Computer Architecture
Lecture 6a: ChargeCache

Hasan Ibrahim Hasan
ETH Zürich
Fall 2018
4 October 2018
ChargeCache
Reducing DRAM Latency by Exploiting Row Access Locality

Hasan Hassan,
Gennady Pekhimenko,
Nandita Vijaykumar,
Vivek Seshadri, Donghyuk Lee,
Oguz Ergin, Onur Mutlu

SAFARI
Carnegie Mellon

TOBB UNIVERSITY OF ECONOMICS AND TECHNOLOGY
kasirga
Executive Summary

- **Goal**: Reduce average DRAM access latency with no modification to the existing DRAM chips

- **Observations**:
  1) A highly-charged DRAM row can be accessed with low latency
  2) A row’s charge is restored when the row is accessed
  3) A recently-accessed row is likely to be accessed again: Row Level Temporal Locality (RLTL)

- **Key Idea**: Track recently-accessed DRAM rows and use lower timing parameters if such rows are accessed again

- **ChargeCache**:
  - Low cost & no modifications to the DRAM
  - Higher performance **(8.6-10.6% on average for 8-core)**
  - Lower DRAM energy **(7.9% on average)**
Outline

1. DRAM Operation Basics
2. Accessing Highly-charged Rows
3. Row Level Temporal Locality (RLTL)
4. ChargeCache
5. Evaluation
6. Conclusion
DRAM Stores Data as Charge

Three steps of charge movement:
1. S
2. R
3. Precharge

DRAM Cell

Sense-Amplifier

CPU

MemCtrl
DRAM Charge over Time

Ready to Access
Ready to Precharge
Ready to Access Charge Level

Data 0
Data 1

Cell
Sense-Amplifier

Cell
Sense-Amplifier

Sensing
Restore
Precharge
time

charge

Act
R/W
Pre

tRCD

tRAS

SAFARI
Outline

1. DRAM Operation Basics
2. Accessing Highly-charged Rows
3. Row Level Temporal Locality (RLTL)
4. ChargeCache
5. Evaluation
6. Conclusion
Accessing Highly-charged Rows

- **Ready to Access**
- **Ready to Precharge**

- **Cell**
- **Sense-Amplifier**

- **Data 0**
- **Data 1**

- **Sensing**
- **Restore**
- **Precharge**

- **tRCD**
- **tRAS**

**SAFARI**
Observation 1

A **highly-charged** DRAM row can be accessed with **low latency**

- tRCD: 44%
- tRAS: 37%

How does a row become highly-charged?
How Does a Row Become Highly-Charged?

DRAM cells lose charge over time

Two ways of restoring a row’s charge:

- Refresh Operation
- Access
Observation 2

A row’s charge is **restored** when the row is **accessed**

How likely is a **recently-accessed** row to be accessed again?
1. DRAM Operation Basics

2. Accessing Highly-charged Rows

3. Row Level Temporal Locality (RLTL)

4. ChargeCache

5. Evaluation

6. Conclusion
Row Level Temporal Locality (RLTL)

A recently-accessed DRAM row is likely to be accessed again.

- $t$-RLTL: Fraction of rows that are accessed within time $t$ after their previous access.

$\text{88ms} - \text{RLTL for eight-core workloads}$
Outline

1. DRAM Operation Basics
2. Accessing Highly-charged Rows
3. Row Level Temporal Locality (RLTL)
4. ChargeCache
5. Evaluation
6. Conclusion
Summary of the Observations

1. A highly-charged DRAM row can be accessed with low latency

2. A row’s charge is restored when the row is accessed

3. A recently-accessed DRAM row is likely to be accessed again:
   
   **Row Level Temporal Locality (RLTL)**
Key Idea

Track **recently-accessed** DRAM rows and use **lower timing parameters** if such rows are accessed again.
ChargeCache Overview

Requests: A  D  A

ChargeCache Hits: Use Default Timings

ChargeCache Miss: Use Default Timings

Memory Controller

ChargeCache

A

D

DRAM

A

B

C

D

E

F
Area and Power Overhead

• Modeled with CACTI

• Area
  – ~5KB for 128-entry ChargeCache
  – 0.24% of a 4MB Last Level Cache (LLC) area

• Power Consumption
  – 0.15 mW on average (static + dynamic)
  – 0.23% of the 4MB LLC power consumption
1. DRAM Operation Basics
2. Accessing Highly-charged Rows
3. Row Level Temporal Locality (RLTL)
4. ChargeCache
5. Evaluation
6. Conclusion
Methodology

• Simulator
  – Ramulator [Kim+, CAL’15]
    https://github.com/CMU-SAFARI/ramulator

• Workloads
  – 22 single-core workloads
    • SPEC CPU2006, TPC, STREAM
  – 20 multi-programmed 8-core workloads
    • By randomly choosing from single-core workloads
  – Execute at least 1 billion representative instructions per core (Pinpoints)

• System Parameters
  – 1/8 core system with 4MB LLC
  – Default tRCD/tRAS of 11/28 cycles
Mechanisms Evaluated
Non-Uniform Access Time Memory Controller (NUAT)  
*Shin et al., HPCA’14*

- **Key idea**: Access only *recently-refreshed* rows with lower timing parameters
  - *Recently-refreshed* rows can be accessed faster
  - Only a small fraction (10-12%) of accesses go to *recently-refreshed* rows

**ChargeCache**

- *Recently-accessed* rows can be accessed faster
- A large fraction (86-97%) of accesses go to *recently-accessed* rows *(RLTL)*
  - 128 entries per core, **On hit**: tRCD=7, tRAS=20 cycles

**Upper Bound: Low Latency DRAM**

- Works as ChargeCache with 100% Hit Ratio
- **On all DRAM accesses**: tRCD=7, tRAS=20 cycles
ChargeCache improves single-core performance
Eight-core Performance

- NUAT: 2.5%
- ChargeCache: 9%
- ChargeCache + NUAT
- LL-DRAM (Upperbound): 13%

ChargeCache significantly improves multi-core performance.
DRAM Energy Savings

ChargeCache reduces DRAM energy

SAFARI
Other Results In The Paper

• Detailed analysis of the Row Level Temporal Locality phenomenon

• ChargeCache hit-rate analysis

• Sensitivity studies
  o Sensitivity to \( t \) in \( t\)-RLTL
  o ChargeCache capacity
Conclusion

• ChargeCache reduces average DRAM access latency at low cost

• **Observations:**
  1) A highly-charged DRAM row can be accessed with low latency
  2) A row’s charge is restored when the row is accessed
  3) A recently-accessed row is likely to be accessed again: **Row Level Temporal Locality (RLTL)**

• **Key Idea:** Track recently-accessed DRAM rows and use lower timing parameters if such rows are accessed again

• **ChargeCache:**
  - Low cost & no modifications to the DRAM
  - Higher performance (8.6-10.6% on average for 8-core)
  - Lower DRAM energy (7.9% on average)
ChargeCache
Reducing DRAM Latency by Exploiting Row Access Locality

Hasan Hassan,
Gennady Pekhimenko,
Nandita Vijaykumar,
Vivek Seshadri, Donghyuk Lee,
Oguz Ergin, Onur Mutlu

SAFARI
Carnegie Mellon

TOBB UNIVERSITY OF ECONOMICS AND TECHNOLOGY
kasirga
Backup Slides
Detailed Design

1. PRE
   Insert Row Address

2. ACT
   Lookup the Address

3. Invalidation Mechanism

Highly-charged Row Address Cache (HCRAC)
RLTL Distribution

Fraction of Accesses

0.125ms - RLTL
0.25ms - RLTL
0.5ms - RLTL
1ms - RLTL
32ms - RLTL

SAFARI
Sensitivity on Capacity

![Graph showing sensitivity on capacity for single-core and eight-core systems. The x-axis represents capacity in thousands of units, ranging from 4 to 8192. The y-axis represents speedup, ranging from 0% to 15%. The graph shows an increasing trend for both single-core and eight-core systems as capacity increases.](image-url)
Hit-rate Analysis

![Graph showing hit-rate analysis for different numbers of ChargeCache entries. The graph compares single-core and eight-core systems, with and without unlimited size configurations. The y-axis represents ChargeCache hit-rate, ranging from 0% to 100%. The x-axis represents the number of ChargeCache entries, ranging from 0 to 1024. The graph illustrates how hit-rate improves as the number of entries increases, with single-core systems generally showing lower hit-rates compared to eight-core systems, especially as the number of entries grows.]
Sensitivity on t-RLTL

The graph shows the sensitivity of t-RLTL for different time durations and core counts. The x-axis represents time durations (1ms, 4ms, 8ms, 16ms) and core counts (single-core, eight-core), while the y-axis shows speedup and charge cache hit rate. The data indicates that as the time duration increases, the speedup decreases, and the hit rate increases. The single-core configuration shows lower speedup compared to the eight-core configuration for the same time durations.