ChargeCache
Reducing DRAM Latency by Exploiting Row Access Locality

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

Memory Control (MemCtrl)

CPU

DRAM Cell

Sense-Amplifier
DRAM Charge over Time

Cell

Sense Amplifier

Sense-Ampifier

Ready to Access

Ready to Precharge

Ready to Access Charge Level

Data 1

Data 0

charge

tRAS

tRCD

Sensing

Restore

Precharge

time

ACT

R/W

PRE

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

Cell

Sense-Amplifier

Ready to Access

Ready to Precharge

Data 0

Data 1

Sensing

Restore

Precharge

charge

tRCD

tRAS

ACT

R/W

PRE

time

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?
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.

\[ 86\% \text{ for single-core workloads} \]

\[ 97\% \text{ for eight-core workloads} \]
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

Memory Controller

ChargeCache

Requests: A D A

ChargeCache Hit: Use Default Timings
Area and Power Overhead

- Modeled with CACTI

- **Area**
  - \(~5\text{KB}\) for 128-entry ChargeCache
  - \(0.24\%\) of a 4MB Last Level Cache (LLC) area

- **Power Consumption**
  - \(0.15\text{ mW}\) on average (static + dynamic)
  - \(0.23\%\) of the 4MB LLC power consumption
Outline

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

• Simulator
  – DRAM 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+, 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
ChargeCache reduces DRAM energy
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
Outline

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

Source code will be available in May
https://github.com/CMU-SAFARI
ChargeCache
Reducing DRAM Latency by Exploiting Row Access Locality

Hasan Hassan,
Gennady Pekhimenko,
Nandita Vijaykumar,
Vivek Seshadri, Donghyuk Lee,
Oguz Ergin, Onur Mutlu
Backup Slides
Detailed Design

Highly-charged Row Address Cache (HCRAC)

1. PRE
   Insert Row Address

2. ACT
   Lookup the Address

3. Invalidation Mechanism
Sensitivity on Capacity

The graph illustrates the speedup as a function of capacity for both single-core and eight-core systems. The speedup increases with capacity, showing a clear advantage for eight-core systems over single-core systems for larger capacities.
Hit-rate Analysis

ChargeCache Hit-Rate

Number of ChargeCache Entries

Single-core
Eight-core
Single-core (Unlimited Size)
Eight-core (Unlimited Size)
Sensitivity on t-RLTL

![Bar chart showing speedup and ChargeCache hit-rate for different core counts and cycle times.](chart.png)