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
20 November 2018
BSC & UPC
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

**Graphic Elements:**
- Heterogeneous Processors and Accelerators
- Hybrid Main Memory
- Persistent Memory/Storage
- Graphics and Vision Processing

**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**
A Motivating Detour: Genome Sequence Analysis
Our Dream

- An embedded device that can perform comprehensive genome analysis in real time (within a minute)
  - Which of these DNAs does this DNA segment match with?
  - What is the likely genetic disposition of this patient to this drug?
  - . . .
What Is a Genome Made Of?

The chromosome is made up of genes

The genes consist of DNA

Chromosome - 23 pairs

Nucleus

Cell

SAFARI The discovery of DNA’s double-helical structure (Watson+, 1953)
DNA Under Electron Microscope

human chromosome #12 from HeLa’s cell
DNA Sequencing

- **Goal:**
  - Find the complete sequence of A, C, G, T’s in DNA.

- **Challenge:**
  - There is no machine that takes long DNA as an input, and gives the complete sequence as output.
  - All sequencing machines chop DNA into pieces and identify relatively small pieces (but not how they fit together).
Untangling Yarn Balls & DNA Sequencing
Genome sequencers... and more! All produce data with different properties.
**Genome Analysis**

1. **Sequencing**

   - Reference: `TTTATCGCTTCCATGACGCAG`
   - Read1: `ATCGC ATCC`
   - Read2: `TATCGC ATC`
   - Read3: `CATCCATGA`
   - Read4: `CGCTTCCAT`
   - Read5: `CCATGACGC`
   - Read6: `TTCCATGAC`

2. **Read Mapping**

3. **Variant Calling**

4. **Scientific Discovery**
Genome Sequence Alignment: Example

Source: By Aaron E. Darling, István Miklós, Mark A. Ragan - Figure 1 from Darling AE, Miklós I, Ragan MA (2008). "Dynamics of Genome Rearrangement in Bacterial Populations". PLOS Genetics. DOI:10.1371/journal.pgen.1000128., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=30550950
Sequencing

Read Mapping

Variant Calling

Scientific Discovery

300 M bases/min

Bottlenecked in Mapping!!

Illumina HiSeq4000

2 M bases/min

(0.6%)
Hash Table Based Read Mappers

- Guaranteed to find all mappings → sensitive
- Can tolerate up to $e$ errors

Personalized copy number and segmental duplication maps using next-generation sequencing

Can Alkan$^{1,2}$, Jeffrey M Kidd$^1$, Tomas Marques-Bonet$^{1,3}$, Gozde Aksay$^1$, Francesca Antonacci$^1$, Fereydoun Hormozdiari$^4$, Jacob O Kitzman$^1$, Carl Baker$^1$, Maika Malig$^1$, Onur Mutlu$^5$, S Cenk Sahinalp$^4$, Richard A Gibbs$^6$ & Evan E Eichler$^{1,2}$

Read Mapping Execution Time Breakdown

- SAM printing: 3%
- candidate alignment locations (CAL): 4%
- Alignment (Edit-distance comp): 100%
Idea

Filter fast before you align

Minimize costly “approximate string comparisons”
Our First Filter: Pure Software Approach

- Download source code and try for yourself
  - Download link to FastHASH

Xin et al. BMC Genomics 2013, 14(Suppl 1):S13
http://www.biomedcentral.com/1471-2164/14/S1/S13

Accelerating read mapping with FastHASH

Hongyi Xin¹, Donghyuk Lee¹, Farhad Hormozdiari², Samihan Yedkar¹, Onur Mutlu¹*, Can Alkan³*

From The Eleventh Asia Pacific Bioinformatics Conference (APBC 2013)
Vancouver, Canada. 21-24 January 2013
Sequence analysis

**Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping**

Hongyi Xin¹,* , John Greth² , John Emmons² , Gennady Pekhimenko¹ , Carl Kingsford³ , Can Alkan⁴,* and Onur Mutlu²,*

Xin+, "**Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping**", Bioinformatics 2015.
An Example Solution: GateKeeper

1. High throughput DNA sequencing (HTS) technologies
2. Read Pre-Alignment Filtering Fast & Low False Positive Rate
3. Read Alignment Slow & Zero False Positives
FPGA-Based Alignment Filtering

- Mohammed Alser, Hasan Hassan, Hongyi Xin, Oguz Ergin, Onur Mutlu, and Can Alkan

"GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping"

*Bioinformatics*, [published online, May 31], 2017.

[Source Code]
[Online link at Bioinformatics Journal]

---

**GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping**

Mohammed Alser, Hasan Hassan, Hongyi Xin, Oguz Ergin, Onur Mutlu, and Can Alkan

*Bioinformatics*, Volume 33, Issue 21, 1 November 2017, Pages 3355–3363,

https://doi.org/10.1093/bioinformatics/btx342

**Published:** 31 May 2017  **Article history ▼**
DNA Read Mapping & Filtering

- **Problem:** Heavily bottlenecked by Data Movement

- GateKeeper FPGA performance limited by DRAM bandwidth [Alser+, Bioinformatics 2017]

- Ditto for SHD on SIMD [Xin+, Bioinformatics 2015]

- **Solution:** Processing-in-memory can alleviate the bottleneck

- However, we need to design mapping & filtering algorithms to fit processing-in-memory
In-Memory DNA Sequence Analysis

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"
  
  _BMC Genomics_, 2018.

  Proceedings of the _16th Asia Pacific Bioinformatics Conference_ (APBC), Yokohama, Japan, January 2018.

  arxiv.org Version (pdf)

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

Jeremie S. 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

SAFARI
Quick Note: Key Principles and Results

- **Two key principles:**
  - Exploit the structure of the genome to minimize computation
  - Morph and exploit the structure of the underlying hardware to maximize performance and efficiency

- **Algorithm-architecture co-design** for DNA read mapping
  - *Speeds up* read mapping by \(~200\text{x}\ (\text{sometimes more})*
  - *Improves accuracy* of read mapping in the presence of errors

Kim et al., “Genome Read In-Memory (GRIM) Filter,” BMC Genomics 2018.
Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions

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

Briefings in Bioinformatics, bby017, https://doi.org/10.1093/bib/bby017
Published: 02 April 2018  Article history ▼

Nanopore Genome Assembly Pipeline

Figure 1. The analyzed genome assembly pipeline using nanopore sequence data, with its five steps and the associated tools for each step.

More on Genome Analysis: Another Talk

Accelerating Genome Analysis
A Primer on an Ongoing Journey

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
May 21, 2018
HiCOMB-17 Keynote Talk
Recall Our Dream

- An embedded device that can perform comprehensive genome analysis in real time (within a minute)

- Still a long ways to go
  - Energy efficiency
  - Performance (latency)
  - Security
  - Huge memory bottleneck
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!
Example: Capacity, Bandwidth & Latency

Memory latency remains almost constant
DRAM Latency Is Critical for Performance

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Bandwidth</th>
<th>Smaller Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Xpoint)</td>
<td>lower latency</td>
<td>higher cost</td>
</tr>
<tr>
<td>(e.g., DL-DRAM, Ti-DRAM)</td>
<td>lower power</td>
<td>higher cost</td>
</tr>
<tr>
<td>(e.g., LPDDR3, LPDDR4)</td>
<td>larger capacity</td>
<td>lower endurance</td>
</tr>
</tbody>
</table>
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>Reduced-Latency DRAM (e.g., RL/TL-DRAM, FLY-RAM)</td>
<td>lower latency</td>
<td>higher cost</td>
</tr>
<tr>
<td>Low-Power DRAM (e.g., LPDDR3, LPDDR4, Voltron)</td>
<td>lower power</td>
<td>higher latency higher cost</td>
</tr>
<tr>
<td>Non-Volatile Memory (NVM) (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

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
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”
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.

![Graph showing relative server failure rate vs. 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](https://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	extsuperscript{1,2,3} Nandita Vijaykumar	extsuperscript{3} Samira Khan	extsuperscript{4,3} Saugata Ghose	extsuperscript{3} Kevin Chang	extsuperscript{3} Gennady Pekhimenko	extsuperscript{5,3} Donghyuk Lee	extsuperscript{6,3} Oguz Ergin	extsuperscript{2} Onur Mutlu	extsuperscript{1,3}

\textsuperscript{1}ETH Zürich \textsuperscript{2}TOBB University of Economics & Technology \textsuperscript{3}Carnegie Mellon University
\textsuperscript{4}University of Virginia \textsuperscript{5}Microsoft Research \textsuperscript{6}NVIDIA Research
Data Retention in Memory [Liu et al., ISCA 2013]

- Retention Time Profile of DRAM looks like this:

  64-128ms

  >256ms

  128-256ms

  Location dependent
  Stored value pattern dependent
  Time dependent
A Curious Discovery [Kim et al., ISCA 2014]

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

Errors per $10^9$ Cells

First Appearance

Module Vintage
Recent DRAM Is More Vulnerable

All modules from 2012–2013 are vulnerable
A Simple Program Can Induce Many Errors

**loop:**

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

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

1. Avoid *cache hits*
   – Flush $X$ from cache

2. Avoid *row hits* to $X$
   – Read $Y$ in another row

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

```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

```
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

Kim+，“Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” ISCA 2014.
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...

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

Project Zero

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
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 ...

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 (II)

“Can gain control of a smart phone deterministically”

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms, CCS’16
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
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.

THROWHAMMER —

Throwhammer: Rowhammer Attacks over the Network and Defenses

Andrei Tatar
VU Amsterdam

Radhesh Krishnan
VU Amsterdam

Elias Athanasopoulos
University of Cyprus

Kaveh Razavi
VU Amsterdam

Cristiano Giuffrida
VU Amsterdam

Herbert Bos
VU Amsterdam
More Security Implications (V)

- Rowhammer over RDMA (II)

The Hacker News
Security in a serious way

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

- [link] 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/isislovecruft/status/1021939922754723841
Probabilistic Activation in Real Life (II)

https://twitter.com/isislovecruft/status/1021939922754723841
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu


[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Source Code and Data]
Future of Memory Reliability

- 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

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

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]
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)
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** → 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)

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):

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

Dally, HiPEAC 2015

- 64-bit DP: 20 pJ
- 256-bit buses
- 256-bit access: 8 kB SRAM
- 26 pJ
- 256 pJ
- 16 nJ (DRAM Rd/Wr)
- 500 pJ
- 50 pJ
- 1 nJ

Efficient off-chip link

SAFARI
Data Movement vs. Computation Energy

Communication Dominates Arithmetic

A memory access consumes ~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)
62.7% of the total system energy is spent on data movement.
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 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
- Many more
- VM Cloning
- Deduplication
- Page Migration

SAFARI
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

Step 2:
Activate row B

DRAM subarray

Row Buffer (4 Kbytes)

Transfer row

Transfer row
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

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

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
```
Performance: Bitmap Index on Ambit

Performance: BitWeaving on Ambit

`select count(*) from T where c1 <= val <= c2`
More on In-DRAM Bulk AND/OR


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

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
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)

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

![Graph showing speedup with 32 and 128 cores](chart.png)
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.
Communications via Remote Function Calls

Message Queue
Tesseract System for Graph Processing

- Logic
- Memory

Crossbar Network

- DRAM Controller
- NI
- In-Order Core
- Message Queue
- PF Buffer
- LP
- Prefetching

Host Processor

Memory-Mapped Accelerator Interface

(Noncacheable, Physically Addressed)

Prefetching

- LP
- PF Buffer
- MTP
Communications In Tesseract (I)
Remote Function Call (Non-Blocking)
Evaluated Systems

- **DDR3-Oo**:
  - 8 OoO 4GHz
  - 8 OoO 4GHz
  - 8 OoO 4GHz
  - 102.4GB/s

- **HMC-Oo**: 640GB/s
  - 8 OoO 4GHz
  - 8 OoO 4GHz
  - 8 OoO 4GHz

- **HMC-MC**: 640GB/s
  - 128 In-Order 2GHz
  - 128 In-Order 2GHz

**Tesseract**

- 32 Tesseract Cores

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

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

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

> 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)]
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}  Rahul Thakur\textsuperscript{3}  Parthasarathy Ranganathan\textsuperscript{3}  Youngsok Kim\textsuperscript{2}  Daehyun Kim\textsuperscript{4,3}  Onur Mutlu\textsuperscript{5,1}

\textsuperscript{1}Google, \textsuperscript{2}Korea University, \textsuperscript{3}University of California, \textsuperscript{4}IBM, \textsuperscript{5}University of Washington
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
Consumer Devices

Consumer devices are everywhere!

Energy consumption is a first-class concern in consumer devices.
Popular Google Consumer 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
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.
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

Small embedded low-power core  
PIM Core

Small fixed-function accelerators  
PIM Accelerator

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

Video Playback
Google’s video codec

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
Quantization

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

Based on our analysis, we conclude that:
- Both functions are good candidates for PIM execution.
- It is feasible to implement them in PIM logic.

A simple data conversion operation that requires shift, addition, and multiplication operations.
Evaluation Methodology

- **System Configuration (gem5 Simulator)**
  - **SoC**: 4 OoO cores, 8-wide issue, 64 kB L1cache, 2MB L2 cache
  - **PIM Core**: 1 core per vault, 1-wide issue, 4-wide SIMD, 32kB L1 cache
  - **3D-Stacked Memory**: 2GB cube, 16 vaults per cube
    - Internal Bandwidth: 256GB/S
    - Off-Chip Channel Bandwidth: 32 GB/s
  - **Baseline Memory**: LPDDR3, 2GB, FR-FCFS scheduler

- *We study each target in isolation and emulate each separately and run them in our simulator*
Normalized Energy

77.7% and 82.6% of energy reduction for texture tiling and packing comes from eliminating data movement.

PIM core and PIM accelerator reduces 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%**
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

ASPLOS 2018
62.7% of the total system energy is spent on data movement
Truly Distributed GPU Processing with PIM?

```
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_idxs = 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)


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

Ashutosh Pattnaik\textsuperscript{1} \hspace{0.2cm} Xulong Tang\textsuperscript{1} \hspace{0.2cm} Adwait Jog\textsuperscript{2} \hspace{0.2cm} Onur Kayiran\textsuperscript{3} \hspace{0.2cm} Asit K. Mishra\textsuperscript{4} \hspace{0.2cm} Mahmut T. Kandemir\textsuperscript{1} \hspace{0.2cm} Onur Mutlu\textsuperscript{5,6} \hspace{0.2cm} Chita R. Das\textsuperscript{1}

\textsuperscript{1}Pennsylvania State University \hspace{0.2cm} \textsuperscript{2}College of William and Mary \hspace{0.2cm} \textsuperscript{3}Advanced Micro Devices, Inc. \hspace{0.2cm} \textsuperscript{4}Intel Labs \hspace{0.2cm} \textsuperscript{5}ETH Zürich \hspace{0.2cm} \textsuperscript{6}Carnegie Mellon University
Accelerating Linked Data Structures


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

---

**Table 2: Baseline Simulation Configuration**

<table>
<thead>
<tr>
<th>Component</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>16 out-of-order cores, 4 GHz, 4-issue</td>
</tr>
<tr>
<td>L1 I/D-Cache</td>
<td>Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs</td>
</tr>
<tr>
<td>On-Chip Network</td>
<td>Crossbar, 2 GHz, 144-bit links</td>
</tr>
<tr>
<td>Main Memory</td>
<td>32 GB, 8 HMCs, daisy-chain (80 GB/s full-duplex)</td>
</tr>
<tr>
<td>HMC</td>
<td>4 GB, 16 vaults, 256 DRAM banks [20]</td>
</tr>
<tr>
<td>– DRAM</td>
<td>FR-FCFS, tCL = tRCD = tRP = 13.75 ns [27]</td>
</tr>
<tr>
<td>– Vertical Links</td>
<td>64 TSVs per vault with 2 Gb/s signaling rate [23]</td>
</tr>
</tbody>
</table>
Simpler PIM: PIM-Enabled Instructions

Automatic Code and Data Mapping


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

Kevin Hsieh† Eiman Ebrahimi† Gwangsun Kim* Niladrish Chatterjee† Mike O'Connor† Nandita Vijaykumar† Onur Mutlu§† Stephen W. Keckler†

†Carnegie Mellon University †NVIDIA *KAIST §ETH Zürich
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
Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

*The University of Texas at Austin    §ETH Zürich
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
Challenge and Opportunity for Future

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

Fundamentally High-Performance (Data-Centric) Computing Architectures
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 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

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>Program/Language</td>
</tr>
<tr>
<td>System Software</td>
</tr>
<tr>
<td><strong>SW/HW Interface</strong></td>
</tr>
<tr>
<td>Micro-architecture</td>
</tr>
<tr>
<td>Logic</td>
</tr>
<tr>
<td>Devices</td>
</tr>
<tr>
<td>Electrons</td>
</tr>
</tbody>
</table>
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


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)

SM (Streaming Multiprocessor)

Main GPU

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?


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


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

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

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

- Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu,
  "Concurrent Data Structures for Near-Memory Computing"
  [Slides (pptx) (pdf)]
Simulation Infrastructures for PIM

- Ramulator extended for PIM
  - Flexible and extensible DRAM simulator
  - Can model many different memory standards and proposals
  - https://github.com/CMU-SAFARI/ramulator

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim\textsuperscript{1} \quad Weikun Yang\textsuperscript{1,2} \quad Onur Mutlu\textsuperscript{1}
\textsuperscript{1}Carnegie Mellon University \quad \textsuperscript{2}Peking University
An FPGA-based Test-bed for PIM?


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

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]
New Applications and Use Cases for PIM

Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"

*BMC Genomics*, 2018.
Proceedings of the *16th Asia Pacific Bioinformatics Conference* (APBC), Yokohama, Japan, January 2018.
arxiv.org Version (pdf)

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

Jeremie S. 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
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]

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

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

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

- Pre-paradigm science: no clear consensus in the field
- Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
- Revolutionary science: underlying assumptions re-examined
Computer Architecture Today

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

- You can invent new paradigms for computation, communication.

- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined

225
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
Four Key Directions

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


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

Fundamentally 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
One Important 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
Principled Design
Another Principled Design

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

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?

Source: http://spectrum.ieee.org/image/MjYzMzAyMg.jpeg
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
For Some Open Problems, See

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]

SAFARI

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://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomt hrpDpMJm05P6J9x](https://www.youtube.com/playlist?list=PL5Q2soXY2Zi-HXxomt hrpDpMJm05P6J9x)

- 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)
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
20 November 2018
BSC & UPC
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://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)
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

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu, "Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions"
[Preliminary arxiv.org version]

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)]
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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

- All are available at

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

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

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]
Ramulator Motivation

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

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

Table 1. Landscape of DRAM-based memory
Ramulator

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

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

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate (MT/s)</th>
<th>Timing (CL-RCD-RP)</th>
<th>Data-Bus (Width×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

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

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim\textsuperscript{1} \quad Weikun Yang\textsuperscript{1,2} \quad Onur Mutlu\textsuperscript{1}
\textsuperscript{1}Carnegie Mellon University \quad \textsuperscript{2}Peking University
Tesseract: Extra Slides
Communications In Tesseract (I)
Communications In Tesseract (III)
Remote Function Call (Non-Blocking)
Effect of Bandwidth & Programming Model

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

Programming Model

- HMC-MC Bandwidth
- HMC-MC + PIM BW
- Tesseract + Conventional BW
- Tesseract (No Prefetching)

Speedup
Reducing Memory Latency
Main Memory Latency Lags Behind

Memory latency remains almost constant
A Closer Look …

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

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

Graph/Tree Processing

Datacenter Workloads
[Kanev+ (Google), ISCA’15]

Long memory latency → performance bottleneck
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
  - ...
Latency Variation in Memory Chips

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


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

  [Link to GitHub repository](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)
AL-DRAM improves single-core performance on a real system
AL-DRAM: Multi-Core Evaluation

AL-DRAM provides higher performance on multi-programmed & multi-threaded workloads

Performance Improvement

Average Improvement

10.4%

all-35-workload
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]
Heterogeneous Latency within A Chip

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]
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.

→ *Dynamic & low cost* latency optimization
**DIVA Online Profiling**

Design-Induced-Variation-Aware

- Slow cells
  - Process variation
  - Random error
- Error-correcting code

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

And, What If …

- … we can sacrifice reliability of some data to access it with even lower latency?
The DRAM Latency PUF:

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

Jeremie S. Kim    Minesh Patel
Hasan Hassan      Onur Mutlu
Motivation

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

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

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

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

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

Low % chance to fail with reduced $t_{RCD}$
DRAM Accesses and Failures

Process variation during manufacturing leads to cells having unique characteristics.

Bitline Charge Sharing

Ready to Access Voltage Level

-wordline
-access transistor
-capacitor
-bitline

Bitline Voltage

0.5 \( V_{dd} \)

\( V_{dd} \)

\( V_{min} \)

Time

\( t_{RCD} \)

SAFARI
DRAM Accesses and Failures

Bitline Voltage

$V_{dd}$

$V_{min}$

$0.5 \cdot V_{dd}$

Ready to Access Voltage Level

 weaker cells have a higher probability to fail

SAFAI
The DRAM Latency PUF Evaluation

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

• The latency error patterns satisfy PUF requirements

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

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

Jeremie S. Kim    Minesh Patel
Hasan Hassan     Onur Mutlu

HPCA 2018

QR Code for the paper
DRAM Latency PUFs

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"
[Lightning Talk Video]
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
Tiered Latency DRAM
What Causes the Long Latency?

DRAM Latency = Subarray Latency + I/O Latency

Dominant Subarray Latency
Why is the Subarray So Slow?

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

Long Bitline

Short Bitline

Faster

Smaller

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

- **Cheaper**
  - GOAL
  - Fancy DRAM
    - Short Bitline
    - 32 cells/bitline
  - Commodity DRAM
    - Long Bitline
    - 64 cells/bitline
    - 128 cells/bitline
    - 256 cells/bitline
    - 512 cells/bitline

- **Faster**
Approximating the Best of Both Worlds

- Long Bitline: Small Area, High Latency
- Our Proposal: Add Isolation Transistors, Short Bitline, Fast
- Short Bitline: Large Area, Low Latency

Need Isolation
Add Isolation Transistors
Long Bitline ➔ Fast
Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM Short Bitline

Small Area

Low Latency

High Latency

Small Area

Low Latency

Large Area

Low Latency

Small area using long bitline

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

- DRAM Latency (tRC) -56%
- DRAM Power +49%
- Power +23%
- DRAM Area Overhead ~3%: mainly due to the isolation transistors

Latency

(52.5ns)

Commodity DRAM
Near TL-DRAM
Far TL-DRAM

Commodity DRAM
Near TL-DRAM
Far TL-DRAM
Trade-Off: Area (Die-Area) vs. Latency

- Cheaper
- Faster

Near Segment

Far Segment

512 cells/bitline

GOAL

32, 64, 128, 256
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, 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>

Figure 2. Performance comparison of DRAM standards

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


- Source code is released under the liberal MIT License
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)
A Deeper Dive into DRAM Reliability Issues
There are Two Other Solution Directions

- **New Technologies:** Replace or (more likely) augment DRAM with a different technology
  - Non-volatile memories

- **Embracing Un-reliability:** Design memories with different reliability and store data intelligently across them

- ...
Exploiting Memory Error Tolerance with Hybrid Memory Systems

Vulnerable data

Tolerant data

Reliable memory

Low-cost memory

On Microsoft’s Web Search workload
Reduces server hardware cost by 4.7 %
Achieves single server availability target of 99.90 %

Heterogeneous-Reliability Memory [DSN 2014]
More on Heterogeneous-Reliability Memory

- Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory" Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Atlanta, GA, June 2014. [Summary] [Slides (pptx) (pdf)] [Coverage on ZDNet]
Root Causes of Disturbance Errors

• *Cause 1: Electromagnetic coupling*
  – Toggling the wordline voltage briefly increases the voltage of adjacent wordlines
  – Slightly opens adjacent rows → Charge leakage

• *Cause 2: Conductive bridges*

• *Cause 3: Hot-carrier injection*

*Confirmed by at least one manufacturer*
RowHammer Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper
7. Solution Space

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

Note: For three modules with the most errors (only first bank)

Most aggressors & victims are adjacent
Note: For three modules with the most errors (only first bank)

Less frequent accesses → Fewer errors
2 Refresh Interval

Note: Using three modules with the most errors (only first bank)

More frequent refreshes → Fewer errors
### Data Pattern

<table>
<thead>
<tr>
<th>Solid</th>
<th>RowStripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111</td>
<td>1111111</td>
</tr>
<tr>
<td>11111111</td>
<td>0000000</td>
</tr>
<tr>
<td>11111111</td>
<td>1111111</td>
</tr>
<tr>
<td>11111111</td>
<td>0000000</td>
</tr>
<tr>
<td>~Solid</td>
<td>~RowStripe</td>
</tr>
<tr>
<td>0000000</td>
<td>0000000</td>
</tr>
<tr>
<td>0000000</td>
<td>1111111</td>
</tr>
<tr>
<td>0000000</td>
<td>0000000</td>
</tr>
<tr>
<td>0000000</td>
<td>1111111</td>
</tr>
</tbody>
</table>

*Errors affected by data stored in other cells*
6. Other Results (in Paper)

• *Victim Cells ≠ Weak Cells (i.e., leaky cells)*
  – Almost no overlap between them

• *Errors not strongly affected by temperature*
  – Default temperature: 50°C
  – At 30°C and 70°C, number of errors changes <15%

• *Errors are repeatable*
  – Across ten iterations of testing, >70% of victim cells had errors in every iteration
6. Other Results (in Paper) cont’d

• As many as 4 errors per cache-line
  – Simple ECC (e.g., SECDED) cannot prevent all errors

• Number of cells & rows affected by aggressor
  – Victims cells per aggressor: \( \leq 110 \)
  – Victims rows per aggressor: \( \leq 9 \)

• Cells affected by two aggressors on either side
  – Very small fraction of victim cells (<100) have an error when either one of the aggressors is toggled
Some Potential Solutions

- Make better DRAM chips  Cost
- Refresh frequently  Power, Performance
- Sophisticated ECC  Cost, Power
- Access counters  Cost, Power, Complexity
Naive Solutions

1. **Throttle accesses to same row**
   - Limit access-interval: \( \geq 500\text{ns} \)
   - Limit number of accesses: \( \leq 128\text{K} \) (=64ms/500ns)

2. **Refresh more frequently**
   - Shorten refresh-interval by \( \sim 7x \)

*Both naive solutions introduce significant overhead in performance and power*
Apple’s Patch for RowHammer


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 and Lenovo 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
  – Enough slack in timing 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
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]
Retrospective on RowHammer & Future

Onur Mutlu,
"The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"
[Slides (pptx) (pdf)]
Fundamentally
Secure, Reliable, Safe
Computing Architectures
Future of Main Memory

- DRAM is becoming less reliable → more vulnerable
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]

Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field

Justin Meza  Qiang Wu*  Sanjeev Kumar*  Onur Mutlu
Carnegie Mellon University  * Facebook, Inc.

SAFARI
DRAM Reliability Reducing

Intuition: quadratic increase in capacity

Graph:
- Y-axis: Relative server failure rate
- X-axis: Chip density (Gb)
- Points:
  - (1, 0.25)
  - (2, 0.5)
  - (4, 1.0)

The graph shows a quadratic increase in the relative server failure rate with an increase in chip density.
Aside: SSD Error Analysis in the Field

- First large-scale field study of flash memory errors

Future of Main Memory

- DRAM is becoming less reliable → more vulnerable

- Due to difficulties in DRAM scaling, other problems may also appear (or they may be going unnoticed)

- Some errors may already be slipping into the field
  - Read disturb errors (Rowhammer)
  - Retention errors
  - Read errors, write errors
  - ...

- These errors can also pose security vulnerabilities
DRAM Data Retention Time Failures

- Determining the data retention time of a cell/row is getting more difficult

- Retention failures may already be slipping into the field
Analysis of Retention Failures [ISCA’13]

Two Challenges to Retention Time Profiling

- Data Pattern Dependence (DPD) of retention time
- Variable Retention Time (VRT) phenomenon
Two Challenges to Retention Time Profiling

- Challenge 1: Data Pattern Dependence (DPD)
  - Retention time of a DRAM cell depends on its value and the values of cells nearby it
  - When a row is activated, all bitlines are perturbed simultaneously
Data Pattern Dependence

- Electrical noise on the bitline affects reliable sensing of a DRAM cell.
- The magnitude of this noise is affected by values of nearby cells via:
  - Bitline-bitline coupling → electrical coupling between adjacent bitlines
  - Bitline-wordline coupling → electrical coupling between each bitline and the activated wordline
Data Pattern Dependence

- Electrical noise on the bitline affects reliable sensing of a DRAM cell
- The magnitude of this noise is affected by values of nearby cells via
  - Bitline-bitline coupling → electrical coupling between adjacent bitlines
  - Bitline-wordline coupling → electrical coupling between each bitline and the activated wordline

- Retention time of a cell depends on data patterns stored in nearby cells
  → need to find the worst data pattern to find worst-case retention time
  → this pattern is location dependent
Two Challenges to Retention Time Profiling

- Challenge 2: Variable Retention Time (VRT)
  - Retention time of a DRAM cell changes randomly over time
    - a cell alternates between multiple retention time states
  - Leakage current of a cell changes sporadically due to a charge trap in the gate oxide of the DRAM cell access transistor
  - When the trap becomes occupied, charge leaks more readily from the transistor’s drain, leading to a short retention time
    - Called Trap-Assisted Gate-Induced Drain Leakage
  - This process appears to be a random process
  - Worst-case retention time depends on a random process
    → need to find the worst case despite this

  

[Kim+ IEEE TED’11]
Newer device families have more weak cells than older ones. Likely a result of technology scaling.
An Example VRT Cell

A cell from E 2Gb chip family
Variable Retention Time

- Many failing cells jump from very high retention time to very low.
- Most failing cells exhibit VRT.
- Min ret time = Max ret time, Expected if no VRT.
- A 2Gb chip family.
More on DRAM Retention Analysis

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
Industry Is Writing Papers About It, Too

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
Mitigation of Retention Issues [SIGMETRICS’14]

- Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa Alameldeen, Chris Wilkerson, and Onur Mutlu,

"The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study"

Handling Data-Dependent Failures [DSN'16]

- Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa Alameldeen, Chris Wilkerson, and Onur Mutlu,

"The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study"

Handling Data-Dependent Failures [CAL’16]

- Samira Khan, Chris Wilkerson, Donghyuk Lee, Alaa R. Alameldeen, and Onur Mutlu,
  "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM"
Handling Variable Retention Time [DSN’15]

- Moinuddin Qureshi, Dae Hyun Kim, Samira Khan, Prashant Nair, and Onur Mutlu,
  "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems"
  Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015.
  [Slides (pptx) (pdf)]
Handling Both DPD and VRT [ISCA’17]


- First experimental analysis of (mobile) LPDDR4 chips
- Analyzes the complex tradeoff space of retention time profiling
- Key idea: enable fast and robust profiling at higher refresh intervals & temp.
Memory reliability is reducing
Reliability issues open up security vulnerabilities
- Very hard to defend against
Rowhammer is an example
- Its implications on system security research are tremendous & exciting

**Good news: We have a lot more to do.**

**Understand:** Solid methodologies for failure modeling and discovery
- Modeling based on real device data – small scale and large scale

**Architect:** Principled co-architecting of system and memory
- Good partitioning of duties across the stack

**Design & Test:** Principled electronic design, automation, testing
- High coverage and good interaction with system reliability methods
If Time Permits: NAND Flash Vulnerabilities

Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
"Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives"

_to appear in Proceedings of the IEEE, 2017._

Meza+, “A Large-Scale Study of Flash Memory Errors in the Field,” SIGMETRICS 2015.

Overview Paper on Flash Reliability

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
  "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives"
Challenge and Opportunity for Future

Fundamentally Secure, Reliable, Safe Computing Architectures
NAND Flash Memory
Reliability and Security
Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,
"Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives"
Evolution of NAND Flash Memory

- Flash memory is widening its range of applications
  - Portable consumer devices, laptop PCs and enterprise servers

Seaung Suk Lee, “Emerging Challenges in NAND Flash Technology”, Flash Summit 2011 (Hynix)
Flash Challenges: Reliability and Endurance

- **P/E cycles (provided)**
  - A few thousand

- **P/E cycles (required)**
  - Writing the full capacity of the drive 10 times per day for 5 years (STEC)
  - > 50k P/E cycles

E. Grochowski et al., “Future technology challenges for NAND flash and HDD products”, Flash Memory Summit 2012
NAND Flash Memory is Increasingly Noisy

Write → Read
Future NAND Flash-based Storage Architecture

Our Goals:
Build reliable error models for NAND flash memory
Design efficient reliability mechanisms based on the model
NAND Flash Error Model

Experimentally characterize and model dominant errors

Luo et al., “Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory,” JSAC 2016

Cai et al., “Neighbor-Cell Assisted Error Correction in MLC NAND Flash Memories”, SIGMETRICS 2014
Cai et al., “Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery”, HPCA 2015

Cai et al., “Error Analysis and Retention-Aware Error Management for NAND Flash Memory, ITJ 2013

Write

- Erase block
- Program page
- Neighbor page prog/read (c-to-c interference)
- Retention

Read
Our Goals and Approach

Goals:
- Understand error mechanisms and develop reliable predictive models for MLC NAND flash memory errors
- Develop efficient error management techniques to mitigate errors and improve flash reliability and endurance

Approach:
- Solid experimental analyses of errors in real MLC NAND flash memory → drive the understanding and models
- Understanding, models, and creativity → drive the new techniques
Experimental Testing Platform

Cai et al., FPGA-based Solid-State Drive prototyping platform, FCCM 2011.
NAND Flash Error Types

- Four types of errors [Cai+, DATE 2012]

- Caused by **common flash operations**
  - Read errors
  - Erase errors
  - Program (interference) errors

- Caused by flash **cell losing charge over time**
  - Retention errors
    - Whether an error happens depends on required retention time
    - Especially problematic in MLC flash because threshold voltage window to determine stored value is smaller
Observations: Flash Error Analysis

- Raw bit error rate increases exponentially with P/E cycles.
- Retention errors are dominant (>99% for 1-year retention time).
- Retention errors increase with retention time requirement.

Cai et al., Error Patterns in MLC NAND Flash Memory, DATE 2012.
More on Flash Error Analysis

Solution to Retention Errors

- Refresh periodically
- Change the period based on P/E cycle wearout
  - Refresh more often at higher P/E cycles
- Use a combination of in-place and remapping-based refresh
One Issue: Read Disturb in Flash Memory

- All scaled memories are prone to read disturb errors
NAND Flash Memory Background

Flash Memory

Block 0
- Read
- Pass
- Pass
- Pass

... 

Block N

Flash Controller
Flash Cell Array
Flash Cell

Floating Gate Transistor (Flash Cell)

$V_{th} = 2.5\, \text{V}$
Flash Read

\[ V_{\text{read}} = 2.5 \, \text{V} \]

\[ V_{\text{th}} = 2 \, \text{V} \]

\[ V_{\text{th}} = 3 \, \text{V} \]
Flash Pass-Through

\[ V_{\text{pass}} = 5 \text{ V} \]

\[ V_{\text{th}} = 2 \text{ V} \]

\[ V_{\text{pass}} = 5 \text{ V} \]

\[ V_{\text{th}} = 3 \text{ V} \]
Read from Flash Cell Array

V_{\text{pass}} = 5.0

Pass (5V)

Page 1

V_{\text{read}} = 2.5

Read (2.5V)

Page 2

V_{\text{pass}} = 5.0

Pass (5V)

Page 3

V_{\text{pass}} = 5.0

Pass (5V)

Page 4

Correct values for page 2:

0 0 1 1
Read Disturb Problem: “Weak Programming”

Effect

Page 1
Pass (5V)

Page 2
Pass (5V)

Page 3
Read (2.5V)

Page 4
Pass (5V)

Repeatedly read page 3 (or any page other than page 2)
Read Disturb Problem: “Weak Programming” Effect

High pass-through voltage induces “weak-programming” effect.
Executive Summary

• **Read disturb errors** limit flash memory lifetime today
  – Apply a *high pass-through voltage* \( V_{\text{pass}} \) to multiple pages on a read
  – Repeated application of \( V_{\text{pass}} \) can alter stored values in unread pages

• We **characterize read disturb** on real NAND flash chips
  – Slightly lowering \( V_{\text{pass}} \) greatly reduces read disturb errors
  – Some flash cells are more prone to read disturb

• **Technique 1:** Mitigate read disturb errors online
  – \( V_{\text{pass}} \) **Tuning** dynamically finds and applies a lowered \( V_{\text{pass}} \) per block
  – Flash memory lifetime improves by 21%

• **Technique 2:** Recover after failure to prevent data loss
  – **Read Disturb Oriented Error Recovery** (RDR) selectively corrects cells more susceptible to read disturb errors
More on Flash Read Disturb Errors

- Yu Cai, Yixin Luo, Saugata Ghose, Erich F. Haratsch, Ken Mai, and Onur Mutlu,
  "Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation"
  Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015.
Large-Scale Flash SSD Error Analysis

- First large-scale field study of flash memory errors


[Slides (pptx) (pdf)] [Coverage at ZDNet] [Coverage on The Register] [Coverage on TechSpot] [Coverage on The Tech Report]
Another Time: NAND Flash Vulnerabilities

- Onur Mutlu,
  "Error Analysis and Management for MLC NAND Flash Memory"
  Technical talk at Flash Memory Summit 2014 (FMS), Santa Clara, CA, August 2014.
  Slides (ppt) (pdf)

Meza+, “A Large-Scale Study of Flash Memory Errors in the Field,” SIGMETRICS 2015.
Flash Memory Programming Vulnerabilities

- Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch,

"Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques"


[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
Other Works on Flash Memory
NAND Flash Error Model

Experimentally characterize and model dominant errors
Luo et al., “Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory”, JSAC 2016

Write

- Erase block
- Program page

→

Neighbor page prog/read (c-to-c interference)

→

Retention

Read

Cai et al., “Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques”, HPCA 2017
Cai et al., “Neighbor-Cell Assisted Error Correction in MLC NAND Flash Memories”, SIGMETRICS 2014
Cai et al., “Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation”, DSN 2015
Cai et al., “Error Analysis and Retention-Aware Error Management for NAND Flash Memory”, ITJ 2013
Cai et al., “Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery”, HPCA 2015
Neighbor-Assisted Error Correction

- Yu Cai, Gulay Yalcin, Onur Mutlu, Eric Haratsch, Osman Unsal, Adrian Cristal, and Ken Mai,
"Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories"
Data Retention

Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery"
[Slides (pptx) (pdf)]
SSD Error Analysis in the Field

- First large-scale field study of flash memory errors
- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, "A Large-Scale Study of Flash Memory Errors in the Field"
  [Slides (pptx) (pdf)] [Coverage at ZDNet] [Coverage on The Register] [Coverage on TechSpot] [Coverage on The Tech Report]
Flash Memory Programming Vulnerabilities

- Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch,

"Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques"
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
Accurate and Online Channel Modeling

More on DRAM Refresh
Tackling Refresh: Solutions

- **Parallelize refreshes with accesses** [Chang+ HPCA’14]

- **Eliminate unnecessary refreshes** [Liu+ ISCA’12]
  - Exploit device characteristics
  - Exploit data and application characteristics

- **Reduce refresh rate and detect+correct errors that occur** [Khan+ SIGMETRICS’14]

- **Understand retention time behavior in DRAM** [Liu+ ISCA’13]
Summary: Refresh-Access Parallelization

• DRAM refresh interferes with memory accesses
  – Degrades system performance and energy efficiency
  – Becomes exacerbated as DRAM density increases

• Goal: Serve memory accesses in parallel with refreshes to reduce refresh interference on demand requests

• Our mechanisms:
  – 1. Enable more parallelization between refreshes and accesses across different banks with new per-bank refresh scheduling algorithms
  – 2. Enable serving accesses concurrently with refreshes in the same bank by exploiting parallelism across DRAM subarrays

• Improve system performance and energy efficiency for a wide variety of different workloads and DRAM densities
  – 20.2% and 9.0% for 8-core systems using 32Gb DRAM at low cost
  – Very close to the ideal scheme without refreshes

Refresh Penalty

Refresh interferes with memory accesses

Refresh delays requests by 100s of ns
Per-bank refresh allows accesses to other banks while a bank is refreshing
Shortcomings of Per-Bank Refresh

• **Problem 1**: Refreshes to different banks are scheduled in a **strict round-robin order**
  – The static ordering is hardwired into DRAM chips
  – Refreshes busy banks with many queued requests when other banks are idle

• **Key idea**: Schedule per-bank refreshes to idle banks opportunistically in a dynamic order
Our First Approach: DARP

- **Dynamic Access-Refresh Parallelization (DARP)**
  - An improved scheduling policy for per-bank refreshes
  - Exploits refresh scheduling flexibility in DDR DRAM

- **Component 1: Out-of-order per-bank refresh**
  - Avoids poor static scheduling decisions
  - Dynamically issues per-bank refreshes to idle banks

- **Component 2: Write-Refresh Parallelization**
  - Avoids refresh interference on latency-critical reads
  - Parallelizes refreshes with a batch of writes
Shortcomings of Per-Bank Refresh

- **Problem 2**: Banks that are being refreshed cannot concurrently serve memory requests
Shortcomings of Per-Bank Refresh

• Problem 2: Refreshing banks cannot concurrently serve memory requests

• Key idea: Exploit subarrays within a bank to parallelize refreshes and accesses across subarrays
Methodology

100 workloads: SPEC CPU2006, STREAM, TPC-C/H, random access

System performance metric: Weighted speedup
Comparison Points

- **All-bank refresh** [DDR3, LPDDR3, …]
- **Per-bank refresh** [LPDDR3]
- **Elastic refresh** [Stuecheli et al., MICRO ‘10]:
  - Postpones refreshes by a time delay based on the predicted rank idle time to avoid interference on memory requests
  - Proposed to schedule all-bank refreshes without exploiting per-bank refreshes
  - Cannot parallelize refreshes and accesses within a rank
- **Ideal (no refresh)**
System Performance

- 7.9%
- 12.3%
- 20.2%

2. Consistent system performance improvement across DRAM densities (within 0.9%, 1.2%, and 3.8% of ideal)
Energy Efficiency

Consistent reduction on energy consumption
More Information on Refresh-Access Parallelization

Kevin Chang, Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu,
"Improving DRAM Performance by Parallelizing Refreshes with Accesses"
Proceedings of the 20th International Symposium on High-Performance Computer Architecture (HPCA), Orlando, FL, February 2014. [Summary] [Slides (pptx) (pdf)]
Tackling Refresh: Solutions

- Parallelize refreshes with accesses [Chang+ HPCA’14]

- Eliminate unnecessary refreshes [Liu+ ISCA’12]
  - Exploit device characteristics
  - Exploit data and application characteristics

- Reduce refresh rate and detect+correct errors that occur [Khan+ SIGMETRICS’14]

- Understand retention time behavior in DRAM [Liu+ ISCA’13]
Most Refreshes Are Unnecessary

Retention Time Profile of DRAM looks like this:

- 64-128ms
- >256ms
- 128-256ms
RAIDR: Eliminating Unnecessary Refreshes

1. Profiling:
   Profile the retention time of all DRAM rows.

2. Binning:
   Store rows into bins by retention time and use Bloom Filters for efficient and scalable storage.

3. Refreshing:
   Memory controller refreshes rows in different bins at different rates and probe Bloom Filters to determine refresh rate of a row.

Can reduce refreshes by ~75% and reduces energy consumption and improves performance.

Refresh control is in DRAM in today’s auto-refresh systems

RAIDR can be implemented in either the controller or DRAM
Overhead of RAIDR in DRAM controller:
1.25 KB Bloom Filters, 3 counters, additional commands issued for per-row refresh (all accounted for in evaluations)
Overhead of RAIDR in DRAM chip:
Per-chip overhead: 20B Bloom Filters, 1 counter (4 Gbit chip)
Total overhead: 1.25KB Bloom Filters, 64 counters (32 GB DRAM)
RAIDR: Results and Takeaways

- System: 32GB DRAM, 8-core; SPEC, TPC-C, TPC-H workloads
- RAIDR hardware cost: 1.25 kB (2 Bloom filters)
- Refresh reduction: 74.6%
- Dynamic DRAM energy reduction: 16%
- Idle DRAM power reduction: 20%
- Performance improvement: 9%
- Benefits increase as DRAM scales in density
RAIDR performance benefits increase with DRAM chip capacity

DRAM Device Capacity Scaling: Energy

RAIDR energy benefits increase with DRAM chip capacity

RAIDR: Eliminating Unnecessary Refreshes

Observation: Most DRAM rows can be refreshed much less often without losing data [Kim+, EDL’09][Liu+ ISCA’13]

Key idea: Refresh rows containing weak cells more frequently, other rows less frequently

1. Profiling: Profile retention time of all rows
2. Binning: Store rows into bins by retention time in memory controller
   Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
3. Refreshing: Memory controller refreshes rows in different bins at different rates

Results: 8-core, 32GB, SPEC, TPC-C, TPC-H
- 74.6% refresh reduction @ 1.25KB storage
- ~16%/20% DRAM dynamic/idle power reduction
- ~9% performance improvement
- Benefits increase with DRAM capacity

More on RAIDR

Tackling Refresh: Solutions

- Parallelize refreshes with accesses [Chang+ HPCA’14]

- Eliminate unnecessary refreshes [Liu+ ISCA’12]
  - Exploit device characteristics
  - Exploit data and application characteristics

- Reduce refresh rate and detect+correct errors that occur [Khan+ SIGMETRICS’14]

- Understand retention time behavior in DRAM [Liu+ ISCA’13]
Motivation: Understanding Retention

- Past works require **accurate and reliable measurement of retention time of each DRAM row**
  - To maintain data integrity while reducing refreshes

- **Assumption:** worst-case retention time of each row can be determined and stays the same at a given temperature
  - Some works propose writing all 1’s and 0’s to a row, and measuring the time before data corruption

- **Question:**
  - Can we reliably and accurately determine retention times of all DRAM rows?
Two Challenges to Retention Time Profiling

- Data Pattern Dependence (DPD) of retention time
- Variable Retention Time (VRT) phenomenon
An Example VRT Cell

A cell from E 2Gb chip family
VRT: Implications on Profiling Mechanisms

- Problem 1: There does not seem to be a way of determining if a cell exhibits VRT without actually observing a cell exhibiting VRT
  - VRT is a memoryless random process [Kim+ JJAP 2010]

- Problem 2: VRT complicates retention time profiling by DRAM manufacturers
  - Exposure to very high temperatures can induce VRT in cells that were not previously susceptible
    → can happen during soldering of DRAM chips
    → manufacturer’s retention time profile may not be accurate

- One option for future work: Use ECC to continuously profile DRAM online while aggressively reducing refresh rate
  - Need to keep ECC overhead in check
More on DRAM Retention Analysis

Tackling Refresh: Solutions

- **Parallelize refreshes with accesses** [Chang+ HPCA’14]

- **Eliminate unnecessary refreshes** [Liu+ ISCA’12]
  - Exploit device characteristics
  - Exploit data and application characteristics

- **Reduce refresh rate and detect+correct errors that occur** [Khan+ SIGMETRICS’14]

- **Understand retention time behavior in DRAM** [Liu+ ISCA’13]
Key Observations:

- **Testing** alone **cannot detect** all possible failures
- Combination of ECC and other mitigation techniques is much more **effective**
  - But degrades performance
- **Testing** can help to reduce the **ECC strength**
  - Even when starting with a **higher strength ECC**

Towards an Online Profiling System

1. Initially Protect DRAM with Strong ECC
2. Periodically Test Parts of DRAM
3. Mitigate errors and reduce ECC

Run tests periodically after a short interval at smaller regions of memory
More on Online Profiling of DRAM

- Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa Alameldeen, Chris Wilkerson, and Onur Mutlu,
"The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study"
How Do We Make RAIDR Work in the Presence of the VRT Phenomenon?
Making RAIDR Work w/ Online Profiling & ECC

- Moinuddin Qureshi, Dae Hyun Kim, Samira Khan, Prashant Nair, and Onur Mutlu,
  "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems"
  Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015.
  [Slides (pptx) (pdf)]
**AVATAR**

**Insight:** Avoid retention failures ➔ Upgrade row on ECC error

**Observation:** Rate of VRT >> Rate of soft error (50x-2500x)

---

**AVATAR mitigates VRT by increasing refresh rate on error**
RESULTS: REFRESH SAVINGS

Retention Testing Once a Year can revert refresh saving from 60% to 70%

AVATAR reduces refresh by 60%-70%, similar to multi rate refresh but with VRT tolerance
AVATAR gets 2/3\(^{rd}\) the performance of NoRefresh. More gains at higher capacity nodes
ENERGY DELAY PRODUCT

AVATAR reduces EDP,
Significant reduction at higher capacity nodes
Moinuddin Qureshi, Dae Hyun Kim, Samira Khan, Prashant Nair, and Onur Mutlu,
"AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems"
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 Refresh: Summary and Conclusions

- DRAM refresh is a critical challenge
  - in scaling DRAM technology efficiently to higher capacities

- Discussed several promising solution directions
  - Parallelize refreshes with accesses [Chang+ HPCA’14]
  - Eliminate unnecessary refreshes [Liu+ ISCA’12]
  - Reduce refresh rate and detect+correct errors that occur [Khan+ SIGMETRICS’14]

- Examined properties of retention time behavior [Liu+ ISCA’13]
  - Enable realistic VRT-Aware refresh techniques [Qureshi+ DSN’15]

- Many avenues for overcoming DRAM refresh challenges
  - Handling DPD/VRT phenomena
  - Enabling online retention time profiling and error mitigation
  - Exploiting application behavior
Other Backup Slides
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, ETH Zürich
Consumer Devices

Consumer devices are everywhere!

Energy consumption is a first-class concern in consumer devices
Popular Google Consumer Workloads

- **Chrome**: Google’s web browser
- **TensorFlow Mobile**: Google’s machine learning framework
- **VP9**: Google’s video codec
  - **YouTube**: Video Playback
  - **YouTube**: Video Capture
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 data movement often comes from simple functions

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

- Small embedded low-power core
- PIM Core
- Small fixed-function accelerators
- PIM Accelerator

Offloading to PIM logic reduces energy by 55.4% and improves performance by 54.2% on average
Goals

1. Understand the **data movement** related bottlenecks in modern consumer workloads.

2. Analyze opportunities to **mitigate** data movement by using **processing-in-memory (PIM)**.

3. Design **PIM logic** that can maximize energy efficiency given the limited area and energy budget in consumer devices.
Outline

- Introduction
- **Background**
  - Analysis Methodology
  - Workload Analysis
- Evaluation
- Conclusion
Potential Solution to Address Data Movement

- **Processing-in-Memory (PIM)**
  - A potential solution to **reduce data movement**
  - **Idea:** move computation close to data

  Reduces data movement
  - Exploits large in-memory bandwidth
  - Exploits shorter access latency to memory

- **Enabled by recent advances in 3D-stacked memory**
Outline

• Introduction
• Background
• **Analysis Methodology**
• Workload Analysis
• Evaluation
• Conclusion
Workload Analysis Methodology

• Workload Characterization
  – Chromebook with an Intel Celeron SoC and 2GB of DRAM
  – Extensively use performance counters within SoC

• Energy Model
  – Sum of the energy consumption within the CPU, all caches, off-chip interconnects, and DRAM
PIM Logic Implementation

SoC <-> Logic Layer

- PIM Core
  - Customized embedded general-purpose core
  - No aggressive ILP techniques
  - 256-bit SIMD unit

- PIM Accelerator
  - Small fixed-function accelerators
  - Multiple copies of customized in-memory logic unit
Workload Analysis

Chrome
Google’s web browser

TensorFlow
Google’s machine learning framework

VP9
Google’s video codec

Video Playback

Video Capture

Google’s video Codec
Workload Analysis

Chrome
Google’s web browser

TensorFlow
Google’s machine learning framework

VP9
Video Playback
Google’s video codec

VP9
Video Capture
Google’s video codec
How Chrome Renders a Web Page

- HTML
  - HTML Parser
- CSS
  - CSS Parser
- Render Tree
- Layout
- Rasterization
- Compositing
How Chrome Renders a Web Page

Loading and Parsing

- HTML
  - HTML Parser
- CSS
  - CSS Parser

Layouting

- Render Tree
- Layout
- Rasterization

Painting

- assembles all layers into a final screen image
- paints those objects and generates the bitmaps

**calculates the visual elements and position of each object**
Browser Analysis

• To satisfy user experience, the browser must provide:
  – Fast **loading** of webpages
  – Smooth **scrolling** of webpages
  – Quick **switching** between browser tabs

• We **focus on two important user interactions:**
  1) **Page Scrolling**
  2) **Tab Switching**
  – Both include **page loading**
Scrolling
What Does Happen During Scrolling?

Rasterization uses **color blitters** to convert the **basic primitives** into **bitmaps**.

- **HTML** → **HTML Parser**
- **CSS** → **CSS Parser**
- **Render Tree** → **Layout**
- **Rasterization** → **Compositing**
- **Color Blitting**
- **Texture Tiling**

To minimize **cache misses** during **compositing**, the graphics driver reorganizes the bitmaps.
41.9% of page scrolling energy is spent on texture tiling and color blitting
A significant portion of total data movement comes from texture tiling and color blitting.

37.7% of total system energy consumption goes to data movement.

SAFARI
A significant portion of total data movement comes from texture tiling and color blitting.

Can we use PIM to mitigate the data movement cost for texture tiling and color blitting?

37.7% of total system energy consumption goes to data movement.
Can We Use PIM for Texture Tiling?

Texture tiling is a good candidate for PIM execution.
Can We Implement Texture Tiling in PIM Logic?

Requires simple primitives: `memcpy`, bitwise operations, and simple arithmetic operations.

- PIM Core: 9.4% of the area available for PIM logic
- PIM Accelerator: 7.1% of the area available for PIM logic

PIM core and PIM accelerator are feasible to implement in-memory Texture Tiling.
Color Blitting Analysis

Generates a large amount of data movement
Accounts for 19.1% of the total system energy during scrolling

Color blitting is a good candidate for PIM execution

Requires low-cost operations: Memset, simple arithmetic, and shift operations

It is feasible to implement color blitting in PIM core and PIM accelerator
Texture tiling and color blitting account for a significant portion (41.9\%) of energy consumption. 37.7\% of total system energy goes to data movement generated by these functions.

1. Both functions can benefit significantly from PIM execution.

2. Both functions are feasible to implement as PIM logic.
Tab Switching
What Happens During Tab Switching?

- **Chrome employs a multi-process architecture**
  - Each tab is a separate process

- **Main operations during tab switching:**
  - Context switch
  - Load the new page
Memory Consumption

- Primary concerns during tab switching:
  - How fast a new tab loads and becomes interactive
  - Memory consumption

Chrome uses **compression** to reduce each tab’s **memory footprint**
Data Movement Study

• To study data movement during tab switching, we emulate a user switching through 50 tabs.

We make two key observations:

1. Compression and decompression contribute to 18.1% of the total system energy.

2. 19.6 GB of data moves between CPU and ZRAM.
Can We Use PIM to Mitigate the Cost?

**CPU-Only**
- Swap out N pages
- Read N Pages
- Compress
- Write back
- Other tasks

**CPU + PIM**
- Swap out N pages
- Other tasks
- Uncompressed Page
- ZRAM
- CPU
- PIM
- No off-chip data movement

PIM core and PIM accelerator are feasible to implement in-memory compression/decompression.
Tab Switching Wrap Up

A large amount of data movement happens during tab switching as Chrome attempts to compress and decompress tabs.

Both functions can benefit from PIM execution and can be implemented as PIM logic.
Workload Analysis

Chrome
Google’s web browser

TensorFlow
Google’s machine learning framework

VP9
YouTube Video Playback
Google’s video codec

VP9
YouTube Video Capture
Google’s video codec
Workload Analysis

Chrome
Google’s web browser

TensorFlow
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

**Matrix** → **Packing** → **Packed Matrix**

**Reorders** elements of matrices to minimize **cache misses** during **matrix multiplication**

- Up to **40%** of the **inference energy** and **31%** of the **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.
Quantization

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

Based on our analysis, we conclude that:

- Both functions are good candidates for **PIM execution**.
- It is **feasible** to implement them in **PIM logic**.

**A simple data conversion operation** that requires **shift**, **addition**, and **multiplication** operations.
Video Playback and Capture

VP9

Compressed video → VP9 Decoder → Display

Captured video → VP9 Encoder → Compressed video

Majority of energy is spent on data movement

Majority of data movement comes from simple functions in decoding and encoding pipelines
Outline

• Introduction
• Background
• Analysis Methodology
• Workload Analysis
• Evaluation
• Conclusion
Evaluation Methodology

- **System Configuration (gem5 Simulator)**
  - **SoC**: 4 OoO cores, 8-wide issue, 64 kB L1 cache, 2MB L2 cache
  - **PIM Core**: 1 core per vault, 1-wide issue, 4-wide SIMD, 32kB L1 cache
  - **3D-Stacked Memory**: 2GB cube, 16 vaults per cube
    - Internal Bandwidth: 256GB/S
    - Off-Chip Channel Bandwidth: 32 GB/s
  - **Baseline Memory**: LPDDR3, 2GB, FR-FCFS scheduler

- **We study each target in isolation and emulate each separately and run them in our simulator**
Normalized Energy

PIM core and PIM accelerator reduces energy consumption on average by 49.1% and 55.4%.

77.7% and 82.6% of energy reduction for texture tiling and packing comes from eliminating data movement.

CPU-Only  PIM-Core  PIM-Acc

Normalized Energy
Offloading these kernels to **PIM core** and **PIM accelerator** improves **performance** on average by **44.6%** and **54.2%**.
Conclusion

- Energy consumption is a **major challenge** in consumer devices
- We conduct an in-depth analysis of popular Google consumer workloads
  - 62.7% of the total system energy is spent on **data movement**
  - Most of the **data movement** comes from **simple functions** that consist of **simple operations**
- We use **PIM** to reduce **data movement cost**
  - We design **lightweight logic** to implement **simple operations** in DRAM
  - Reduces **total energy** by 55.4% on average
  - Reduces **execution time** by 54.2% on average
End of Backup Slides
Brief Self Introduction

Onur Mutlu
- Full Professor @ ETH Zurich CS, since September 2015
- 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/
- omutlu@gmail.com (Best way to reach me)
- https://people.inf.ethz.ch/omutlu/projects.htm

Research and Teaching in:
- Computer architecture, computer systems, hardware security, bioinformatics
- Memory and storage systems
- Hardware security, safety, predictability
- Fault tolerance
- Hardware/software cooperation
- Architectures for bioinformatics, health, medicine
- ...

507