Memory Systems
and Memory-Centric Computing Systems
Lecture 5, Topic 4: Low-Latency Memory

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
13 July 2018
HiPEAC ACACES Summer School 2018
Eliminating the Adoption Barriers

How to Enable Adoption of Processing in Memory
Barriers to Adoption of PIM

1. Functionality of and applications for PIM
2. Ease of programming (interfaces and compiler/HW support)
3. System support: coherence & virtual memory
4. Runtime systems for adaptive scheduling, data mapping, access/sharing control
5. Infrastructures to assess benefits and feasibility
We Need to Revisit the Entire Stack

```
We Need to Revisit the Entire Stack

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons
```
Key Challenge 1: Code Mapping

**Challenge 1:** Which operations should be executed in memory vs. in CPU?
Key Challenge 2: Data Mapping

- **Challenge 2:** How should data be mapped to different 3D memory stacks?
How to Do the Code and Data Mapping?


[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]
How to Schedule Code?

  
  Proceedings of the 25th International Conference on Parallel Architectures and Compilation Techniques (PACT), Haifa, Israel, September 2016.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik\textsuperscript{1} Xulong Tang\textsuperscript{1} Adwait Jog\textsuperscript{2} Onur Kayiran\textsuperscript{3}
Asit K. Mishra\textsuperscript{4} Mahmut T. Kandemir\textsuperscript{1} Onur Mutlu\textsuperscript{5,6} Chita R. Das\textsuperscript{1}

\textsuperscript{1}Pennsylvania State University \textsuperscript{2}College of William and Mary
\textsuperscript{3}Advanced Micro Devices, Inc. \textsuperscript{4}Intel Labs \textsuperscript{5}ETH Zürich \textsuperscript{6}Carnegie Mellon University
Challenge: Coherence for Hybrid CPU-PIM Apps

<table>
<thead>
<tr>
<th>App</th>
<th>Components</th>
<th>Radii</th>
<th>PageRank</th>
<th>Components</th>
<th>Radii</th>
<th>PageRank</th>
<th>Components</th>
<th>Radii</th>
<th>PageRank</th>
<th>HTAP-256</th>
<th>HTAP-128</th>
<th>GMean</th>
</tr>
</thead>
<tbody>
<tr>
<td>arXiV</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Gnutella</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Enron</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>IMDB</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

Traditional coherence

No coherence overhead
Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"

LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

Amirali Boroumand†, Saugata Ghose†, Minesh Patel†, Hasan Hassan†‡, Brandon Lucia†,
Kevin Hsieh†, Krishna T. Malladi*, Hongzhong Zheng*, and Onur Mutlu‡†

†Carnegie Mellon University  *Samsung Semiconductor, Inc.  ‡TOBB ETÜ  †ETH Zürich

SAFARI
How to Support Virtual Memory?


Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh†  Samira Khan‡  Nandita Vijaykumar†
Kevin K. Chang†  Amirali Boroumand†  Saugata Ghose†  Onur Mutlu§†
†Carnegie Mellon University  ‡University of Virginia  §ETH Zürich
How to Design Data Structures for PIM?

[Slides (pptx) (pdf)]

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu
Computer Science Department
Brown University
zhiyu.liu@brown.edu

Irina Calciu
VMware Research Group
icalciu@vmware.com

Maurice Herlihy
Computer Science Department
Brown University
mph@cs.brown.edu

Onur Mutlu
Computer Science Department
ETH Zürich
onur.mutlu@inf.ethz.ch
Simulation Infrastructures for PIM

- **Ramulator** extended for PIM
  - Flexible and extensible DRAM simulator
  - Can model many different memory standards and proposals
  - Kim+, “**Ramulator: A Flexible and Extensible DRAM Simulator**”, IEEE CAL 2015.
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)
An FPGA-based Test-bed for PIM?


- Flexible
- Easy to Use (C++ API)
- Open-source

  github.com/CMU-SAFARI/SoftMC
New Applications and Use Cases for PIM

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"  
  *BMC Genomics*, 2018.  
  Proceedings of the *16th Asia Pacific Bioinformatics Conference (APBC)*, Yokohama, Japan, January 2018.  
  arxiv.org Version (pdf)

GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Jeremie S. Kim\(^1,6\)*, Damla Senol Cali\(^1\), Hongyi Xin\(^2\), Donghyuk Lee\(^3\), Saugata Ghose\(^1\), Mohammed Alser\(^4\), Hasan Hassan\(^6\), Oguz Ergin\(^5\), Can Alkan\(^4\)* and Onur Mutlu\(^6,1\)*

*From* The Sixteenth Asia Pacific Bioinformatics Conference 2018  
Yokohama, Japan. 15-17 January 2018
Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand
Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, Onur Mutlu
Genome Read In-Memory (GRIM) Filter:
Fast Seed Location Filtering in DNA Read Mapping using Processing-in-Memory Technologies

Jeremie Kim,
Damla Senol, Hongyi Xin, Donghyuk Lee,
Saugata Ghose, Mohammed Alser, Hasan Hassan,
Oguz Ergin, Can Alkan, and Onur Mutlu
Executive Summary

- **Genome Read Mapping** is a very important problem and is the first step in many types of genomic analysis
  - Could lead to improved health care, medicine, quality of life

- Read mapping is an approximate string matching problem
  - Find the best fit of 100 character strings into a 3 billion character dictionary
  - **Alignment** is currently the best method for determining the similarity between two strings, but is very expensive

- We propose an in-memory processing algorithm **GRIM-Filter** for accelerating read mapping, by reducing the number of required alignments

- We implement GRIM-Filter using in-memory processing within **3D-stacked memory** and show up to 3.7x speedup.
The layout of bit vectors in a bank enables filtering many bins in parallel.

Customized logic for accumulation and comparison per genome segment:
- Low area overhead, simple implementation.
GRIM-Filter Performance

1.8x-3.7x performance benefit across real data sets
GRIM-Filter False Positive Rate

5.6x-6.4x False Positive reduction across real data sets
Conclusions

- We propose an in memory filter algorithm to accelerate end-to-end genome read mapping by reducing the number of required alignments.

- Compared to the previous best filter:
  - We observed 1.8x-3.7x speedup
  - We observed 5.6x-6.4x fewer false positives

- GRIM-Filter is a universal filter that can be applied to any genome read mapper.
In-Memory DNA Sequence Analysis

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu,
  "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"
  *BMC Genomics*, 2018.
  Proceedings of the 16th Asia Pacific Bioinformatics Conference (APBC), Yokohama, Japan, January 2018.
  arxiv.org Version (pdf)

GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Jeremie S. Kim\(^1,6^*\), Damla Senol Cali\(^1\), Hongyi Xin\(^2\), Donghyuk Lee\(^3\), Saugata Ghose\(^1\), Mohammed Alser\(^4\), Hasan Hassan\(^6\), Oguz Ergin\(^5\), Can Alkan\(^4^*\) and Onur Mutlu\(^6,1^*\)

*From* The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018
Open Problems: PIM Adoption

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun
Carnegie Mellon University

Onur Mutlu
ETH Zürich and Carnegie Mellon University

Enabling the Paradigm Shift
Computer Architecture Today

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication, and storage

- Recommended book: Thomas Kuhn, “The Structure of Scientific Revolutions” (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined
You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly).

You can invent new paradigms for computation, communication,

Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)

- Pre-paradigm science: no clear consensus in the field
- Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
- Revolutionary science: underlying assumptions re-examined
What Will You Learn in This Course?

- **Memory Systems and Memory-Centric Computing Systems**
  - July 9-13, 2018

- Topic 1: Main Memory Trends and Basics
- Topic 2: Memory Reliability & Security: RowHammer and Beyond
- Topic 3: In-memory Computation
- **Topic 4: Low-Latency (and Low-Energy) Memory**
- Topic 5 (unlikely): Enabling and Exploiting Non-Volatile Memory
- Topic 6 (unlikely): Flash Memory and SSD Scaling

- Major Overview Reading:
  - Mutlu and Subramaniam, “Research Problems and Opportunities in Memory Systems,” SUPERFRI 2014.
Agenda

- Brief Introduction
- A Motivating Example
- Memory System Trends
- What Will You Learn In This Course
  - And, how to make the best of it...
- Memory Fundamentals
- Key Memory Challenges and Solution Directions
  - Security, Reliability, Safety
  - Energy and Performance: Data-Centric Systems
  - Latency and Latency-Reliability Tradeoffs
- Summary and Future Lookout
Four Key Directions

- Fundamentally **Secure/Reliable/Safe** Architectures
- Fundamentally **Energy-Efficient** Architectures
  - Memory-centric (Data-centric) Architectures
- Fundamentally **Low-Latency** Architectures
- Architectures for **Genomics, Medicine, Health**
Maslow’s Hierarchy of Needs, A Third Time


Source: https://www.simplypsychology.org/maslow.html
Challenge and Opportunity for Future

Fundamentally Low-Latency Computing Architectures
Memory Latency: Fundamental Tradeoffs
Review: Memory Latency Lags Behind

Memory latency remains almost constant
DRAM Latency Is Critical for Performance

In-memory Databases
[Mao+, EuroSys’12; Clapp+ (Intel), IISWC’15]

Graph/Tree Processing
[Xu+, IISWC’12; Umuroglu+; FPL’15]

In-Memory Data Analytics
[Clapp+ (Intel), IISWC’15; Awan+, BDCloud’15]

Datacenter Workloads
[Kanew+ (Google), ISCA’15]
DRAM Latency Is Critical for Performance

Long memory latency → performance bottleneck
The Memory Latency Problem

- High memory latency is a significant limiter of system performance and energy-efficiency.

- It is becoming increasingly so with higher memory contention in multi-core and heterogeneous architectures:
  - Exacerbating the bandwidth need
  - Exacerbating the QoS problem

- It increases processor design complexity due to the mechanisms incorporated to tolerate memory latency.
Retrospective: Conventional Latency Tolerance Techniques

- Caching [initially by Wilkes, 1965]
  - Widely used, simple, effective, but inefficient, passive
  - Not all applications/phases exhibit temporal or spatial locality

- Prefetching [initially in IBM 360/91, 1967]
  - Works well for regular memory access patterns
  - Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive

- Multithreading [initially in CDC 6600, 1964]
  - Works well if there are multiple threads
  - Improving single thread performance using multithreading hardware is an ongoing research effort

- Out-of-order execution [initially by Tomasulo, 1967]
  - Tolerates cache misses that cannot be prefetched
  - Requires extensive hardware resources for tolerating long latencies

None of These Fundamentally Reduce Memory Latency
Two Major Sources of Latency Inefficiency

- Modern DRAM is not designed for low latency
  - Main focus is cost-per-bit (capacity)

- Modern DRAM latency is determined by worst case conditions and worst case devices
  - Much of memory latency is unnecessary

Our Goal: Reduce Memory Latency at the Source of the Problem
What Causes the Long Memory Latency?
Why the Long Memory Latency?

- **Reason 1: Design of DRAM Micro-architecture**
  - Goal: Maximize capacity/area, not minimize latency

- **Reason 2: “One size fits all” approach to latency specification**
  - Same latency parameters for all temperatures
  - Same latency parameters for all DRAM chips (e.g., rows)
  - Same latency parameters for all parts of a DRAM chip
  - Same latency parameters for all supply voltage levels
  - Same latency parameters for all application data
  - ...

SAFARI
Tackling the Fixed Latency Mindset

- Reliable operation latency is actually very heterogeneous
  - Across temperatures, chips, parts of a chip, voltage levels, ...

- Idea: **Dynamically find out and use the lowest latency one can reliably access a memory location with**
  - Adaptive-Latency DRAM [HPCA 2015]
  - Flexible-Latency DRAM [SIGMETRICS 2016]
  - Design-Induced Variation-Aware DRAM [SIGMETRICS 2017]
  - Voltron [SIGMETRICS 2017]
  - DRAM Latency PUF [HPCA 2018]
  - ...

- We would like to find sources of latency heterogeneity and exploit them to minimize latency
Latency Variation in Memory Chips

Heterogeneous manufacturing & operating conditions → latency variation in timing parameters
Why is Latency High?

- **DRAM latency**: Delay as specified in DRAM standards
  - Doesn’t reflect true DRAM device latency
- Imperfect manufacturing process → latency variation
- **High standard latency** chosen to increase yield
What Causes the Long Memory Latency?

- **Conservative timing margins!**

- DRAM timing parameters are set to cover the worst case

- **Worst-case temperatures**
  - 85 degrees vs. common-case
  - to enable a wide range of operating conditions

- **Worst-case devices**
  - DRAM cell with smallest charge across any acceptable device
  - to tolerate process variation at acceptable yield

- This leads to large timing margins for the common case
Understanding and Exploiting Variation in DRAM Latency
DRAM Stores Data as Charge

Three steps of charge movement:
1. Sensing
2. Restore
3. Precharge
**DRAM Charge over Time**

**Why does DRAM need the extra timing margin?**
Two Reasons for Timing Margin

1. Process Variation
   - DRAM cells are not equal
   - Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence
DRAM Cells are Not Equal

Ideal

Real

Smallest Cell

Largest Cell

Large variation in cell size
Large variation in charge
Large variation in access latency
Process Variation

Small cell can store small charge

- Small cell capacitance
- High contact resistance
- Slow access transistor

High access latency
Two Reasons for Timing Margin

1. Process Variation
   - DRAM cells are not equal
   - Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence
   - DRAM leaks more charge at higher temperature
   - Leads to extra timing margin for cells that operate at low temperature
Charge Leakage Temperature

Cells store small charge at high temperature and large charge at low temperature → Large variation in access latency
DRAM Timing Parameters

• **DRAM timing parameters are dictated by the worst-case**
  – The smallest cell with the smallest charge in all DRAM products
  – Operating at the highest temperature

• **Large timing margin for the common-case**
Adaptive-Latency DRAM [HPCA 2015]

- **Idea:** Optimize DRAM timing for the common case
  - Current temperature
  - Current DRAM module

- Why would this reduce latency?
  - A DRAM cell can store much more charge in the common case (low temperature, strong cell) than in the worst case
  - More charge in a DRAM cell
    - Faster sensing, charge restoration, precharging
    - Faster access (read, write, refresh, ...)

Extra Charge $\rightarrow$ Reduced Latency

1. Sensing
   Sense **cells with extra charge** faster
   $\rightarrow$ **Lower sensing latency**

2. Restore
   No need to fully restore **cells with extra charge**
   $\rightarrow$ **Lower restoration latency**

3. Precharge
   No need to fully precharge bitlines for **cells with extra charge**
   $\rightarrow$ **Lower precharge latency**
DRAM Characterization Infrastructure


- Flexible
- Easy to Use (C++ API)
- Open-source

  
  github.com/CMU-SAFARI/SoftMC
SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

Hasan Hassan\textsuperscript{1,2,3} Nandita Vijaykumar\textsuperscript{3} Samira Khan\textsuperscript{4,3} Saugata Ghose\textsuperscript{3} Kevin Chang\textsuperscript{3} Gennady Pekhimenko\textsuperscript{5,3} Donghyuk Lee\textsuperscript{6,3} Oguz Ergin\textsuperscript{2} Onur Mutlu\textsuperscript{1,3}

\textsuperscript{1}ETH Zürich \quad \textsuperscript{2}TOBB University of Economics & Technology \quad \textsuperscript{3}Carnegie Mellon University \quad \textsuperscript{4}University of Virginia \quad \textsuperscript{5}Microsoft Research \quad \textsuperscript{6}NVIDIA Research
Observation 1. Faster Sensing

Typical DIMM at Low Temperature

More Charge
Strong Charge Flow
Faster Sensing

Timing (tRCD)
17% ↓
No Errors

115 DIMM Characterization

Typical DIMM at Low Temperature
⇒ More charge ⇒ Faster sensing
Observation 2. Reducing Restore Time

Typical DIMM at Low Temperature

Less Leakage ➔ Extra Charge
No Need to Fully Restore Charge

115 DIMM Characterization

Read ($t_{\text{RAS}}$)
37% ↓

Write ($t_{\text{WR}}$)
54% ↓
No Errors

Typical DIMM at lower temperature
➔ More charge ➔ Restore time reduction
AL-DRAM

- **Key idea**
  - Optimize DRAM timing parameters online

- **Two components**
  - DRAM manufacturer provides multiple sets of reliable DRAM timing parameters at different temperatures for each DIMM
  - System monitors DRAM temperature & uses appropriate DRAM timing parameters

DRAM Temperature

- **DRAM temperature measurement**
  - Server cluster: Operates at under 34°C
  - Desktop: Operates at under 50°C
  - **DRAM standard optimized for 85 °C**

**DRAM operates at low temperatures in the common-case**

- Previous works – Maintain low DRAM temperature
  - David+ ICAC 2011
  - Liu+ ISCA 2007
  - Zhu+ Itherm 2008
Latency Reduction Summary of 115 DIMMs

• Latency reduction for read & write (55°C)
  – Read Latency: 32.7%
  – Write Latency: 55.1%

• Latency reduction for each timing parameter (55°C)
  – Sensing: 17.3%
  – Restore: 37.3% (read), 54.8% (write)
  – Precharge: 35.2%
AL-DRAM: Real System Evaluation

- **System**
  - **CPU**: AMD 4386 (8 Cores, 3.1GHz, 8MB LLC)

---

### D18F2x200 dct[0]_mp[1:0] DDR3 DRAM Timing 0

Reset: 0F05_0505h. See 2.9.3 [DCT Configuration Registers].

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:30</td>
<td>Reserved.</td>
</tr>
<tr>
<td>29:24</td>
<td><strong>Tras:</strong> row active strobe. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from an activate command to a precharge command, both to the same chip select bank.</td>
</tr>
<tr>
<td>23:21</td>
<td>Reserved.</td>
</tr>
<tr>
<td>20:16</td>
<td><strong>Trp:</strong> row precharge time. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from a precharge command to an activate command or auto refresh command, both to the same bank.</td>
</tr>
</tbody>
</table>

- 07h-00h: Reserved
- 2Ah-08h: <Tras> clocks
- 3Fh-2Bh: Reserved
AL-DRAM: Single-Core Evaluation

AL-DRAM improves performance on a real system
AL-DRAM provides higher performance for multi-programmed & multi-threaded workloads
Reducing Latency Also Reduces Energy

- AL-DRAM reduces DRAM power consumption by 5.8%
- Major reason: reduction in row activation time
AL-DRAM: Advantages & Disadvantages

- **Advantages**
  + Simple mechanism to reduce latency
  + Significant system performance and energy benefits
    + Benefits higher at low temperature
  + Low cost, low complexity

- **Disadvantages**
  - Need to determine reliable operating latencies for different temperatures and different DIMMs → higher testing cost
    (might not be that difficult for low temperatures)
More on AL-DRAM

Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu,
"Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case"
[Slides (pptx) (pdf)] [Full data sets]
Different Types of Latency Variation

- AL-DRAM exploits latency variation
  - Across time (different temperatures)
  - Across chips

- Is there also latency variation within a chip?
  - Across different parts of a chip
Variation in Activation Errors

Modern DRAM chips exhibit significant variation in activation latency.

Results from 7500 rounds over 240 chips:

- No ACT Errors
- Many errors
- Very few errors
- Rife w/ errors

Modern DRAM chips exhibit significant variation in activation latency.

Bit Error Rate (BER)

- $10^{-10}$
- $10^{-9}$
- $10^{-8}$
- $10^{-7}$
- $10^{-6}$
- $10^{-5}$
- $10^{-4}$
- $10^{-3}$
- $10^{-2}$
- $10^{-1}$
- $10^0$

tRCD (ns)

- 1.3 ns
- 2.5
- 5.0
- 7.5
- 10.0
- 12.5

Standard
Spatial Locality of Activation Errors

Activation errors are concentrated at certain columns of cells
Mechanism to Reduce DRAM Latency

• Observation: DRAM timing errors (slow DRAM cells) are concentrated on certain regions

• Flexible-LatencY (FLY) DRAM
  – A software-transparent design that reduces latency

• Key idea:
  1) Divide memory into regions of different latencies
  2) Memory controller: Use lower latency for regions without slow cells; higher latency for other regions

FLY-DRAM Configurations

Results

FLY-DRAM improves performance by exploiting spatial latency variation in DRAM

FLY-DRAM: Advantages & Disadvantages

- **Advantages**
  - Reduces latency significantly
  - Exploits significant within-chip latency variation

- **Disadvantages**
  - Need to determine reliable operating latencies for different parts of a chip → higher testing cost
  - Slightly more complicated controller
Analysis of Latency Variation in DRAM Chips

Kevin Chang, Abhijith Kashyap, Hasan Hassan, Samira Khan, Kevin Hsieh, Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Tianshi Li, and Onur Mutlu,

"Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"
[Slides (pptx) (pdf)]
[Source Code]
Why Is There
Spatial Latency Variation
Within a Chip?
What Is Design-Induced Variation?

Systematic variation in cell access times caused by the physical organization of DRAM.
DIVA Online Profiling

Profile only slow regions to determine min. latency → Dynamic & low cost latency optimization
**DIVA Online Profiling**

Design-Induced-Variation-Aware

- slow cells
- process variation
- random error

- inherently slow design-induced variation
- localized error

- error-correcting code

Combine error-correcting codes & online profiling → Reliably reduce DRAM latency
DIVA-DRAM reduces latency more aggressively and uses ECC to correct random slow cells.
DIVA-DRAM: Advantages & Disadvantages

- **Advantages**
  
  ++ Automatically finds the lowest reliable operating latency at system runtime (lower production-time testing cost)
  
  + Reduces latency more than prior methods (w/ ECC)
  
  + Reduces latency at high temperatures as well

- **Disadvantages**

  - Requires knowledge of inherently-slow regions
  
  - Requires ECC (Error Correcting Codes)
  
  - Imposes overhead during runtime profiling
Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms

Donghyuk Lee, NVIDIA and Carnegie Mellon University
Samira Khan, University of Virginia
Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Carnegie Mellon University
Gennady Pekhimenko, Vivek Seshadri, Microsoft Research
Onur Mutlu, ETH Zürich and Carnegie Mellon University

Understanding & Exploiting the Voltage-Latency-Reliability Relationship
High DRAM Power Consumption

- **Problem**: High DRAM (memory) power in today’s systems

>40% in POWER7 (Ware+, HPCA’10) >40% in GPU (Paul+, ISCA’15)
Low-Voltage Memory

• Existing DRAM designs to help reduce DRAM power by lowering supply voltage conservatively
  – Power $\propto$ Voltage$^2$

• DDR3L (low-voltage) reduces voltage from 1.5V to 1.35V (-10%)

• LPDDR4 (low-power) employs low-power I/O interface with 1.2V (lower bandwidth)

Can we reduce DRAM power and energy by further reducing supply voltage?
Goals

1. Understand and characterize the various characteristics of DRAM under *reduced voltage*

2. Develop a mechanism that reduces DRAM energy by *lowering voltage* while keeping performance loss within a target
Key Questions

• How does reducing voltage affect reliability (errors)?

• How does reducing voltage affect DRAM latency?

• How do we design a new DRAM energy reduction mechanism?
Supply Voltage Control on DRAM

Adjust the supply voltage to every chip on the same module

Supply Voltage
Custom Testing Platform

**SoftMC** [Hassan+, HPCA’17]: FPGA testing platform to

1) Adjust supply voltage to DRAM modules
2) Schedule DRAM commands to DRAM modules

Existing systems: DRAM commands not exposed to users

https://github.com/CMU-SAFARI/DRAM-Voltage-Study
Tested DRAM Modules

- **124 DDR3L** (low-voltage) DRAM chips
  - 31 SO-DIMMs
  - **1.35V** (DDR3 uses 1.5V)
  - Density: 4Gb per chip
  - Three major vendors/manufacturers

- Iteratively read every bit in each 4Gb chip under a wide range of supply voltage levels: 1.35V to 1.0V (-26%)
Reliability Worsens with Lower Voltage

Reducing voltage below $V_{\text{min}}$ causes an increasing number of errors.

Errors induced by reduced-voltage operation.

Fraction of Cache Lines with Errors (%)

- Vendor A
- Vendor B
- Vendor C

Min. voltage ($V_{\text{min}}$) without errors

Nominal Voltage

SAFARI
Source of Errors

Detailed circuit simulations (SPICE) of a DRAM cell array to model the behavior of DRAM operations

https://github.com/CMU-SAFARI/DRAM-Voltage-Study

Reliable low-voltage operation requires higher latency
DIMMs Operating at Higher Latency

Measured minimum latency that does not cause errors in DRAM modules

DRAM requires longer latency to access data without errors at lower voltage.
Spatial Locality of Errors

A module under 1.175V (12% voltage reduction)

Errors concentrate in certain regions
Summary of Key Experimental Observations

• **Voltage-induced errors** increase as voltage reduces further below $V_{\text{min}}$

• Errors exhibit **spatial locality**

• **Increasing the latency** of DRAM operations mitigates voltage-induced errors
DRAM Voltage Adjustment to Reduce Energy

- **Goal**: Exploit the trade-off between voltage and latency to reduce energy consumption

- **Approach**: Reduce DRAM voltage **reliably**
  - Performance loss due to increased latency at lower voltage

![Graph showing performance and power savings with different supply voltages.](image)
Voltron Overview

How do we predict performance loss due to increased latency under low DRAM voltage?

User specifies the **performance loss target**

Select the **minimum** DRAM voltage without violating the target
Linear Model to Predict Performance

Voltron

User specifies the performance loss target

Select the minimum DRAM voltage without violating the target

Application’s characteristics

[1.3V, 1.25V, ...] DRAM Voltage

Linear regression model

[-1%, -3%, ...] Predicted performance loss

Min. Voltage Target

Final Voltage
Regression Model to Predict Performance

• Application’s characteristics for the model:
  – **Memory intensity**: Frequency of last-level cache misses
  – **Memory stall time**: Amount of time memory requests stall commit inside CPU

• Handling multiple applications:
  – Predict a performance loss for each application
  – Select the minimum voltage that satisfies the performance target for all applications
Comparison to Prior Work

- **Prior work**: Dynamically scale frequency and voltage of the entire DRAM based on bandwidth demand [David+, ICAC’11]
  - **Problem**: Lowering voltage on the peripheral circuitry decreases channel frequency (memory data throughput)
- **Voltron**: Reduce voltage to only DRAM array without changing the voltage to peripheral circuitry
Exploiting Spatial Locality of Errors

Key idea: Increase the latency only for DRAM banks that observe errors under low voltage

– **Benefit**: Higher performance
Voltron Evaluation Methodology

- **Cycle-level simulator:** Ramulator [CAL’15]
  - McPAT and DRAMPower for energy measurement
    
    https://github.com/CMU-SAFARI/ramulator

- **4-core** system with DDR3L memory

- **Benchmarks:** SPEC2006, YCSB

- **Comparison to prior work:** MemDVFS [David+, ICAC’11]
  - Dynamic DRAM frequency and voltage scaling
  - Scaling based on the memory bandwidth consumption
Energy Savings with Bounded Performance

MemDVFS | Voltron
[David+, ICAC’11]

More savings for high bandwidth applications

CPU+DRAM

Energy Savings (%)

Performance Loss (%)

Low | High

Memory Intensity

3.2% | 7.3%

-1.6% | -1.8%

Meets performance target
Voltron: Advantages & Disadvantages

- **Advantages**
  + Can trade-off between voltage and latency to improve energy or performance
  + Can exploit the high voltage margin present in DRAM

- **Disadvantages**
  - Requires finding the reliable operating voltage for each chip \(\rightarrow\) higher testing cost
Analysis of Latency-Voltage in DRAM Chips

Kevin Chang, A. Giray Yaglikci, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu,
"Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms"

Understanding Reduced-Voltage Operation in Modern DRAM Chips: Characterization, Analysis, and Mechanisms

Kevin K. Chang† Abdullah Giray Yaşlıkçı† Saugata Ghose† Aditya Agrawal¶ Niladrish Chatterjee¶
Abhijith Kashyap† Donghyuk Lee¶ Mike O’Connor¶,‡ Hasan Hassan§ Onur Mutlu§,†
†Carnegie Mellon University ‡NVIDIA ‡The University of Texas at Austin §ETH Zürich
And, What If …

- … we can sacrifice reliability of some data to access it with even lower latency?
The DRAM Latency PUF:
Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim  Minesh Patel
Hasan Hassan    Onur Mutlu
Motivation

• A PUF is function that generates a signature unique to a given device

• Used in a Challenge-Response Protocol
  - Each device generates a unique PUF response depending the inputs
  - A trusted server authenticates a device if it generates the expected PUF response
DRAM Latency Characterization of 223 LPDDR4 DRAM Devices

• Latency failures come from accessing DRAM with reduced timing parameters.

• Key Observations:
  1. A cell’s latency failure probability is determined by random process variation
  2. Latency failure patterns are repeatable and unique to a device
DRAM Latency PUF Key Idea

High % chance to fail with reduced $t_{RCD}$

Low % chance to fail with reduced $t_{RCD}$
Process variation during manufacturing leads to cells having unique characteristics.

Bitline Charge Sharing

Ready to Access Voltage Level

ACTIVATE
SA Enable
READ

$V_{dd}$

$V_{min}$
DRAM Accesses and Failures

- **Bitline Voltage**
  - $V_{dd}$
  - $V_{min}$
  - $0.5\, V_{dd}$

- **Ready to Access Voltage Level**

- **weaker cells have a higher probability to fail**

- **Process Phases**
  - ACTIVATE
  - SA Enable
  - READ

- **Time**
  - $t_{RCD}$

**SAFARI**
The DRAM Latency PUF Evaluation

• We generate PUF responses using latency errors in a region of DRAM

• The latency error patterns satisfy PUF requirements

• The DRAM Latency PUF generates PUF responses in 88.2ms
Results

• DL-PUF is **orders of magnitude faster** than prior DRAM PUFs!
The DRAM Latency PUF:
Quickly Evaluating Physical Unclonable Functions
by Exploiting the Latency-Reliability Tradeoff
in Modern Commodity DRAM Devices

Jeremie S. Kim    Minesh Patel
Hasan Hassan      Onur Mutlu

QR Code for the paper

HPCA 2018

Systems@ETH Zürich

ETH Zürich

Carnegie Mellon
DRAM Latency PUFs

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"


[Lightning Talk Video]
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]

The DRAM Latency PUF:
Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices

Jeremie S. Kim†§ Minesh Patel§ Hasan Hassan§ Onur Mutlu§†
†Carnegie Mellon University §ETH Zürich
Reducing Refresh Latency
On Reducing Refresh Latency

- Anup Das, Hasan Hassan, and Onur Mutlu, "VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency"
  Proceedings of the 55th Design Automation Conference (DAC), San Francisco, CA, USA, June 2018.

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
Why the Long Memory Latency?

- **Reason 1: Design of DRAM Micro-architecture**
  - Goal: Maximize capacity/area, not minimize latency

- **Reason 2: “One size fits all” approach to latency specification**
  - Same latency parameters for all **temperatures**
  - Same latency parameters for all **DRAM chips (e.g., rows)**
  - Same latency parameters for all **parts of a DRAM chip**
  - Same latency parameters for all **supply voltage levels**
  - Same latency parameters for all **application data**
  - ...
Tiered Latency DRAM
What Causes the Long Latency?

DRAM Latency = Subarray Latency + I/O Latency
Why is the Subarray So Slow?

- **Long bitline**
  - Amortizes sense amplifier cost → Small area
  - Large bitline capacitance → High latency & power
Trade-Off: Area (Die Size) vs. Latency

Long Bitline

Short Bitline

Faster

Smaller

Trade-Off: Area vs. Latency
Trade-Off: Area (Die Size) vs. Latency

- **Fancy DRAM Short Bitline**
- **Commodity DRAM Long Bitline**
- **GOAL**: 512 cells/bitline

- Cheaper
- Faster
Approximating the Best of Both Worlds

<table>
<thead>
<tr>
<th>Long Bitline</th>
<th>Our Proposal</th>
<th>Short Bitline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Area</td>
<td>Add Isolation Transistors</td>
<td>Large Area</td>
</tr>
<tr>
<td>High Latency</td>
<td>Fast</td>
<td>Low Latency</td>
</tr>
</tbody>
</table>

Need Isolation

Add Isolation Transistors
Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM Short Bitline

Small Area

High Latency

Low Latency

Small Area

Large Area

Low Latency

Low Latency

Small area using long bitline

Low Latency
Commodity DRAM vs. TL-DRAM [HPCA 2013]

- DRAM Latency ($t_{RC}$) • DRAM Power

- DRAM Area Overhead

~3%: mainly due to the isolation transistors
Trade-Off: Area (Die-Area) vs. Latency

- Cheaper
- Normalized DRAM Area
- Latency (ns)
- 64 cells/bitline
- 32 cells/bitline
- 128 cells/bitline
- 256 cells/bitline
- 512 cells/bitline

- Near Segment
- Far Segment

GOAL

Faster
Leveraging Tiered-Latency DRAM

- TL-DRAM is a **substrate** that can be leveraged by the hardware and/or software

- Many potential uses

  1. Use near segment as hardware-managed **inclusive** cache to far segment
  2. Use near segment as hardware-managed **exclusive** cache to far segment
  3. Profile-based page mapping by operating system
  4. Simply replace DRAM with TL-DRAM

Near Segment as Hardware-Managed Cache

- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?
Inter-Segment Migration

• **Goal**: Migrate source row into destination row
• **Naïve way**: Memory controller reads the source row *byte by byte* and writes to destination row *byte by byte* → High latency
Inter-Segment Migration

• Our way:
  – Source and destination cells *share bitlines*
  – Transfer data from source to destination across *shared bitlines* concurrently

![Diagram of inter-segment migration with Far Segment, Near Segment, Isolation Transistor, and Sense Amplifier.]
Inter-Segment Migration

• Our way:
  – Source and destination cells *share bitlines*
  – Transfer data from source to destination *shared bitlines* concurrently

Migration is overlapped with source row access
Additional ~4ns over row access latency

Step 1: Activate source row
Step 2: Activate destination row to connect cell and bitline
Near Segment as Hardware-Managed Cache

- **Challenge 1:** How to efficiently migrate a row between segments?
- **Challenge 2:** How to efficiently manage the cache?
Using near segment as a cache improves performance and reduces power consumption

By adjusting the near segment length, we can trade off cache capacity for cache latency.
More on TL-DRAM

- Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu,

"Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture"

Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013. Slides (pptx)
LISA: Low-Cost Inter-Linked Subarrays
[HPCA 2016]
Problem: Inefficient Bulk Data Movement

Bulk data movement is a key operation in many applications

– `memmove` & `memcpy`: 5% cycles in Google’s datacenter [Kanev+ ISCA’15]

Long latency and high energy
Goal: Provide a new substrate to enable wide connectivity between subarrays
Key Idea and Applications

- **Low-cost Inter-linked subarrays (LISA)**
  - Fast bulk data movement between subarrays
  - Wide datapath via isolation transistors: 0.8% DRAM chip area

- **LISA is a versatile substrate → new applications**
  - **Fast bulk data copy**: Copy latency 1.363ms→0.148ms (9.2x) → 66% speedup, -55% DRAM energy
  - **In-DRAM caching**: Hot data access latency 48.7ns→21.5ns (2.2x) → 5% speedup
  - **Fast precharge**: Precharge latency 13.1ns→5.0ns (2.6x) → 8% speedup
New DRAM Command to Use LISA

Row Buffer Movement (RBM): Move a row of data in an activated row buffer to a precharged one

RBM transfers an entire row b/w subarrays
RBM Analysis

• The range of RBM depends on the DRAM design
  – Multiple RBMs to move data across > 3 subarrays

  Subarray 1
  Subarray 2
  Subarray 3

• Validated with SPICE using worst-case cells
  – NCSU FreePDK 45nm library

• 4KB data in 8ns (w/ 60% guardband)
  → 500 GB/s, 26x bandwidth of a DDR4-2400 channel

• 0.8% DRAM chip area overhead [O+ ISCA’14]
1. Rapid Inter-Subarray Copying (RISC)

- **Goal**: Efficiently copy a row across subarrays
- **Key idea**: Use *RBM* to form a new command sequence

1. Activate *src* row
2. *RBM* $SA_1 \rightarrow SA_2$

Reduces row-copy latency by 9.2x, DRAM energy by 48.1x
2. Variable Latency DRAM (VILLA)

- **Goal**: Reduce DRAM latency with low area overhead
- **Motivation**: Trade-off between area and latency

Long Bitline (DDRx)

Short Bitline (RLDRAM)

Shorter bitlines $\rightarrow$ faster activate and precharge time

High area overhead: >40%
2. Variable Latency DRAM (VILLA)

- **Key idea**: Reduce access latency of hot data via a heterogeneous DRAM design [Lee+ HPCA’13, Son+ ISCA’13]
- **VILLA**: Add fast subarrays as a cache in each bank

Challenge: VILLA cache requires frequent movement of data rows

Reduces hot data access latency by 2.2x at only 1.6% area overhead
3. Linked Precharge (LIP)

- **Problem:** The precharge time is limited by the strength of one precharge unit
- **Linked Precharge (LIP):** LISA precharges a subarray using multiple precharge units

Reduces precharge latency by 2.6x (43% guardband)
More on LISA

Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
[Slides (pptx) (pdf)]
[Source Code]

Low-Cost Inter-Linked Subarrays (LISA):
Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair*, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi*, and Onur Mutlu†
†Carnegie Mellon University  *Georgia Institute of Technology
Reducing Memory Latency by Exploiting Memory Access Patterns
ChargeCache: 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)
DRAM Charge over Time

- Cell
- Sense Amplifier
- Ready to Access
- Ready to Precharge
- Data 0
- Data 1
- tRCD
- tRAS
Accessing Highly-charged Rows

Ready to Access

Ready to Precharge

Cell

Sense-Amplifier

charge

time

Data 0

Data 1

Sensing

Restore

Precharge

R/W

PRE

tRAS

tRCD

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

<table>
<thead>
<tr>
<th>Fraction of Accesses</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>w4</th>
<th>w5</th>
<th>w6</th>
<th>w7</th>
<th>w8</th>
<th>w9</th>
<th>w10</th>
<th>w11</th>
<th>w12</th>
<th>w13</th>
<th>w14</th>
<th>w15</th>
<th>w16</th>
<th>w17</th>
<th>w18</th>
<th>w19</th>
<th>w20</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

88%—RLTL for eight-core workloads
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
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
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
ChargeCache improves single-core performance
Eight-core Performance

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

Speedup significantly improves multi-core performance
DRAM Energy Savings

ChargeCache reduces DRAM energy
More on ChargeCache

Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Vivek Seshadri, Donghyuk Lee, Oguz Ergin, and Onur Mutlu,
"ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality"
[Slides (pptx) (pdf)]
[Source Code]

ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality

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

†Carnegie Mellon University  *TOBB University of Economics & Technology
Summary: Low-Latency Memory
Summary: Tackling Long Memory Latency

- **Reason 1:** Design of DRAM Micro-architecture
  - Goal: Maximize capacity/area, not minimize latency

- **Reason 2:** “One size fits all” approach to latency specification
  - Same latency parameters for all temperatures
  - Same latency parameters for all DRAM chips (e.g., rows)
  - Same latency parameters for all parts of a DRAM chip
  - Same latency parameters for all supply voltage levels
  - Same latency parameters for all application data
  - ...
Challenge and Opportunity for Future

Fundamentally Low Latency Computing Architectures
On DRAM Power Consumption
VAMPIRE DRAM Power Model

[Abstract]

What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study

Saugata Ghose† Abdullah Giray Yaşlıkçı‡† Raghav Gupta† Donghyuk Lee§
Kais Kudrolli† William X. Liu† Hasan Hassan‡ Kevin K. Chang†
Niladrisih Chatterjee§ Aditya Agrawal§ Mike O’Connor§¶ Onur Mutlu‡†
†Carnegie Mellon University ‡ETH Zürich §NVIDIA ¶University of Texas at Austin
Conclusion
Agenda

- Brief Introduction
- A Motivating Example
- Memory System Trends
- What Will You Learn In This Course
  - And, how to make the best of it...
- Memory Fundamentals
- Key Memory Challenges and Solution Directions
  - Security, Reliability, Safety
  - Energy and Performance: Data-Centric Systems
  - Latency and Latency-Reliability Tradeoffs
- Summary and Future Lookout
Four Key Directions

- Fundamentally Secure/Reliable/Safe Architectures

- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures

- Fundamentally Low-Latency Architectures

- Architectures for Genomics, Medicine, Health
What Have We Learned In This Course?

- **Memory Systems and Memory-Centric Computing Systems**
  - July 9-13, 2018

  - **Topic 1:** Main Memory Trends and Basics
  - **Topic 2:** Memory Reliability & Security: RowHammer and Beyond
  - **Topic 3:** In-memory Computation
  - **Topic 4:** Low-Latency (and Low-Energy) Memory
  - **Topic 5** (unlikely): Enabling and Exploiting Non-Volatile Memory
  - **Topic 6** (unlikely): Flash Memory and SSD Scaling

- **Major Overview Reading:**
  - Mutlu and Subramaniam, “Research Problems and Opportunities in Memory Systems,” SUPERFRI 2014.
Some Solution Principles (So Far)

- More data-centric system design
  - Do not center everything around computation units

- Better cooperation across layers of the system
  - Careful co-design of components and layers: system/arch/device
  - Better, richer, more expressive and flexible interfaces

- Better-than-worst-case design
  - Do not optimize for the worst case
  - Worst case should not determine the common case

- Heterogeneity in design (specialization, asymmetry)
  - Enables a more efficient design (No one size fits all)
It Is Time to …

- ... design **principled system architectures** to solve the memory problem

- ... design complete systems to be balanced, high-performance, and energy-efficient, i.e., data-centric (or memory-centric)

- **make memory a key priority** in system design and optimize it & integrate it better into the system

- **This** can
  - Lead to **orders-of-magnitude** improvements
  - **Enable new applications & computing platforms**
  - **Enable better understanding of nature**
  - ...
We Need to Revisit the Entire Stack

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons
Course Materials and Beyond

- **Website for Course Slides and Papers**
  - https://people.inf.ethz.ch/omutlu/projects.htm
  - Final lecture notes and readings (for all topics)
You Can Contact Me Any Time

- **My Contact Information**
  - Onur Mutlu
  - [omutlu@gmail.com](mailto:omutlu@gmail.com)
  - [https://people.inf.ethz.ch/omutlu/index.html](https://people.inf.ethz.ch/omutlu/index.html)
  - +41-79-572-1444 (my cell phone)
  - You can contact me any time with questions and ideas.
Thank You!
Keep in Touch!
Memory Systems
and Memory-Centric Computing Systems
Lecture 5, Topic 4: Low-Latency Memory

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
13 July 2018
HiPEAC ACACES Summer School 2018
Readings, Videos, Reference Materials
Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND, RACHATA AUSAVARUNGNIRUN
Carnegie Mellon University

ONUR MUTLU
ETH Zürich and Carnegie Mellon University


Onur Mutlu and Lavanya Subramanian, "Research Problems and Opportunities in Memory Systems"

Invited Article in Supercomputing Frontiers and Innovations (SUPERFRI), 2014/2015.

Onur Mutlu,
"The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"


[Slides (pptx) (pdf)]

The RowHammer Problem
and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu
Onur Mutlu,
"Memory Scaling: A Systems Architecture Perspective"

Technical talk at MemCon 2013 (MEMCON), Santa Clara, CA, August 2013. [Slides (pptx) (pdf)] [Video] [Coverage on StorageSearch]

Memory Scaling: A Systems Architecture Perspective

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD’s reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642
Related Videos and Course Materials (I)

- Graduate Computer Architecture Course Lecture Videos (2017, 2015, 2013)
- Parallel Computer Architecture Course Materials (Lecture Videos)
Related Videos and Course Materials (II)

- **Memory Systems Short Course Materials**
  (Lecture Video on Main Memory and DRAM Basics)
Some Open Source Tools (I)

- **Rowhammer** – Program to Induce RowHammer Errors
  - [https://github.com/CMU-SAFARI/rowhammer](https://github.com/CMU-SAFARI/rowhammer)

- **Ramulator** – Fast and Extensible DRAM Simulator
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)

- **MemSim** – Simple Memory Simulator
  - [https://github.com/CMU-SAFARI/memsim](https://github.com/CMU-SAFARI/memsim)

- **NOCulator** – Flexible Network-on-Chip Simulator
  - [https://github.com/CMU-SAFARI/NOCulator](https://github.com/CMU-SAFARI/NOCULATOR)

- **SoftMC** – FPGA-Based DRAM Testing Infrastructure
  - [https://github.com/CMU-SAFARI/SoftMC](https://github.com/CMU-SAFARI/SoftMC)

- Other open-source software from my group
  - [https://github.com/CMU-SAFARI/](https://github.com/CMU-SAFARI/)
  - [http://www.ece.cmu.edu/~safari/tools.html](http://www.ece.cmu.edu/~safari/tools.html)
Some Open Source Tools (II)

- MQSim – A Fast Modern SSD Simulator
  - https://github.com/CMU-SAFARI/MQSim

- Mosaic – GPU Simulator Supporting Concurrent Applications
  - https://github.com/CMU-SAFARI/Mosaic

- IMPICA – Processing in 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFARI/IMPICA

- SMLA – Detailed 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFARI/SMLA

- HWASim – Simulator for Heterogeneous CPU-HWA Systems
  - https://github.com/CMU-SAFARI/HWASim

- Other open-source software from my group
  - https://github.com/CMU-SAFARI/
  - http://www.ece.cmu.edu/~safari/tools.html
More Open Source Tools (III)

- A lot more open-source software from my group
  - [https://github.com/CMU-SAFARI/](https://github.com/CMU-SAFARI/)
  - [http://www.ece.cmu.edu/~safari/tools.html](http://www.ece.cmu.edu/~safari/tools.html)
Referenced Papers

- All are available at

  https://people.inf.ethz.ch/omutlu/projects.htm

  http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]
Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory
Ramulator

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Req/sec (10^3)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Stream</td>
<td>Random</td>
<td>Stream</td>
</tr>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
</tr>
<tr>
<td>DRAMSim2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
</tr>
<tr>
<td>NVMMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>

Table 3. Comparison of five simulators using two traces
Case Study: Comparison of DRAM Standards

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate (MT/s)</th>
<th>Timing (CL-RCD-RP)</th>
<th>Data-Bus (Width x Chan.)</th>
<th>Rank-per-Chan</th>
<th>BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>DDR4</td>
<td>2,400</td>
<td>16-16-16</td>
<td>64-bit x 1</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>SALP†</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR3</td>
<td>1,600</td>
<td>12-15-15</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR4</td>
<td>2,400</td>
<td>22-22-22</td>
<td>32-bit x 2*</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>GDDR5 [12]</td>
<td>6,000</td>
<td>18-18-18</td>
<td>64-bit x 1</td>
<td>1</td>
<td>44.7</td>
</tr>
<tr>
<td>HBM</td>
<td>1,000</td>
<td>7-7-7</td>
<td>128-bit x 8*</td>
<td>1</td>
<td>119.2</td>
</tr>
<tr>
<td>WIO</td>
<td>266</td>
<td>7-7-7</td>
<td>128-bit x 4*</td>
<td>1</td>
<td>15.9</td>
</tr>
<tr>
<td>WIO2</td>
<td>1,066</td>
<td>9-10-10</td>
<td>128-bit x 8*</td>
<td>1</td>
<td>127.2</td>
</tr>
</tbody>
</table>

Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim\textsuperscript{1} Weikun Yang\textsuperscript{1,2} Onur Mutlu\textsuperscript{1}
\textsuperscript{1}Carnegie Mellon University \textsuperscript{2}Peking University