitlines. We explain the two phases of the equalization process in detail.

Phase 1: At time $t = 0^+$ (i.e., shortly after EQ is asserted), M_2 and M_3 enter saturation mode, where the saturation current is $I_{d_{sat_2}}$. The bitline capacitor voltage discharges until it drops (or increases

[2]Partial refresh is different from *partial restore* [41], where charge in a DRAM cell is partially replenished after a *read/write* access to the cell.



(a) DRAM equalization circuit. **(b) Bitlines after wordline activation.** **(c) Sneak paths and parasitic capacitance coupling.** **(d) Latch-based voltage sense amplifier [40].**

Figure 2: Circuits used for our analytical model.

for the complementary bitline) by V_{tn_2}. The time required for *Phase 1* is calculated by

$$t_o = \frac{C_{bl}\Delta V_{bl_i}(t_o)}{I_{d_{sat_2}}} = \frac{C_{bl}V_{tn_2}}{\frac{\beta_{n_2}}{2}(V_g - V_{eq} - V_{tn_2})^2} \tag{1}$$

where $\beta_{n_2} = \mu_{n_2}C_{ox_{n_2}}\frac{W_{n_2}}{L_{n_2}}$ is the NMOS process parameter calculated using electron mobility μ_{n_2}, gate oxide capacitance $C_{ox_{n_2}}$, channel width W_{n_2} and channel length L_{n_2}.

Phase 2: At time $t = t_o^+$, the M_2 and M_3 transistors enter into the linear region with ON resistance r_{on_2}. In this phase, given enough time, the bitline capacitor voltage eventually reaches V_{eq}. For *Phase 2*, the bitline voltage can be formulated as a function of time τ_{eq} as

$$V_{bl_i}(\tau_{eq}) = V_{eq} + \left(V_{bl_i}(t_o) - V_{eq}\right)e^{\frac{-(\tau_{eq}-t_o)}{R_{eq}C_{bl}}}, \text{ where}$$
$$R_{eq} = R_{bl} + r_{on_2} = R_{bl} + \frac{1}{\beta_{n_2}(V_g - V_{eq} - V_{tn_2})} \tag{2}$$

2.2 Pre-sensing Delay

In Figure 2b, we show the state of two DRAM cells after wordline activation, where the access transistor of each activated cell is turned on. During this phase, an activated cell shares its charge with the bitline that should be ideally precharged to $\frac{V_{dd}}{2}$.

Let $V_s(\tau_{eq}^+)$ be the voltage on a DRAM cell after equalization. Applying the charge conservation principle, the bitline voltage during charge-sharing at time t is

$$\Delta V_{bl_i}(t) = \frac{C_s}{C_s + C_{bl}}\left|V_{s_{i,j}}(\tau_{eq}) - V_{bl_i}(\tau_{eq})\right|\left[1 - U(t)\right]$$
$$\text{where } U(t) = \frac{C_s \cdot e^{\frac{-(t-\tau_{eq})}{R_{pre}C_{bl}}} + C_{bl} \cdot e^{\frac{-(t-\tau_{eq})}{R_{pre}C_s}}}{C_s + C_{bl}} \tag{3}$$

where i and j are used to denote the DRAM cell at the intersection of the i^{th} bitline and the j^{th} wordline, and $R_{pre} = r_{on_1} + R_{bl}$, where r_{on_1} is the ON resistance of the access transistor. As time $t \to \infty$, $U(t) \to 0$, and thus

$$\Delta V_{bl_i}(t) \to \frac{C_s}{C_s + C_{bl}}\left|V_{s_{i,j}}(\tau_{eq}) - V_{bl_i}(\tau_{eq})\right| = V_{sense_i} \tag{4}$$

where V_{sense_i} is the maximum possible change in voltage on bitline B_i. Equation 3 can be rewritten for time $t = \tau_{pre}$ as

$$\Delta V_{bl_i}(\tau_{pre}) = V_{sense_i}\left[1 - U(\tau_{pre})\right] \tag{5}$$

In the presence of leakage paths and parasitic components, the maximum voltage change V_{sense_i} would be lower. Figure 2c shows the charge leakage paths in the presence of bitline to bitline (C_{bb}) and bitline to wordline parasitic (C_{bw}) capacitance coupling. Applying the charge conservation principle, we find:

$$dQ_s = dQ_1 + dQ_2 + dQ_3 + dQ_4 \tag{6}$$

where dQ_s is the change in the DRAM cell charge. dQ_1, dQ_2, dQ_3 and dQ_4 are the changes in the i^{th} bitline capacitor charge, i^{th} to $(i-1)^{\text{th}}$ bitline-to-bitline parasitic, i^{th} to $(i+1)^{\text{th}}$ bitline-to-bitline parasitic and bitline-to-wordline parasitic, respectively, shown by the arrows in Figure 2c. These delta changes are

$$dQ_s = C_s\left|V_{s_{i,j}}(\tau_{eq}) - V_{s_{i,j}}(t)\right|$$
$$dQ_1 = C_{bl}V_{sense_i} \text{ and } dQ_2 = C_{bb}\left[V_{sense_i} - V_{sense_{i-1}}\right]$$
$$dQ_3 = C_{bb}\left[V_{sense_i} - V_{sense_{i+1}}\right] \text{ and } dQ_4 = C_{bw}V_{sense_i}$$

We solve Equation 6 using the voltage equality for the bitline B_i and the DRAM cell i.e., $V_{s_{i,j}}(t) = V_{bl_i}(t)$ as

$$V_{sense_i} = K_1\mathcal{L}_{self_{i,j}} + K_2V_{sense_{i-1}} + K_2V_{sense_{i+1}} \tag{7}$$
$$\text{where } \mathcal{L}_{self_{i,j}} = \left|V_{s_{i,j}}(\tau_{eq}) - V_{bl_i}(\tau_{eq})\right|$$
$$K_1 = \frac{C_s}{C_s + C_{bl} + 2C_{bb} + C_{bw}} \text{ and } K_2 = \frac{C_{bb}}{C_s + C_{bl} + 2C_{bb} + C_{bw}}$$

As seen from Equation 7, the maximum voltage change on a bitline depends not only on the voltage difference between a DRAM cell and the bitline, but also on the maximum voltage change on the neighboring bitlines. This cyclic dependency is *not* considered in existing modeling techniques [26]. *Our contribution is a closed-form solution of Equation 7 for N bitlines of the j^{th} wordline, as follows:*

$$\mathbf{V_{sense}} = K_1\mathbf{K}^{-1} \cdot \mathcal{L}_{\mathbf{self}}, \text{ where}$$
$$\mathbf{K} = \begin{bmatrix} k_{0,0} & k_{0,1} & \cdots & k_{0,N-1} \\ k_{1,0} & k_{1,1} & \cdots & k_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{N-1,0} & k_{N-1,1} & \cdots & k_{N-1,N-1} \end{bmatrix}, k_{x,y} = \begin{cases} 1 & \text{for } x = y \\ -K_2 & \text{for } |x-y| = 1 \\ 0 & \text{otherwise} \end{cases}$$
$$\mathbf{V_{sense}} = \begin{bmatrix} V_{sense_0} & V_{sense_1} & \cdots & V_{sense_{N-1}} \end{bmatrix}^T$$
$$\mathcal{L}_{\mathbf{self}} = \begin{bmatrix} \mathcal{L}_{self_{0,j}} & \mathcal{L}_{self_{1,j}} & \cdots & \mathcal{L}_{self_{N-1,j}} \end{bmatrix}^T \tag{8}$$

2.3 Post-sensing Delay

Figure 2d shows the circuit of a latch-based voltage sense amplifier [40]. The circuit amplifies the voltage difference on I_x and I_y, and drives the output terminals O_x and O_y to V_{dd} and V_{ss}, respectively, or vice versa. There are four sub-phases of the post-sensing delay.

Phase 1: Right after the sense amplifier is enabled, the PMOS transistors M_6, M_8, M_{11}, and M_{12} are in cut-off mode, whereas the NMOS transistors M_5, M_7, M_9, and M_{10} are in saturation mode, and the tail NMOS transistor M_{13} is in the linear region.

During *Phase 1*, the saturation current of M_5 and M_7 gradually discharges the voltage of the output nodes O_x and O_y (pre-charged to V_{dd}), respectively. The rate of discharge depends on the respective saturation currents $I_{d_{sat_9}}$ and $I_{d_{sat_{10}}}$. Thus, a voltage difference starts building up at the output terminals. This continues until the voltage at one of the output terminals drops to $V_{dd} - V_{tp}$, when the corresponding PMOS transistor M_{11} or M_{12} turns ON. Assuming $I_x > I_y$, the time delay is

$$t_1 = \frac{C_{bl} V_{tp}}{I_{d_{sat_{10}}}}, I_{d_{sat_{10}}} = \beta_n (V_{eq} - V_{thn})^2 \left(1 - \frac{0.75}{1 + \frac{V_{dd} - V_{thn}}{V_{eq} - V_{thn}}}\right)^2 \quad (9)$$

Phase 2: Once one of the PMOS transistors (e.g., M_6 or M_8) turns ON, the corresponding output voltage starts increasing, which increases the voltage difference due to the positive feedback in the circuit. The delay of this process is

$$t_2 = \frac{C_{bl}}{gm_e} \ln\left(\frac{1}{V_{tp}} \sqrt[2]{\frac{I_{d_{sat_{10}}}}{\beta_n}} \frac{V_{dd} - V_{tp} - V_{eq}}{\Delta V_{bl_i}(\tau_{pre})}\right) \quad (10)$$

where gm_e is the effective transconductance of the cross-coupled inverter.

Phase 3: In this phase, the voltage of the output terminals is driven to V_{dd} or V_{ss}. Let $V_{residue}$ be the marginal voltage difference between the output terminals. The delay in this process is

$$t_3 \approx R_{post} C_{bl} \ln \frac{V_{eq}}{V_{residue}} \text{ where } R_{post} = R_{bl} + r_{on} \quad (11)$$

Phase 4: In this phase, the DRAM cell capacitor is driven to the desired logic value using the voltage developed on the bitlines. Assuming the memory controller allocates τ_{post} time for the complete post-sensing process, the voltage restored to the cell is

$$V_{s_{i,j}}(\tau_{post}) = V_{s_{i,j}}(\tau_{pre}) + V_a \left(1 - e^{\frac{-(\tau_{post} - t_1 - t_2 - t_3)}{R_{post} C_{post}}}\right) \text{ or}$$

$$V_{s_{i,j}}(\tau_{post}) = V_{residue} + V_b \left(e^{\frac{-(\tau_{post} - t_1 - t_2 - t_3)}{R_{post} C_{post}}}\right), \quad (12)$$

$$V_a = V_{dd} - V_{s_{i,j}}(\tau_{pre}) - \tau_{pre} \text{ and } V_b = V_{s_{i,j}}(\tau_{pre}) - V_{residue}$$

where $C_{post} = C_s + C_{bl} + 2C_{bb} + C_w$.

Finally, we calculate the refresh cycle time, *tRFC*, as

$$tRFC = \tau_{eq} + \tau_{pre} + \tau_{post} + \tau_{fixed} \quad (13)$$

where τ_{fixed} is the aggregate of the other fixed delays (e.g., time to assert and deassert a wordline).

The analytical model presented in this section allows us to determine *tRFC* for full and partial refresh operations. The analytical model is available online as an open-source tool [38] to engender future research.

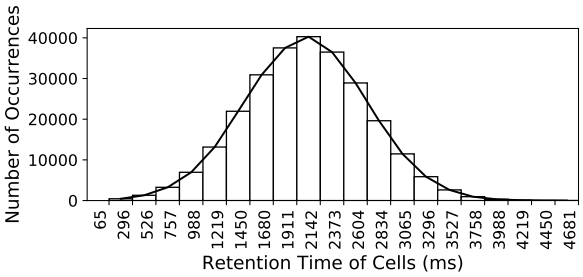
3 VARIABLE REFRESH LATENCY DRAM

Our Variable Refresh Latency DRAM (VRL-DRAM) mechanism reduces the refresh overhead by using low-latency partial refresh operations whenever possible. We define the number of partial refreshes that a DRAM cell can reliably sustain as *mean partial refreshes to sensing failure (MPRSF)*. The MPRSF of a row is the minimum of the MPRSFs of all cells in the row. VRL-DRAM determines the MPRSF of a DRAM row using (1) the analytical model we develop in Section 2 and (2) DRAM retention time profiling data which we assume is available, e.g., using methods in previous works [16, 27, 32, 33]. VRL-DRAM adapts the refresh command scheduling policy of the memory controller to *selectively* issue partial and full refresh commands to reduce the refresh overhead while ensuring data integrity.

3.1 Determining the Reduced Refresh Latency and MPRSF

When using partial refresh, there is a trade-off between the latency reduction in a partial refresh operation (i.e., the amount of charge restored on refresh) and the number of partial refresh operations a row can sustain, i.e., MPRSF. We define the latency of partial and full refresh operations as $\tau_{partial}$ and τ_{full}, respectively.

Our goal is to find a $\tau_{partial}$ that maximizes the refresh overhead reduction, given the retention time distribution data of the DRAM chip, which can be collected using a profiler such as [32, 33]. For our evaluations, we assume a typical DRAM retention time distribution [27], as we show in Figure 3a. In Figure 3b, we show how we bin the DRAM rows into four refresh periods based on their retention times. Based on those refresh periods, we determine the best $\tau_{partial}$ and MPRSF for each row.



(a) DRAM retention time distribution from Liu et al. [27].

Refresh period (ms)	Number of rows in a bank
64	68
128	101
192	145
256	7878

(b) Refresh rates after binning of rows in a DRAM bank.

Figure 3: Retention time binning of DRAM cells.

If we use a large value for $\tau_{partial}$ (e.g., close to τ_{full}), DRAM cells of a row would be refreshed almost to their full capacity after every partial refresh operation. Hence, there would only be negligible reduction of refresh overhead from using partial refreshes. On the other hand, if we use a small value for $\tau_{partial}$, DRAM cells would be refreshed to a state with much lower charge compared to their full charge capacity. In such a case, a DRAM row would be less likely to retain correct data until the next refresh operation. In the worst case, the DRAM row would have 0 MPRSF, which would prevent using partial refresh for the row. Thus, VRL-DRAM would not provide any benefit. Therefore, we need to *intelligently* choose a value for $\tau_{partial}$.

We perform thorough simulations to select the best value for $\tau_{partial}$, which results in maximum refresh overhead reduction. Our simulation framework uses four data patterns (i.e., all 0's, all

1's, alternate 0's/1's and random) [17, 28] to determine $\tau_{partial}$ by taking into account data pattern dependence of DRAM cells. Via our simulations, we find the following breakdown for $\tau_{partial}$ and τ_{full}, in DRAM cycles:

$\tau_{partial} = tRFC|_{\tau_{eq}=1, \tau_{pre}=2, \tau_{post}=4, \tau_{fixed}=4}$ (partial refresh) = 11 cycles

$\tau_{full} = tRFC|_{(\tau_{eq}=1, \tau_{pre}=2, \tau_{post}=12, \tau_{fixed}=4}$ (full refresh) = 19 cycles

3.2 Implementation

VRL-DRAM schedules a full or partial refresh operation based on the MPRSF of the row to be refreshed. Let m_i be the MPRSF of the i^{th} row of the DRAM bank. VRL-DRAM issues a full refresh at every m_i^{th} refresh period of the row. For all other refreshes of the row, VRL-DRAM issues a partial refresh with $\tau_{partial}$ latency. We propose two variants of our mechanism: VRL and VRL-Access.

VRL-DRAM can be implemented entirely inside the memory controller. Algorithm 1 provides the pseudo-code of the logic that the memory controller implements to support VRL-DRAM. Two variables (`mprsf` and `rcount`) store the MPRSF and the number of refresh operations that have been issued to a DRAM row, respectively. In the actual hardware implementation, those two variables can be defined as `nbits`-wide counters.[3] We evaluate the performance of VRL-DRAM with `nbits = 2`, which leads to a very low-cost implementation. In Line 1, the memory controller gets a set of rows that need to be refreshed. Then, the memory controller compares the `rcount` and `mprsf` values of each row. If `rcount` is equal to `mprsf`, the memory controller issues a *full* refresh using τ_{full} latency (Line 4) to fully restore the charge of the cells in the refreshed row. If `rcount` is not equal to `mprsf`, the memory controller issues a *partial* refresh using $\tau_{partial}$ latency (Line 7) to reduce the refresh overhead.

Algorithm 1: VRL-DRAM Refresh Scheduling Policy.

```
   Input: τ_partial, τ_full, mprsf, rcount
   Output: mprsf, rcount
1  rows-to-refresh = Get rows to be refreshed;
2  foreach r ∈ rows-to-refresh do
3      if rcount[r] == mprsf[r] then
4          Set tRFC = τ_full, rcount[r] = 0 and refresh row r;
5      end
6      else
7          Set tRFC = τ_partial, rcount[r] += 1 and refresh row r;
8      end
9  end
```

The second variant of VRL-DRAM optimizes the baseline scheme in Algorithm 1 by exploiting the memory access patterns of the workloads running on the system. In this optimized version, called *VRL-Access*, we take advantage of the fact that a DRAM activation caused by a read or write access fully restores the charge in the DRAM row. Thus, the memory controller can use $\tau_{partial}$ for the subsequent refresh operations to the same row as dictated by MPRSF. Thus, on a read or write access to a row, the memory controller resets the value of *rcount* to 0.

[3] We evaluate the area overhead of VRL-DRAM for different `nbits` values in Section 4.3.

4 RESULTS

We evaluate VRL-DRAM using an in-house simulator and 90nm technology parameters [37]. Our framework can be extended with small effort to other technology nodes.

4.1 Evaluation with Memory Traces

We evaluate VRL-DRAM using benchmarks from the PARSEC-3.0 suite [2] and one server benchmark `bgsave` [19]. Memory traces for these benchmarks are generated using Ramulator [19]. These traces are input to our in-house simulator configured to simulate an 8192x32 (*rows* × *columns*) memory bank. We compare the refresh performance overhead (as measured in cycles spent refreshing the bank) of VRL-DRAM variants (VRL and VRL-Access) to that of RAIDR [27]. Figure 4 reports these comparisons. All results are normalized to the refresh overhead of RAIDR. As seen from the figure, refresh overheads of RAIDR and VRL are fixed for all benchmarks because these techniques are application independent. The refresh overhead of VRL is 23% lower than that of RAIDR. This reduction is due to using low-latency *tRFC* in VRL for partial refresh operations. VRL-Access further reduces the refresh overhead over VRL and RAIDR by 13% and 34%, respectively, on average across our applications. We also find that, VRL-DRAM reduces refresh power by 12% over RAIDR (evaluated using the DRAMPower tool [3]).

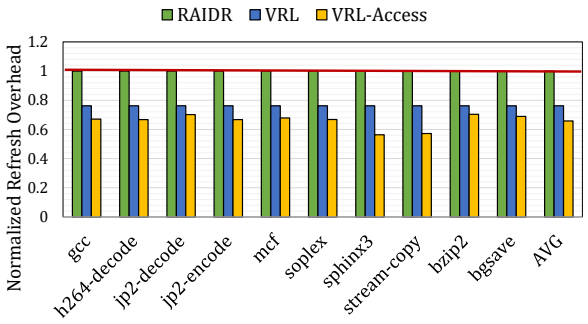


Figure 4: Refresh performance overhead with real traces.

4.2 Accuracy of our Analytical Model

Figure 5 plots voltage responses for bitline $\mathtt{B}_i$ and $\bar{\mathtt{B}}_i$ during voltage equalization, prior to activating a wordline. We report results for three approaches: (1) our two-phase analytical model (Equation 2), (2) the single-cell capacitor model of Li et al. [26], and (3) SPICE simulation of the equalization circuit in Figure 2a. As seen in Figure 5, results for all three approaches are similar for $\bar{\mathtt{B}}_i$ (plotted *below* the dashed line at 0.6V). However, for $\mathtt{B}_i$ (plotted *above* the dashed line), our analytical model is closer to SPICE results than the model used in [26]. Accuracy results are consistently similar for the charge sharing circuit of Figure 2b and the post-sensing charge replenishment circuit of Figure 2d.

Table 1 reports the pre-sensing time τ_{pre} (in memory cycles) needed to refresh a DRAM cell to 95% of its capacity. We report results for six memory configurations using three approaches: SPICE simulation, single-cell capacitor model [26], and our analytical model. The table also reports the corresponding simulation time. As seen in the table, τ_{pre} estimated using our analytical model is within 0%-12.5% of that obtained using SPICE simulation, demonstrating the high accuracy of our model. The accuracy of the single cell model is within 6.25%-62.5% of the SPICE simulation. In terms

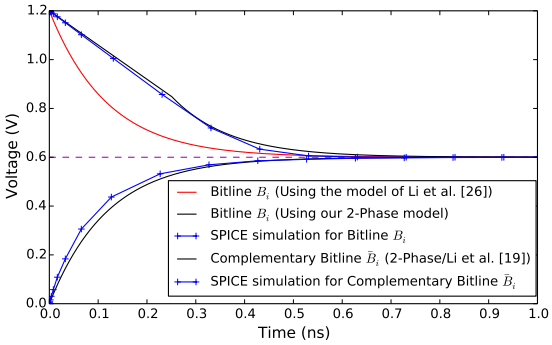


Figure 5: Voltage response during the equalization stage.

of simulation time, the single cell model is the fastest. Our analytical model is a few orders of magnitude faster than SPICE and its speed scales proportionally with DRAM size.

Table 1: Accuracy trade-offs of our analytical model.

Bank Size	Pre-sensing time (cycles)			Simulation time		
	SPICE Sim.	Single cell Model	Our Model	SPICE Sim.	Single cell Model	Our Model
2048x32	7	6	7	1h 36m	5ms	10s
2048x128	8	6	8	1h 57m	5ms	51s
8192x32	9	6	9	4h 23m	21ms	20s
8192x128	11	6	10	5h 17m	21ms	108s
16384x32	14	6	12	17h 12m	44ms	90s
16384x128	16	6	14	21h 1m	44ms	209s

4.3 Area Overhead of VRL-DRAM

Table 2 reports the area overhead of VRL-DRAM for a DRAM bank of size 8192x32, synthesized at the 90nm feature size [37]. We provide results for three different values of `nbits` (see Section 3.2). The area overhead of VRL-DRAM is within 1-2% of the area of a DRAM bank.

Table 2: Area overhead of VRL-DRAM at 90nm.

`nbits`	Logic area (μm^2)	% DRAM bank area (μm^2)
2	105	0.97%
3	152	1.4%
4	200	1.85%

5 CONCLUSIONS

We introduce VRL-DRAM, a mechanism to reduce the refresh latency overhead in modern DRAM chips. VRL-DRAM requires a long-latency full refresh of a DRAM row *only when necessary*. It otherwise schedules *short-latency* partial refreshes, reducing the refresh overhead. We develop a circuit-level analytical model to determine the number of partial refreshes that can be sustained by a DRAM cell without losing data integrity. Our model incorporates memory content in neighboring cells, sneak paths, and bitline/wordline parasitic capacitance coupling, and is freely available online [38]. Using memory traces from real workloads, we demonstrate that VRL-DRAM reduces the refresh performance overhead by 34% over a state-of-the-art approach, RAIDR [27]. We conclude that VRL-DRAM is an effective method for reducing DRAM refresh latency.

REFERENCES

[1] I. Bhati *et al.*, "DRAM Refresh Mechanisms, Penalties, and Trade-Offs," in *TC*, 2016.
[2] C. Bienia *et al.*, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008.
[3] K. Chandrasekar *et al.*, "Towards Variation-Aware System-Level Power Estimation of DRAMs: An Empirical Approach," in *DAC*, 2013.
[4] K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[5] K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[6] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[7] N. Gulur *et al.*, "MicroRefresh: Minimizing Refresh Overhead in DRAM Caches," in *MEMSYS*, 2016.
[8] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[9] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[10] S. Hong *et al.*, "Low-Voltage DRAM Sensing Scheme with Offset-Cancellation Sense Amplifier," in *JSSC*, 2002.
[11] JEDEC, "Double Data Rate (DDR) SDRAM Specification," in *JESD79E*, 2005.
[12] M. Jung *et al.*, "A Platform to Analyze DDR3 DRAM's Power and Retention Time," in *IEEE Design & Test*, 2017.
[13] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[14] B. Keeth, "DRAM Circuit Design: Fundamental and High-Speed Topics." John Wiley & Sons, 2008.
[15] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[16] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," in *MICRO*, 2017.
[17] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[18] J. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.
[19] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," in *CAL*, 2016.
[20] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[21] Y. Kim *et al.*, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[22] J. Kotra *et al.*, "Hardware-Software Co-design to Mitigate DRAM Refresh Overheads: A Case for Refresh-Aware Process Scheduling," in *ASPLOS*, 2017.
[23] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[24] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[25] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[26] Y. Li *et al.*, "DRAM Yield Analysis and Optimization by a Statistical Design Approach," in *TCAS-I*, 2011.
[27] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[28] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[29] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.
[30] O. Mutlu, "The RowHammer Problem and Other Issues we may Face as Memory Becomes Denser," in *DATE*, 2017.
[31] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," in *SUPERFRI*, 2014.
[32] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.
[33] M. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[34] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[35] V. Seshadri *et al.*, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[36] H. Shin *et al.*, "Timing Window Wiper: A New Scheme for Reducing Refresh Power of DRAM," in *ASP-DAC*, 2017.
[37] E. Sicard, "Introducing 90 nm Technology in Microwind3," in *Technology*, 2004.
[38] VRL-DRAM Analytical Model, https://github.com/anupkdas-nus/VRL-DRAM.
[39] L. Watters *et al.*, "Use of Voltage Equalization in Signal-Sensing Circuits," US Patent 5,841,718. 1998.
[40] B. Wicht *et al.*, "Yield and Speed Optimization of a Latch-Type Voltage Sense Amplifier," in *JSSC*, 2004.
[41] X. Zhang *et al.*, "Restore Truncation for Performance Improvement in Future DRAM Systems," in *HPCA*, 2016.