Memory Systems
and Memory-Centric Computing Systems

Part 3: Computation in Memory

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
3 September 2019
Perugia NiPS Summer School
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?
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.

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

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

The Performance Perspective (Today)

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

![Box plot showing cache-bound cycles for various workloads.](image)

**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
- DRAM Rd/Wr: 16 nJ
- Efficient off-chip link: 500 pJ
- 1 nJ
- 50 pJ
- 26 pJ
- 256 pJ

20 mm
A memory access consumes \( \sim 100\text{-}1000\times \) the energy of a complex addition.
Data Movement vs. Computation Energy

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

---

[1]: Reducing data Movement Energy via Online Data Clustering and Encoding (MICRO’16)
[2]: Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms (IISWC’14)
62.7% of the total system energy is spent on data movement.
We Do Not Want to Move Data!

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

- Enable computation with **minimal data movement**

- **Compute where it makes sense** *(where data resides)*

- Make computing architectures more **data-centric**
Goal: Processing Inside Memory

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

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

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

*SAFARI*
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
Starting Simple: Data Copy and Initialization

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

- Forking
- Zero initialization (e.g., security)
- Checkpointing
- VM Cloning
- Deduplication
- Page Migration
- Many more
Today’s Systems: Bulk Data Copy

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

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

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

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

Idea: Two consecutive ACTivates
Negligible HW cost

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

11.6\times latency reduction, 74\times energy reduction
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)

- Subarray
- Bank I/O

0.01% area cost
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
In-DRAM NOT: Dual Contact Cell

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

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

Performance: In-DRAM Bitwise Operations

Figure 9: Throughput of bitwise operations on various systems.
# Energy of In-DRAM Bitwise Operations

<table>
<thead>
<tr>
<th>Design</th>
<th>not</th>
<th>and/or</th>
<th>nand/nor</th>
<th>xor/xnor</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM &amp; DDR3</td>
<td>93.7</td>
<td>137.9</td>
<td>137.9</td>
<td>137.9</td>
</tr>
<tr>
<td>Channel Energy</td>
<td>1.6</td>
<td>3.2</td>
<td>4.0</td>
<td>5.5</td>
</tr>
<tr>
<td>(nJ/KB) (↓)</td>
<td>59.5X</td>
<td>43.9X</td>
<td>35.1X</td>
<td>25.1X</td>
</tr>
</tbody>
</table>

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

Ambit vs. DDR3: Performance and Energy

Performance Improvement

Energy Reduction

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

Bulk Bitwise Operations in Workloads

- Bitmap indices (database indexing)
- BitWeaving (database queries)
- Set operations
- Encryption algorithms
- 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 \textit{range queries} and \textit{joins}
- Many bitwise operations to perform a query

\begin{itemize}
  \item age < 18
  \item 18 < age < 25
  \item 25 < age < 60
  \item age > 60
\end{itemize}
Performance: Bitmap Index on Ambit

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

>5.4-6.6X Performance Improvement

Performance: BitWeaving on Ambit

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

More on In-DRAM Bulk AND/OR

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

"Fast Bulk Bitwise AND and OR in DRAM"


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  \textsuperscript{2}NVIDIA Research  \textsuperscript{3}Intel  \textsuperscript{4}ETH Zürich  \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.
Yet Another Review from ISCA 2016

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...
Suggestion to Community

We Need to Fix the Reviewer Accountability Problem
Takeaway

Main Memory Needs
Intelligent Controllers
Takeaway

Our Community Needs Accountable Reviewers
Initial RowHammer Reviews

Disturbance Errors in DRAM: Demonstration, Characterization, and Prevention

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

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

<table>
<thead>
<tr>
<th>+ Abstract</th>
<th>+ Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review #66A</td>
<td>OveMer 1</td>
</tr>
<tr>
<td>Review #66B</td>
<td>Nov 5</td>
</tr>
<tr>
<td>Review #66C</td>
<td>WriQua 4</td>
</tr>
<tr>
<td>Review #66D</td>
<td>RevExp 4</td>
</tr>
<tr>
<td>Review #66E</td>
<td>4</td>
</tr>
<tr>
<td>Review #66F</td>
<td>2</td>
</tr>
</tbody>
</table>
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.
More ...

Reviews from ISCA 2014

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-qQSnBcdo

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>
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
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
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]; RLDDRAM3 (2011) [29]</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

![Graph Speedup Diagram]

<table>
<thead>
<tr>
<th>Cores</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>128...</td>
<td>+42%</td>
</tr>
</tbody>
</table>

Speedup: 0, 1, 2, 3, 4
Key Bottlenecks in Graph Processing

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

1. Frequent random memory accesses

2. Little amount of computation
**Tesseract System for Graph Processing**

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

- **Host Processor**
- **Memory**
- **Logic**

Memory-Mapped Accelerator Interface (Noncacheable, Physically Addressed)

Crossbar Network

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

Host Processor

Memory-Mapped Accelerator Interface
(Noncacheable, Physically Addressed)

In-Order Core

Message Queue

Communications via Remote Function Calls

Crossbar Network

Communications In Tesseract (I)

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

![Diagram of graph vertices and weights]
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}

![Diagram of vaults and vertices](image-url)
Communications In Tesseract (III)

```javascript
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

Vault #1

Vault #2
Remote Function Call (Non-Blocking)

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

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

Memory

Logic

Host Processor

Memory-Mapped Accelerator Interface
(Noncacheable, Physically Addressed)

Crossbar Network

Message Queue

Prefetching

DRAM Controller

NI

Prefetching

LP

PF Buffer

MTP

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="DDR3-OoO" alt="Diagram" /></td>
<td><img src="HMC-OoO" alt="Diagram" /></td>
<td><img src="HMC-MC" alt="Diagram" /></td>
<td><img src="Tesseract" alt="Diagram" /></td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>32 Tesseract Cores</td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td></td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td></td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td></td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td></td>
</tr>
<tr>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td>8 OoO 4GHz</td>
<td></td>
</tr>
</tbody>
</table>

102.4GB/s | 640GB/s | 640GB/s | 8TB/s

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

>13X Performance Improvement

On five graph processing algorithms

- DDR3-OoO: +56%
- HMC-OoO: +25%
- HMC-MC: 9.0x
- Tesseract: 11.6x
- Tesseract-LP: 13.8x

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

Memory Bandwidth Consumption

- DDR3-OoO: 80GB/s
- HMC-OoO: 190GB/s
- HMC-MC: 243GB/s
- Tesseract: 1.3TB/s
- Tesseract-LP: 2.2TB/s
- Tesseract-LP-MTP: 2.9TB/s
Effect of Bandwidth & Programming Model

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

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC-MC</td>
<td>1.0x</td>
</tr>
<tr>
<td>HMC-MC + PIM BW</td>
<td>2.3x</td>
</tr>
<tr>
<td>Tesseract + Conventional BW</td>
<td>3.0x</td>
</tr>
<tr>
<td>Tesseract (No Prefetching)</td>
<td>6.5x</td>
</tr>
</tbody>
</table>

Bandwidth

- HMC-MC: 1.0x
- HMC-MC + PIM BW: 2.3x
- Tesseract + Conventional BW: 3.0x
- Tesseract (No Prefetching): 6.5x
Tesseract Graph Processing System Energy

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

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
"A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"
[Slides (pdf)] [Lightning Session Slides (pdf)]

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

Junwhan Ahn  Sungpack Hong§  Sungjoo Yoo  Onur Mutlu†  Kiyoung Choi
junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University  §Oracle Labs  †Carnegie Mellon University
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
Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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
- **Video Playback**: Google’s video codec
- **Video Capture**: Google’s video codec
Energy Cost of Data Movement

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

Potential solution: move computation close to data

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

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

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

Small embedded low-power core

Small fixed-function accelerators

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

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

Video Playback
VP9
Google’s video codec

Video Capture
VP9
Google’s video codec
TensorFlow Mobile

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

A simple data reorganization process that requires simple arithmetic

Matrix $\rightarrow$ Packing $\rightarrow$ Packed Matrix

**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
Quantization

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

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

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


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

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

SAFARI
Truly Distributed GPU Processing with PIM?

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Main GPU

Logic layer

Crossbar switch

Vault Ctrl

Vault Ctrl

// Work out which pixel we are working on.
const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
const int colIdx = blockIdx.y;
const int sliceIdx = threadIdx.z;

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

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

Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, 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)]
Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

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

\textsuperscript{1}Pennsylvania State University  \textsuperscript{2}College of William and Mary  \textsuperscript{3}Advanced Micro Devices, Inc.  \textsuperscript{4}Intel Labs  \textsuperscript{5}ETH Zürich  \textsuperscript{6}Carnegie Mellon University
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
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
**PEI: PIM-Enabled Instructions (Ideas)**

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

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

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

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

In-Memory Addition

- Host Processor
- Main Memory
- Value: 8 bytes in
- w.next_rank: 0 bytes out
- `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)

```
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>

120

SAFARI
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**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Host-Only</th>
<th>PIM-Only</th>
<th>Locality-Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HJ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
PEI Performance Delta: Small Data Sets

(Small Inputs, Baseline: Host-Only)
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)

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

**Legend:**
- PIM-Only
- Locality-Aware
PEI Energy Consumption

- Small
- Medium
- Large

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

Host-Only
PIM-Only
Locality-Aware
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


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

Junwhan Ahn  Sungjoo Yoo  Onur Mutlu†  Kiyoungh Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr

Seoul National University  †Carnegie Mellon University

SAFARI
Automatic Code and Data Mapping

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

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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

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

---

- 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
Efficient Automatic Data Coherence Support

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

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


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

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

†Carnegie Mellon University  *Samsung Semiconductor, Inc.  §TOBB ETÜ  ††ETH Zürich
Efficient Automatic Data Coherence Support

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

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


CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

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

†Carnegie Mellon University
‡Simon Fraser University
§ETH Zürich
*KMUTNB
$Samsung Semiconductor, Inc.
Challenge and Opportunity for Future

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
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
PIM Review and Open Problems

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

Onur Mutlu\textsuperscript{a,b}, Saugata Ghose\textsuperscript{b}, Juan Gómez-Luna\textsuperscript{a}, Rachata Ausavarungrun\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 Ausavarungrun, "Processing Data Where It Makes Sense: Enabling In-Memory Computation"
[arXiv version]

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

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

†Carnegie Mellon University  §ETH Zürich

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

Key Challenge 1: Code Mapping

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

![Diagram showing main GPU, 3D-stacked memory, SM (Streaming Multiprocessor), Crossbar switch, and Vault Ctrl]
Key Challenge 2: Data Mapping

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

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

  

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
How to Schedule Code? (II)

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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

*The University of Texas at Austin †Apple ‡NVIDIA §ETH Zürich & Carnegie Mellon University
Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

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

The diagram illustrates the performance differences between traditional coherence and no coherence for various applications under two conditions: CPU-only and different PIM configurations.

- **Traditional coherence**
  - Shows the performance of applications under normal coherence settings.

- **No coherence overhead**
  - Demonstrates the performance with zero coherence overhead.

The applications tested include:
- Components
- Radii
- PageRank
- Components
- Radii
- PageRank
- Components
- Radii
- PageRank
- HTAP-256
- HTAP-128
- GMean

The diagram uses different colors to represent different PIM configurations and shows speedup metrics.
How to Maintain Coherence? (I)

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

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


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

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

†Carnegie Mellon University  *Samsung Semiconductor, Inc.  §TOBB ETÜ  †ETH Zürich
How to 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†
Minesh Patel*
Hasan Hassan*
Rachata Ausavarungnirun†‡
Kevin Hsieh†
Nastaran Hajinazar‡
Krishna T. Malladi§
Hongzhong Zheng§
Onur Mutlu*†

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

SAFARI
How to Support Virtual Memory?


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

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


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

SAFARI
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.
  - [GitHub Repository](https://github.com/CMU-SAFARI/ramulator-pim)
  - [GitHub Repository](https://github.com/CMU-SAFARI/ramulator)
  - [Source Code for Ramulator-PIM](https://github.com/CMU-SAFARI/ramulator-pim)
Performance & Energy Models for PIM

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

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

[Slides (pptx) (pdf)]

[Poster (pptx) (pdf)]

[Source Code for Ramulator-PIM]

---

**NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning**

Gagandeep Singh\textsuperscript{a,c}  
Juan Gómez-Luna\textsuperscript{b}  
Stefano Corda\textsuperscript{a,c}  
Sander Stujik\textsuperscript{a}  
Giovanni Mariani\textsuperscript{c}  
Onur Mutlu\textsuperscript{b}  
Geraldo F. Oliveira\textsuperscript{b}  
Henk Corporaal\textsuperscript{a}

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

SAFARI
Hasan Hassan et al., **SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies** HPCA 2017.

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

github.com/CMU-SAFARI/SoftMC
Simulation Infrastructures for PIM (in SSDs)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,

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


[Slides (pptx) (pdf)]
[Source Code]

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 \textit{16th Asia Pacific Bioinformatics Conference (APBC)}, Yokohama, Japan, January 2018.
  \url{arxiv.org Version (pdf)}

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

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

\textit{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
Onur Mutlu, Saugata Ghose, Juan Gómez-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


SAFARI

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

Fundamentally High-Performance (Data-Centric) Computing Architectures
Computing Architectures with Minimal Data Movement
One Important Takeaway

Main Memory Needs
Intelligent Controllers
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
PIM: Concluding Remarks
Concluding Remarks

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

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

- Enable computation capability inside and close to memory

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

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

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

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

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

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

*Proceedings of the IEEE, Sept. 2017*

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

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

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

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]

Memory Systems
and Memory-Centric Computing Systems

Part 3: Computation in Memory

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
3 September 2019
Perugia NiPS Summer School

SAFARI
ETH Zürich
Carnegie Mellon