ETH zürich Carnegie Mellon University UNICAMP SAFARI

Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration

Yaohua Wang, Arash Tavakkol, Lois Orosa, Saugata Ghose,
Nika Mansouri Ghiasi, Minesh Patel, Jeremie S. Kim,
Hasan Hassan, Mohammad Sadrosadati, Onur Mutlu

October 22, 2018

Executive Summary

- DRAM access latency has **not decreased** significantly for decades
  - A DRAM cell stores data as charge, where the charge leaks over time
  - Requires **row activation** and **charge restoration: high-latency** operations
  - We can reduce the latency by only ***partially restoring*** a cell's charge level
- Key observations
  - There is **significant potential to reduce latency** if we partially restore DRAM cells that **will be *reactivated* soon**
  - If we reduce restoration too much, we cannot use mechanisms that reduce the activation latency: **need to strike a balance**
- **CAL: Charge-Level-Aware Look-Ahead Partial Restoration**
  - **Simple prediction** for rows that will be reactivated soon (98% accuracy)
  - **Maximizes the total DRAM access latency reduction** using both partial restoration and reduced activation latency
  - 8-core workloads: **14.7% speedup, 11.3% energy reduction** on average

Outline

Background: Accessing Data in DRAM

Key Observations on Partial Restoration

Charge-Level-Aware Look-Ahead Partial Restoration

Evaluation

Conclusion

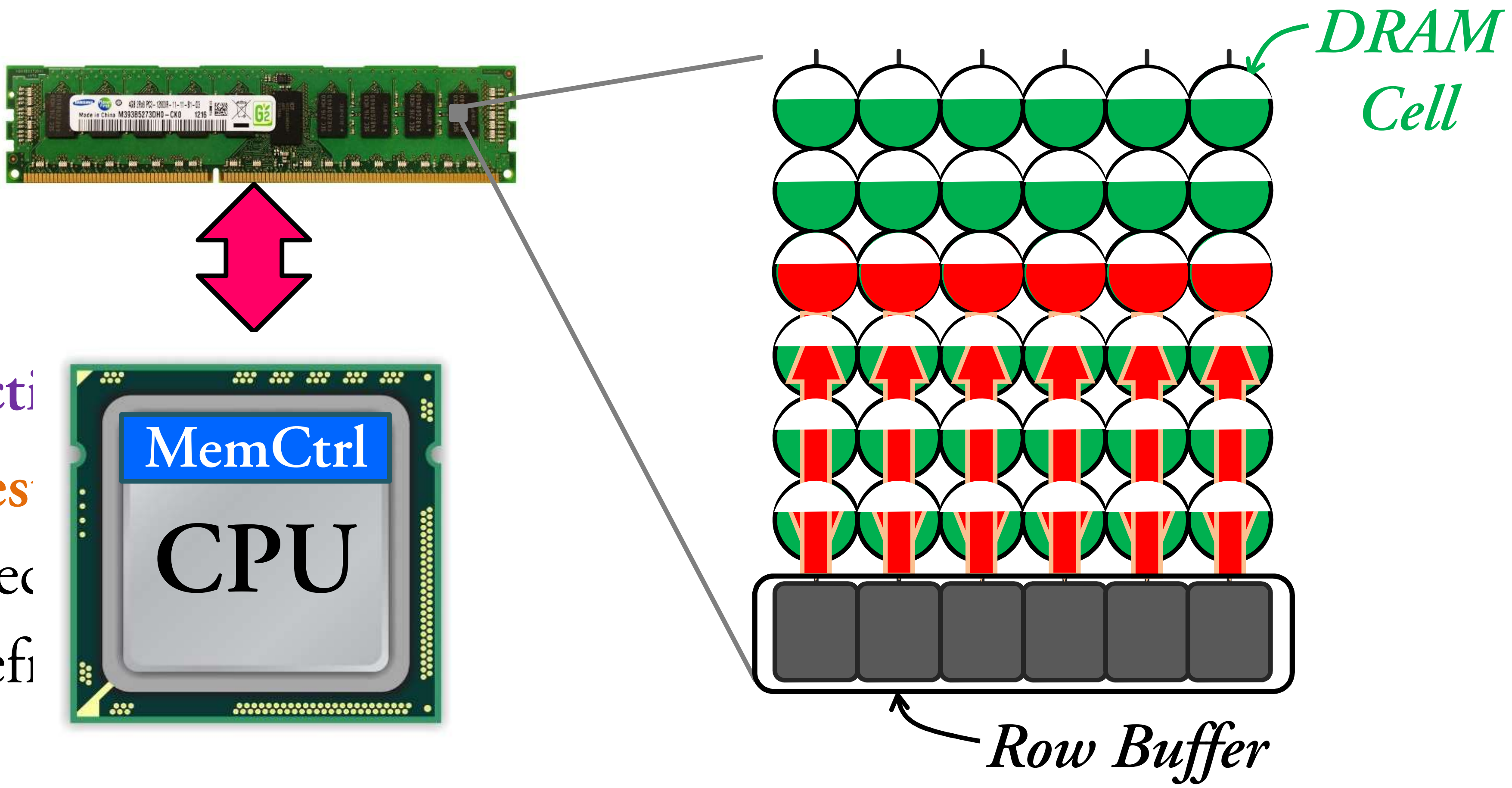
Fundamental DRAM Operations

Acti

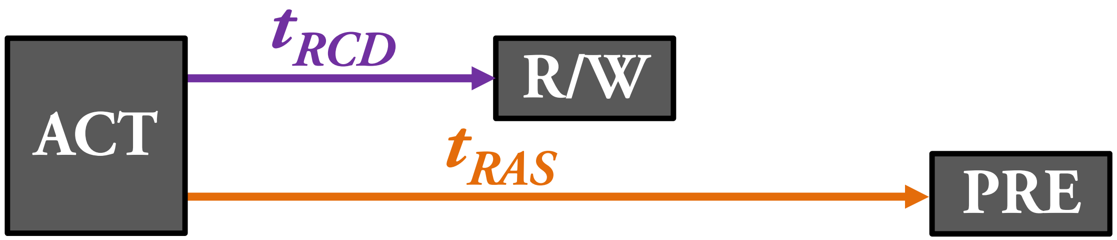
Res

Prec

Refi



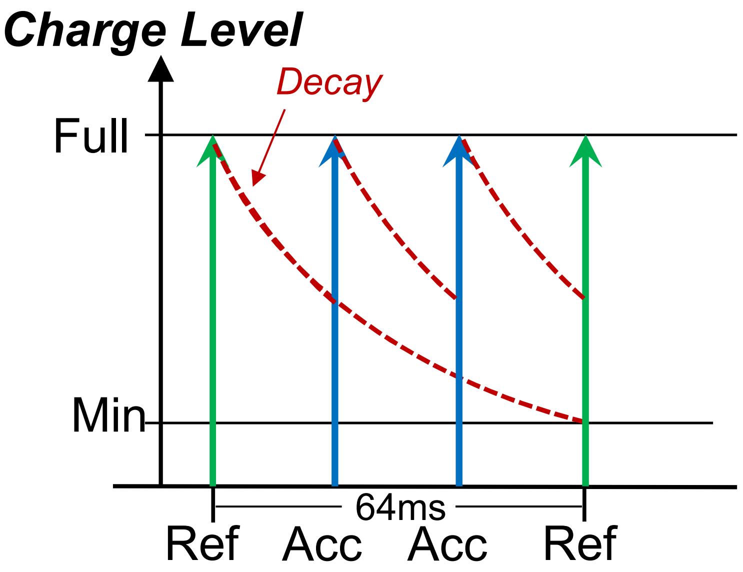
Timeline



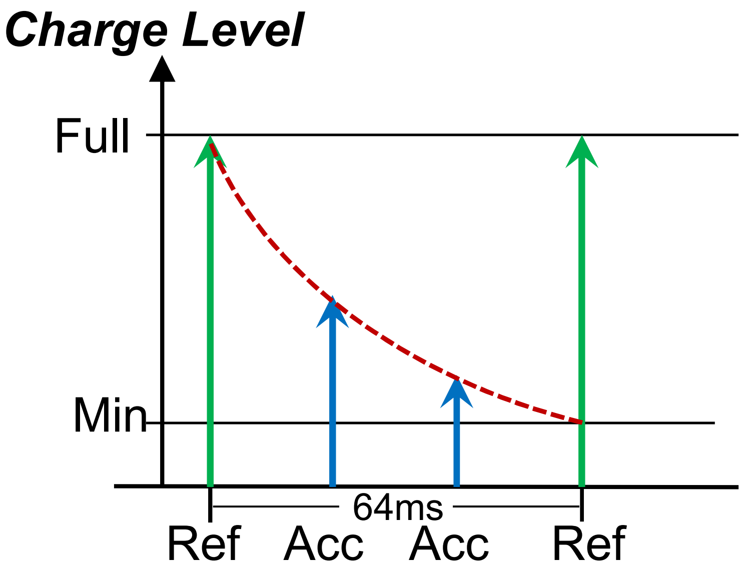
Reducing Latency with Partial Restoration

- **Prior work** [Zhang+ HPCA '16] **exploits the charge level of a DRAM cell by *partially restoring* the cell's charge**
  - Refreshes are regularly scheduled every 64ms
  - Baseline: charge level is **fully restored** after refresh *or* access
  - Apply **partial restoration for *soon-to-be refreshed* cells**

Full Charge Restoration



Partial Restoration



Outline

SAFARI

Background: Accessing Data in DRAM

Key Observations on Partial Restoration

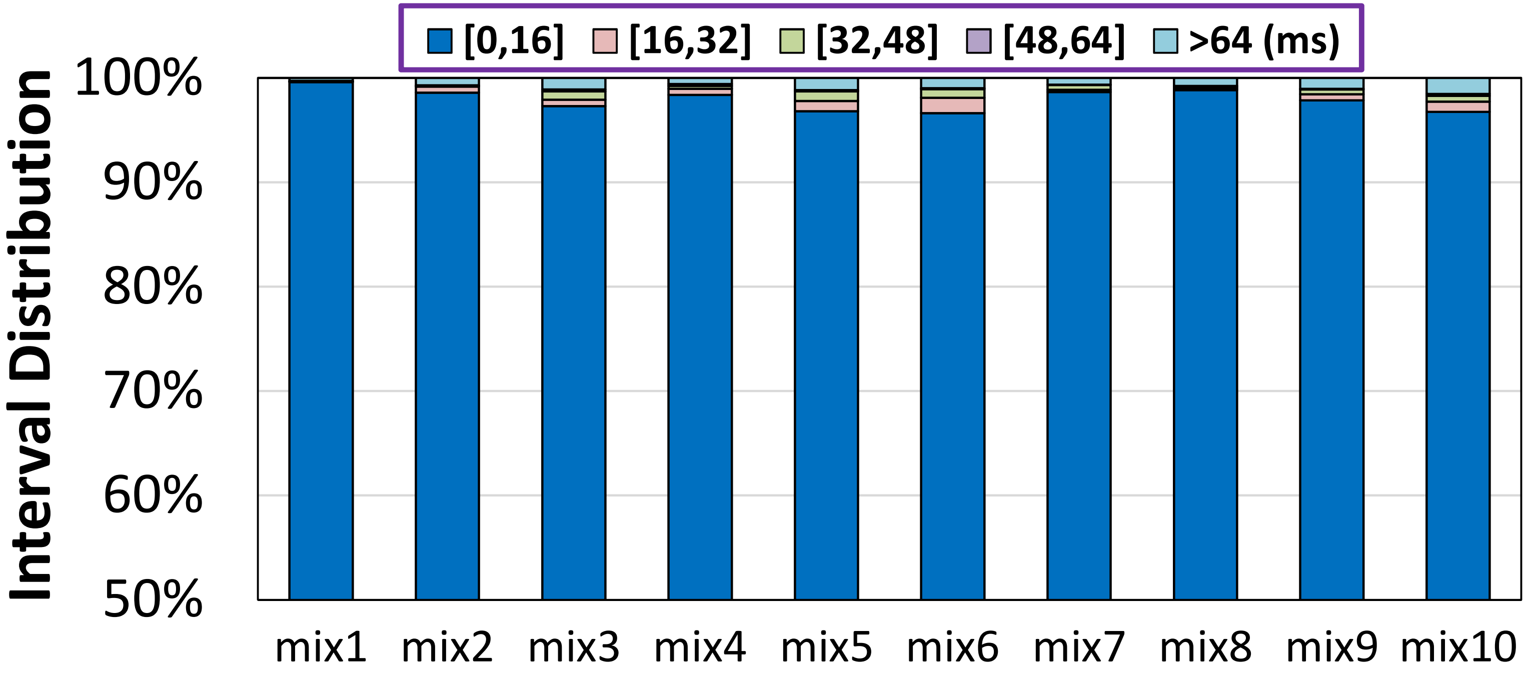
Charge-Level-Aware Look-Ahead Partial Restoration

Evaluation

Conclusion

More Significant Opportunities for Partial Restoration

- We can partially restore a cell if it will be reactivated soon



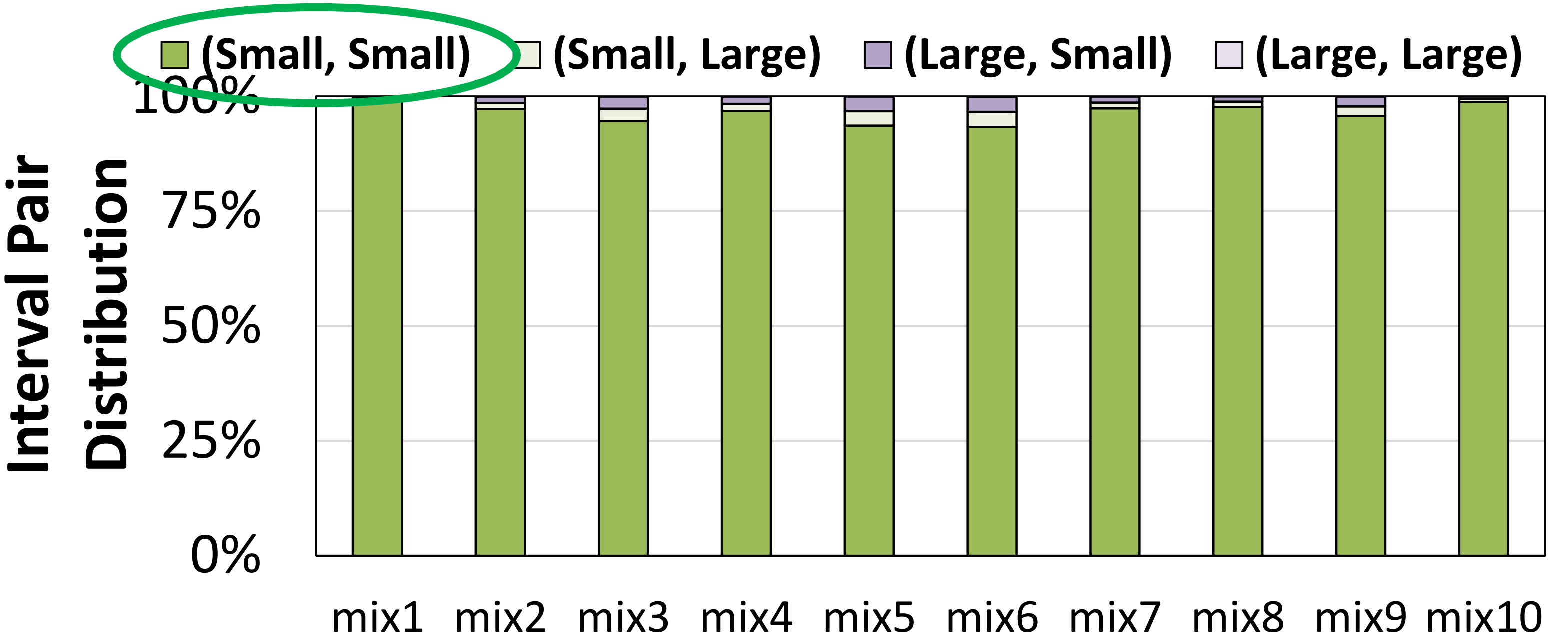
- On average, 95% of access-to-access intervals are <16ms

Large potential benefits for partial restoration, but **how can we predict when a cell will be reactivated?**

Predicting If a Cell Is Reactivated Soon (in 16ms)

SAFARI

- The next access-to-access interval of a cell can be accurately predicted by the last access-to-access interval



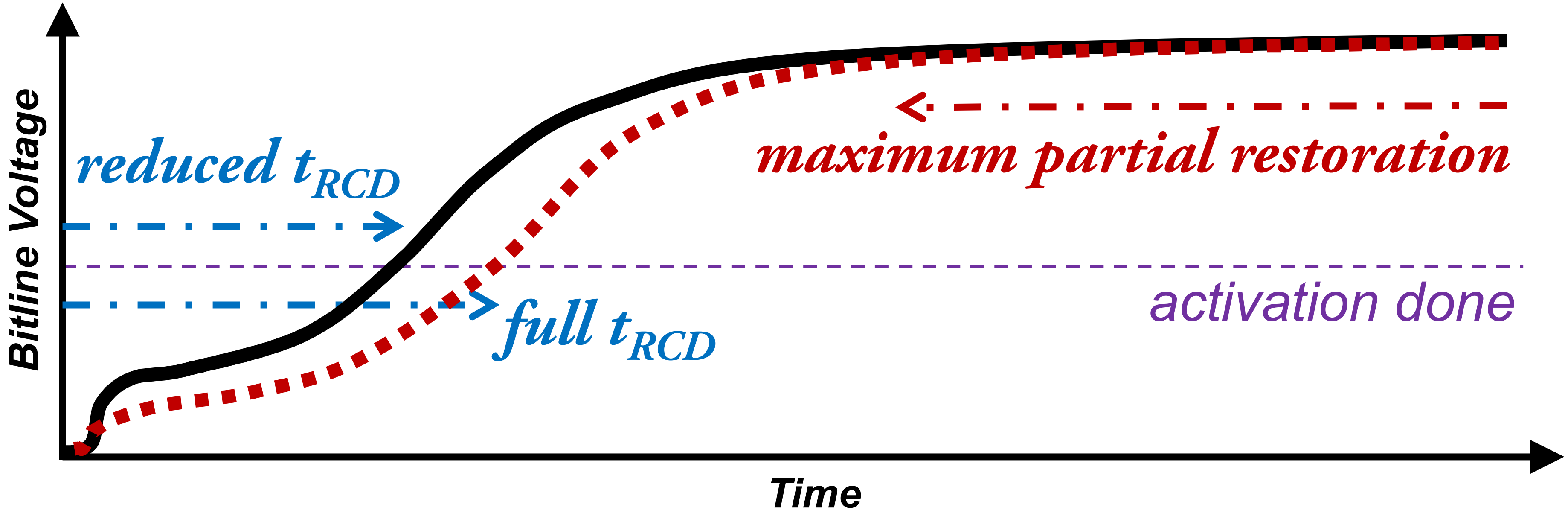
PREDICTION:

If the last interval is small, the next interval is small

98% accuracy for 8-core workloads

Balancing Activation/Restoration Latency Reductions

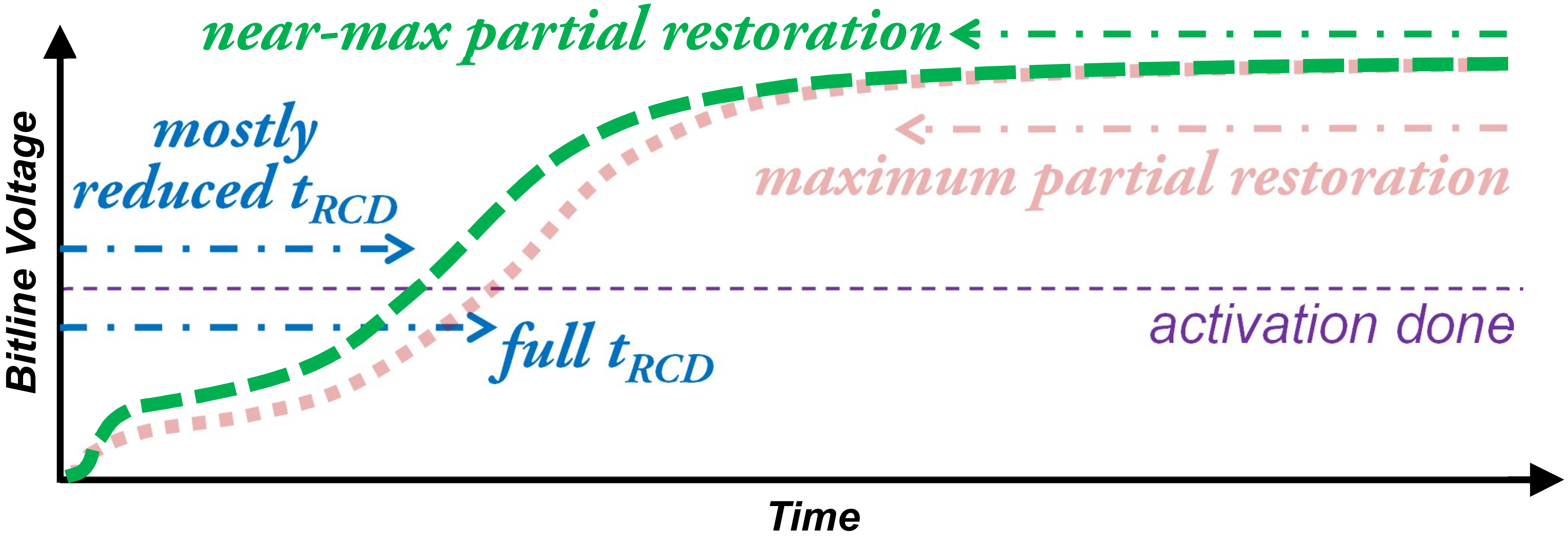
- We can also reduce the activation latency (t_{RCD}) for DRAM cells with high charge levels [Hassan+ HPCA 2016]



If we maximize the benefits of partial restoration, cells no longer have high charge at activation: **we can no longer reduce the activation latency**

Minimizing the Total DRAM Access Latency

- We need to trade off the latency reductions of both reduced activation latency and partial restoration



We need to find an optimal balance:
reduce *most* of the activation and restoration latencies
(details in the paper)

Summary of Key Observations

We can apply partial restoration on soon-to-be-reactivated DRAM cells

We can predict with very high accuracy if a cell will be reactivated soon

We can minimize the DRAM access latency by trading off reduced activation latencies and partial restoration

Outline

Background: Accessing Data in DRAM

Key Observations on Partial Restoration

Charge-Level-Aware Look-Ahead Partial Restoration

Evaluation

Conclusion

Overview of CAL

KEY IDEAS

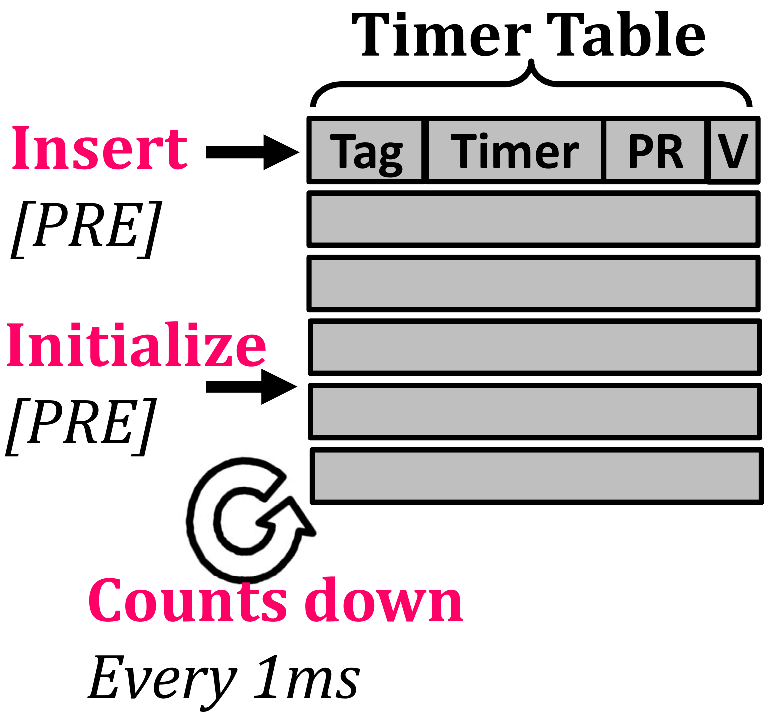
1. Track and use a DRAM row's last access-to-access interval to predict whether the row will be reactivated again soon

2. Reduce both the restoration and activation latencies, based on the prediction and next refresh

- CAL needs to track
  - Last access-to-access interval of each row
  - Whether the row was partially restored
- We add a ***timer table*** to the memory controller

Tracking Access-to-Access Intervals

- When a row is accessed initially
  - Insert an entry in the timer table
  - Initialize the entry's timer to 15ms
- Timer counts down every 1ms
- When a row is accessed again, check the timer in the table
  - < 1ms since last access: apply both partial restoration and reduced activation
  - 1ms –15ms since last access: use only partial restoration
- If timer reaches 0, or if entry is evicted
  - Fully restore the row if it was partially restored before



Outline

Background: Accessing Data in DRAM

Key Observations on Partial Restoration

Charge-Level-Aware Look-Ahead Partial Restoration

Evaluation

Conclusion

Evaluation Methodology

- **Simulation Platform**
  - Ramulator: open-source DRAM simulator [Kim+, CAL'15]
    https://github.com/CMU-SAFARI/ramulator
  - Energy model: includes CPU, caches, off-chip links, and DRAM
- **CAL Parameters**
  - **8-way** cache-like set-associative timer table
  - **256** entries, LRU replacement policy
  - Area overhead: 0.11% of a 16MB last-level cache
- **20 single-core workloads**
  - SPEC CPU2006, TPC, BioBench, Memory Scheduling Championship
  - Categorized into memory intensive and memory non-intensive
- **20 eight-core multiprogrammed workloads**
  - Vary the memory intensiveness from 25% to 100% of applications
- **Baseline: DDR4 DRAM**

Comparison Points

- **Base:** Baseline DDR4 configuration

- **CC:** ChargeCache
  - Reduces activation latency for highly-charged rows
  - [Hassan+, HPCA '16]

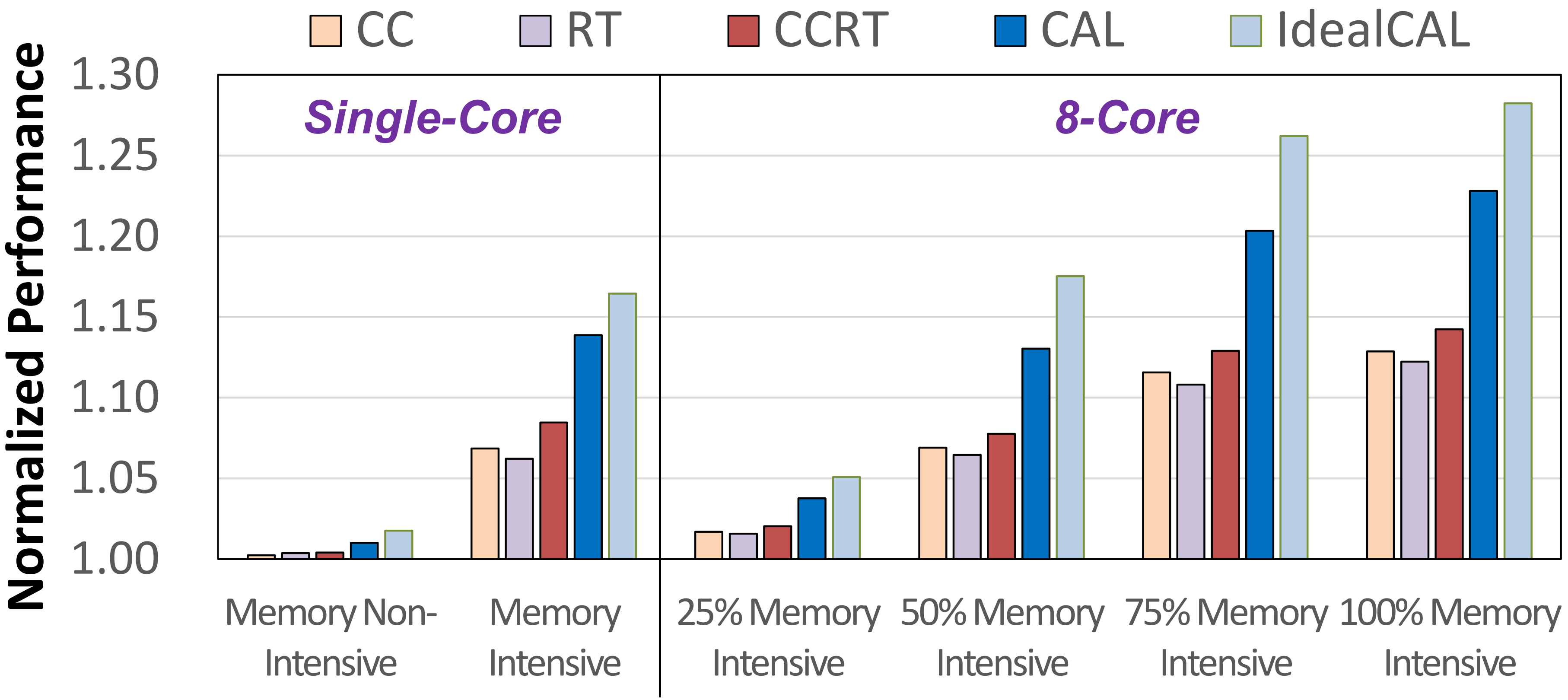
- **RT:** Restore Truncation
  - Reduces restoration latency for soon-to-be-refreshed rows
  - [Zhang+, HPCA '16]

- **CCRT:** combination of ChargeCache and Restore Truncation

- **IdealCAL:** idealized version of CAL

Performance Improvement Over DDR4 Baseline

SAFARI

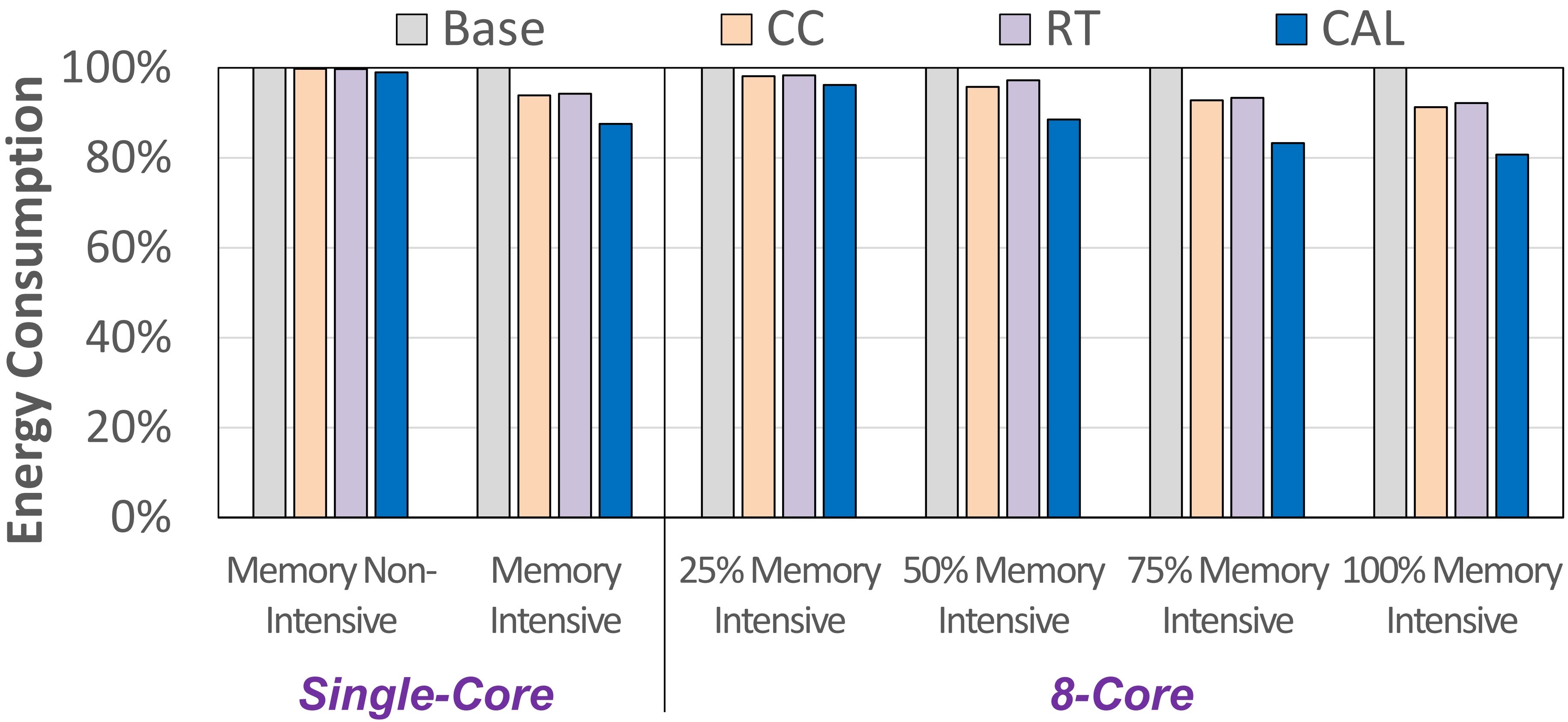


CAL significantly reduces the DRAM access latency

7.4% speedup for single-core workloads

14.7% speedup for eight-core workloads

Energy Savings Over DDR4 Baseline



CAL is effective at reducing energy consumption

10.1% reduction for memory-intensive single-core workloads

11.3% reduction for eight-core workloads

Other Results in Our Paper

- Explanations for access-to-access interval distribution

- Detailed analysis of the trade-off between activation and restoration latency reductions

- Handling large access-to-access intervals in the timer table

- Sensitivity studies on
    - Timer table size
    - Restoration level
    - Row management and address mapping policies
    - DRAM temperature

Conclusion

- DRAM access latency is the performance bottleneck

- Key observations
  - There is **significant potential to reduce latency** if we partially restore DRAM cells that **will be *reactivated* soon**
  - If we reduce restoration too much, we cannot use mechanisms that reduce the activation latency: **need to strike a balance**

- **CAL: Charge-Level-Aware Look-Ahead Partial Restoration**
  - **Track and use a DRAM row's last access-to-access interval** to predict whether the row will be reactivated again soon
  - **Reduce both the restoration and activation latencies,** based on the prediction and next refresh

- Significant DRAM access latency reductions at low cost: 14.7% speedup, 11.3% energy reduction 8-core workloads

Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration

Yaohua Wang, Arash Tavakkol, Lois Orosa, Saugata Ghose,
Nika Mansouri Ghiasi, Minesh Patel, Jeremie S. Kim,
Hasan Hassan, Mohammad Sadrosadati, Onur Mutlu

October 22, 2018

Backup Slides

Partial Restoration

- **Partial restoration parameters**
  - Conservatively chosen timing parameters
  - Charge level aware rather than aggressive partial restoration

- **Possible combination with DRAM profiling mechanisms to apply partial restoration only on strong cells, that can hold their charge level for a longer time**
  - Studies for cell profiling
  - [Patel+, ISCA2017]

15ms VS 16ms

- **Our analysis for access-to-access interval is based on the 16ms sub-window (Section 3 in the paper)**
  - Demonstrated by prior work,16ms is small enough to yield reasonable restoration latency reductions
  - Smaller interval does not add much

- **Our timer table design employs a 4-bit timer, which can count down from 15**
  - This provides us at least 1ms redundancy for us to issue ACT/PRE (i.e. fully restore), when the timer reaches 0.
  - Redundancy is necessary because DRAM commands have to obey timing constraints, there maybe some scheduled commands/refresh waiting to finish their operations.
  - 1ms seconds is large enough to guarantee that we can save the partially restored charge level

Per-core Timer Table Design

- The timer table can be implemented as either a single shared table or as per-core tables.

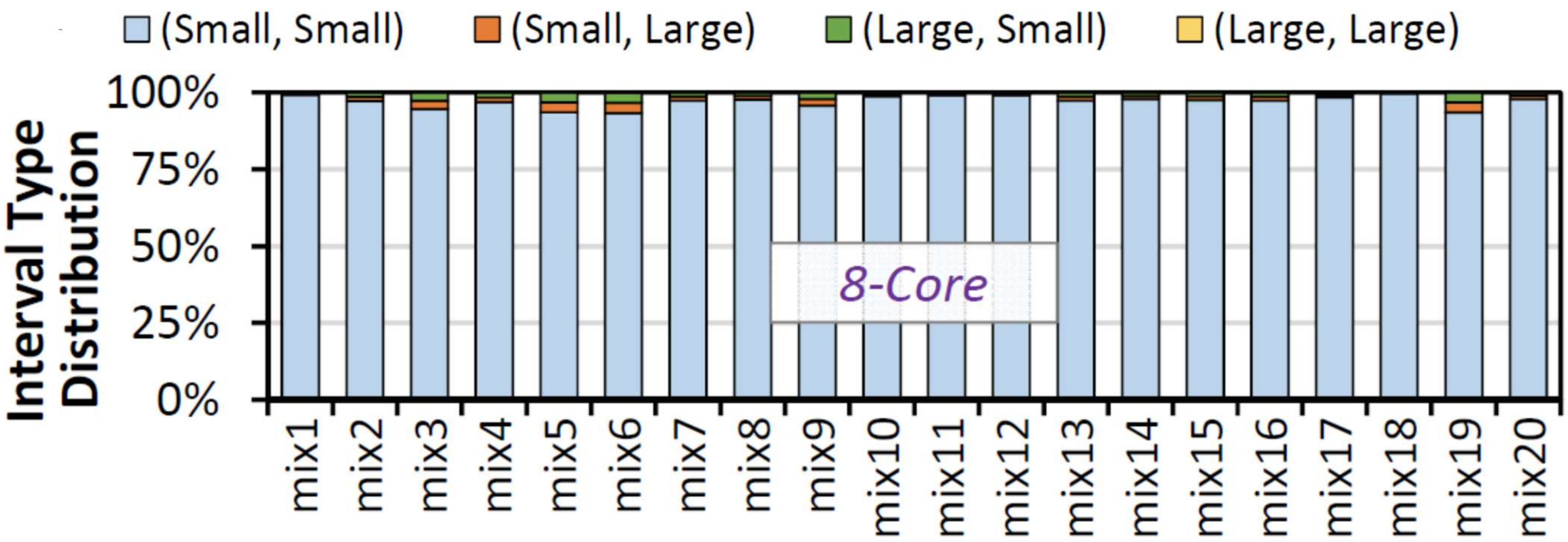
- In our evaluation, we assume that each core has a dedicated timer table.

- We choose per-core timer tables to avoid the need to tune the optimal size of a shared table based on the core count, and to simplify table organization.

Memory Intensity and Access Interval

SAFARI

- According to our analysis, memory intensity has little effect on our interval distribution



- Mix1-5: 25% memory intensive; Mix6-10: 50% memory intensive; Mix11-15, 75% memory intensive; 100% memory intensive

What If Applications with Large Access Intervals

- Our analysis of access intervals in Section 3 comes from a wide variety of benchmarks from multiple benchmark suits
  - SPEC CPU2006, TPC, BioBench, Memory Scheduling Championship
- The access locality contribute to our accuracy of interval prediction
- 16ms access interval is large enough to filter out the diversities among benchmarks, leading to a high prediction accuracy
  - With decrease interval (e.g., 8ms), the prediction accuracy would decrease by roughly 10%
- While applications with mostly >16ms access-to-access interval may exist, they tend to be memory-none intensive, memory latency is not as critical

The Uniform Interval Distributions

- Refresh happens at fixed time intervals, independent of the memory access pattern.
- Due to a combination of the access locality and the high number of row conflicts
- 16ms interval helps to filter out the diversities of distributions

Handling Large Access-to-access intervals

- **Two cases to save data by ACT and PRE command pairs**
  - Miss prediction: when a row timer reaches zero
  - Fail to track the partially restored row: when an entry is evicted
  - An (ACT, PRE) command pair issued immediately
- **1ms redundancy is provided to guarantee timing requirement**
- **Less than 0.1% of performance overhead introduced by the (ACT, PRE) command pairs**

Trade-off between tRCD and tRAS reduction

SAFARI

- Heuristic of the trade-off

$$PRBenefit(V_x) = \frac{\Delta t_{RAS}(Vx)}{\Delta t_{RCD}(V_{full}) - \Delta t_{RCD}(Vx)} \qquad (1)$$

  - V_x is the voltage level a DRAM cell is partially restored to. According to our conservative estimation it should be between 0.75Vdd to 0.975Vdd

  - We choose 0.85Vdd as the optimal V_x that maximizes PRBenefit

3. Area and Power Overhead

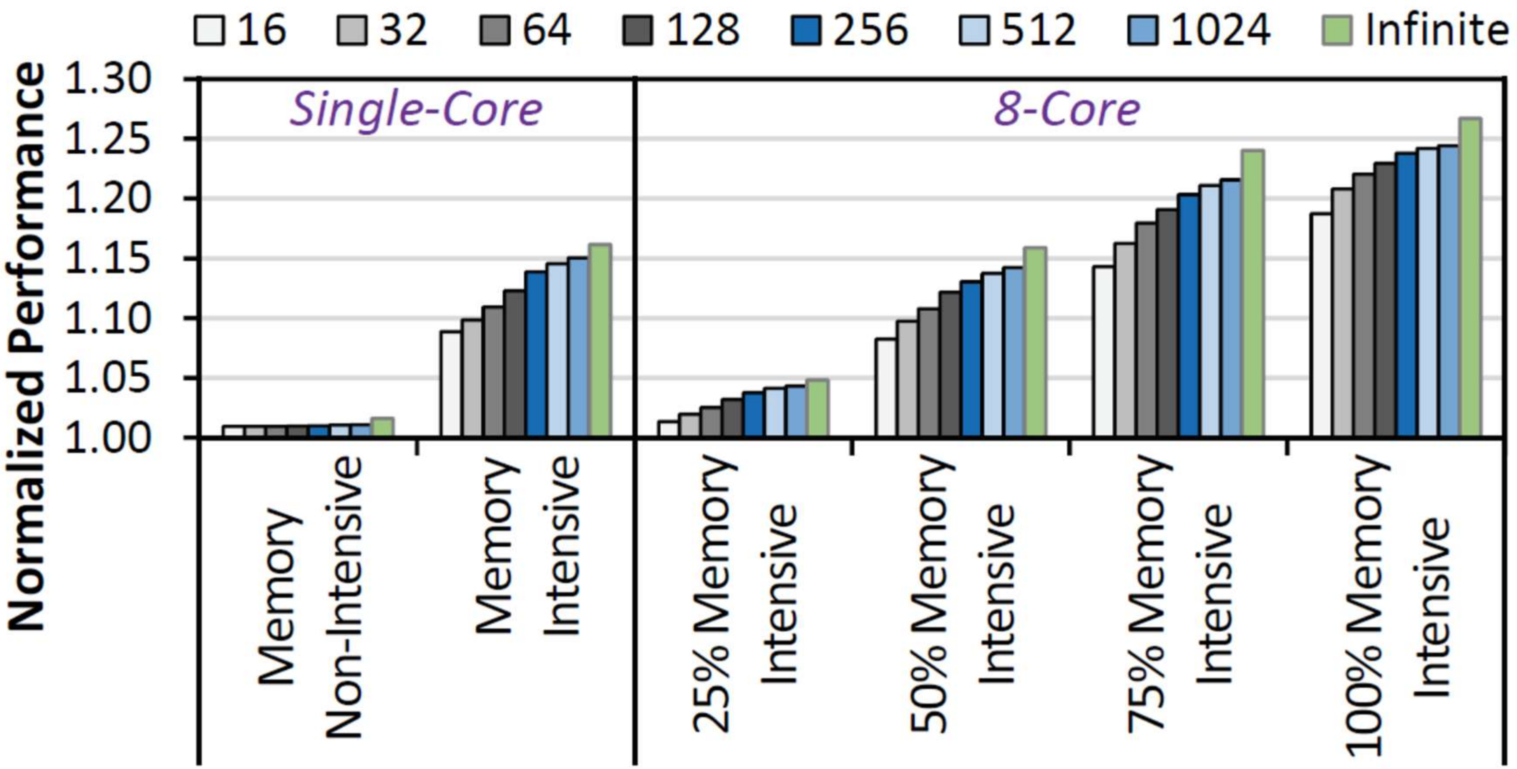
- Per-core 256-entry timer table

- 7.7KB with 0.034mm^2 area overhead, 0.11% of 16MB LLC

- 0.202mW power consumption, 0.08% of LLC

- New commands with reduced activation and restoration latency
  - Reserved undefined encoding
  - No additional command bus bits are needed

Timer Table Size

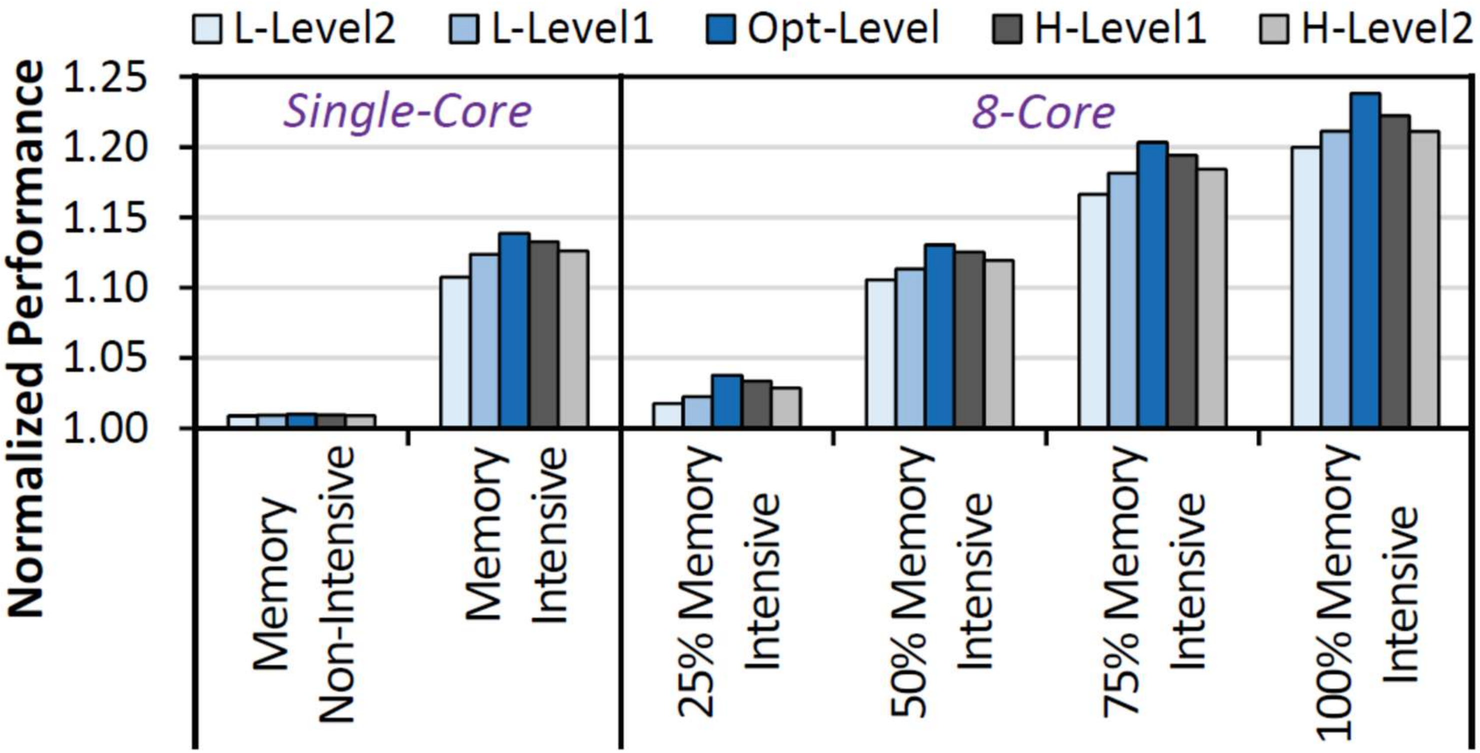


A larger table capacity provides higher performance

The improvement diminish at larger table capacities

Different Restoration Levels

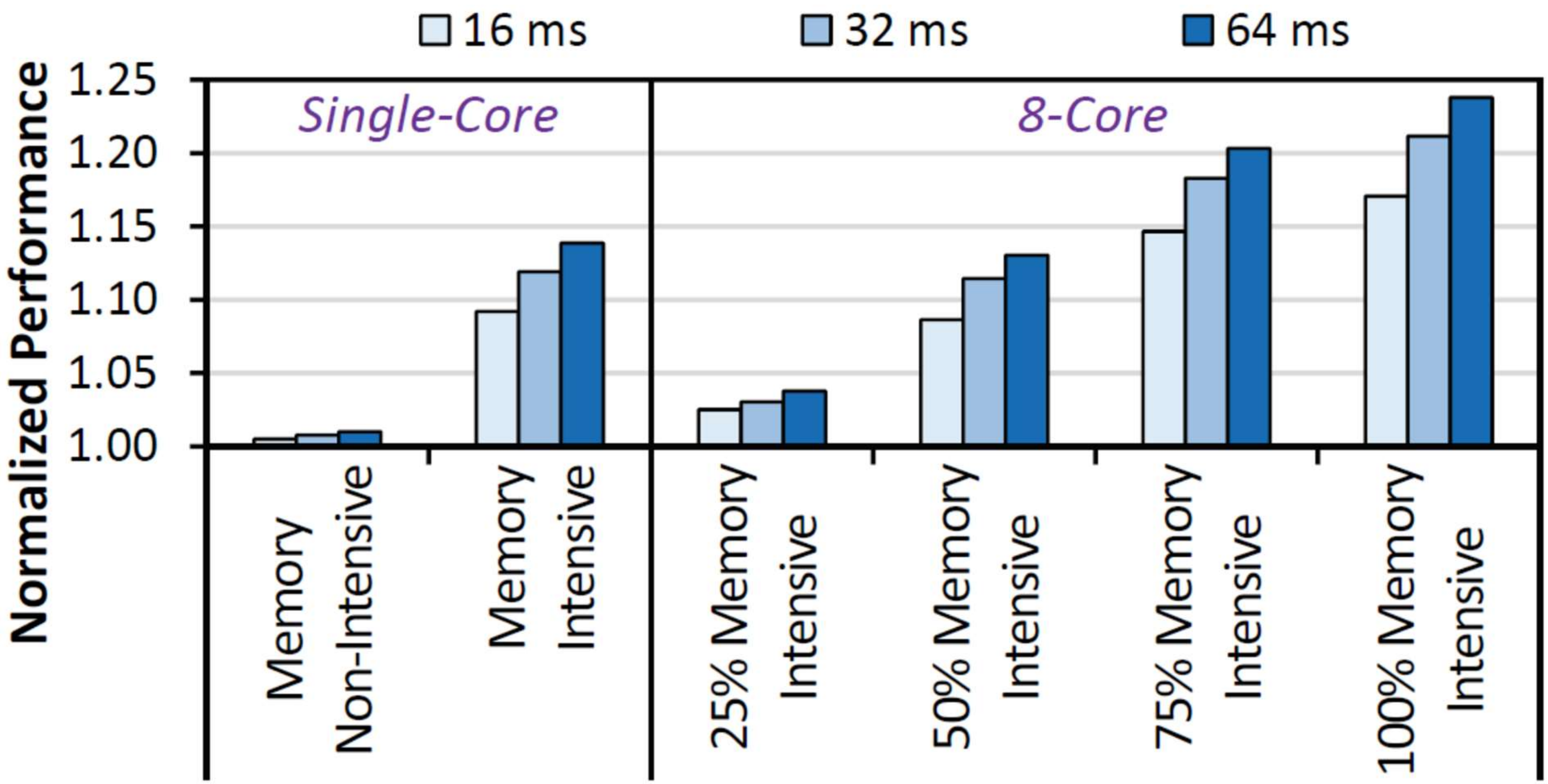
SAFARI



The restoration level plays an important role in balancing between restoration and activation latency reduction.

Different Refresh Interval

SAFARI



CAL is still effective at reduced refresh intervals
CAL can be extended to support partial refresh

Activation and Restoration Latency Reduction Trade-off

- There exists trade-off between activation and restoration latency reductions

- Proper partial restoration level
  - Achieve **significantly reduced** restoration latency
  - A **smaller** reduction of activation latency
  - Achieving the benefits of both

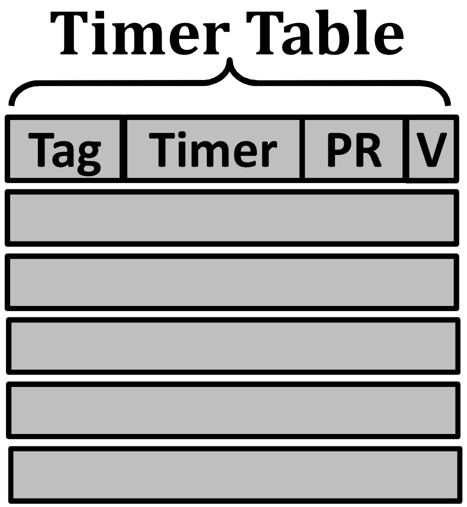
- A simple and effective trade-off heuristic to approximate the maximum benefit from the trade-off (see paper)

Key Idea & Structure of CAL

SAFARI

1. Uses the last access-to-access interval of a row to **predict** whether the row will be reactivated again soon

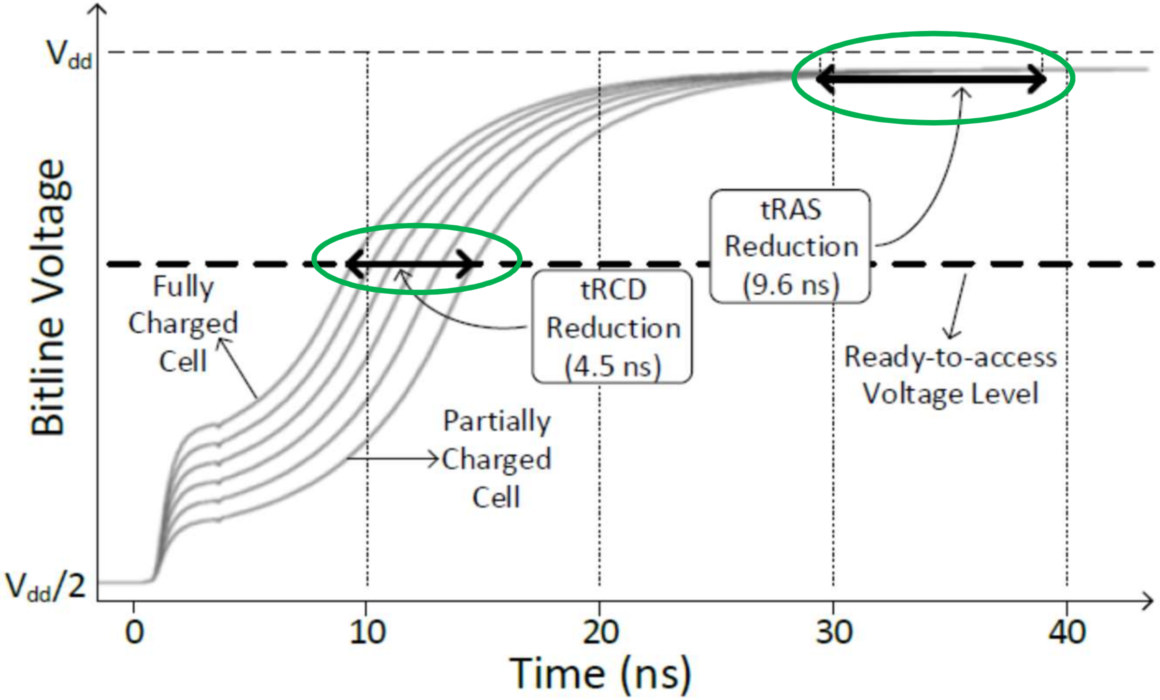
2. Reduces both restoration and activation latencies, based on the prediction and next refresh



- Tag: stores the DRAM row address
- Timer: records the time elapsed since last precharge
- Partial Restored bit (PR): set to 1 when a row is partially restored
- Valid bit (V)

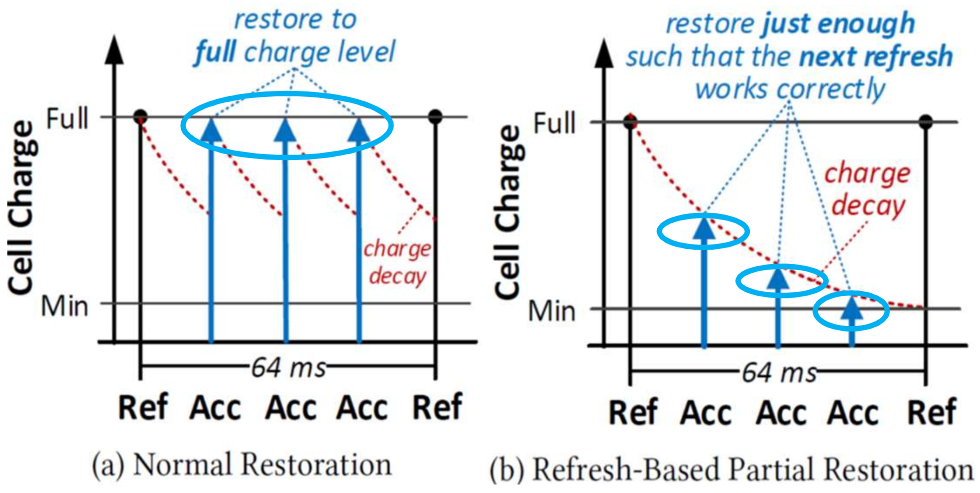
Exploiting Charge Levels to Reduce Latency

- Prior works exploit the charge level of a DRAM cell to reduce the restoration and activation latencies
  - High charge level can reduce activation latency
    - » Cells accessed in the last 1ms



Exploiting Charge Levels to Reduce Latency

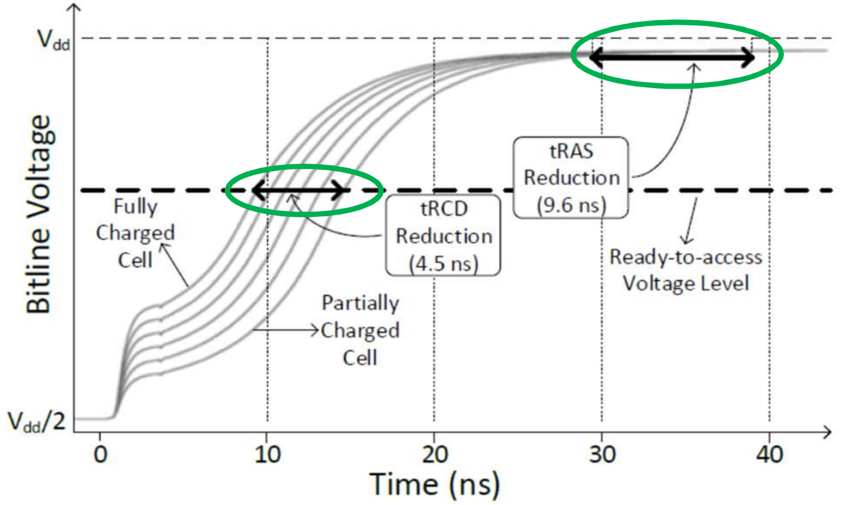
- Prior works exploit the charge level of a DRAM cell to reduce the restoration and activation latencies
  - DRAM cells' charge level can be partially restored to reduce Restoration latency
    - » Partially restore the charge level of soon-to-be refreshed cells



(a) Normal Restoration

(b) Refresh-Based Partial Restoration

  - High charge level can reduce activation latency
    - » Cells accessed in the last 1ms


V_{dd}
$V_{dd}/2$
Bitline Voltage
Fully
Charged
Cell
tRCD
Reduction
(4.5 ns)
tRAS
Reduction
(9.6 ns)
Ready-to-access
Voltage Level
Partially
Charged
Cell
Time (ns)