Sub-Agenda: In-Memory Computation

- 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
Observation and Opportunity

- **High latency and high energy caused by data movement**
  - Long, energy-hungry interconnects
  - Energy-hungry electrical interfaces
  - Movement of large amounts of data

- **Opportunity:** Minimize data movement by performing computation directly (near) where the data resides
  - Processing in memory (PIM)
  - In-memory computation/processing
  - Near-data processing (NDP)
  - General concept applicable to any data storage & movement unit (caches, SSDs, main memory, network, controllers)
Four Key Issues in Future Platforms

- 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 (Human) Hierarchy of Needs, Revisited


Source: https://www.simplypsychology.org/maslow.html
Do We Want This?

Source: V. Milutinovic
Or This?

Source: V. Milutinovic
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)

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

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

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

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

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

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com
The Memory Bottleneck


**Runahead Execution: An Effective Alternative to Large Instruction Windows**
It’s the Memory, Stupid!

RICHARD SITES

It’s the Memory, Stupid!
When we started the Alpha architecture design in 1988, we estimated a 25-year lifetime and a relatively modest 32% per year compounded performance improvement of implementations over that lifetime (1,000× total). We guestimated about 10× would come from CPU clock improvement, 10× from multiple instruction issue, and 10× from multiple processors.

5, 1996 MICROPROCESSOR REPORT

An Informal Interview on Memory

Madeleine Gray and Onur Mutlu, "It’s the memory, stupid’: A conversation with Onur Mutlu"

HiPEAC info 55, HiPEAC Newsletter, October 2018.
[Shorter Version in Newsletter]
[Longer Online Version with References]

‘It’s the memory, stupid’: A conversation with Onur Mutlu

‘We’re beyond computation; we know how to do computation really well, we can optimize it, we can build all sorts of accelerators ... but the memory – how to feed the data, how to get the data into the accelerators – is a huge problem.’

This was how ETH Zürich and Carnegie Mellon Professor Onur Mutlu opened his course on memory systems and memory-centric computing systems at HiPEAC’s summer school, ACACES18. A prolific publisher – he recently bagged the top spot on the International Symposium on Computer Architecture (ISCA) hall of fame – Onur is passionate about computation and communication that are efficient and secure by design. In advance of our Computing Systems Week focusing on data centres, storage, and networking, which takes place next week in Heraklion, HiPEAC picked his brains on all things data-based.
The Performance Perspective (Today)

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

  ![Graph showing pipeline slot breakdown for various workloads.](image_url)

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 20pJ
- 256-bit buses
- 256-bit access 8 kB SRAM
- 1 nJ
- 50 pJ
- 26 pJ
- 256 pJ
- 16 nJ
- 500 pJ
- DRAM Rd/Wr
- Efficient off-chip link

SAFARI
Data Movement vs. Computation Energy

Communication Dominates Arithmetic

Dally, HiPEAC 2015

A memory access consumes ~100-1000X the energy of a complex addition.
Data Movement vs. Computation Energy

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

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


62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand¹
Rachata Ausavarungrunrung¹
Aki Kuusela³
Allan Knies³

Saugata Ghose¹
Eric Shiu³
Parthasarathy Ranganathan³

Youngsok Kim²
Rahul Thakur³
Daehyun Kim⁴,³

Onur Mutlu⁵,¹

SAFARI
We Do Not Want to Move Data!

A memory access consumes $\sim 100-1000X$ the energy of a complex addition.
We Need A Paradigm Shift To ...

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

Many questions ... How do we design the:

- compute-capable memory & controllers?
- processor chip and in-memory units?
- software and hardware interfaces?
- system software, compilers, languages?
- algorithms and theoretical foundations?
Why In-Memory Computation Today?

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

- **Pull from Systems and Applications**
  - Data access is a major system and application bottleneck
  - Systems are energy limited
  - Data movement much more energy-hungry than computation
UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine
  - Includes standard DIMM modules, with a large number of DPU processors combined with DRAM chips.

- Replaces standard DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - Large amounts of compute & memory bandwidth

We Need to Think Differently from the Past Approaches
Sub-Agenda: In-Memory Computation

- 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

Bulk Data Copy

Bulk Data Initialization
Bulk Data Copy and Initialization

The Impact of Architectural Trends on Operating System Performance
Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod, Emmett Witchel, and Anoop Gupta

Hardware Support for Bulk Data Movement in Server Platforms
Li Zhao†, Ravi Iyer‡ Srihari Makineni‡, Laxmi Bhuyan† and Don Newell‡
†Department of Computer Science and Engineering, University of California, Riverside, CA 92521
Email: {zhao, bhuyan}@cs.ucr.edu
‡Communications Technology Lab, Intel Corporation, Hillsboro, Oregon

Architecture Support for Improving Bulk Memory Copying and Initialization Performance
Xiaowei Jiang, Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA

Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA
Starting Simple: Data Copy and Initialization

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

- Forking
- Zero initialization (e.g., security)
- Checkpointing
- VM Cloning
- Deduplication
- Page Migration

*Many more*

[Image: Diagram illustrating various data copy and initialization techniques with icons and text labels.]
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)

11.6X latency reduction, 74X energy reduction
RowClone: Intra-Subarray

Data gets copied

\[ \frac{V_{DD}}{2} \delta \]

Amplify the difference

Sources and destinations

\[ 0 \]

Sense Amplifier (Row Buffer)

\[ \frac{V_{DD}}{2} \]

\[ 0 \]
RowClone: Intra-Subarray (II)

1. **Activate** src row (copy data from src to row buffer)

2. **Activate** dst row (disconnect src from row buffer, connect dst – copy data from row buffer to dst)
RowClone: Inter-Bank

Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction
Generalized RowClone

- **Inter Subarray Copy** (Use Inter-Bank Copy Twice)
- **Inter Bank Copy** (Pipelined Internal RD/WR)
- **Intra Subarray Copy** (2 ACTs)

0.01% area cost
RowClone: Fast Row Initialization

Fix a row at Zero
(0.5% loss in capacity)
RowClone: Bulk Initialization

- Initialization with arbitrary data
  - Initialize one row
  - Copy the data to other rows

- Zero initialization (most common)
  - Reserve a row in each subarray (always zero)
  - Copy data from reserved row (FPM mode)
  - 6.0x lower latency, 41.5x lower DRAM energy
  - 0.2% loss in capacity
RowClone: Latency & Energy Benefits

Latency Reduction

Energy Reduction

Very low cost: 0.01% increase in die area
Copy and Initialization in Workloads

Fraction of Memory Traffic

- bootup
- compile
- forkbench
- mcached
- mysql
- shell

Zero Copy Write Read
RowClone: Application Performance

% Compared to Baseline

- IPC Improvement
- Energy Reduction

Applications:
- bootup
- compile
- forkbench
- mcached
- mysql
- shell

[Bar chart showing performance improvements and energy reductions for different applications.]
End-to-End System Design

- Application
- Operating System
- ISA
- Microarchitecture
- DRAM (RowClone)

How to communicate occurrences of bulk copy-initialization across layers?

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?
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 Bulk Bitwise AND/OR Operation

- **BULKAND A, B  C**
- Semantics: Perform a bitwise AND of two rows A and B and store the result in row C

- R0 – reserved zero row, R1 – reserved one row
- D1, D2, D3 – Designated rows for triple activation

1. RowClone A into D1
2. RowClone B into D2
3. RowClone R0 into D3
4. ACTIVATE D1,D2,D3
5. RowClone Result into C
More on In-DRAM Bulk AND/OR

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

"Fast Bulk Bitwise AND and OR in DRAM"


---

Fast Bulk Bitwise AND and OR in DRAM

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

*Carnegie Mellon University †Intel Pittsburgh
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

In-DRAM NOT Operation

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 (nJ/KB)</th>
<th>and/or (X)</th>
<th>nand/nor (X)</th>
<th>xor/xnor (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM &amp; 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>(nJ/KB)</td>
<td>(↓) 59.5</td>
<td>43.9</td>
<td>35.1</td>
<td>25.1</td>
</tr>
</tbody>
</table>

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

Ambit vs. DDR3: Performance and Energy

- Performance Improvement
- Energy Reduction

Bulk Bitwise Operations in Workloads

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

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

- Alternative to B-tree and its variants
- Efficient for performing range queries and joins
- Many bitwise operations to perform a query
**Performance: Bitmap Index on Ambit**

![Bar chart showing performance improvement]

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

>5.4-6.6X Performance Improvement

Performance: BitWeaving on Ambit

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

More on In-DRAM Bulk AND/OR

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

"Fast Bulk Bitwise AND and OR in DRAM"

*IEEE Computer Architecture Letters (CAL), April 2015.*

---

**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 In-DRAM Bitwise Operations


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

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

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


In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
Challenge: Intelligent Memory Device

Does memory have to be dumb?
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
A Detour
on the Review Process
Ambit Sounds Good, No?

Review from ISCA 2016

Paper summary
The paper proposes to extend DRAM to include bulk, bit-wise logical operations directly between rows within the DRAM.

Strengths
- Very clever/novel idea.
- Great potential speedup and efficiency gains.

Weaknesses
- Probably won't ever be built. Not practical to assume DRAM manufacturers with change DRAM in this way.
Another Review

Another Review from ISCA 2016

Strengths
The proposed mechanisms effectively exploit the operation of the DRAM to perform efficient bitwise operations across entire rows of the DRAM.

Weaknesses
This requires a modification to the DRAM that will only help this type of bitwise operation. It seems unlikely that something like that will be adopted.
Weaknesses

The core novelty of Buddy RAM is almost all circuits-related (by exploiting sense amps). I do not find architectural innovation even though the circuits technique benefits architecturally by mitigating memory bandwidth and relieving cache resources within a subarray. The only related part is the new ISA support for bitwise operations at DRAM side and its induced issue on cache coherence.
The Reviewer Accountability Problem

Acknowledgments

We thank the reviewers of ISCA 2016/2017, MICRO 2016/2017, and HPCA 2017 for their valuable comments. We
We Have a Mindset Issue...

- There are many other similar examples from reviews...
  - For many other papers...

- And, we are not even talking about JEDEC yet...

- How do we fix the mindset problem?

- By doing more research, education, implementation in alternative processing paradigms

We need to work on enabling the better future...
Aside: A Recommended Book

Even if the performance analysis is correctly done and presented, it may not be enough to persuade your audience—the decision makers—to follow your recommendations. The list shown in Box 10.2 is a compilation of reasons for rejection heard at various performance analysis presentations. You can use the list by presenting it immediately and pointing out that the reason for rejection is not new and that the analysis deserves more consideration. Also, the list is helpful in getting the competing proposals rejected!

There is no clear end of an analysis. Any analysis can be rejected simply on the grounds that the problem needs more analysis. This is the first reason listed in Box 10.2. The second most common reason for rejection of an analysis and for endless debate is the workload. Since workloads are always based on the past measurements, their applicability to the current or future environment can always be questioned. Actually workload is one of the four areas of discussion that lead a performance presentation into an endless debate. These “rat holes” and their relative sizes in terms of time consumed are shown in Figure 10.26. Presenting this cartoon at the beginning of a presentation helps to avoid these areas.

FIGURE 10.26 Four issues in performance presentations that commonly lead to endless discussion.
Box 10.2 Reasons for Not Accepting the Results of an Analysis

1. This needs more analysis.
2. You need a better understanding of the workload.
3. It improves performance only for long I/O's, packets, jobs, and files, and most of the I/O's, packets, jobs, and files are short.
4. It improves performance only for short I/O’s, packets, jobs, and files, but who cares for the performance of short I/O’s, packets, jobs, and files; it’s the long ones that impact the system.
5. It needs too much memory/CPU/bandwidth and memory/CPU/bandwidth isn’t free.
6. It only saves us memory/CPU/bandwidth and memory/CPU/bandwidth is cheap.
7. There is no point in making the networks (similarly, CPUs/disks/…) faster; our CPUs/disks (any component other than the one being discussed) aren’t fast enough to use them.
8. It improves the performance by a factor of \( x \), but it doesn’t really matter at the user level because everything else is so slow.
9. It is going to increase the complexity and cost.
10. Let us keep it simple stupid (and your idea is not stupid).
11. It is not simple. (Simplicity is in the eyes of the beholder.)
12. It requires too much state.
13. Nobody has ever done that before. (You have a new idea.)
14. It is not going to raise the price of our stock by even an eighth. (Nothing ever does, except rumors.)
15. This will violate the IEEE, ANSI, CCITT, or ISO standard.
16. It may violate some future standard.
17. The standard says nothing about this and so it must not be important.
18. Our competitors don’t do it. If it was a good idea, they would have done it.
19. Our competition does it this way and you don’t make money by copying others.
20. It will introduce randomness into the system and make debugging difficult.
21. It is too deterministic; it may lead the system into a cycle.
22. It’s not interoperable.
23. This impacts hardware.
24. That’s beyond today’s technology.
25. It is not self-stabilizing.
26. Why change—it’s working OK.
Suggestion to Community

We Need to Fix the Reviewer Accountability Problem
Takeaway

Main Memory Needs

Intelligent Controllers
Takeaway

Research Community Needs

Accountable Reviewers
Suggestions to Reviewers

- Be fair; you do not know it all
- Be open-minded; you do not know it all
- Be accepting of diverse research methods: there is no single way of doing research
- Be constructive, not destructive
- Do not have double standards...

Do not block or delay scientific progress for non-reasons
RowClone & Bitwise Ops in Real DRAM Chips

ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs

Fei Gao
feig@princeton.edu
Department of Electrical Engineering
Princeton University

Georgios Tziantzioulis
georgios.tziantzioulis@princeton.edu
Department of Electrical Engineering
Princeton University

David Wentzlaff
wentzlaf@princeton.edu
Department of Electrical Engineering
Princeton University

RowClone & Bitwise Ops in Real DRAM Chips

Figure 4: Timeline for a single bit of a column in a row copy operation. The data in $R_1$ is loaded to the bit-line, and overwrites $R_2$.

Figure 5: Logical AND in ComputeDRAM. $R_1$ is loaded with constant zero, and $R_2$ and $R_3$ store operands (0 and 1). The result ($0 = 1 \land 0$) is finally set in all three rows.
Row Copy in ComputeDRAM

Bitline is above $V_{DD}/2$ when R2 is activated.
Bitwise AND in ComputeDRAM

$V_{dd}/2$

$R_3 = 00_2$
Operand: 1

$R_4 = 01_2$
Constant: 0

T2 very short
PRE cannot close R1
R3 will appear on the address bus
ACT(R2) will activate R3 and R2

$T_1 = T_2 = 0$ idle cycle
Experimental Methodology

Figure 9: (a) Schematic diagram of our testing framework. (b) Picture of our testbed. (c) Thermal picture when the DRAM is heated to 80 °C.
# Experimental Methodology

## Table 1: Evaluated DRAM modules

<table>
<thead>
<tr>
<th>Group ID: Vendor_Size_Freq(MHz)</th>
<th>Part Num</th>
<th># Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKhynix_2G_1333</td>
<td>HMT325S6BFR8C-H9</td>
<td>6</td>
</tr>
<tr>
<td>SKhynix_4G_1333</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SKhynix_4G_6400_S2S32</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SKhynix_4G_6400_S2S32</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SKhynix_4G_6400_S2S32</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Samsung_4G_1333</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Samsung_4G_1333</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Micron_2G_1333</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Micron_2G_1333</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Elpida_2G_1333</td>
<td>EBJ21UE8BDS0-DJ-F</td>
<td>2</td>
</tr>
<tr>
<td>Nanya_4G_1333</td>
<td>NT4GC64B8HG0NS-CG</td>
<td>2</td>
</tr>
<tr>
<td>TimeTec_4G_1333</td>
<td>78AP10NUS2R2-4G</td>
<td>2</td>
</tr>
<tr>
<td>Corsair_4G_1333</td>
<td>CMSA8GX3M2A1333C9</td>
<td>2</td>
</tr>
</tbody>
</table>

32 DDR3 Modules

~256 DRAM Chips
Proof of Concept

- How they test these memory modules:
  - Vary $T_1$ and $T_2$, observe what happens.

### SoftMC Experiment

1. Select a random subarray
2. Fill subarray with random data
3. Issue ACT-PRE-ACTs with given $T_1$ & $T_2$
4. Read out subarray
5. Find out how many columns in a row support either operation
   - Row-wise success ratio
Each grid represents the success ratio of operations for a specific DDR3 module.
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li¹, Cong Xu², Qiaosha Zou¹,⁵, Jishen Zhao³, Yu Lu⁴, and Yuan Xie¹

University of California, Santa Barbara¹, Hewlett Packard Labs²
University of California, Santa Cruz³, Qualcomm Inc.⁴, Huawei Technologies Inc.⁵
{shuangchenli, yuanxie}@ece.ucsb.edu¹
Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs \( n \)-row bitwise operations inside NVM in one step.
Other Examples of
“Why Change? It’s Working OK!”
Mindset Issues Are Everywhere

- “Why Change? It’s Working OK!” mindset limits progress
- There are many such examples in real life
- Examples of Bandwidth Waste in Real Life
- Examples of Latency and Queueing Delays in Real Life
- Example of Where to Build a Bridge over a River
Another Example
# Initial RowHammer Reviews

Disturbance Errors in DRAM: Demonstration, Characterization, and Prevention

Rejected (R2)  863kB  
| Friday 31 May 2013 2:00:53pm PDT 
| b9bf06021da54cddf4cd0b3565558a181868b972

You are an **author** of this paper.

<table>
<thead>
<tr>
<th>OveMer</th>
<th>Nov</th>
<th>WriQua</th>
<th>RevExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**SAFARI**
This is an excellent test methodology paper, but there is no micro-architectural or architectural content.

- Whereas they show disturbance may happen in DRAM array, authors don't show it can be an issue in realistic DRAM usage scenario.
- Lacks architectural/microarchitectural impact on the DRAM disturbance analysis.

The mechanism investigated by the authors is one of many well known disturb mechanisms. The paper does not discuss the root causes to sufficient depth and the importance of this mechanism compared to others. Overall the length of the sections restating known information is much too long in relation to new work.
Experimental DRAM Testing Infrastructure

## Tested DRAM Modules

(129 total)

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Module</th>
<th>Date* (yy-mm)</th>
<th>Timing†</th>
<th>Organization</th>
<th>Chip</th>
<th>Victims-per-Module</th>
<th>R_{66} (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
<td>10-08</td>
<td>1066</td>
<td>50.625</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>10-20</td>
<td>1066</td>
<td>50.625</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2.5</td>
<td>10-20</td>
<td>1066</td>
<td>50.625</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2.7</td>
<td>11-24</td>
<td>1066</td>
<td>49.125</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.12</td>
<td>11-50</td>
<td>1066</td>
<td>49.125</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2.14</td>
<td>12-22</td>
<td>1600</td>
<td>50.625</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.16</td>
<td>12-22</td>
<td>1600</td>
<td>50.625</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.18</td>
<td>12-26</td>
<td>1600</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.30</td>
<td>12-40</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.31</td>
<td>13-02</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.34</td>
<td>13-14</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.35</td>
<td>13-16</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.37</td>
<td>13-20</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A2.40</td>
<td>13-28</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>14-04</td>
<td>1600</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A3.41</td>
<td>14-04</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>08-49</td>
<td>1066</td>
<td>50.625</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B2</td>
<td>09-49</td>
<td>1066</td>
<td>50.625</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>10-19</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B4</td>
<td>10-31</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B5</td>
<td>11-13</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B6</td>
<td>11-16</td>
<td>1066</td>
<td>50.625</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B7</td>
<td>11-19</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B8</td>
<td>11-25</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B9</td>
<td>11-37</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B10</td>
<td>11-46</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B12</td>
<td>12-12</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B13</td>
<td>12-49</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B14</td>
<td>12-01</td>
<td>1866</td>
<td>47.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B15</td>
<td>12-10</td>
<td>1866</td>
<td>47.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B16</td>
<td>12-25</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B17</td>
<td>12-26</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B18</td>
<td>13-02</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B19</td>
<td>13-14</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B20</td>
<td>13-20</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B21</td>
<td>13-28</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B22</td>
<td>13-40</td>
<td>1600</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B23</td>
<td>13-41</td>
<td>1600</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C1</td>
<td>10-18</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>10-20</td>
<td>1066</td>
<td>50.625</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>C3</td>
<td>10-22</td>
<td>1066</td>
<td>50.625</td>
<td>0.5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>C4</td>
<td>10-26</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C5</td>
<td>10-31</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C6</td>
<td>10-51</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td>11-12</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C8</td>
<td>11-19</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C9</td>
<td>11-31</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C10</td>
<td>11-42</td>
<td>1333</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C11</td>
<td>11-48</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C12</td>
<td>12-08</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C13</td>
<td>12-15</td>
<td>1600</td>
<td>49.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C14</td>
<td>12-16</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C15</td>
<td>12-20</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C16</td>
<td>12-23</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C17</td>
<td>12-24</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C18</td>
<td>12-25</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C19</td>
<td>12-30</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C20</td>
<td>12-32</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C21</td>
<td>12-33</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C22</td>
<td>13-11</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C23</td>
<td>13-35</td>
<td>1600</td>
<td>48.125</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

* We report the manufacture date marked on the chip packages, which is more accurate than other dates that can be gleaned from a module.
† We report timing constraints stored in the module’s on-board ROM [33], which is read by the system BIOS to calibrate the memory controller.
§ We report DRAM die versions marked on the chip packages, which typically progress in the following manner: A → B → C → ... → Z.

Table 3. Sample population of 129 DDR3 DRAM modules, categorized by manufacturer and sorted by manufacture date.
Fast Forward 6 Months
PAPER WEAKNESSES

1) The disturbance error (a.k.a coupling or cross-talk noise induced error) is a known problem to the DRAM circuit community.

2) What you demonstrated in this paper is so called DRAM row hammering issue - you can even find a Youtube video showing this! - http://www.youtube.com/watch?v=i3-qQSnBcd0

2) The architectural contribution of this study is too insignificant.

PAPER WEAKNESSES

- Row Hammering appears to be well-known, and solutions have already been proposed by industry to address the issue.

- The paper only provides a qualitative analysis of solutions to the problem. A more robust evaluation is really needed to know whether the proposed solution is necessary.
Final RowHammer Reviews

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

Accepted  639kB  21 Nov 2013 10:53:11pm CST  
f039be2735313b39304ae1c6296523867a485610

You are an author of this paper.

<table>
<thead>
<tr>
<th>Review</th>
<th>OveMer</th>
<th>Nov</th>
<th>WriQua</th>
<th>RevConAnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>#41A</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>#41B</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>#41C</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>#41D</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>#41E</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>#41F</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

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

Project Zero

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
Onur Mutlu and Jeremie Kim, "RowHammer: A Retrospective"


[Preliminary arXiv version]
RowHammer in 2020
RowHammer in 2020 (I)

- Jeremie S. Kim, Minesh Patel, A. Giray Yaglikci, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu,

"Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques"


[Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (20 minutes)]
[Lightning Talk Video (3 minutes)]

Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques

Jeremie S. Kim§†, Minesh Patel§, A. Giray Yaglıkçi§, Hasan Hassan§, Roknoddin Azizi§, Lois Orosa§, Onur Mutlu§†

§ETH Zürich    †Carnegie Mellon University
RowHammer in 2020 (II)

- Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh"


[Slides (pptx) (pdf)]
[Talk Video (17 minutes)]
[Source Code]
[Web Article]

Best paper award.

TRRespass: Exploiting the Many Sides of Target Row Refresh

Pietro Frigo*† Emanuele Vannacci*† Hasan Hassan§ Victor van der Veen¶
Onur Mutlu§ Cristiano Giuffrida* Herbert Bos* Kaveh Razavi*

*Vrije Universiteit Amsterdam
†ETH Zürich
§ETH Zürich
¶Qualcomm Technologies Inc.
RowHammer in 2020 (III)

- Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu,

"Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers"


[Slides (pptx) (pdf)]
[Talk Video (17 minutes)]

Are We Susceptible to Rowhammer?
An End-to-End Methodology for Cloud Providers

Lucian Cojocar, Jeremie Kim§†, Minesh Patel§, Lillian Tsai†,
Stefan Saroiu, Alec Wolman, and Onur Mutlu§†
Microsoft Research, §ETH Zürich, †CMU, ‡MIT
Session 1A: Security & Privacy I

5:00 PM CEST - 5:15 PM CEST
Graphene: Strong yet Lightweight Row Hammer Protection
Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, Jae W. Lee (Seoul National University)

5:15 PM CEST - 5:30 PM CEST
Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Memory
Alexander Freij, Shougang Yuan, Huiyang Zhou (NC State University); Yan Selikhin (University of Central Florida)

5:30 PM CEST - 5:45 PM CEST
PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses
Zhi Zhang (University of New South Wales and Data61, CSIRO, Australia); Yueqiang Cheng (Baidu Security); Dongxi Liu; Surya Nepal (Data61, CSIRO, Australia); Zhi Wang (Florida State University); Yuval Yarom (University of Adelaide and Data61, CSIRO, Australia)
RowHammer in 2020 (V)

Session #5: Rowhammer

Session chair: Michael Franz (UC Irvine)

**RAMBleed: Reading Bits in Memory Without Accessing Them**
Andrew Kwong (University of Michigan), Daniel Genkin (University of Michigan), Daniel Gruss (Data61)

**Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers**
Lucian Cojocar (Microsoft Research), Jeremie Kim (ETH Zurich, CMU), Minesh Patel (ETH Zurich, Microsoft Research), Onur Mutlu (ETH Zurich, CMU)

**Leveraging EM Side-Channel Information to Detect Rowhammer Attacks**
Zhenkai Zhang (Texas Tech University), Zihao Zhan (Vanderbilt University), Daniel Balasubramanian, Peter Volgyesi (Vanderbilt University), Xenophon Koutsovorkos (Vanderbilt University)

**TRRespass: Exploiting the Many Sides of Target Row Refresh**
Pietro Frigo (Vrije Universiteit Amsterdam, The Netherlands), Emanuele Vannacci (Vrije Universiteit Veen (Qualcomm Technologies, Inc.), Onur Mutlu (ETH Zürich), Cristiano Giuffrida (Vrije Universiteit Amsterdam, The Netherlands), Kaveh Razavi (Vrije Universiteit Amsterdam, The Netherlands)
DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips
Fan Yao, University of Central Florida; Adnan Siraj Rakin and Deliang Fan, Arizona State University

Show details ▶
More to Come…
Follow Your Passion
(Do not get derailed by naysayers)
Suggestion to Researchers: Principle: Resilience

Be Resilient
Focus on learning and scholarship
Principle: Learning and Scholarship

The quality of your work defines your impact
An Interview on Research and Education

- Computing Research and Education (@ ISCA 2019)
  - [https://www.youtube.com/watch?v=8ffSEKZhmvo&list=PL5Q2soXY2Zi_4oP9LdL3cc8G6NIjD2Ydz](https://www.youtube.com/watch?v=8ffSEKZhmvo&list=PL5Q2soXY2Zi_4oP9LdL3cc8G6NIjD2Ydz)

- Maurice Wilkes Award Speech (10 minutes)
  - [https://www.youtube.com/watch?v=tcQ3zZ3JpuA&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=15](https://www.youtube.com/watch?v=tcQ3zZ3JpuA&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnV2YFBJl&index=15)
More Thoughts and Suggestions

- Onur Mutlu, "Some Reflections (on DRAM)"
  Award Speech for ACM SIGARCH Maurice Wilkes Award, at the ISCA Awards Ceremony, Phoenix, AZ, USA, 25 June 2019.
  [Slides (pptx) (pdf)]
  [Video of Award Acceptance Speech (Youtube; 10 minutes) (Youku; 13 minutes)]
  [Video of Interview after Award Acceptance (Youtube; 1 hour 6 minutes) (Youku; 1 hour 6 minutes)]
  [News Article on "ACM SIGARCH Maurice Wilkes Award goes to Prof. Onur Mutlu"]

- Onur Mutlu, "How to Build an Impactful Research Group"
  57th Design Automation Conference Early Career Workshop (DAC), Virtual, 19 July 2020.
  [Slides (pptx) (pdf)]
Sub-Agenda: In-Memory Computation

- 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
We Need to Think Differently from the Past Approaches
Memory as an Accelerator

Memory similar to a “conventional” accelerator
Processing in Memory:
Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory
Opportunity: 3D-Stacked Logic+Memory

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

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDARAM3 (2011) [29]</td>
</tr>
<tr>
<td>3D-Stacked</td>
<td>WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13];</td>
</tr>
<tr>
<td>Academic</td>
<td>SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27];</td>
</tr>
<tr>
<td></td>
<td>SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37];</td>
</tr>
<tr>
<td></td>
<td>Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33];</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory

Several Questions in 3D-Stacked PIM

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

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

- Large graphs are everywhere (circa 2015)

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

- Scalable large-scale graph processing is challenging

  - 32 Cores
  - 128...

  ![Graph Speedup Chart]

  - +42% Speedup
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)

Memory

Logic

Crossbar Network

In-Order Core

LP
PF Buffer
MTP
Message Queue

DRAM Controller

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

Host Processor
Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed

Memory

Logic

In-Order Core

Communications via Remote Function Calls

Message Queue

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

Non-blocking Remote Function Call

Can be delayed until the nearest barrier
Remote Function Call (Non-Blocking)

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

```cpp
put(w.id, function() { w.next_rank += value; })
```
Tesseract System for Graph Processing

Host Processor
Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed

Logic
Memory
Crossbar Network

Prefetching
LP
PF Buffer
MTP
Message Queue
DRAM Controller
NI

Host Processor

Memory

Logic

Crossbar Network

Prefetching
LP
PF Buffer
MTP
Message Queue
DRAM Controller
NI

SAFARI
Evaluated Systems

<table>
<thead>
<tr>
<th>DDR3-OoO</th>
<th>HMC-OoO</th>
<th>HMC-MC</th>
<th>Tesseract</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram of DDR3-OoO" /></td>
<td><img src="image2" alt="Diagram of HMC-OoO" /></td>
<td><img src="image3" alt="Diagram of HMC-MC" /></td>
<td><img src="image4" alt="Diagram of Tesseract" /></td>
</tr>
<tr>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>32 Tesseract Cores</strong></td>
</tr>
<tr>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>128 In-Order 2GHz</strong></td>
<td></td>
</tr>
<tr>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>128 In-Order 2GHz</strong></td>
<td></td>
</tr>
<tr>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>128 In-Order 2GHz</strong></td>
<td></td>
</tr>
<tr>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>8 OoO 4GHz</strong></td>
<td><strong>128 In-Order 2GHz</strong></td>
<td></td>
</tr>
<tr>
<td><strong>102.4GB/s</strong></td>
<td><strong>640GB/s</strong></td>
<td><strong>640GB/s</strong></td>
<td><strong>8TB/s</strong></td>
</tr>
</tbody>
</table>

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

>13X Performance Improvement

On five graph processing algorithms

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

Speedup

>56%  >25%  9.0x  11.6x  13.8x

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

Memory Bandwidth Consumption

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

Memory Bandwidth Consumption
Effect of Bandwidth & Programming Model

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

Programming Model:
- HMC-MC
- HMC-MC + PIM BW
- Tesseract + Conventional BW
- Tesseract (No Prefetching)

Bandwidth Speedup:
- 2.3x
- 3.0x
- 6.5x

(No Prefetching)
Tesseract Graph Processing System Energy

- Memory Layers
- Logic Layers
- Cores

> 8X Energy Reduction

HMC-OoO

Tesseract with Prefetching

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

- **Advantages**
  - Specialized graph processing accelerator using PIM
  - Large system performance and energy benefits
  - Takes advantage of 3D stacking for an important workload
  - More general than just graph processing

- **Disadvantages**
  - Changes a lot in the system
    - New programming model
    - Specialized Tesseract cores for graph processing
  - Cost
  - Scalability limited by off-chip links or graph partitioning
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)]
Sub-Agenda: In-Memory Computation

- 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
Several Questions in 3D-Stacked PIM

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

What is the minimal processing-in-memory support we can provide?
- With minimal changes to system and programming
Several Questions in 3D-Stacked PIM

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

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming
3D-Stacked PIM on Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu,

"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"


Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

Consumer devices are everywhere!

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

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

VP9
Google’s video codec

Video Playback
YouTube

Video Capture
YouTube

Google’s video codec
Energy Cost of Data Movement

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

Potential solution: move computation close to data

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

2\textsuperscript{nd} key observation: a significant fraction of the data movement often comes from simple functions

We can design lightweight logic to implement these \textit{simple functions} in memory

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

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

VP9
Google’s video codec

Video Playback

Video Capture

SAFARI
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.
Normalized Energy

- **CPU-Only**
- **PIM-Core**
- **PIM-Acc**

Normalized Energy

<table>
<thead>
<tr>
<th>Activity</th>
<th>Chrome Browser</th>
<th>TensorFlow Mobile</th>
<th>Video Playback and Capture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture Tiling</td>
<td>1</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Color Blitting</td>
<td>1</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Compression</td>
<td>0.6</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Decompression</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Packing</td>
<td>0.6</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Quantization</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>Sub-Pixel Interpolation</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Deblocking Filter</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Motion Estimation</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**PIM core** and **PIM accelerator** reduce **energy consumption** on average by **49.1%** and **55.4%**
Offloading these kernels to PIM core and PIM accelerator improves performance on average by 44.6% and 54.2%
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
How Chrome Renders a Web Page

Loading and Parsing

- HTML
  - HTML Parser
- CSS
  - CSS Parser

Layouting

- Render Tree
- Layout

Rasterization

- Rasterization
- Compositing

Painting

- Paints those objects and generates the bitmaps
- Assembles all layers into a final screen image

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
Tab Switching
What Happens During Tab Switching?

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

  ![Diagram showing Chrome Process and multiple tab processes](image)

• **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

Memory

- Uncompressed Pages
- Other tasks
- ZRAM

CPU + PIM

- Swap out N pages
- Other tasks
- Compress
- ZRAM

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.
More on PIM for Mobile Devices


62.7% of the total system energy is spent on data movement

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand\(^1\) Rachata Ausavarungnirun\(^1\) Saugata Ghose\(^1\) Youngsok Kim\(^2\)
Aki Kuusela\(^3\) Allan Knies\(^3\) Eric Shiu\(^3\) Rahul Thakur\(^3\) Daehyun Kim\(^4,3\)
Parthasarathy Ranganathan\(^3\) Onur Mutlu\(^5,1\)
Truly Distributed GPU Processing with PIM?

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Main GPU

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl

Another diagram:

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

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

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


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
Accelerating GPU Execution with PIM (II)


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

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

Ashutosh 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
Accelerating Linked Data Structures

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

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

Kevin Hsieh† Samira Khan‡ Nandita Vijaykumar†
Kevin K. Chang† Amirali Boroumand† Saugata Ghose† Onur Mutlu§†
†Carnegie Mellon University ‡University of Virginia §ETH Zürich
Accelerating Dependent Cache Misses

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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

*The University of Texas at Austin †Apple ‡NVIDIA §ETH Zürich & Carnegie Mellon University
Accelerating Runahead Execution

Milad Hashemi, Onur Mutlu, and Yale N. Patt,
"Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016.
[Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

*The University of Texas at Austin  §ETH Zürich
Accelerating Climate Modeling

Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal,
"NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"
Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, September 2020.
[Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (23 minutes)]
Nominated for the Stamatis Vassiliadis Memorial Award.

NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh\textsuperscript{a,b,c}, Dionysios Diamantopoulos\textsuperscript{c}, Christoph Hagleitner\textsuperscript{c}, Juan Gómez-Luna\textsuperscript{b}, Sander Stuijk\textsuperscript{a}, Onur Mutlu\textsuperscript{b}, Henk Corporaal\textsuperscript{a}
\textsuperscript{a}Eindhoven University of Technology \hspace{1cm} \textsuperscript{b}ETH Zürich \hspace{1cm} \textsuperscript{c}IBM Research Europe, Zurich
Accelerating Approximate String Matching

- Damla Senol Cali, Gurpreet S. Kalsi, Zulal Bingol, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu,

"GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis"


[Lighting Talk Video (1.5 minutes)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (18 minutes)]
[Slides (pptx) (pdf)]

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali†‡, Gurpreet S. Kalsi‡, Zülal Bingöl†, Can Firtina§, Lavanya Subramanian†, Jeremie S. Kim‡, Rachata Ausavarungnirun©, Mohammed Alser§, Juan Gomez-Luna§, Amirali Boroumand†, Anant Nori‡, Allison Scibisz†, Sreenivas Subramoney§, Can Alkan†, Saugata Ghose†, Onur Mutlu†,

†Carnegie Mellon University  ‡Processor Architecture Research Lab, Intel Labs  ©Bilkent University  §ETH Zürich
©King Mongkut’s University of Technology North Bangkok  *University of Illinois at Urbana–Champaign

SAFARI
Accelerating Time Series Analysis

- Ivan Fernandez, Ricardo Quislant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu,

"NATSA: A Near-Data Processing Accelerator for Time Series Analysis"

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

Ivan Fernandez§ Ricardo Quislant§ Christina Giannoula† Mohammed Alser‡
Juan Gómez-Luna‡ Eladio Gutiérrez§ Oscar Plata§ Onur Mutlu‡

§University of Malaga †National Technical University of Athens ‡ETH Zürich
Several Questions in 3D-Stacked PIM

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

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming
Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoungh Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"
[Slides (pdf)] [Lightning Session Slides (pdf)]
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
Simple PIM Operations as ISA Extensions (II)

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

Host Processor

Main Memory

Conventional Architecture

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

**In-Memory Addition**

- **8 bytes in**
- **0 bytes out**

**Host Processor**

**Main Memory**

pim.add r1, (r2)
Always Executing in Memory? Not A Good Idea

Increased Memory Bandwidth Consumption
Caching very effective

Reduced Memory Bandwidth Consumption due to In-Memory Computation

More Vertices
PEI: PIM-Enabled Instructions (Example)

```c
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>
PIM-Enabled Instructions

- Key to practicality: single-cache-block restriction
  - Each PEI can access *at most one last-level cache block*
  - Similar restrictions exist in atomic instructions

- Benefits
  - **Localization**: each PEI is bounded to one memory module
  - **Interoperability**: easier support for cache coherence and virtual memory
  - **Simplified locality monitoring**: data locality of PEIs can be identified simply by the cache control logic
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
Evaluated Data-Intensive Applications

- Ten emerging data-intensive workloads
  - Large-scale graph processing
    - Average teenage follower, BFS, PageRank, single-source shortest path, weakly connected components
  - In-memory data analytics
    - Hash join, histogram, radix partitioning
  - Machine learning and data mining
    - Streamcluster, SVM-RFE

- Three input sets (small, medium, large) for each workload to show the impact of data locality
PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: Host-Only)
PEI Performance: Large Data Sets

Normalized Amount of Off-chip Transfer

- ATF
- BFS
- PR
- SP
- WCC
- HJ
- HG
- RP
- SC
- SVM

Types:
- Host-Only
- PIM-Only
- Locality-Aware
PEI Performance Delta: Small Data Sets

(Small Inputs, Baseline: Host-Only)

-60% -40% -20% 0% 20% 40% 60%

ATF  BFS  PR  SP  WCC  HJ  HG  RP  SC  SVM  GM

PIM-Only  Locality-Aware
PEI Performance: Small Data Sets

Normalized Amount of Off-chip Transfer

- ATF
- BFS
- PR
- SP
- WCC
- HJ
- HG
- RP
- SC
- SVM

Host-Only
PIM-Only
Locality-Aware
PEI Performance Delta: Medium Data Sets

(Medium Inputs, Baseline: Host-Only)

-10% 0% 10% 20% 30% 40% 50% 60% 70%

ATF BFS PR SP WCC HJ HG RP SC SVM GM

PIM-Only  Locality-Aware

SAFARI
PEI Energy Consumption

![Bar chart showing energy consumption for different sizes: Small, Medium, Large.](chart)

- **Host-Only**
- **PIM-Only**
- **Locality-Aware**

**Legend:**
- **Dark Blue:** Cache
- **Light Blue:** Host-side PCU
- **Light Cyan:** HMC Link
- **Light Purple:** Memory-side PCU
- **Light Green:** DRAM
- **Light Yellow:** PMU

**Dimensions:**
- Small: [Energy Consumption](chart)
- Medium: [Energy Consumption](chart)
- Large: [Energy Consumption](chart)
PEI: Advantages & Disadvantages

- **Advantages**
  + Simple and low cost approach to PIM
  + No changes to programming model, virtual memory
  + Dynamically decides where to execute an instruction

- **Disadvantages**
  - Does not take full advantage of PIM potential
    - Single cache block restriction is limiting
Simpler PIM: PIM-Enabled Instructions

[Slides (pdf)] [Lightning Session Slides (pdf)]
Automatic Code and Data Mapping

Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler,
"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]
Automatic Offloading of Critical Code


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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

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

- Milad Hashemi, Onur Mutlu, and Yale N. Patt, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads". Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016. [Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

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

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

"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
Challenge and Opportunity for Future

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures
Fundamentally
High-Performance
(Data-Centric)
Computing Architectures
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
Sub-Agenda: In-Memory Computation

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

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

1. Functionality of and applications & software for PIM

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

3. System support: coherence & virtual memory

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

5. Infrastructures to assess benefits and feasibility

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

We can get there step by step
Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun,
"Processing Data Where It Makes Sense: Enabling In-Memory Computation"
[arXiv version]

PIM Review and Open Problems (II)

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

Saugata Ghose† Amirali Boroumand† Jeremie S. Kim†§ Juan Gómez-Luna§ Onur Mutlu§†
†Carnegie Mellon University §ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
[Preliminary arXiv version]
Key Challenge 1: Code Mapping

- Challenge 1: Which operations should be executed in memory vs. in CPU?

Code snippet:
```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;
}
```
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?

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

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

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


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

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

\textsuperscript{1}Pennsylvania State University \textsuperscript{2}College of William and Mary \textsuperscript{3}Advanced Micro Devices, Inc. \textsuperscript{4}Intel Labs \textsuperscript{5}ETH Zürich \textsuperscript{6}Carnegie Mellon University
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)]
Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

*The University of Texas at Austin   §ETH Zürich
Challenge: Coherence for Hybrid CPU-PIM Apps

![Graph showing speedup comparison between Traditional coherence and No coherence overhead. The graph compares different applications and their performance with various coherence strategies.]

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

Speedup

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Applications:
- Components
- Radii
- PageRank
- Components
- Radii
- PageRank
- Components
- Radii
- PageRank
- HTAP-256
- HTAP-128
- GMean

arXiv
Gnutella
Enron
IMDB

No coherence overhead
How to Maintain Coherence? (I)


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 Maintain Coherence? (II)

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

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


CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

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

†Carnegie Mellon University
○Simon Fraser University
*ETH Zürich
§Samsung Semiconductor, Inc.
‡KMUTNB

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

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng, Onur Mutlu
Specialized accelerators are now everywhere!

Recent advancement in 3D-stacked technology enabled Near-Data Accelerators (NDA)
Coherence For NDAs

Challenge: Coherence between NDAs and CPUs

(1) Large cost of off-chip communication

(2) NDA applications generate a large amount of off-chip data movement

It is impractical to use traditional coherence protocols
# Existing Coherence Mechanisms

We extensively study existing **NDA coherence mechanisms** and make **three key observations**:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>These mechanisms <strong>eliminate</strong> a significant portion of <strong>NDA’s benefits</strong></td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>The <strong>majority of off-chip coherence traffic</strong> generated by these mechanisms is <strong>unnecessary</strong></td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>Much of the <strong>off-chip traffic</strong> can be <strong>eliminated</strong> if the <strong>coherence mechanism</strong> has <strong>insight</strong> into the <strong>memory accesses</strong></td>
</tr>
</tbody>
</table>
An Optimistic Approach

We find that an optimistic approach to coherence can address the challenges related to NDA coherence.

1. Gain insights *before* any coherence checks happen.
2. Perform *only the necessary* coherence requests.
We propose CoNDA, a mechanism that uses optimistic NDA execution to avoid unnecessary coherence traffic.
We propose CoNDA, a mechanism that uses optimistic NDA execution to avoid unnecessary coherence traffic. CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism.
CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng, Onur Mutlu
How to Maintain Coherence? (II)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

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

◊Carnegie Mellon University
†Carnegie Mellon University
‡ETH Zürich
§KMUTNB
$Samsung Semiconductor, Inc.

SAFARI
How to Support Virtual Memory?

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

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

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


Concurrent Data Structures for Near-Memory Computing

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

Irina Calciu
VMware Research Group
icalciu@vmware.com

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

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

- **Ramulator** extended for PIM
  - Flexible and extensible DRAM simulator
  - Can model many different memory standards and proposals
  - Kim+, “**Ramulator: A Flexible and Extensible DRAM Simulator**”, IEEE CAL 2015.
  - [https://github.com/CMU-SAFARI/ramulator-pim](https://github.com/CMU-SAFARI/ramulator-pim)
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)
  - [Source Code for Ramulator-PIM](https://github.com/CMU-SAFARI/ramulator-pim)
Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning"

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

[Slides (pptx) (pdf)]
[Poster (pptx) (pdf)]
[Source Code for Ramulator-PIM]
An FPGA-based Test-bed for PIM?


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

*github.com/CMU-SAFARI/SoftMC*
MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol†, Juan Gómez-Luna†, Mohammad Sadrosadati†, Saugata Ghose‡, Onur Mutlu††

†ETH Zürich  ‡Carnegie Mellon University
New Applications and Use Cases for PIM

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

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

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

From The Sixteenth Asia Pacific Bioinformatics Conference 2018  
Yokohama, Japan. 15-17 January 2018
Genome Read In-Memory (GRIM) Filter:
Fast Seed Location Filtering in DNA Read Mapping using Processing-in-Memory Technologies

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

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

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

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

- We implement GRIM-Filter using **in-memory processing** within **3D-stacked memory** and show up to **3.7x speedup**.
Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand
Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, Onur Mutlu

SAFARI  Carnegie Mellon  Google  SAMSUNG  SEOUL NATIONAL UNIVERSITY  ETH Zürich
Accelerating Climate Modeling

- Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal,

"NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"

Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, September 2020.
[Slides (pptx) (pdf)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (23 minutes)]

Nominated for the Stamatis Vassiliadis Memorial Award.

NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh\textsuperscript{a,b,c}, Dionysios Diamantopoulos\textsuperscript{c}, Christoph Hagleitner\textsuperscript{c}, Juan Gómez-Luna\textsuperscript{b}, Sander Stuijk\textsuperscript{a}, Onur Mutlu\textsuperscript{b}, Henk Corporaal\textsuperscript{a}

\textsuperscript{a}Eindhoven University of Technology
\textsuperscript{b}ETH Zürich
\textsuperscript{c}IBM Research Europe, Zurich

SAFARI
Accelerating Approximate String Matching


[Lighting Talk Video (1.5 minutes)]
[Lightning Talk Slides (pptx) (pdf)]
[Talk Video (18 minutes)]
[Slides (pptx) (pdf)]
Accelerating Time Series Analysis

- Ivan Fernandez, Ricardo Quislant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu,

"NATSA: A Near-Data Processing Accelerator for Time Series Analysis"

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

Ivan Fernandez§ Ricardo Quislant§ Christina Giannoula† Mohammed Alser‡
Juan Gómez-Luna‡ Eladio Gutiérrez§ Oscar Plata§ Onur Mutlu‡

§University of Malaga †National Technical University of Athens ‡ETH Zürich
Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu$^{a,b}$, Saugata Ghose$^b$, Juan Gómez-Luna$^a$, Rachata Ausavarunrnirun$^{b,c}$

$^a$ETH Zürich
$^b$Carnegie Mellon University
$^c$King Mongkut’s University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"
*Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.*
A Workload and Programming Ease Driven Perspective of Processing-in-Memory

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

†Carnegie Mellon University  §ETH Zürich

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

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
One Important Takeaway

Main Memory Needs

Intelligent Controllers
Enabling the Paradigm Shift
Recall: Computer Architecture Today

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

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

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

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

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

  - 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.
UPMEM Processing-in-DRAM Engine (2019)

- **Processing in DRAM Engine**

  - Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.

- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - **Large amounts of** compute & memory bandwidth

---

Sub-Agenda: In-Memory Computation

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


Challenge and Opportunity for Future

Fundamentally
High-Performance
(Data-Centric)
Computing Architectures
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
PIM: 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

Source: By Toni_V, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=4087256
Another Principled Design

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


The Overarching Principle

Zoomorphic architecture

From Wikipedia, the free encyclopedia

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

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

- It is time to design principled system architectures to solve the memory problem

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

- Enable computation capability inside and close to memory

- This can
  - Lead to orders-of-magnitude improvements
  - Enable new applications & computing platforms
  - Enable better understanding of nature
  - ...
The Future of Processing in Memory is Bright

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

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

Yet, we have to
- Think across the stack
- Design enabling systems
We Need to Revisit the Entire Stack

We can get there step by step
If In Doubt, See Other Doubtful Technologies

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

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

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

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarunngirun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"  
Invited paper in Microprocessors and Microsystems (MICPRO), June 2019.
[arXiv version]

PIM Review and Open Problems (II)

A Workload and Programming Ease Driven Perspective of Processing-in-Memory
Saugata Ghose† Amirali Boroumand† Jeremie S. Kim‡§ Juan Gómez-Luna§ Onur Mutlu§†
†Carnegie Mellon University §ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
Invited Article in IBM Journal of Research & Development, Special Issue on
[Preliminary arXiv version]

SAFARI
Computer Architecture
Lecture 6: Computation in Memory

Prof. Onur Mutlu
ETH Zürich
Fall 2020
8 October 2020
We Did Not Cover The Later Slides. They Are For Your Benefit.
Accelerating Linked Data Structures

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

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

Kevin Hsieh† Samira Khan‡ Nandita Vijaykumar†
Kevin K. Chang† Amirali Boroumand† Saugata Ghose† Onur Mutlu§†
†Carnegie Mellon University ‡University of Virginia §ETH Zürich

SAFARI
Executive Summary

• Our Goal: Accelerating pointer chasing inside main memory

• Challenges: Parallelism challenge and Address translation challenge

• Our Solution: In-Memory Pointer Chasing Accelerator (IMPICA)
  • Address-access decoupling: enabling parallelism in the accelerator with low cost
  • IMPICA page table: low cost page table in logic layer

• Key Results:
  • 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  • 6% - 41% reduction in energy consumption
Linked Data Structures

• Linked data structures are widely used in many important applications.
The Problem: Pointer Chasing

• Traversing linked data structures requires chasing pointers

Find(A)

Serialized and irregular access pattern
6X cycles per instruction in real workloads
Our Goal

Accelerating pointer chasing inside main memory

Find(A)

DRAM layers

Logic layer

Data (A)

CPU

Find (A)
Parallelism Challenge

CPU core

- Comp
- Memory access
- Comp

CPU core

- Comp
- Memory access
- Comp

In-Memory Accelerator

- Comp
- Memory access
- Comp
- Comp
- Memory access
- Comp

Time

Slower for two operations
Parallelism Challenge and Opportunity

• A simple in-memory accelerator can still be **slower** than multiple CPU cores

• **Opportunity:** a pointer-chasing accelerator spends a long time waiting for memory

```
  CPU core  CPU core  CPU core
  |
  v
Accelerator
  |
  v
Comp | Memory access (10-15X of Comp) | Comp
```
Our Solution: Address-Access Decoupling

Address-access decoupling enables parallelism in both engines with low cost
Address Translation Challenge

The page table walk requires multiple memory accesses

No TLB/MMU on the memory side
Duplicating it is costly and creates compatibility issue

Page table walk

Virtual Address

47

#PML4 #PDPT #PGD

PML4 PDPT PGD PGT $2^n$
Our Solution: IMPICA Page Table

- Completely **decouple the page table of IMPICA from the page table of the CPUs**

Map linked data structure into IMPICA regions

IMPICA page table is a partial-to-any mapping
IMPICA Page Table: Mechanism

Virtual Address

Bit [47:41]

Region Table

Flat page table saves one memory access

Bit [40:21]

Virtual Address

Bit [20:12]

Flat Page Table (2MB)

Tiny region table is almost always in the cache

Bit [11:0]

Small Page Table (4KB)

Physical Address
Evaluation Methodology

• Simulator: gem5

• System Configuration
  • CPU
    • 4 OoO cores, 2GHz
    • Cache: 32KB L1, 1MB L2
  • IMPICA
    • 1 core, 500MHz, 32KB Cache
  • Memory Bandwidth
    • 12.8 GB/s for CPU, 51.2 GB/s for IMPICA

• Our simulator code is open source
  • https://github.com/CMU-SAFARI/IMPICA
Result – Microbenchmark Performance

- Baseline + extra 128KB L2
- IMPICA

- Linked List: 1.9X
- Hash Table: 1.3X
- B-Tree: 1.2X
Result – Database Performance

**Database Throughput**
- Baseline + extra 128KB L2: +2%
- Baseline + extra 1MB L2: +5%
- IMPICA: +16%

**Database Latency**
- Baseline + extra 128KB L2: -0%
- Baseline + extra 1MB L2: -4%
- IMPICA: -13%
System Energy Consumption

![Normalized Energy Comparison](chart)

- **Linked List**: Baseline + extra 128KB L2 saves 41% vs IMPICA.
- **Hash Table**: IMPICA saves 24% vs Baseline + extra 128KB L2.
- **B-Tree**: IMPICA saves 10% vs Baseline + extra 128KB L2.
- **DBx1000**: IMPICA saves 6% vs Baseline + extra 128KB L2.
### Area and Power Overhead

<table>
<thead>
<tr>
<th>Component</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (Cortex-A57)</td>
<td>5.85 mm² per core</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>5 mm² per MB</td>
</tr>
<tr>
<td>Memory Controller</td>
<td>10 mm²</td>
</tr>
<tr>
<td>IMPICA (+32KB cache)</td>
<td>0.45 mm²</td>
</tr>
</tbody>
</table>

- **Power overhead**: average power increases by 5.6%