Processing Data Where It Makes Sense in Modern Computing Systems: Enabling In-Memory Computation

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
23 March 2018
DATE Emerging Memory Workshop Keynote Talk
Current Research Focus Areas

Research Focus: Computer architecture, HW/SW, bioinformatics, security

- Memory and storage (DRAM, flash, emerging), interconnects
- Heterogeneous & parallel systems, GPUs, systems for data analytics
- System/architecture interaction, new execution models, new interfaces
- Hardware security, energy efficiency, fault tolerance, performance
- Genome sequence analysis & assembly algorithms and architectures
- Biologically inspired systems & system design for bio/medicine

Broad research spanning apps, systems, logic with architecture at the center
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
GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies

Jeremie S. Kim\textsuperscript{1,6*}, Damla Senol Cali\textsuperscript{1}, Hongyi Xin\textsuperscript{2}, Donghyuk Lee\textsuperscript{3}, Saugata Ghose\textsuperscript{1}, Mohammed Alser\textsuperscript{4}, Hasan Hassan\textsuperscript{6}, Oguz Ergin\textsuperscript{5}, Can Alkan*\textsuperscript{4}, and Onur Mutlu*\textsuperscript{6,1}
New Genome Sequencing Technologies

Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks, and Future Directions

Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu

1 Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
2 Department of Computer Engineering, Bilkent University, Bilkent, Ankara, Turkey
3 Department of Computer Science, Systems Group, ETH Zürich, Zürich, Switzerland

Memory & Storage
Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor.

Main memory system must scale (in size, technology, efficiency, cost, and management algorithms) to maintain performance growth and technology scaling benefits.
Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor.

Main memory system must scale (in size, technology, efficiency, cost, and management algorithms) to maintain performance growth and technology scaling benefits.
Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor.

Main memory system must scale (in size, technology, efficiency, cost, and management algorithms) to maintain performance growth and technology scaling benefits.
Memory System: A *Shared Resource* View

Most of the system is dedicated to storing and moving data.
State of the Main Memory System

- Recent technology, architecture, and application trends
  - lead to new requirements
  - exacerbate old requirements

- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements

- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging

- We need to rethink the main memory system
  - to fix DRAM issues and enable emerging technologies
  - to satisfy all requirements
Major Trends Affecting Main Memory (I)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores/agents
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
Example: The Memory Capacity Gap

- Memory capacity per core expected to drop by 30% every two years
- Trends worse for memory bandwidth per core!
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
[Kanev+ (Google), ISCA’15]
DRAM Latency Is Critical for Performance

In-memory Databases

Graph/Tree Processing

Long memory latency → performance bottleneck

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

Datacenter Workloads
[Kanev+ (Google), ISCA’15]
Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer’03]
  - >40% power in DRAM [Ware, HPCA’10][Paul, ISCA’15]
  - DRAM consumes power even when not used (periodic refresh)
- DRAM technology scaling is ending
Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy
Major Trends Affecting Main Memory (V)

- DRAM scaling has already become increasingly difficult
  - Increasing cell leakage current, reduced cell reliability, increasing manufacturing difficulties [Kim+ ISCA 2014], [Liu+ ISCA 2013], [Mutlu IMW 2013], [Mutlu DATE 2017]
  - Difficult to significantly improve capacity, energy

- Emerging memory technologies are promising
Major Trends Affecting Main Memory (V)

- DRAM scaling has already become increasingly difficult
  - Increasing cell leakage current, reduced cell reliability, increasing manufacturing difficulties [Kim+ ISCA 2014], [Liu+ ISCA 2013], [Mutlu IMW 2013], [Mutlu DATE 2017]
  - **Difficult to significantly improve capacity, energy**

- **Emerging memory technologies** are promising

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Bandwidth</th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3D-Stacked DRAM</strong></td>
<td>higher bandwidth</td>
<td>smaller capacity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduced-Latency DRAM (e.g., RL/TL-DRAM, FLY-RAM)</td>
<td>lower latency</td>
<td>higher cost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low-Power DRAM (e.g., LPDDR3, LPDDR4, Voltron)</td>
<td>lower power</td>
<td>higher latency</td>
<td>higher cost</td>
<td></td>
</tr>
<tr>
<td>Non-Volatile Memory (NVM) (e.g., PCM, STTRAM, ReRAM, 3D Xpoint)</td>
<td>larger capacity</td>
<td>higher latency</td>
<td>higher dynamic power</td>
<td>lower endurance</td>
</tr>
</tbody>
</table>
Major Trend: Hybrid Main Memory

Yoon+, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.

CPU

DRAM

Fast, **durable**
Small, leaky, volatile, high-cost

PCM

Large, non-volatile, low-cost
Slow, **wears out**, high active energy

Hardware/software manage data allocation and movement to achieve the best of multiple technologies.
Foreshadowing

Main Memory Needs

Intelligent Controllers
Industry Is Writing Papers About It, Too

DRAM Process Scaling Challenges

- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance
  - Leakage current of cell access transistors increasing

- **tWR**
  - Contact resistance between the cell capacitor and access transistor increasing
  - On-current of the cell access transistor decreasing
  - Bit-line resistance increasing

- **VRT**
  - Occurring more frequently with cell capacitance decreasing
Call for Intelligent Memory Controllers

DRAM Process Scaling Challenges

- Refresh
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng, **John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

Samsung Electronics, Hwasung, Korea / *Samsung Electronics, San Jose / **Intel
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
Maslow’s (Human) Hierarchy of Needs


- We need to start with reliability and security...
The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]

- DRAM capacity, cost, and energy/power hard to scale
As Memory Scales, It Becomes Unreliable

- Data from all of Facebook’s servers worldwide
- Meza+, “Revisiting Memory Errors in Large-Scale Production Data Centers,” DSN’15.

**Graph:**

- **Y-axis:** Relative server failure rate
- **X-axis:** Chip density (Gb)
- **Intuition:** quadratic increase in capacity
Large-Scale Failure Analysis of DRAM Chips

- Analysis and modeling of memory errors found in all of Facebook’s server fleet

- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field" Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015. [Slides (pptx) (pdf)] [DRAM Error Model]
Infrastructures to Understand Such Issues

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)
Infrastructures to Understand Such Issues

SoftMC: Open Source DRAM Infrastructure


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

github.com/CMU-SAFARI/SoftMC
SoftMC

- https://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
\textsuperscript{4}University of Virginia \quad \textsuperscript{5}Microsoft Research \quad \textsuperscript{6}NVIDIA Research
Data Retention in Memory [Liu et al., ISCA 2013]

- Retention Time Profile of DRAM looks like this:

  - Location dependent
  - Stored value pattern dependent
  - Time dependent

- 64-128ms
- >256ms
- 128-256ms

Location dependent
Stored value pattern dependent
Time dependent
One can predictably induce errors in most DRAM memory chips
A simple hardware failure mechanism can create a widespread system security vulnerability.
Modern DRAM is Prone to Disturbance Errors

Repeatedly reading a row enough times (before memory gets refreshed) induces **disturbance errors** in adjacent rows in most real DRAM chips you can buy today.

*Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors*, (Kim et al., ISCA 2014)
Most DRAM Modules Are Vulnerable

A company

86%
(37/43)

Up to
1.0×10^7
errors

B company

83%
(45/54)

Up to
2.7×10^6
errors

C company

88%
(28/32)

Up to
3.3×10^5
errors

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)
Recent DRAM Is More Vulnerable
Recent DRAM Is More Vulnerable
Recent DRAM Is More Vulnerable

All modules from 2012–2013 are vulnerable
One Can Take Over an Otherwise-Secure System

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology...

Project Zero

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
Security Implications

It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after.
More Security Implications

“We can gain unrestricted access to systems of website visitors.”

Not there yet, but ...

ROOT privileges for web apps!

Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript (DIMVA’16)

Source: https://lab.dsst.io/32c3-slides/7197.html
More Security Implications

“Can gain control of a smart phone deterministically”
More Security Implications?
More on RowHammer Analysis

- Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu,
"Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors"
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Source Code and Data]
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
Call for Intelligent Memory Controllers

DRAM Process Scaling Challenges

- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance.

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng, **John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

Samsung Electronics, Hwasung, Korea / *Samsung Electronics, San Jose / **Intel
Aside: Intelligent Controller for NAND Flash

HAPS-52 Mother Board
USB Daughter Board
Virtex-V FPGA (NAND Controller)
USB Jack

Aside: Intelligent Controller for NAND Flash

Proceedings of the IEEE, Sept. 2017

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
Takeaway

Main Memory Needs

Intelligent Controllers
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
Three Key Systems Trends

1. Data access is a major bottleneck
   - Applications are increasingly data hungry

2. Energy consumption is a key limiter

3. Data movement energy dominates compute
   - Especially true for off-chip to on-chip movement
The Need for More Memory 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**
  - [Kanev+ (Google), ISCA’15]
The Performance Perspective (1996-2005)

- “It’s the Memory, Stupid!” (Richard Sites, MPR, 1996)

The Performance Perspective

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"

Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

- Onur Mutlu § Jared Stark † Chris Wilkerson ‡ Yale N. Patt §

§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

†Microprocessor Research
Intel Labs
jared.w.stark@intel.com

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com
The Performance Perspective (Today)

- All of Google’s Data Center Workloads (2015):

The Performance Perspective (Today)

- All of Google’s Data Center Workloads (2015):

![Box plot graph showing cache-bound cycles (%)](image)

**Figure 11**: Half of cycles are spent stalled on caches.

The Energy Perspective

Communication Dominates Arithmetic

Dally, HiPEAC 2015
A memory access consumes \( \sim 1000X \) the energy of a complex addition.
Data Movement vs. Computation Energy

- **Data movement** is a major system energy bottleneck
  - Comprises 41% of mobile system energy during web browsing [2]
  - Costs ~115 times as much energy as an ADD operation [1, 2]

[1]: Reducing data Movement Energy via Online Data Clustering and Encoding (MICRO’16)
[2]: Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms (IISWC’14)
Challenge and Opportunity for Future

High Performance and Energy Efficient
Maslow’s (Human) Hierarchy of Needs, Revisited


The Problem

Data access is the major performance and energy bottleneck

Our current design principles cause great energy waste (and great performance loss)
The Problem

Processing of data is performed far away from the data.
A Computing System

- Three key components
  - Computation
  - Communication
  - Storage/memory

Burks, Goldstein, von Neumann, “Preliminary discussion of the logical design of an electronic computing instrument,” 1946.
A Computing System

- Three key components
  - Computation
  - Communication
  - Storage/memory

Burks, Goldstein, von Neumann, “Preliminary discussion of the logical design of an electronic computing instrument,” 1946.

Today’s Computing Systems

- Are overwhelmingly processor centric
- All data processed in the processor $\rightarrow$ at great system cost
- Processor is heavily optimized and is considered the master
- Data storage units are dumb and are largely unoptimized (except for some that are on the processor die)
Yet …

- “It’s the Memory, Stupid!” (Richard Sites, MPR, 1996)

Perils of Processor-Centric Design

- Grossly-imbalanced systems
  - Processing done only in **one place**
  - Everything else just stores and moves data: **data moves a lot**
    - Energy inefficient
    - Low performance
    - Complex

- Overly complex and bloated processor (and accelerators)
  - To tolerate data access from memory
  - Complex hierarchies and mechanisms
    - Energy inefficient
    - Low performance
    - Complex
Most of the system is dedicated to storing and moving data
We Do Not Want to Move Data!

A memory access consumes $\sim 1000X$ the energy of a complex addition.
62.7% of the total system energy is spent on data movement.

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand\textsuperscript{1}  
Rachata Ausavarungnirun\textsuperscript{1}  
Aki Kuusela\textsuperscript{3}  
Allan Knies\textsuperscript{3}  
Saugata Ghose\textsuperscript{1}  
Eric Shiu\textsuperscript{3}  
Rahul Thakur\textsuperscript{3}  
Parthasarathy Ranganathan\textsuperscript{3}  
Youngsok Kim\textsuperscript{2}  
Daehyun Kim\textsuperscript{4,3}  
Onur Mutlu\textsuperscript{5,1}
We Need A Paradigm Shift To …

- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric
Goal: Processing Inside Memory

Many questions ... How do we design the:
- compute-capable memory & controllers?
- processor chip?
- software and hardware interfaces?
- system software and languages?
- algorithms?
Why In-Memory Computation Today?

- Push from Technology
  - DRAM Scaling at jeopardy
    - Controllers close to DRAM...
  - Industry open to new memory architectures

- Pull from Systems and Applications
  - Data access is a major system and application bottleneck
  - Systems are energy limited
  - Data movement much more energy-hungry than computation
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
Processing in Memory: Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory
Approach 1: Minimally Changing DRAM

- DRAM has great capability to perform bulk data movement and computation internally with small changes
  - Can exploit internal connectivity to move data
  - Can exploit analog computation capability
  - ...

Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM

- RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
- Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
- Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)
- "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology” (Seshadri et al., MICRO 2017)
Starting Simple: Data Copy and Initialization

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

Forking

Zero initialization (e.g., security)

Checkpointing

VM Cloning

Deduplication

Page Migration

Many more
Today’s Systems: Bulk Data Copy

1) High latency
2) High bandwidth utilization
3) Cache pollution
4) Unwanted data movement

1046ns, 3.6uJ (for 4KB page copy via DMA)
Future Systems: In-Memory Copy

1) Low latency
2) Low bandwidth utilization
3) No cache pollution
4) No unwanted data movement

1046ns, 3.6uJ → 90ns, 0.04uJ
RowClone: In-DRAM Row Copy

Idea: Two consecutive ACTivates
Negligible HW cost

Step 1: Activate row A
Transfer row

Step 2: Activate row B
Transfer row

Data Bus

4 Kbytes

8 bits

DRAM subarray

Row Buffer (4 Kbytes)
RowClone: Latency and Energy Savings

More on RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,

"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"

Proceedings of the 46th International Symposium on Microarchitecture (MICRO), Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri vseshadr@cs.cmu.edu
Yoongu Kim yoongukim@cmu.edu
Chris Fallin* cfallin@c1f.net
Donghyuk Lee donghyuk1@cmu.edu
Rachata Ausavarungnirun rachata@cmu.edu
Gennady Pekhimenko gpekhime@cs.cmu.edu
Yixin Luo yixinluo@andrew.cmu.edu
Onur Mutlu onur@cmu.edu
Phillip B. Gibbons† phillip.b.gibbons@intel.com
Michael A. Kozuch† michael.a.kozuch@intel.com
Todd C. Mowry tcm@cs.cmu.edu

Carnegie Mellon University †Intel Pittsburgh
Memory as an Accelerator

CPU core
CPU core
mini-CPU core
video core
imaging core
GPU (throughput) core
GPU (throughput) core
GPU (throughput) core
GPU (throughput) core
LLC
Memory Controller
Memory Bus
Memory similar to a “conventional” accelerator

Specialized compute-capability in memory
In-Memory Bulk Bitwise Operations

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement

- New memory technologies enable even more opportunities
  - Memristors, resistive RAM, phase change mem, STT-MRAM, ...
  - Can operate on data with minimal movement
In-DRAM AND/OR: Triple Row Activation

Final State

$$AB + BC + AC$$

$$C(A + B) + \sim C(AB)$$

In-DRAM NOT: Dual Contact Cell

Idea:
Feed the negated value in the sense amplifier into a special row

Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Performance: In-DRAM Bitwise Operations

Figure 9: Throughput of bitwise operations on various systems.

Energy of In-DRAM Bitwise Operations

<table>
<thead>
<tr>
<th>Design</th>
<th>not</th>
<th>and/or</th>
<th>nand/nor</th>
<th>xor/xnor</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>93.7</td>
<td>137.9</td>
<td>137.9</td>
<td>137.9</td>
</tr>
<tr>
<td>Ambit</td>
<td>1.6</td>
<td>3.2</td>
<td>4.0</td>
<td>5.5</td>
</tr>
<tr>
<td>↓</td>
<td>59.5X</td>
<td>43.9X</td>
<td>35.1X</td>
<td>25.1X</td>
</tr>
</tbody>
</table>

Table 3: Energy of bitwise operations. (↓) indicates energy reduction of Ambit over the traditional DDR3-based design.

Ambit vs. DDR3: Performance and Energy

- Performance Improvement
- Energy Reduction

Bulk Bitwise Operations in Workloads

- Bitmap indices (database indexing)
- Set operations
- Encryption algorithms
- BitWeaving (database queries)
- BitFunnel (web search)
- DNA sequence mapping

[1] Li and Patel, BitWeaving, SIGMOD 2013
Example Data Structure: Bitmap Index

- Alternative to B-tree and its variants
- Efficient for performing range queries and joins
- Many bitwise operations to perform a query

age < 18 18 < age < 25 25 < age < 60 age > 60

Bitmap 1 Bitmap 2 Bitmap 3 Bitmap 4
Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

Performance: BitWeaving on Ambit

Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving

More on In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,

"Fast Bulk Bitwise AND and OR in DRAM"


---

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*, Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

* Carnegie Mellon University      † Intel Pittsburgh
More on Ambit


Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri\(^{1,5}\) Donghyuk Lee\(^{2,5}\) Thomas Mullins\(^{3,5}\) Hasan Hassan\(^{4}\) Amirali Boroumand\(^{5}\)
Jeremie Kim\(^{4,5}\) Michael A. Kozuch\(^{3}\) Onur Mutlu\(^{4,5}\) Phillip B. Gibbons\(^{5}\) Todd C. Mowry\(^{5}\)

\(^{1}\)Microsoft Research India \(^{2}\)NVIDIA Research \(^{3}\)Intel \(^{4}\)ETH Zürich \(^{5}\)Carnegie Mellon University
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
Challenge: Intelligent Memory Device

Does memory have to be dumb?
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
Opportunity: 3D-Stacked Logic+Memory

Other “True 3D” technologies under development
### DRAM Landscape (circa 2015)

<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

Two Key Questions in 3D-Stacked PIM

- How can we accelerate important applications if we use 3D-stacked memory as a coarse-grained accelerator?
  - what is the architecture and programming model?
  - what are the mechanisms for acceleration?

- What is the minimal processing-in-memory support we can provide?
  - without changing the system significantly
  - while achieving significant benefits
Graph Processing

- Large graphs are everywhere (circa 2015)

- Scalable large-scale graph processing is challenging

36 Million Wikipedia Pages
1.4 Billion Facebook Users
300 Million Twitter Users
30 Billion Instagram Photos

32 Cores
128... +42%

Speedup
0 1 2 3 4
Key Bottlenecks in Graph Processing

for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}

1. Frequent random memory accesses
2. Little amount of computation
Tesseract System for Graph Processing

Interconnected set of 3D-stacked memory+logic chips with simple cores

Host Processor
Memory
Logic

In-Order Core
Message Queue
DRAM Controller

Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed)

Crossbar Network

SAFARI Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
Tesseract System for Graph Processing

Host Processor
Memory-Mapped Accelerator Interface (Noncacheable, Physically Addressed)

Logic
Memory

Communications via Remote Function Calls
Message Queue

In-Order Core

Crossbar Network
Tesseract System for Graph Processing
Evaluated Systems

**DDR3-OoO**
- 8 OoO 4GHz
- 8 OoO 4GHz

**HMC-OoO**
- 8 OoO 4GHz
- 8 OoO 4GHz
- 8 OoO 4GHz
- 8 OoO 4GHz

**HMC-MC**
- 128 In-Order 2GHz
- 128 In-Order 2GHz
- 128 In-Order 2GHz
- 128 In-Order 2GHz

**Tesseract**
- 32 Tesseract Cores

**Sendwidths**
- 102.4GB/s
- 640GB/s
- 640GB/s
- 8TB/s

SAFARI Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
Tesseract Graph Processing Performance

>13X Performance Improvement

On five graph processing algorithms

- DDR3-OoO
- HMC-OoO
- HMC-MC
- Tesseract
- Tesseract-LP
- Tesseract-LP-MTP

Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing" ISCA 2015.
Tesseract Graph Processing Performance

Memory Bandwidth Consumption

<table>
<thead>
<tr>
<th>Case</th>
<th>Memory Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3-OoO</td>
<td>80GB/s</td>
</tr>
<tr>
<td>HMC-OoO</td>
<td>190GB/s</td>
</tr>
<tr>
<td>HMC-MC</td>
<td>243GB/s</td>
</tr>
<tr>
<td>Tesseract</td>
<td>1.3TB/s</td>
</tr>
<tr>
<td>Tesseract-LP</td>
<td>2.2TB/s</td>
</tr>
<tr>
<td>Tesseract-LP-MTP</td>
<td>2.9TB/s</td>
</tr>
</tbody>
</table>
Tesseract Graph Processing System Energy

Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
More on Tesseract

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,

"A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"


[Slides (pdf)] [Lightning Session Slides (pdf)]
PIM on Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand¹  Saugata Ghose¹  Youngsok Kim²
Rachata Ausavarungnirun¹  Eric Shiu³  Rahul Thakur³  Daehyun Kim⁴,³
Aki Kuusela³  Allan Knies³  Parthasarathy Ranganathan³  Onur Mutlu⁵,¹
Truly Distributed GPU Processing with PIM?

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Main GPU

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl

3D-stacked memory

SM (Streaming Multiprocessor)

Main GPU

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl

```
void applyScaleFactorsKernel(uint8_T * const out,
                           uint8_T const * const in,
                           const double *factor,
                           size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows +
                       sliceIdx*numRows*numCols;
```
Accelerating GPU Execution with PIM (I)

  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]
Accelerating GPU Execution with PIM (II)

  
  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 Pattnaik¹  Xulong Tang¹  Adwait Jog²  Onur Kayıran³  Asit K. Mishra⁴  Mahmut T. Kandemir¹  Onur Mutlu⁵,⁶  Chita R. Das¹

¹Pennsylvania State University  ²College of William and Mary  ³Advanced Micro Devices, Inc.  ⁴Intel Labs  ⁵ETH Zürich  ⁶Carnegie Mellon University
Two Key Questions in 3D-Stacked PIM

- How can we accelerate important applications if we use 3D-stacked memory as a coarse-grained accelerator?
  - what is the architecture and programming model?
  - what are the mechanisms for acceleration?

- What is the minimal processing-in-memory support we can provide?
  - without changing the system significantly
  - while achieving significant benefits
Simpler PIM: PIM-Enabled Instructions


PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn  Sungjoo Yoo  Onur Mutlu†  Kiyoung Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University  †Carnegie Mellon University

SAFARI
Automatic Code and Data Mapping

[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]
Challenge and Opportunity for Future

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures
Challenge and Opportunity for Future Fundamentally Low-Latency (Data-Centric) Computing Architectures
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
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

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?

```c
void applyScaleFactorsKernel( uint8_t * const out, uint8_t T const * const in, const double *factor, size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows + sliceIdx*numRows*numCols;
}
```
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 \quad \textsuperscript{2}College of William and Mary
\textsuperscript{3}Advanced Micro Devices, Inc. \quad \textsuperscript{4}Intel Labs \quad \textsuperscript{5}ETH Zürich \quad \textsuperscript{6}Carnegie Mellon University
Challenge: Coherence for Hybrid CPU-PIM Apps

![Graph showing speedup for various applications with different coherence models.](chart)

- **Traditional coherence**
- **No coherence overhead**

- **CPU-only**
- **FG**
- **CG**
- **NC**
- **LazyPIM**
- **Ideal-PIM**

**Applications:**
- arXiv
- Gnutella
- Enron
- IMDB
- GMean

**Legend:**
-的传统 coherence
- No coherence overhead
How to Maintain Coherence?

- 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
How to Support Virtual Memory?

Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
"Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"
Proceedings of the 34th IEEE International Conference on Computer Design (ICCD), Phoenix, AZ, USA, October 2016.

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)

---

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim\(^1\)  Weikun Yang\(^1,2\)  Onur Mutlu\(^1\)

\(^1\)Carnegie Mellon University  \(^2\)Peking University
An FPGA-based Test-bed for PIM?


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

`github.com/CMU-SAFARI/SoftMC`
Genome Read Mapping in PIM
Goals

- Understand the primitives, architectures, and benefits of PIM by carefully examining many important workloads
- Develop a common workload suite for PIM research
Genome Read In-Memory (GRIM) Filter: 
Fast Location Filtering in DNA Read Mapping 
with Emerging Memory Technologies

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

Carnegie Mellon
Insan Doğramacı Bilkent University
TOBB UNIVERSITY OF ECONOMICS AND TECHNOLOGY
ETH Zürich
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.
PIM-Based DNA Sequence Analysis

- To Appear in APBC 2018 and BMC Genomics 2018.

GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies

Jeremie S. Kim1,6*, Damla Senol Cali1, Hongyi Xin2, Donghyuk Lee3, Saugata Ghose1, Mohammed Alser4, Hasan Hassan6, Oguz Ergin5, Can Alkan*4, and Onur Mutlu*6,1

SAFARI
Agenda

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
Maslow’s Hierarchy of Needs, A Third Time


Source: https://www.simplypsychology.org/maslow.html
Fundamentally Energy-Efficient (Data-Centric) Computing Architectures
Challenge and Opportunity for Future

Fundamentally Low-Latency (Data-Centric) Computing Architectures
Concluding Remarks
A Quote from A Famous Architect

“architecture [...] based upon principle, and not upon precedent”
Precedent-Based Design?

- “architecture [...] based upon principle, and not upon precedent”
Principled Design

“architecture [...] based upon principle, and not upon precedent”
The Overarching Principle

Organic architecture

From Wikipedia, the free encyclopedia

Organic architecture is a philosophy of architecture which promotes harmony between human habitation and the natural world through design approaches so sympathetic and well integrated with its site, that buildings, furnishings, and surroundings become part of a unified, interrelated composition.

A well-known example of organic architecture is Fallingwater, the residence Frank Lloyd Wright designed for the Kaufmann family in rural Pennsylvania. Wright had many choices to locate a home on this large site, but chose to place the home directly over the waterfall and creek creating a close, yet noisy dialog with the rushing water and the steep site. The horizontal striations of stone masonry with daring cantilevers of colored beige concrete blend with native rock outcroppings and the wooded environment.
Another Example: Precedent-Based Design
Principled Design
Another Principled Design

Source: http://www.arcspace.com/exhibitions/unsorted/santiago-calatrava/
Principle Applied to Another Structure

The Overarching Principle

Zoomorphic architecture

From Wikipedia, the free encyclopedia

Zoomorphic architecture is the practice of using animal forms as the inspirational basis and blueprint for architectural design. "While animal forms have always played a role adding some of the deepest layers of meaning in architecture, it is now becoming evident that a new strand of biomorphism is emerging where the meaning derives not from any specific representation but from a more general allusion to biological processes."[1]

Some well-known examples of Zoomorphic architecture can be found in the TWA Flight Center building in New York City, by Eero Saarinen, or the Milwaukee Art Museum by Santiago Calatrava, both inspired by the form of a bird’s wings.[3]
Overarching Principle for Computing?
Concluding Remarks

- 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)

- Enable computation capability inside and close to memory

- This can
  - Lead to orders-of-magnitude improvements
  - Enable new applications & computing platforms
  - Enable better understanding of nature
  - ...

The Future of Processing in Memory is Bright

- Regardless of challenges
  - in underlying technology and overlying problems/requirements

Can enable:
- Orders of magnitude improvements
- New applications and computing systems

Yet, we have to
- Think across the stack
- Design enabling systems
If In Doubt, See Other Doubtful Technologies

- A very “doubtful” emerging technology
  - for at least two decades

Proceedings of the IEEE, Sept. 2017

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
Processing Data Where It Makes Sense in Modern Computing Systems: Enabling In-Memory Computation

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
23 March 2018
DATE Emerging Memory Workshop Keynote Talk
Acknowledgments

- My current and past students and postdocs
  - Rachata Ausavarungnirun, Abhishek Bhowmick, Amirali Boroumand, Rui Cai, Yu Cai, Kevin Chang, Saugata Ghose, Kevin Hsieh, Tyler Huberty, Ben Jaiyen, Samira Khan, Jeremie Kim, Yoongu Kim, Yang Li, Jamie Liu, Lavanya Subramanian, Donghyuk Lee, Yixin Luo, Justin Meza, Gennady Pekhimenko, Vivek Seshadri, Lavanya Subramanian, Nandita Vijaykumar, HanBin Yoon, Jishen Zhao, ...

- My collaborators
  - Can Alkan, Chita Das, Phil Gibbons, Sriram Govindan, Norm Jouppi, Mahmut Kandemir, Mike Kozuch, Konrad Lai, Ken Mai, Todd Mowry, Yale Patt, Moinuddin Qureshi, Partha Ranganathan, Bikash Sharma, Kushagra Vaid, Chris Wilkerson, ...
Funding Acknowledgments

- NSF
- GSRC
- SRC
- CyLab
- AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware
Some Open Source Tools

- **Rowhammer**
  - [https://github.com/CMU-SAFA RI/rowhammer](https://github.com/CMU-SAFA RI/rowhammer)

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

- **MemSim**
  - [https://github.com/CMU-SAFA RI/memsim](https://github.com/CMU-SAFA RI/memsim)

- **NOCulator**
  - [https://github.com/CMU-SAFA RI/NOCulator](https://github.com/CMU-SAFA RI/NOCulator)

- **DRAM Error Model**
  - [http://www.ece.cmu.edu/~safari/tools/memerr/index.html](http://www.ece.cmu.edu/~safari/tools/memerr/index.html)

- **Other open-source software from my group**
  - [https://github.com/CMU-SAFA RI/](https://github.com/CMU-SAFA RI/)
  - [http://www.ece.cmu.edu/~safari/tools.html](http://www.ece.cmu.edu/~safari/tools.html)
Tesseract: Extra Slides
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}

Vault #1

Vault #2

v
&w
w
for (v: graph.vertices) {
    for (w: v.successors) {
        put(w.id, function() { w.next_rank += weight * v.rank; });
    }
}

barrier();

Non-blocking Remote Function Call
Can be delayed until the nearest barrier

Vault #1

Vault #2
Remote Function Call (Non-Blocking)

1. Send function address & args to the remote core
2. Store the incoming message to the message queue
3. Flush the message queue when it is full or a synchronization barrier is reached

```javascript
put(w.id, function() { w.next_rank += value; })
```
Effect of Bandwidth & Programming Model

- **HMC-MC Bandwidth (640GB/s)**
- **Tesseract Bandwidth (8TB/s)**

**Bandwidth**:
- **HMC-MC**: 1.0x
- **HMC-MC + PIM BW**: 2.3x
- **Tesseract + Conventional BW**: 3.0x
- **Tesseract (No Prefetching)**: 6.5x

**Programming Model**:
- **HMC-MC**: 0.0x
- **HMC-MC + PIM BW**: 2.3x
- **Tesseract + Conventional BW**: 3.0x
- **Tesseract (No Prefetching)**: 6.5x
Reducing Memory Latency
Main Memory Latency Lags Behind

- **Capacity**
- **Bandwidth**
- **Latency**

Memory latency remains almost constant
A Closer Look …

Figure 1: DRAM latency trends over time [20, 21, 23, 51].

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

In-memory Databases

Graph/Tree Processing

Long memory latency → performance bottleneck

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

Datacenter Workloads
[Kanev+ (Google), ISCA’15]
Why the Long Latency?

- **Design of DRAM uArchitecture**
  - Goal: Maximize capacity/area, not minimize latency

- **“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
Heterogeneous manufacturing & operating conditions → latency variation in timing parameters
DRAM Characterization Infrastructure

DRAM Characterization Infrastructure


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

  github.com/CMU-SAFARI/SoftMC
SoftMC: Open Source DRAM Infrastructure

https://github.com/CMU-SAFARI/SoftMC

SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

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

$^1$ETH Zürich $^2$TOBB University of Economics & Technology $^3$Carnegie Mellon University
$^4$University of Virginia $^5$Microsoft Research $^6$NVIDIA Research
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]
  - ...

- We would like to find sources of latency heterogeneity and exploit them to minimize latency
Adaptive-Latency 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

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)
  - **DRAM**: 4GByte DDR3-1600 (800Mhz Clock)
  - **OS**: Linux
  - **Storage**: 128GByte SSD

- **Workload**
  - 35 applications from SPEC, STREAM, Parsec, Memcached, Apache, GUPS

---

**Table: DDR3 DRAM Timing 0**

<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>07h-00h</td>
<td>Reserved.</td>
</tr>
<tr>
<td>2Ah-08h</td>
<td>&lt;Tras&gt; clocks</td>
</tr>
<tr>
<td>3Fh-2Bh</td>
<td>Reserved.</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 refresh command, both to the same bank.</td>
</tr>
</tbody>
</table>
AL-DRAM improves single-core performance on a real system
AL-DRAM provides higher performance on 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
More on Adaptive-Latency 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]

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case

Donghyuk Lee  Yoongu Kim  Gennady Pekhimenko  
Samira Khan  Vivek Seshadri  Kevin Chang  Onur Mutlu  
Carnegie Mellon University

SAFARI

196
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]

Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization

Kevin K. Chang\textsuperscript{1} Abhijith Kashyap\textsuperscript{1} Hasan Hassan\textsuperscript{1,2}
Saugata Ghose\textsuperscript{1} Kevin Hsieh\textsuperscript{1} Donghyuk Lee\textsuperscript{1} Tianshi Li\textsuperscript{1,3}
Gennady Pekhimenko\textsuperscript{1} Samira Khan\textsuperscript{4} Onur Mutlu\textsuperscript{5,1}

\textsuperscript{1}Carnegie Mellon University \textsuperscript{2}TOBB ETÜ \textsuperscript{3}Peking University \textsuperscript{4}University of Virginia \textsuperscript{5}ETH Zürich

SAFARI
What Is Design-Induced Variation?

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

Design-Induced-Variation-Aware

Profile only slow regions to determine min. latency

→ Dynamic & low cost latency optimization
Design-Induced-Variation-Aware

slow cells
process variation
random error

error-correcting code

inherently slow
design-induced variation
localized error

online profiling

Combine error-correcting codes & online profiling
→ Reliably reduce DRAM latency
DIVA-DRAM reduces latency more aggressively and uses ECC to correct random slow cells.
Design-Induced Latency Variation in DRAM

- Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu,

"Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms"

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
Voltron: Exploiting the Voltage-Latency-Reliability Relationship
Executive Summary

- **DRAM (memory) power is significant in today’s systems**
  - Existing low-voltage DRAM reduces voltage *conservatively*

- **Goal:** Understand and exploit the reliability and latency behavior of real DRAM chips under *aggressive reduced-voltage operation*

- **Key experimental observations:**
  - Huge voltage margin -- Errors occur beyond some voltage
  - Errors exhibit *spatial locality*
  - Higher operation latency mitigates voltage-induced errors

- **Voltron:** A new DRAM energy reduction mechanism
  - Reduce DRAM voltage *without introducing errors*
  - Use a *regression model* to select voltage that does not degrade performance beyond a chosen target → 7.3% system energy reduction
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 Yaglıkçı† 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?
Tiered Latency DRAM
What Causes the Long Latency?

DRAM Latency = Subarray Latency + I/O Latency

Dominant Subarray I/O
Why is the Subarray So Slow?

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

Long Bitline

Faster
Smaller

Short Bitline

Faster
Smaller

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

Normalized DRAM Area
Latency (ns)

- 64 cells/bitline
- 32 cells/bitline
- 128 cells/bitline
- 256 cells/bitline
- 512 cells/bitline

Commodity DRAM
Long Bitline
Cheaper
Faster
Fancy DRAM
Short Bitline

Goal: Faster and Cheaper
Approximating the Best of Both Worlds

Long Bitline
Small Area
High Latency

Our Proposal

Short Bitline
Large Area
Low Latency

Need Isolation
Add Isolation Transistors

Fast
Approximating the Best of Both Worlds

- Long Bitline Tiered-Latency DRAM
- Short Bitline

<table>
<thead>
<tr>
<th>Small Area</th>
<th>Small Area</th>
<th>Large Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Latency</td>
<td>Low Latency</td>
<td>Low Latency</td>
</tr>
</tbody>
</table>

Small area using long bitline
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: 64 cells/bitline
Far Segment: 512 cells/bitline

GOAL: Cheaper and Faster

Faster

0 10 20 30 40 50 60 70
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

Using near segment as a cache improves performance and reduces power consumption

Challenge and Opportunity for Future

Fundamentally

Low Latency

Computing Architectures
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, TLDRA, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<table>
<thead>
<tr>
<th>Simulator (clang -O3)</th>
<th>Cycles ($10^6$) Random</th>
<th>Cycles ($10^6$) Stream</th>
<th>Runtime (sec.) Random</th>
<th>Runtime (sec.) Stream</th>
<th>Req/sec ($10^3$) Random</th>
<th>Req/sec ($10^3$) Stream</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
<td>133</td>
<td>402</td>
<td>2.1</td>
</tr>
<tr>
<td>DRAMSim2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
<td>49</td>
<td>114</td>
<td>1.2</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
<td>53</td>
<td>133</td>
<td>4.5</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
<td>6</td>
<td>8</td>
<td>1.6</td>
</tr>
<tr>
<td>NVMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
<td>15</td>
<td>20</td>
<td>4,230.0</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.

Source code is released under the liberal MIT License

- [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)
End of Backup Slides
Brief Self Introduction

Onur Mutlu

- Full Professor @ ETH Zurich CS, since September 2015 (officially May 2016)
- Strecker Professor @ Carnegie Mellon University ECE/CS, 2009-2016, 2016-...
- PhD from UT-Austin, worked at Google, VMware, Microsoft Research, Intel, AMD
- [https://people.inf.ethz.ch/omutlu/](https://people.inf.ethz.ch/omutlu/)
- [omutlu@gmail.com](mailto:omutlu@gmail.com) (Best way to reach me)
- [https://people.inf.ethz.ch/omutlu/projects.htm](https://people.inf.ethz.ch/omutlu/projects.htm)

Research and Teaching in:

- Computer architecture, computer systems, security, bioinformatics
- Memory and storage systems
- Hardware security
- Fault tolerance
- Hardware/software cooperation
- ...