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
19 November 2019
ICCD Keynote

SAFARI
ETH Zürich
Carnegie Mellon
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!
DRAM Capacity, Bandwidth & Latency

- **Capacity**: 128x improvement
- **Bandwidth**: 20x improvement
- **Latency**: 1.3x improvement

**Graph Details**:
- **Years**: 1999 to 2017
- **Improvement (log)**: 1, 10, 100

**Legend**:
- Black triangle: Capacity
- Green square: Bandwidth
- Red circle: Latency

**SAFARI**
DRAM 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 Is Critical for Performance

In-memory Databases

Graph/Tree Processing

Memory → performance bottleneck

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

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

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

VP9
YouTube
Video Playback
Google’s video codec

VP9
YouTube
Video Capture
Google’s video codec
DRAM Is Critical for Performance

Chrome

TensorFlow Mobile

Memory → performance bottleneck

Video Playback
Google's video codec

Video Capture
Google's video codec
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
Energy Waste in Mobile Devices


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} \hspace{1cm} Saugata Ghose\textsuperscript{1} \hspace{1cm} Youngsok Kim\textsuperscript{2} \\
Rachata Ausavarungnirun\textsuperscript{1} \hspace{1cm} Eric Shiu\textsuperscript{3} \hspace{1cm} Rahul Thakur\textsuperscript{3} \\
Aki Kuusela\textsuperscript{3} \hspace{1cm} Parthasarathy Ranganathan\textsuperscript{3} \hspace{1cm} Daehyun Kim\textsuperscript{4,3} \\
Allan Knies\textsuperscript{3} \hspace{1cm} Onur Mutlu\textsuperscript{5,1} \\

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

CPU

DRAM Ctrl | PCM Ctrl

DRAM

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

Phase Change Memory (or Tech. X)

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

Yoon+, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.
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...
How Reliable/Secure/Safe is This Bridge?

Source: http://www.technologystudent.com/struct1/tacom1.png
Collapse of the “Galloping Gertie”

Source: AP
How Secure Are These People?

Security is about preventing unforeseen consequences
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.

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-SAFA/SoftMC
SoftMC

- [https://github.com/CMU-SAFARI/SoftMC](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
Retention Time Profile of DRAM looks like this:

- Location dependent
- Stored value pattern dependent
- Time dependent

64-128ms

>256ms

128-256ms
Main Memory Needs
Intelligent Controllers
More on DRAM Refresh (I)


RAIDR: Retention-Aware Intelligent DRAM Refresh

Jamie Liu Ben Jaiyen Richard Veras Onur Mutlu
Carnegie Mellon University
More on DRAM Refresh (II)

Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu,
"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"
Proceedings of the 40th International Symposium on Computer Architecture (ISCA), Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu
More on DRAM Refresh (III)

- Minesh Patel, Jeremie S. Kim, and Onur Mutlu,
  "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions"
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

- First experimental analysis of (mobile) LPDDR4 chips
- Analyzes the complex tradeoff space of retention time profiling
- Idea: enable fast and robust profiling at higher refresh intervals & temperatures

The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions

Minesh Patel$^\ddagger$  Jeremie S. Kim$^\ddagger$\$  Onur Mutlu$^\ddagger$

$^\ddagger$ETH Zürich  $^\ddagger$Carnegie Mellon University
One can predictably induce errors in most DRAM memory chips
DRAM RowHammer

A simple hardware failure mechanism can create a widespread system security vulnerability.
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 $1.0 \times 10^7$ errors

B company
83% (45/54)
Up to $2.7 \times 10^6$ errors

C company
88% (28/32)
Up to $3.3 \times 10^5$ 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
A Simple Program Can Induce Many Errors

```assembly
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```

Download from: https://github.com/CMU-SAFARI/rowhammer
A Simple Program Can Induce Many Errors

1. Avoid *cache hits*
   - Flush \( \mathbf{X} \) from cache

2. Avoid *row hits* to \( \mathbf{X} \)
   - Read \( \mathbf{Y} \) in another row

Download from: [https://github.com/CMU-SAFARI/rowhammer](https://github.com/CMU-SAFARI/rowhammer)
A Simple Program Can Induce Many Errors

```
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```

Download from: https://github.com/CMU-SAFARI/rowhammer
A Simple Program Can Induce Many Errors

```
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```

Download from: https://github.com/CMU-SAFARI/rowhammer
A Simple Program Can Induce Many Errors

```
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```

Download from: https://github.com/CMU-SAFARI/rowhammer
## Observed Errors in Real Systems

<table>
<thead>
<tr>
<th>CPU Architecture</th>
<th>Errors</th>
<th>Access-Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Haswell (2013)</td>
<td>22.9K</td>
<td>12.3M/sec</td>
</tr>
<tr>
<td>Intel Ivy Bridge (2012)</td>
<td>20.7K</td>
<td>11.7M/sec</td>
</tr>
<tr>
<td>Intel Sandy Bridge (2011)</td>
<td>16.1K</td>
<td>11.6M/sec</td>
</tr>
<tr>
<td>AMD Piledriver (2012)</td>
<td>59</td>
<td>6.1M/sec</td>
</tr>
</tbody>
</table>

A real reliability & security issue

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

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

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
RowHammer Security Attack Example

- “Rowhammer” is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
  - Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

- We tested a selection of laptops and found that a subset of them exhibited the problem.
- We built two working privilege escalation exploits that use this effect.
  - Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn+, 2015)

- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.
- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries (PTEs).
- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.

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

Rowhammer
Security Implications

Rowhammer

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 (I)

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

Not there yet, but ...

**ROWHAMMER.js**

ROOT privileges for web apps!

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

“Can gain control of a smart phone deterministically”

Hammer And Root

Millions of Androids

Source: https://fossbytes.com/drammer-rowhammer-attack-android-root-devices/
More Security Implications (III)

- Using an integrated GPU in a mobile system to remotely escalate privilege via the WebGL interface

"GRAND PWNING UNIT" —

Drive-by Rowhammer attack uses GPU to compromise an Android phone

JavaScript based GLitch pwns browsers by flipping bits inside memory chips.

DAN GOODIN - 5/3/2018, 12:00 PM

Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU

- Pietro Frigo
  Vrije Universiteit
  Amsterdam
  p.frigo@vu.nl

- Cristiano Giuffrida
  Vrije Universiteit
  Amsterdam
  giuffrida@cs.vu.nl

- Herbert Bos
  Vrije Universiteit
  Amsterdam
  herbertb@cs.vu.nl

- Kaveh Razavi
  Vrije Universiteit
  Amsterdam
  kaveh@cs.vu.nl
More Security Implications (IV)

- Rowhammer over RDMA (I)

THROWHAMMER —

Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

DAN GOODIN - 5/10/2018, 5:26 PM

Throwhammer: Rowhammer Attacks over the Network and Defenses

Andrei Tatar  
VU Amsterdam

Radhesh Krishnan  
VU Amsterdam

Elias Athanasopoulos  
University of Cyprus

Cristiano Giuffrida  
VU Amsterdam

Herbert Bos  
VU Amsterdam

Kaveh Razavi  
VU Amsterdam
More Security Implications (V)

- Rowhammer over RDMA (II)

Nethammer—Exploiting DRAM Rowhammer Bug Through Network Requests

Nethammer: Inducing Rowhammer Faults through Network Requests

Moritz Lipp
Graz University of Technology

Misiker Tadesse Aga
University of Michigan

Michael Schwarz
Graz University of Technology

Daniel Gruss
Graz University of Technology

Clémentine Maurice
Univ Rennes, CNRS, IRISA

Lukas Raab
Graz University of Technology

Lukas Lamster
Graz University of Technology
More Security Implications?
Apple’s Patch for RowHammer

- https://support.apple.com/en-gb/HT204934

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5

Impact: A malicious application may induce memory corruption to escalate privileges

Description: A disturbance error, also known as Rowhammer, exists with some DDR3 RAM that could have led to memory corruption. **This issue was mitigated by increasing memory refresh rates.**

CVE-ID

CVE-2015-3693: Mark Seaborn and Thomas Dullien of Google, working from original research by Yoongu Kim et al (2014)

HP, Lenovo, and other vendors released similar patches
Our Solution to RowHammer

• **PARA:** *Probabilistic Adjacent Row Activation*

• Key Idea
  – After closing a row, we activate (i.e., refresh) one of its neighbors with a low probability: $p = 0.005$

• Reliability Guarantee
  – When $p=0.005$, errors in one year: $9.4 \times 10^{-14}$
  – By adjusting the value of $p$, we can vary the strength of protection against errors
Advantages of PARA

• **PARA refreshes rows infrequently**
  – Low power
  – Low performance-overhead
    • Average slowdown: **0.20%** (for 29 benchmarks)
    • Maximum slowdown: **0.75%**

• **PARA is stateless**
  – Low cost
  – Low complexity

• **PARA is an effective and low-overhead solution to prevent disturbance errors**
Requirements for PARA

• If implemented in DRAM chip (done today)
  – Enough slack in timing and refresh parameters
  – Plenty of slack today:
    • Chang et al., “Understanding Latency Variation in Modern DRAM Chips,” SIGMETRICS 2016.
    • Lee et al., “Design-Induced Latency Variation in Modern DRAM Chips,” SIGMETRICS 2017.
    • Chang et al., “Understanding Reduced-Voltage Operation in Modern DRAM Devices,” SIGMETRICS 2017.

• If implemented in memory controller
  – Better coordination between memory controller and DRAM
  – Memory controller should know which rows are physically adjacent
Probabilistic Activation in Real Life (I)

https://twitter.com/isislovecruf/status/1021939922754723841
More on RowHammer Analysis

Future of Memory Reliability

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

[Slides (pptx) (pdf)]
Onur Mutlu and Jeremie Kim, "RowHammer: A Retrospective"


[Preliminary arXiv version]

---

RowHammer: A Retrospective

Onur Mutlu§‡

§ETH Zürich

Jeremie S. Kim‡§

‡Carnegie Mellon University
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

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

USB Jack

Virtex-II Pro (USB controller)

1x-nm NAND Flash

Virtex-V FPGA (NAND Controller)

NAND Daughter Board


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
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 Need for More Memory Performance

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

VP9
YouTube
Video Playback
Google’s video codec

VP9
YouTube
Video Capture
Google’s video codec
Do We Want This?

Source: V. Milutinovic
Or This?

Source: V. Milutinovic
Maslow’s (Human) Hierarchy of Needs, Revisited


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

High Performance, Energy Efficient, Sustainable
The Problem

Data access is the major performance and energy bottleneck

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

I expect that over the coming decade memory subsystem design will be the only important design issue for microprocessors.

---

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):

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

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
The Energy Perspective

Communication Dominates Arithmetic

64-bit DP 20pJ

256-bit buses

256-bit access 8 kB SRAM

20mm

16 nJ DRAM Rd/Wr

500 pJ Efficient off-chip link

50 pJ

1 nJ

26 pJ

256 pJ

Dally, HiPEAC 2015
Data Movement vs. Computation Energy

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)
Energy Waste in Mobile Devices


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}  

SAFARI
We Do Not Want to Move Data!

Communication Dominates Arithmetic

Dally, HiPEAC 2015

A memory access consumes \( \sim 1000X \) the energy of a complex addition
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 and in-memory units?
- software and hardware interfaces?
- system software, compilers, languages?
- algorithms and theoretical foundations?
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

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
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

3) No cache pollution
1) Low latency

2) Low bandwidth utilization
4) No unwanted data movement

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

Idea: Two consecutive ACTivates
Negligible HW cost

Step 1: Activate row A
Step 2: Activate row B

DRAM subarray
Row Buffer (4 Kbytes)
Data Bus
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)]
Memory as an Accelerator

Memory similar to a “conventional” accelerator
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

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

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

Performance: In-DRAM Bitwise Operations

**Figure 9:** Throughput of bitwise operations on various systems.

## Energy of In-DRAM Bitwise Operations

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

<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>**DRAM &amp; **</td>
<td>DDR3</td>
<td>93.7</td>
<td>137.9</td>
<td>137.9</td>
</tr>
<tr>
<td>Channel Energy</td>
<td>Ambit</td>
<td>1.6</td>
<td>3.2</td>
<td>4.0</td>
</tr>
<tr>
<td><em>(nJ/KB)</em></td>
<td>(↓)</td>
<td>59.5X</td>
<td>43.9X</td>
<td>35.1X</td>
</tr>
</tbody>
</table>

Ambit vs. DDR3: Performance and Energy

- **Performance Improvement**
- **Energy Reduction**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance Improvement</th>
<th>Energy Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>50</td>
<td>32X</td>
</tr>
<tr>
<td>and/or</td>
<td>40</td>
<td>35X</td>
</tr>
<tr>
<td>nand/nor</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>xor/xnor</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

Bulk Bitwise Operations in Workloads

- Bitmap indices (database indexing)
- Set operations
- Encryption algorithms
- BitWeaving (database queries)
- BitFunnel (web search)
- DNA sequence mapping
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
```
Performance: Bitmap Index on Ambit

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

>5.4-6.6X Performance Improvement

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"
More on Ambit


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

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

\textsuperscript{1}Microsoft Research India \textsuperscript{2}NVIDIA Research \textsuperscript{3}Intel \textsuperscript{4}ETH Zürich \textsuperscript{5}Carnegie Mellon University
More on In-DRAM Bulk Bitwise Execution

- Vivek Seshadri and Onur Mutlu,
  "In-DRAM Bulk Bitwise Execution Engine"
  [Preliminary arXiv version]

In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
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
Processing in Memory: Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory
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]; RLDRA M3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory

Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming
Graph Processing

- Large graphs are everywhere (circa 2015)
  - 36 Million Wikipedia Pages
  - 1.4 Billion Facebook Users
  - 300 Million Twitter Users
  - 30 Billion Instagram Photos

- Scalable large-scale graph processing is challenging
  - Speedup
    - 32 Cores
    - 128...
    - +42%
Key Bottlenecks in Graph Processing

```java
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-Mapped Accelerator Interface (Noncacheable, Physically Addressed)

Crossbar Network

Memory

Logic

In-Order Core

LP

PF Buffer

MTP

Message Queue

DRAM Controller

NI

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)

- Memory
  - Crossbar Network
  - In-Order Core
  - DRAM Controller
  - NI

- Logic
  - Communications via Remote Function Calls
  - Message Queue

- Host Processor

- Memory

- Logic
Tesseract System for Graph Processing

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

Logic

Memory

Crossbar Network

Prefetching

DRAM Controller

LP
PF Buffer
MTP

Message Queue

NI
Evaluated Systems

<table>
<thead>
<tr>
<th>DDR3-OoO</th>
<th>HMC-OoO</th>
<th>HMC-MC</th>
<th>Tesseract</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 In-Order 2GHz</td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 In-Order 2GHz</td>
</tr>
<tr>
<td>128 In-Order 2GHz</td>
<td>128 In-Order 2GHz</td>
<td>128 In-Order 2GHz</td>
<td>128 In-Order 2GHz</td>
</tr>
</tbody>
</table>

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

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

Memory Bandwidth Consumption

- DDR3-OoO: 80GB/s
- HMC-OoO: 190GB/s
- HMC-MC: 243GB/s
- Tesseract: 1.3TB/s
- Tesseract-LP: 2.2TB/s
- Tesseract-LP-MTP: 2.9TB/s
Tesseract Graph Processing System Energy

HMC-OoO

> 8X Energy Reduction

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)]

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

Junwhan Ahn  Sungpack Hong$  Sungjoo Yoo  Onur Mutlu†  Kιyoung Choi
junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University  $Oracle Labs  †Carnegie Mellon University
Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming
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 Boroumand\textsuperscript{1} 
Rachata Ausavarungnirun\textsuperscript{1} 
Aki Kuusela\textsuperscript{3} 
Saugata Ghose\textsuperscript{1} 
Eric Shiu\textsuperscript{3} 
Allan Knies\textsuperscript{3} 
Youngsok Kim\textsuperscript{2} 
Rahul Thakur\textsuperscript{3} 
Parthasarathy Ranganathan\textsuperscript{3} 
Daehyun Kim\textsuperscript{4,3} 
Onur Mutlu\textsuperscript{5,1}

SAFARI
Consumer Devices

Consumer devices are everywhere!

Energy consumption is a first-class concern in consumer devices.
Four Important Workloads

**Chrome**
Google’s web browser

**TensorFlow Mobile**
Google’s machine learning framework

**VP9**
YouTube Video Playback
Google’s video codec

**VP9**
YouTube Video Capture
Google’s video codec
Energy Cost of Data Movement

**1st key observation:** 62.7% of the total system energy is spent on data movement.

**Potential solution:** move computation close to data.

**Challenge:** limited area and energy budget.
Using PIM to Reduce Data Movement

2nd key observation: a significant fraction of the data movement often comes from simple functions.

We can design lightweight logic to implement these simple functions in memory.

Offloading to PIM logic reduces energy and improves performance, on average, by 55.4% and 54.2%.
Workload Analysis

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

VP9
Video Playback
Google’s video codec

VP9
Video Capture
Google’s video codec
57.3% of the inference energy is spent on data movement.

54.4% of the data movement energy comes from packing/unpacking and quantization.
Packing

Reorders elements of matrices to minimize cache misses during matrix multiplication

Up to 40% of the inference energy and 31% of inference execution time

Packing’s data movement accounts for up to 35.3% of the inference energy

A simple data reorganization process that requires simple arithmetic
Quantization

Converts **32-bit floating point** to **8-bit integers** to improve inference execution time and energy consumption.

- Up to **16.8%** of the inference energy and **16.1%** of inference execution time.
- Majority of quantization energy comes from data movement.

A simple **data conversion** operation that requires **shift, addition, and multiplication** operations.
PIM core and PIM accelerator reduce energy consumption on average by 49.1% and 55.4%
Offloading these kernels to **PIM core** and **PIM accelerator** improves **performance** on average by **44.6%** and **54.2%**
More on PIM for Mobile Devices


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} \quad Rachata Ausavarungnirun\textsuperscript{1} \quad Eric Shiu\textsuperscript{3} \quad Rahul Thakur\textsuperscript{3} \quad Daehyun Kim\textsuperscript{4,3} \\
Saugata Ghose\textsuperscript{1} \quad Youngsok Kim\textsuperscript{2} \quad Parthasarathy Ranganathan\textsuperscript{3} \quad Onur Mutlu\textsuperscript{5,1} \\
Aki Kuusela\textsuperscript{3} \quad Allan Knies\textsuperscript{3}
Truly Distributed GPU Processing with PIM?

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Main GPU

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl

Logic layer

SM

Global

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)

Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrich Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler,

"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"

[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]
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
Accelerating Linked Data Structures

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
Accelerating Dependent Cache Misses

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller"
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi*, Khubaib†, Eiman Ebrahimi‡, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin  †Apple  ‡NVIDIA  §ETH Zürich & Carnegie Mellon University
Two Key Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming
PEI: PIM-Enabled Instructions (Ideas)

- **Goal:** Develop mechanisms to get the most out of near-data processing with minimal cost, minimal changes to the system, no changes to the programming model

- **Key Idea 1:** Expose each PIM operation as a cache-coherent, virtually-addressed host processor instruction (called PEI) that operates on only a single cache block
  - e.g., __pim_add(&w.next_rank, value) \(\rightarrow\) pim.add r1, (r2)
  - No changes sequential execution/programming model
  - No changes to virtual memory
  - Minimal changes to cache coherence
  - No need for data mapping: Each PEI restricted to a single memory module

- **Key Idea 2:** Dynamically decide where to execute a PEI (i.e., the host processor or PIM accelerator) based on simple locality characteristics and simple hardware predictors
  - Execute each operation at the location that provides the best performance
for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        w.next_rank += value;
    }
}

Host Processor

Main Memory

Conventional Architecture

64 bytes in
64 bytes out
for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        __pim_add(&w.next_rank, value);
    }
}

pim.add r1, (r2)
PEI: PIM-Enabled Instructions (Example)

for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        __pim_add(&w.next_rank, value);
    }
}
pfence();

- Executed either in memory or in the processor: dynamic decision
  - Low-cost locality monitoring for a single instruction
- Cache-coherent, virtually-addressed, single cache block only
- Atomic between different PEIs
- Not atomic with normal instructions (use pfence for ordering)

Table 1: Summary of Supported PIM Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>R</th>
<th>W</th>
<th>Input</th>
<th>Output</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-byte integer increment</td>
<td>O</td>
<td>O</td>
<td>0 bytes</td>
<td>0 bytes</td>
<td>AT</td>
</tr>
<tr>
<td>8-byte integer min</td>
<td>O</td>
<td>O</td>
<td>8 bytes</td>
<td>0 bytes</td>
<td>BFS, SP, WCC</td>
</tr>
<tr>
<td>Floating-point add</td>
<td>O</td>
<td>O</td>
<td>8 bytes</td>
<td>0 bytes</td>
<td>PR</td>
</tr>
<tr>
<td>Hash table probing</td>
<td>O</td>
<td>X</td>
<td>8 bytes</td>
<td>9 bytes</td>
<td>HJ</td>
</tr>
<tr>
<td>Histogram bin index</td>
<td>O</td>
<td>X</td>
<td>1 byte</td>
<td>16 bytes</td>
<td>HG, RP</td>
</tr>
<tr>
<td>Euclidean distance</td>
<td>O</td>
<td>X</td>
<td>64 bytes</td>
<td>4 bytes</td>
<td>SC</td>
</tr>
<tr>
<td>Dot product</td>
<td>O</td>
<td>X</td>
<td>32 bytes</td>
<td>8 bytes</td>
<td>SVM</td>
</tr>
</tbody>
</table>
Example (Abstract) PEI uArchitecture

Host Processor

Out-Of-Order Core

L1 Cache

L2 Cache

Last-Level Cache

HMC Controller

3D-stacked Memory

PCU

DRAM Controller

PCU

DRAM Controller

PCU

DRAM Controller

PMU

(PETI Mgmt Unit)

PIM Directory

Locality Monitor

Example PEI uArchitecture
PEI: Initial Evaluation Results

- Initial evaluations with **10 emerging data-intensive workloads**
  - Large-scale graph processing
  - In-memory data analytics
  - Machine learning and data mining
  - Three input sets (small, medium, large) for each workload to analyze the impact of data locality

- Pin-based cycle-level x86-64 simulation

**Performance Improvement and Energy Reduction:**
- 47% average speedup with large input data sets
- 32% speedup with small input data sets
- 25% avg. energy reduction in a single node with large input data sets
PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: Host-Only)

47% Performance Improvement
PEI Energy Consumption

25% Energy Reduction

Host-Only
PIM-Only
Locality-Aware

Small
Medium
Large

Cache
HMC Link
DRAM
Host-side PCU
Memory-side PCU
PMU

SAFARI
Simpler PIM: PIM-Enabled Instructions

Automatic Code and Data Mapping


[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]
Automatic Offloading of Critical Code

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller"
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi*, Khubaib†, Eiman Ebrahimi‡, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin  †Apple  ‡NVIDIA  §ETH Zürich & Carnegie Mellon University
Automatic Offloading of Prefetch Mechanisms

[Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi*, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin  §ETH Zürich
Efficient Automatic Data Coherence Support

- 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
Efficient Automatic Data Coherence Support

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

"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"


CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand†
Brandon Lucia†
Saugata Ghose†
Rachata Ausavarungnirun†‡
Nastaran Hajinazar♦†
Krishna T. Malladi§
Minesh Patel*
Hasan Hassan*
Kevin Hsieh†
Hongzhong Zheng§
Onur Mutlu*†

†Carnegie Mellon University
♦Simon Fraser University
‡ETH Zürich
§KMUTNB
§Samsung Semiconductor, Inc.
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
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 & software for PIM

2. Ease of programming (interfaces and compiler/HW support)

3. System support: coherence & virtual memory

4. Runtime and compilation systems for adaptive scheduling, data mapping, access/sharing control

5. Infrastructures and models to assess benefits and feasibility

All can be solved with change of mindset
We Need to Revisit the Entire Stack

We can get there step by step
PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu\textsuperscript{a,b}, Saugata Ghose\textsuperscript{b}, Juan Gómez-Luna\textsuperscript{a}, Rachata Ausavarungnirun\textsuperscript{b,c}

\textsuperscript{a}ETH Zürich
\textsuperscript{b}Carnegie Mellon University
\textsuperscript{c}King Mongkut’s University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"

Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.

[arXiv version]

SAFARI

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose† Amirali Boroumand‡ Jeremie S. Kim‡§ Juan Gómez-Luna§ Onur Mutlu§†

†Carnegie Mellon University §ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu, "Processing-in-Memory: A Workload-Driven Perspective"
[Preliminary arXiv version]

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 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;
```

3D-stacked memory (memory stack)

Main GPU

SM (Streaming Multiprocessor)

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl
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?

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 Pattnaik¹ Xulong Tang¹ Adwait Jog² Onur Kayiran³
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
Challenge: Coherence for Hybrid CPU-PIM Apps

Traditional coherence

CPU-only
FG
CG
NC
LazyPIM
No coherence overhead

Ideal-PIM
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?


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
  - [Source Code for Ramulator-PIM](https://github.com/CMU-SAFARI/ramulator-pim)
  - [Source Code for 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](https://github.com/CMU-SAFARI/SoftMC)
Simulation Infrastructures for PIM (in SSDs)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,
  "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"
  [Slides (pptx) (pdf)]
  [Source Code]

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices
Arash Tavakkol†, Juan Gómez-Luna†, Mohammad Sadrosadati†, Saugata Ghose‡, Onur Mutlu†‡
†ETH Zürich ‡Carnegie Mellon University

SAFARI
Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira, Stefano Corda, Sander Stujik, Onur Mutlu, and Henk Corporaal, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning". 

Proceedings of the 56th Design Automation Conference (DAC), Las Vegas, NV, USA, June 2019.

[Slides (pptx) (pdf)]
[Poster (pptx) (pdf)]
[Source Code for Ramulator-PIM]
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"

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. Kim¹,⁶*, Damla Senol Cali¹, Hongyi Xin², Donghyuk Lee³, Saugata Ghose¹, Mohammed Alser⁴, Hasan Hassan⁶, Oguz Ergin⁵, Can Alkan⁴* and Onur Mutlu⁶,¹*

*From The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018*
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**.
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

SAFARI, Carnegie Mellon, Google, Samsung, Seoul National University, ETH Zürich
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

[Preliminary arxiv.org version]

PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu\textsuperscript{a,b}, Saugata Ghose\textsuperscript{b}, Juan Gómez-Luna\textsuperscript{a}, Rachata Ausavarungnirun\textsuperscript{b,c}

\textsuperscript{a}ETH Zürich
\textsuperscript{b}Carnegie Mellon University
\textsuperscript{c}King Mongkut’s University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"

Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.

[arXiv version]

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose† Amirali Boroumand‡ Jeremie S. Kim†§ Juan Gómez-Luna§ Onur Mutlu§†

†Carnegie Mellon University  §ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu, "Processing-in-Memory: A Workload-Driven Perspective"
[Preliminary arXiv version]
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
Fundamentally Energy-Efficient (Data-Centric) Computing Architectures
Challenge and Opportunity for Future

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

Computing Architectures with Minimal Data Movement
Key Takeaway

Main Memory Needs

Intelligent Controllers
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

Source: http://cookiemagik.deviantart.com/art/Train-station-207266944
Another Principled Design
Another Principled Design
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
We Need to Revisit the Entire Stack

We can get there step by step
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
PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu\textsuperscript{a,b}, Saugata Ghose\textsuperscript{b}, Juan Gómez-Luna\textsuperscript{a}, Rachata Ausavarungnirun\textsuperscript{b,c}

\textsuperscript{a}ETH Zürich
\textsuperscript{b}Carnegie Mellon University
\textsuperscript{c}King Mongkut’s University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"

Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.

[arXiv version]

\url{https://arxiv.org/pdf/1903.03988.pdf}
A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose† Amirali Boroumand† Jeremie S. Kim†§ Juan Gómez-Luna§ Onur Mutlu§†

†Carnegie Mellon University §ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu, "Processing-in-Memory: A Workload-Driven Perspective"
[Preliminary arXiv version]

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

- Alibaba, AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware
- NSF
- NIH
- GSRC
- SRC
- CyLab
Acknowledgments

SAFARI
SAFARI Research Group
safari.ethz.ch

Think BIG, Aim HIGH!

https://safari.ethz.ch
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
19 November 2019
ICCD Keynote
Slides Not Covered

But Could Be Useful
Readings, Videos, Reference Materials
Accelerated Memory Course (~6.5 hours)

- **ACACES 2018**
  - Memory Systems and Memory-Centric Computing Systems
  - Taught by Onur Mutlu July 9-13, 2018
  - ~6.5 hours of lectures

- **Website for the Course including Videos, Slides, Papers**
  - [https://safari.ethz.ch/memory_systems/ACACES2018/](https://safari.ethz.ch/memory_systems/ACACES2018/)
  - [https://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomthrpDpMJm05P6J9x](https://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomthrpDpMJm05P6J9x)

- **All Papers are at:**
  - [https://people.inf.ethz.ch/omutlu/projects.htm](https://people.inf.ethz.ch/omutlu/projects.htm)
  - Final lecture notes and readings (for all topics)
Longer Memory Course (~18 hours)

- **Tu Wien 2019**
  - Memory Systems and Memory-Centric Computing Systems
  - Taught by Onur Mutlu June 12-19, 2019
  - ~18 hours of lectures

- **Website for the Course including Videos, Slides, Papers**
  - [https://www.youtube.com/playlist?list=PL5Q2soXY2Zi_gntM55VoMIKlw7YrXOhbl](https://www.youtube.com/playlist?list=PL5Q2soXY2Zi_gntM55VoMIKlw7YrXOhbl)

- **All Papers are at:**
  - [https://people.inf.ethz.ch/omutlu/projects.htm](https://people.inf.ethz.ch/omutlu/projects.htm)
  - Final lecture notes and readings (for all topics)
Some Overview Talks

- Future Computing Architectures
  - [https://www.youtube.com/watch?v=kgiZlSOcGFM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=1](https://www.youtube.com/watch?v=kgiZlSOcGFM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=1)

- Enabling In-Memory Computation
  - [https://www.youtube.com/watch?v=oHqsNbxgdzM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=7](https://www.youtube.com/watch?v=oHqsNbxgdzM&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=7)

- Accelerating Genome Analysis
  - [https://www.youtube.com/watch?v=hPnSmfwu2-A&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=9](https://www.youtube.com/watch?v=hPnSmfwu2-A&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=9)

- Rethinking Memory System Design
  - [https://www.youtube.com/watch?v=F7xZLNMIY1E&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=3](https://www.youtube.com/watch?v=F7xZLNMIY1E&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=3)
Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"


[arXiv version]

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]
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
Onur Mutlu and Jeremie Kim, "RowHammer: A Retrospective"


[Preliminary arXiv version]

RowHammer: A Retrospective

Onur Mutlu§‡, Jeremie S. Kim‡§

§ETH Zürich ‡Carnegie Mellon University
Related Videos and Course Materials (I)

- 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-SAFA/RI/MQSim
- Mosaic – GPU Simulator Supporting Concurrent Applications
  - https://github.com/CMU-SAFA/RI/Mosaic
- IMPICA – Processing in 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFA/RI/IMPICA
- SMLA – Detailed 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFA/RI/SMLA
- HWASim – Simulator for Heterogeneous CPU-HWA Systems
  - https://github.com/CMU-SAFA/RI/HWASim

- Other open-source software from my group
  - https://github.com/CMU-SAFA/RI/
  - 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/
  - 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

Ramlulator: 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×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 × 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 × 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 × 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 × 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 × 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 × 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 × 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 × 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 × 8*</td>
<td>1</td>
<td>127.2</td>
</tr>
</tbody>
</table>

Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards

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

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim¹  Weikun Yang¹,²  Onur Mutlu¹
¹Carnegie Mellon University  ²Peking University