Computer Architecture

Lecture 3: Processing using Memory

Prof. Onur Mutlu
ETH Zürich
Fall 2022
6 October 2022
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
  - Processing using Memory
  - Processing near 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 AI/ML, Genomics, Medicine, Health
Maslow’s (Human) Hierarchy of Needs, Revisited


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

Source: V. Milutinovic
Challenge and Opportunity for Future

High Performance, Energy Efficient, Sustainable
(All at the Same Time)
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)
It’s the Memory, Stupid!

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

**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**

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

The Performance Perspective

Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"
One of the 15 computer architecture papers of 2003 selected as Top Picks by IEEE Micro.

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**
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):

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

SAFARI

24

64-bit DP
20pJ

256-bit buses

256-bit access
8 kB SRAM

20mm

16 nJ

DRAM
Rd/Wr

Efficient
off-chip link

500 pJ

1 nJ

50 pJ

26 pJ

256 pJ
A memory access consumes \(~100-1000\times\) 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)
Energy Waste in Mobile Devices


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

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

SAFARI
We Do Not Want to Move Data!

Communication Dominates Arithmetic

A memory access consumes $\sim 100\text{-}1000X$ 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?
A Modern Primer on Processing in Memory

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

SAFARI Research Group

\textsuperscript{a}ETH Zürich
\textsuperscript{b}Carnegie Mellon University
\textsuperscript{c}University of Illinois at Urbana-Champaign
\textsuperscript{d}King Mongkut’s University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun, "A Modern Primer on Processing in Memory"

SAFARI

A Modern Primer on Processing in Memory

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

SAFARI Research Group

\textsuperscript{a}ETH Zürich
\textsuperscript{b}Carnegie Mellon University
\textsuperscript{c}University of Illinois at Urbana-Champaign
\textsuperscript{d}King Mongkut’s University of Technology North Bangkok

Abstract

Modern computing systems are overwhelmingly designed to move data to computation. This design choice goes directly against at least three key trends in computing that cause performance, scalability and energy bottlenecks: (1) data access is a key bottleneck as many important applications are increasingly data-intensive, and memory bandwidth and energy do not scale well, (2) energy consumption is a key limiter in almost all computing platforms, especially server and mobile systems, (3) data movement, especially off-chip to on-chip, is very expensive in terms of bandwidth, energy and latency, much more so than computation. These trends are especially severely-felt in the data-intensive server and energy-constrained mobile systems of today.

At the same time, conventional memory technology is facing many technology scaling challenges in terms of reliability, energy, and performance. As a result, memory system architects are open to organizing memory in different ways and making it more intelligent, at the expense of higher cost. The emergence of 3D-stacked memory plus logic, the adoption of error correcting codes inside the latest DRAM chips, proliferation of different main memory standards and chips, specialized for different purposes (e.g., graphics, low-power, high bandwidth, low latency), and the necessity of designing new solutions to serious reliability and security issues, such as the RowHammer phenomenon, are an evidence of this trend.

This chapter discusses recent research that aims to practically enable computation close to data, an approach we call processing-in-memory (PIM). PIM places computation mechanisms in or near where the data is stored (i.e., inside the memory chips, in the logic layer of 3D-stacked memory, or in the memory controllers), so that data movement between the computation units and memory is reduced or eliminated. While the general idea of PIM is not new, we discuss motivating trends in applications as well as memory circuits/technology that greatly exacerbate the need for enabling it in modern computing systems. We examine at least two promising new approaches to designing PIM systems to accelerate important data-intensive applications: (1) processing using memory by exploiting analog operational properties of DRAM chips to perform massively-parallel operations in memory, with low-cost changes, (2) processing near memory by exploiting 3D-stacked memory technology design to provide high memory bandwidth and low memory latency to in-memory logic. In both approaches, we describe and tackle relevant cross-layer research, design, and adoption challenges in devices, architecture, systems, and programming models. Our focus is on the development of in-memory processing designs that can be adopted in real computing platforms at low cost. We conclude by discussing work on solving key challenges to the practical adoption of PIM.

Keywords: memory systems, data movement, main memory, processing-in-memory, near-data processing, computation-in-memory, processing using memory, processing near memory, 3D-stacked memory, non-volatile memory, energy efficiency, high-performance computing, computer architecture, computing paradigm, emerging technologies, memory scaling, technology scaling, dependable systems, robust systems, hardware security, system security, latency, low-latency computing
1. Introduction

Main memory, built using the Dynamic Random Access Memory (DRAM) technology, is a major component in nearly all computing systems, including servers, cloud platforms, mobile/embedded devices, and sensor systems. Across all of these systems, the data working set sizes of modern applications are rapidly growing, while the need for fast analysis of such data is increasing. Thus, main memory is becoming an increasingly significant bottleneck across a wide variety of computing systems and applications [1–26]. Alleviating the main memory bottleneck requires the memory capacity, energy, cost, and performance to all scale in an efficient manner across technology generations. Unfortunately, it has become increasingly difficult in recent years, especially the past decade, to scale all of these dimensions [1, 2, 27–59], and thus the main memory bottleneck has been worsening.

A major reason for the main memory bottleneck is the high energy and latency cost associated with data movement. In modern computers, to perform any operation on data that resides in main memory, the processor must retrieve the data from main memory. This requires the memory controller to issue commands to a DRAM module across a relatively slow and power-hungry off-chip bus (known as the memory channel). The DRAM module sends the requested data across the memory channel, after which the data is placed in the caches and registers. The CPU can perform computation on the data once the data is in its registers. Data movement from the DRAM to the CPU incurs long latency and consumes a significant amount of energy [7–9, 60–64]. These costs are often exacerbated by the fact that much of the data brought into the caches is not reused by the CPU [62, 63, 65, 66], providing little benefit in return for the high latency and energy cost.

The cost of data movement is a fundamental issue with the processor-centric nature of contemporary computer systems. The CPU is considered to be the master in the system, and computation is performed only in the processor (and accelerators). In contrast, data storage and communication units, including the main memory, are treated as unintelligent workers that are incapable of computation. As a result of this processor-centric design paradigm, data moves a lot in the system between the computation units and communication/storage units so that computation can be done on it. With the increasingly data-centric nature of contemporary and emerging applications, the processor-centric design paradigm leads to great inefficiency in performance, energy and cost. For example, most of the real estate within a single compute...
Processing Data
Where It Makes Sense
Processing in/near Memory: An Old Idea


IEEE TRANSACTIONS ON COMPUTERS, VOL. C-18, NO. 8, AUGUST 1969

Cellular Logic-in-Memory Arrays

WILLIAM H. KAUTZ, MEMBER, IEEE

Abstract—As a direct consequence of large-scale integration, many advantages in the design, fabrication, testing, and use of digital circuitry can be achieved if the circuits can be arranged in a two-dimensional iterative, or cellular, array of identical elementary networks, or cells. When a small amount of storage is included in each cell, the same array may be regarded either as a logically enhanced memory array, or as a logic array whose elementary gates and connections can be “programmed” to realize a desired logical behavior.

In this paper the specific engineering features of such cellular logic-in-memory (CLIM) arrays are discussed, and one such special-purpose array, a cellular sorting array, is described in detail to illustrate how these features may be achieved in a particular design. It is shown how the cellular sorting array can be employed as a single-address, multiword memory that keeps in order all words stored within it. It can also be used as a content-addressed memory, a pushdown memory, a buffer memory, and (with a lower logical efficiency) a programmable array for the realization of arbitrary switching functions. A second version of a sorting array, operating on a different sorting principle, is also described.

Index Terms—Cellular logic, large-scale integration, logic arrays logic in memory, push-down memory, sorting, switching functions.

CELL EQUATIONS: \( \hat{x} = \overline{x}y + wy \)

\( s_y = wcx \), \( r_y = wc \overline{x} \)

\( \hat{z} = M(x, y, z) = x\overline{y} + z(x+\overline{y}) \)

Fig. 1. Cellular sorting array I.

https://doi.org/10.1109/T-C.1969.222754
Processing in/near Memory: An Old Idea


A Logic-in-Memory Computer

HAROLD S. STONE

Abstract—If, as presently projected, the cost of microelectronic arrays in the future will tend to reflect the number of pins on the array rather than the number of gates, the logic-in-memory array is an extremely attractive computer component. Such an array is essentially a microelectronic memory with some combinational logic associated with each storage element.
Why In-Memory Computation Today?

- **Huge problems with Memory Technology**
  - Memory technology scaling is not going well (e.g., RowHammer)
  - Many scaling issues demand intelligence in memory

- **Huge demand from Applications & Systems**
  - Data access bottleneck
  - Energy & power bottlenecks
  - Data movement energy dominates computation energy
  - Need all at the same time: performance, energy, sustainability
  - We can improve all metrics by minimizing data movement

- **Designs are squeezed in the middle**
Processing-in-Memory Landscape Today

And, many other experimental chips and startups
Why In-Memory Computation Today?

- **Huge problems with Memory Technology**
  - Memory technology scaling is not going well (e.g., RowHammer)
  - Many scaling issues demand intelligence in memory

- **Huge demand from Applications & Systems**
  - Data access bottleneck
  - Energy & power bottlenecks
  - Data movement energy dominates computation energy
  - Need all at the same time: performance, energy, sustainability
  - We can improve all metrics by minimizing data movement

- **Designs are squeezed in the middle**
The Push from Circuits and Devices

Main Memory Needs

Intelligent Controllers
Memory Scaling Issues Are Real

- Onur Mutlu,
  "Memory Scaling: A Systems Architecture Perspective"
  Proceedings of the 5th International Memory Workshop (IMW), Monterey, CA, May 2013. Slides (pptx) (pdf)
  EETimes Reprint

Memory Scaling: A Systems Architecture Perspective

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/

Application Scaling Issues Are Real

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
"A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
Top Picks Honorable Mention by IEEE Micro.

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
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

UPMEM Memory Modules

• E19: 8 chips DIMM (1 rank). DPUs @ 267 MHz
• P21: 16 chips DIMM (2 ranks). DPUs @ 350 MHz
2,560-DPU Processing-in-Memory System

Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
IZZAT EL HAJJ, American University of Beirut, Lebanon
IVAN FERNANDEZ, ETH Zürich, Switzerland and University of Malaga, Spain
CHRISTINA CIANNOLA, ETH Zürich, Switzerland and NTUA, Greece
GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
ONUR MUTLU, ETH Zürich, Switzerland

Many modern workloads, such as neural networks, databases, and graph processing, are fundamentally memory-bound. For such workloads, the data movement between main memory and CPU cores imposes a significant overhead in terms of both latency and energy. A major reason is that this communication happens through a narrow bus with high latency and limited bandwidth, and the low data reuse in memory-bound workloads is manifest in the cost of main memory access. Fundamentally addressing this data movement bottleneck requires a paradigm where the memory system assumes an active role in computing by integrating processing capabilities. This paradigm is known as processing-in-memory (PIM).

Recent research explores different forms of PIM architectures, motivated by the emergence of new 3D-stacked memory technologies that integrate memory with a logic layer where processing elements can be easily placed. Past works evaluate these architectures in simulation or, at best, with simplified hardware prototypes. In contrast, the UPMEM company has designed and manufactured the first publicly available real-world PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called DRAM Processing Units (DPU), integrated in the same chip.

This paper provides the first comprehensive analysis of the first publicly available real-world PIM architecture. We make two key contributions. First, we conduct an experimental characterization of the UPMEM-based PIM system using microbenchmarks to assess various architecture limits such as compute throughput and memory bandwidth, yielding new insights. Second, we present PIM benchmarks (in-memory benchmarks), a benchmark suite of 16 workloads from different application domains (e.g., dense and sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing), which we identify as memory-bound. We evaluate the performance and scaling characteristics of PIM benchmarks on the UPMEM PIM architecture, and compare their performance and energy consumption to their state-of-the-art CPU and GPU counterparts. Our extensive evaluation conducted on two real UPMEM-based PIM systems with 140 and 2,560 DPU provides new insights about suitability of different workloads to the PIM system, programming recommendations for software designers, and suggestions and hints for hardware and architecture designers of future PIM systems.

Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System

JUAN GÓMEZ-LUNA¹, IZZAT EL HAJJ², IVAN FERNANDEZ¹,³, CHRISTINA GIANNOTTA¹,⁴, GERALDO F. OLIVEIRA¹, AND ONUR MUTLU¹

¹ETH Zürich
²American University of Beirut
³University of Malaga
⁴National Technical University of Athens

Corresponding author: Juan Gómez-Luna (e-mail: juang@ethz.ch).

https://github.com/CMU-SAFARI/prim-benchmarks
Samsung Develops Industry’s First High Bandwidth Memory with AI Processing Power

The new architecture will deliver over twice the system performance and reduce energy consumption by more than 70%

Samsung Electronics, the world leader in advanced memory technology, today announced that it has developed the industry’s first High Bandwidth Memory (HBM) integrated with artificial intelligence (AI) processing power – the HBM-PIM. The new processing-in-memory (PIM) architecture brings powerful AI computing capabilities inside high-performance memory, to accelerate large-scale processing in data centers, high performance computing (HPC) systems and AI-enabled mobile applications.

Kwangil Park, senior vice president of Memory Product Planning at Samsung Electronics stated, “Our groundbreaking HBM-PIM is the industry’s first programmable PIM solution tailored for diverse AI-driven workloads such as HPC, training and inference. We plan to build upon this breakthrough by further collaborating with AI solution providers for even more advanced PIM-powered applications.”

Samsung Function-in-Memory DRAM (2021)

- FIMDRAM based on HBM2

**Chip Specification**

- 128DQ / 8CH / 16 banks / BL4
- 32 PCU blocks (1 FIM block/2 banks)
- 1.2 TFLOPS (4H)
- FP16 ADD / Multiply (MUL) / Multiply-Accumulate (MAC) / Multiply-and-Add (MAD)

[3D Chip Structure of HBM with FIMDRAM]
Programmable Computing Unit

- Configuration of PCU block
  - Interface unit to control data flow
  - Execution unit to perform operations
  - Register group
    - 32 entries of CRF for instruction memory
    - 16 GRF for weight and accumulation
    - 16 SRF to store constants for MAC operations

[Block diagram of PCU in FIMDRAM]
# Samsung Function-in-Memory DRAM (2021)

## Available instruction list for FIM operation

<table>
<thead>
<tr>
<th>Type</th>
<th>CMD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Point</td>
<td>ADD</td>
<td>FP16 addition</td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td>FP16 multiplication</td>
</tr>
<tr>
<td></td>
<td>MAC</td>
<td>FP16 multiply-accumulate</td>
</tr>
<tr>
<td></td>
<td>MAD</td>
<td>FP16 multiply and add</td>
</tr>
<tr>
<td>Data Path</td>
<td>MOVE</td>
<td>Load or store data</td>
</tr>
<tr>
<td></td>
<td>FILL</td>
<td>Copy data from bank to GRFs</td>
</tr>
<tr>
<td>Control Path</td>
<td>NOP</td>
<td>Do nothing</td>
</tr>
<tr>
<td></td>
<td>JUMP</td>
<td>Jump instruction</td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
<td>Exit instruction</td>
</tr>
</tbody>
</table>

*Available instruction list for FIM operation*

---

ISSCC 2021 / SESSION 25 / DRAM / 25.4

25.4 A 20nm 60Gb Function-In-Memory DRAM, Based on HBM2 with a 1.2TLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

Young Soon Kee 1, Suk Ha Lee 1, Jeon Hee Lee 1, Sang Hyeok Lee 1, Jang Hyeong Nam 1, Kwon Hyung Kwon 1, Ju Min Ryu 1, Jeong Pil Son 1, Seung Gi Cho 1, Hak Soo Yu 1, Ha Seok Lee 1, Seo Yong Kim 1, Younseong Che 1, Jin Suk Kim 1, Jeongmin Choi 1, Hyun-Soon Shin 1, Jin Kwon 1, Bong I. Park 1, Young Min Kim 1, Myeong Jun Song 1, Ahn Chul 1, Bambino Kim 1, Geo Young Kim 1, Eun Seong Kim 1, Chong Wook Kang 1, Hyeo Min Lee 1, Seongmin Son 1, Jeonho Song 1, JeongHyun Youn 1, Kyohyun Seo 1, Nam Soon Kim 1

1Sungshin Electronics, Seoul, Korea
2Samsung Electronics, San Jose, CA
3Samsung Electronics, Suwon, Korea
Chip Implementation

- Mixed design methodology to implement FIMDRAM
  - Full-custom + Digital RTL
Lecture on FIMDRAM/HBM-PIM

FIMDRAM: Bank-level Parallelism

- Unlike standard DRAM devices, all banks can be accessed concurrently for 8x higher bandwidth (with 16 pCHs)
- In AB-PIM mode, a memory command triggers a PIM instruction in the CRF

Processing-in-Memory Course: Lecture 4: Real-world PIM: Samsung HBM-PIM Architecture - Spring 2022

673 views • Streamed live on Mar 31, 2022

Onur Mutlu Lectures
25.9K subscribers

https://youtu.be/_CpWJGK9N04
Samsung AxDIMM (2021)

- DIMM-based PIM
  - DLRM recommendation system

Lecture on AxDIMM

Processing-in-Memory Course: Lecture 9: Real-world PIM: Samsung AxDIMM - Spring 2022

https://youtu.be/J_prUKfnv7Q
SK Hynix AiM: Chip Implementation (2022)

- 4 Gb AiM die with 16 processing units (PUs)

**AiM Die Photograph**

**1 Process Unit (PU) Area**

<table>
<thead>
<tr>
<th></th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>0.19mm²</td>
</tr>
<tr>
<td>MAC</td>
<td>0.11mm²</td>
</tr>
<tr>
<td>Activation Function (AF)</td>
<td>0.02mm²</td>
</tr>
<tr>
<td>Reservoir Cap.</td>
<td>0.05mm²</td>
</tr>
<tr>
<td>Etc.</td>
<td>0.01mm²</td>
</tr>
</tbody>
</table>

MAC 58%, AF 11%, Reservoir Cap. 26%, Etc. 5%

Lee et al., A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications, ISSCC 2022
SK Hynix AiM: System Organization (2022)

- GDDR6-based AiM architecture

Lee et al., A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications, ISSCC 2022
Lecture on Accelerator-in-Memory

AiM: Adder Tree: Bank-wide Mantisa Shift (BWMS)
- Find MAX EX of 16 EXs
- Obtain the differences
- Shift all MAs by the differences
- Perform MA additions

Processing-in-Memory Course: Lecture 6: Real-world PIM: SK Hynix AiM - Spring 2022

https://youtu.be/NDL77Xdcbss?t=159
Alibaba HB-PNM: Overall Architecture (2022)

- **3D-stacked logic die and DRAM die** vertically bonded by hybrid bonding (HB)

---

Niu et al., 184QPS/W 64Mb/mm2 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System, ISSCC 2022
Lecture on HB-PNM

Match Engine: Distance Calculator

- Distance calculator obtains similarity between input query and feature vectors
- It computes Hamming distance of two 512-bit vectors
- Distance is filtered by root of max-heap

Processing-in-Memory Course: Lecture 10: Real-world PIM: Alibaba HB-PNM - Spring 2022

410 views • Premiered May 12, 2022

https://youtu.be/OZjKnn-DbwA
FPGA-based Processing Near Memory


FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications

Gagandeep Singh\textsuperscript{\diamond}  Mohammed Alser\textsuperscript{\diamond}  Damla Senol Cali\textsuperscript{\#}
Dionysios Diamantopoulos\textsuperscript{\nabla}  Juan Gómez-Luna\textsuperscript{\diamond}
Henk Corporaal\textsuperscript{*}  Onur Mutlu\textsuperscript{\#\#}

\textsuperscript{\diamond}ETH Zürich  \textsuperscript{\#\#}Carnegie Mellon University
\textsuperscript{*}Eindhoven University of Technology  \textsuperscript{\nabla}IBM Research Europe

SAFARI
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
The Push from Circuits and Devices

Main Memory Needs

Intelligent Controllers
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**
  - Processing using Memory
  - Processing near Memory
- How to Enable Adoption of Processing in Memory
- Conclusion
5.2. Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM)

Many recent works take advantage of the memory technology innovations that we discuss in Section 5.1 to enable and implement PIM. We find that these works generally take one of two approaches, which are categorized in Table 1: (1) processing using memory or (2) processing near memory. We briefly describe each approach here. Sections 6 and 7 will provide example approaches and more detail for both.

Table 1: Summary of enabling technologies for the two approaches to PIM used by recent works. Adapted from [341] and extended.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Example Enabling Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Using Memory</td>
<td>SRAM, DRAM, Phase-change memory (PCM), Magnetic RAM (MRAM), Resistive RAM (RRAM)/memristors</td>
</tr>
<tr>
<td>Processing Near Memory</td>
<td>Logic layers in 3D-stacked memory, Silicon interposers, Logic in memory controllers, Logic in memory chips (e.g., near bank), Logic in memory modules, Logic near caches, Logic near/in storage devices</td>
</tr>
</tbody>
</table>

Processing in Memory: Two Approaches

1. Processing using Memory
2. Processing near Memory
Approach 1: Processing Using Memory

- Take advantage of operational principles of memory to perform bulk data movement and computation in memory
  - 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‡ Srichari 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

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
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

CPU — L1 — L2 — L3 — MC

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

Idea: Two consecutive ACTivates with Negligible HW cost

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

\[ V_{DD}/2 + \delta \]

Data gets copied

Sense Amplifier (Row Buffer)

Amplify the difference
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)
- Subarray
- Bank
- Bank I/O

0.01% area cost
RowClone: Latency and Energy Savings

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

- Inter-Subarray: 11.6x
- Inter-Bank: 1.9x
- Intra-Subarray: 1.0x
- Copy: 6.0x

Energy Reduction

- Inter-Subarray: 74.4x
- Inter-Bank: 3.2x
- Intra-Subarray: 1.5x
- Copy: 41.5x
- Zero: 1.0x

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

![Bar chart showing the fraction of memory traffic for different workloads: bootup, compile, forkbench, mcached, mysql, and shell. The chart indicates the percentage of zero copy, copy, write, and read operations for each workload.](chart.png)
RowClone: Application Performance

% Compared to Baseline

- IPC Improvement
- Energy Reduction

bootup  compile  forkbench  mcached  mysql  shell
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?
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)]
Lecture on RowClone & Processing using DRAM

Mindset: Memory as an Accelerator

Memory similar to a “conventional” accelerator

https://www.youtube.com/watch?v=n6Pwg1qax_E&list=PL5Q2soXY2Zi_7UBNmC9B8Yr5JSwTG9yH4&index=4
Mindset: Memory as an Accelerator

Memory similar to a “conventional” accelerator
RowClone Strengths
Strengths of the Paper

- Simple, novel mechanism to solve an important problem
- Effective and low hardware overhead
- Intuitive idea!
- Greatly improves performance and efficiency (assuming data is mapped nicely)
- Seems like a clear win for data initialization (without mapping requirements)
- Makes software designer’s life easier
  - If copies are 10x-100x cheaper, how to design software?

- Paper tackles many low-level and system-level issues
- Well-written, insightful paper
RowClone Weaknesses
Weaknesses

- Requires data to be mapped in the same subarray to deliver the largest benefits
  - Helps less if data movement is not within a subarray
  - Does not help if data movement is across DRAM channels
- Inter-subarray copy is very inefficient
- Causes many changes in the system stack
  - End-to-end design spans applications to circuits
  - Software-hardware cooperative solution might not always be easy to adopt
- Cache coherence and data reuse cause real overheads
- Evaluation is done solely in simulation
- Evaluation does not consider multi-chip systems
- Are these the best workloads to evaluate?
Recall: Try to Avoid Rat Holes

Performance Analysis Rat Holes

Workload  Metrics  Configuration  Details

Source: https://www.cse.wustl.edu/~jain/iucee/ftp/k_10adp.pdf
Improvements on RowClone
RowClone Extensions and Follow-Up Work

- Can we do faster inter-subarray copy?
  - Yes, see LISA [Chang et al., HPCA 2016]

- Can we enable data movement at smaller granularities within a bank?
  - Yes, see FIGARO [Wang et al., MICRO 2020]

- Can we do better inter-bank copy?
  - Yes, see Network-on-Memory [CAL 2020]

- Can similar ideas and DRAM properties be used to perform computation on data?
  - Yes, see Ambit [Seshadri et al., CAL 2015, MICRO 2017]
LISA: Increasing Connectivity in DRAM

Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"

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

Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair*, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi*, and Onur Mutlu†

†Carnegie Mellon University  *Georgia Institute of Technology
Moving Data Inside DRAM?

Goal: Provide a new substrate to enable wide connectivity between subarrays.
Key Idea and Applications

- **Low-cost Inter-linked subarrays (LISA)**
  - Fast bulk data movement between subarrays
  - Wide datapath via isolation transistors: 0.8% DRAM chip area

- **LISA is a versatile substrate → new applications**
  - Fast bulk data copy: Copy latency 1.363ms→0.148ms (9.2x)
    → 66% speedup, -55% DRAM energy
  - In-DRAM caching: Hot data access latency 48.7ns→21.5ns (2.2x)
    → 5% speedup
  - Fast precharge: Precharge latency 13.1ns→5.0ns (2.6x)
    → 8% speedup
More on LISA

Kevin K. Chang, Prashant J. Nair, Saugata Ghose, Donghyuk Lee, Moinuddin K. Qureshi, and Onur Mutlu,
"Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM"
[Slides (pptx) (pdf)]
[Source Code]

Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM

Kevin K. Chang†, Prashant J. Nair*, Donghyuk Lee†, Saugata Ghose†, Moinuddin K. Qureshi*, and Onur Mutlu†

†Carnegie Mellon University  *Georgia Institute of Technology
FIGARO: Fine-Grained In-DRAM Copy

- Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S. Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, and Onur Mutlu,
"FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching"

FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching

Yaohua Wang* Lois Orosa† Xiangjun Peng○* Yang Guo* Saugata Ghose○† Minesh Patel† Jeremie S. Kim† Juan Gómez Luna† Mohammad Sadrosadati§ Nika Mansouri Ghiasi† Onur Mutlu††

*National University of Defense Technology  †ETH Zürich ○Chinese University of Hong Kong
○University of Illinois at Urbana–Champaign ††Carnegie Mellon University  §Institute of Research in Fundamental Sciences
Network-On-Memory: Fast Inter-Bank Copy

- Seyyed Hossein SeyyedAghaei Rezaei, Mehdi Modarressi, Rachata Ausavarungnirun, Mohammad Sadrosadati, Onur Mutlu, and Masoud Daneshtalab,

"NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories"

Mindset: Memory as an Accelerator

Memory similar to a “conventional” accelerator
In-DRAM Bulk Bitwise 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"

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

Proceedings of the 50th International Symposium on Microarchitecture (MICRO), Boston, MA, USA, October 2017.

[Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology](https://example.com)

Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
In-DRAM Bulk Bitwise Execution Paradigm


In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
SIMDRAM Framework for in-DRAM Computing


- [2-page Extended Abstract]
- [Short Talk Slides (pptx) (pdf)]
- [Talk Slides (pptx) (pdf)]
- [Short Talk Video (5 mins)]
- [Full Talk Video (27 mins)]

**SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM**

*Nastaran Hajinazar*¹,²
Nika Mansouri Ghiasi¹

*Geraldo F. Oliveira*¹
Minesh Patel¹
Juan Gómez-Luna¹

Sven Gregorio¹
Mohammed Alser¹
Onur Mutlu¹

João Dinis Ferreira¹
Saugata Ghose³

¹ETH Zürich ²Simon Fraser University ³University of Illinois at Urbana–Champaign
Extensions and Follow-Up Work (II)

- Can this idea be evaluated on a real system? How?
  - Yes, see the ComputeDRAM paper [MICRO 2019]

- Can similar ideas be used in other types of memories?
  Phase Change Memory? RRAM? STT-MRAM?
  - Yes, see the Pinatubo paper [DAC 2016]

- Can charge sharing properties be used for other functions?
  - Yes, see the D-RaNGe [HPCA 2019], DL-PUF [HPCA 2018], QUAC-TRNG [ISCA 2021] works on random numbers & PUFs

- Can we have more efficient solutions to
  - Cache coherence (minimize overhead)
  - Data reuse after copy and initialization
RowClone in Off-the-Shelf DRAM Chips

- Idea: Violate DRAM timing parameters to mimic RowClone

ComputDRAM: 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

Real Processing Using Memory Prototype

- End-to-end RowClone & TRNG using off-the-shelf DRAM chips
- Idea: Violate DRAM timing parameters to mimic RowClone

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun§†, Juan Gómez Luna§, Konstantinos Kanellopoulos§, Behzad Salami§*
Hasan Hassan§, Oğuz Ergin†, Onur Mutlu§
§ETH Zürich, †TOBB ETÜ, *BSC

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Real Processing Using Memory Prototype

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Building a PiDRAM Prototype

To build PiDRAM's prototype on Xilinx ZC706 boards, developers need to use the two sub-projects in this directory. `fpga-zynq` is a repository branched off of UC Berkeley's `fpga-zynq` repository. We use `fpga-zynq` to generate rocket chip designs that support end-to-end DRAM PuM execution. `controller-hardware` is where we keep the main Vivado project and Verilog sources for PiDRAM's memory controller and the top level system design.

Rebuilding Steps

1. Navigate into `fpga-zynq` and read the README file to understand the overall workflow of the repository
   ○ Follow the readme in `fpga-zynq/rocket-chip/riscv-tools` to install dependencies
2. Create the Verilog source of the rocket chip design using the `ZynqCopyFPGAConfig`
   ○ Navigate into zc706, then run `make rocket CONFIG=ZynqCopyFPGAConfig -j number of cores`
3. Copy the generated Verilog file (should be under zc706/src) and overwrite the same file in `controller-hardware/source/hdl/iml/rocket-chip`
4. Open the Vivado project in `controller-hardware/Vivado_Project` using Vivado 2016.2
5. Generate a bitstream
6. Copy the bitstream (`system_top.bit`) to `fpga-zynq/zc706`
7. Use the `./build_script.sh` to generate the new `boot.bin` under `fpga-images-zc706`, you can use this file to program the FPGA using the SD-Card
   ○ For details, follow the relevant instructions in `fpga-zynq/README.md`

You can run programs compiled with the RISC-V Toolchain supplied within the `fpga-zynq` repository. To install the toolchain, follow the instructions under `fpga-zynq/rocket-chip/riscv-tools`.

Generating DDR3 Controller IP sources

We cannot provide the sources for the Xilinx PhY IP we use in PiDRAM's memory controller due to licensing issues. We describe here how to regenerate them using Vivado 2016.2. First, you need to generate the IP RTL files:

1. Open IP Catalog
2. Find "Memory Interface Generator (MIG 7 Series)" IP and double click

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li\textsuperscript{1,*}, Cong Xu\textsuperscript{2}, Qiaosha Zou\textsuperscript{1,5}, Jishen Zhao\textsuperscript{3}, Yu Lu\textsuperscript{4}, and Yuan Xie\textsuperscript{1}

University of California, Santa Barbara\textsuperscript{1}, Hewlett Packard Labs\textsuperscript{2}
University of California, Santa Cruz\textsuperscript{3}, Qualcomm Inc.\textsuperscript{4}, Huawei Technologies Inc.\textsuperscript{5}
{shuangchenli, yuanxie}@ece.ucsb.edu\textsuperscript{1}
Takeaways
Key Takeaways

- A novel method to accelerate data copy and initialization
- Simple and effective
- Hardware/software cooperative
- Good potential for work building on it to extend it
  - To different granularities
  - To make things more efficient and effective
  - Many works have already built on the paper (see LISA, FIGARO, NoM, Ambit, ComputeDRAM, and other works in Google Scholar)
- Easy to read and understand paper
RowClone: Memory as an Accelerator

Memory similar to a “conventional” accelerator
Mindset: Processing using 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)
In-Memory

Bulk Bitwise Operations
In-Memory Bulk Bitwise Operations

- We can support **in-DRAM COPY, ZERO, AND, OR, NOT, MAJ**
- At low cost
- **Using inherent 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

\[ \frac{1}{2}V_{\text{DD}} + \delta \]

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\(^\dagger\), Onur Mutlu\(^*\), Phillip B. Gibbons\(^\dagger\), Todd C. Mowry\(^*\)

\(^*\)Carnegie Mellon University \quad \(^\dagger\)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

Figure 5: Bitwise NOT using a dual contact capacitor

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

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

Ambit vs. DDR3: Performance and Energy

- Performance Improvement
- Energy Reduction

Bulk Bitwise Operations in Workloads

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

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

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

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

Bitmap 1  Bitmap 2  Bitmap 3  Bitmap 4
Performance: Bitmap Index on Ambit

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

>5.4-6.6X Performance Improvement 

Performance: BitWeaving on Ambit

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

More on In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
"Fast Bulk Bitwise AND and OR in DRAM"
More on In-DRAM Bitwise Operations


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

Vivek Seshadri¹,⁵ Donghyuk Lee²,⁵ Thomas Mullins³,⁵ Hasan Hassan⁴ Amirali Boroumand⁵ Jeremie Kim⁴,⁵ Michael A. Kozuch³ Onur Mutlu⁴,⁵ Phillip B. Gibbons⁵ Todd C. Mowry⁵

¹Microsoft Research India  ²NVIDIA Research  ³Intel  ⁴ETH Zürich  ⁵Carnegie 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
SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar\(^1,2\) Nika Mansourí Ghiasi\(^1\) *Geraldo F. Oliveira\(^1\) Minesh Patel\(^1\) Sven Gregorio\(^1\) João Dinis Ferreira\(^1\)
Juan Gómez-Luna\(^1\) Mohammed Alser\(^1\)Onur Mutlu\(^1\) Saugata Ghose\(^3\)

\(^1\)ETH Zürich \(^2\)Simon Fraser University \(^3\)University of Illinois at Urbana–Champaign
SIMDRAM Key Idea

- **SIMDRAM**: An end-to-end processing-using-DRAM framework that provides the programming interface, the ISA, and the hardware support for:
  
  - **Efficiently** computing **complex** operations in DRAM
  
  - Providing the ability to implement **arbitrary** operations as required
  
  - Using an **in-DRAM** massively-parallel SIMD substrate that requires **minimal** changes to DRAM architecture
SIMDRAM: PuM Substrate

- SIMDRAM framework is built around a DRAM substrate that enables two techniques:

**1. Vertical data layout**
- most significant bit (MSB)
- least significant bit (LSB)

**2. Majority-based computation**
\[ C_{out} = AB + AC_{in} + BC_{in} \]

**Pros compared to the conventional horizontal layout:**
- Implicit shift operation
- Massive parallelism

**Pros compared to AND/OR/NOT-based computation:**
- Higher performance
- Higher throughput
- Lower energy consumption
SIMDRAM Framework: Overview

**Step 1: Generate MAJ logic**

User Input: Desired operation

MAJ logic

**Step 2: Generate sequence of DRAM commands**

User Input: New SIMDRAM enabled application

SIMDRAM Output: New SIMDRAM µProgram

**Step 3: Execution according to µProgram**

User Input: foo () {

SIMDRAM Output: Instruction result in memory

`bbop_new`
SIMDRAM Framework: Step 1

**User Input**

- Desired operation: AND/OR/NOT logic
- MAJ/NOT logic

**Step 1: Generate MAJ logic**

**SIMDRAM Output**

- New SIMDRAM μProgram
- μProgram
- Main memory
- ISA

**Step 2: Generate sequence of DRAM commands**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/ACT/PRE
- done

**SIMDRAM Output**

- Instruction result in memory
- ACT/PRE

**User Input**

- SIMDRAM-enabled application

```plaintext
foo () {
    bbop_new
}
```

**Control Unit**

**Step 3: Execution according to μProgram**

**Memory Controller**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/PRE/PRE
- done
Step 1: Naïve MAJ/NOT Implementation

Naïvely converting AND/OR/NOT-implementation to MAJ/NOT-implementation leads to an unoptimized circuit
Step 1: Efficient MAJ/NOT Implementation

Greedy optimization algorithm\(^4\)

Step 1 generates an **optimized** MAJ/NOT-implementation of the desired operation

---

SIMDRAM Framework: Step 2

**User Input**

**Desired operation**

- AND/OR/NOT logic

**SIMDRAM Output**

- **Step 2: Generate sequence of DRAM commands**
  - ACT/PRE
  - ACT/PRE
  - ACT/PRE
  - ACT/ACT/PRE
  - done

**Main memory**

- **New SIMDRAM µProgram**
  - µProgram

**User Input**

**SIMDRAM-enabled application**

```c
foo () {
  bbop_new
}
```

**SIMDRAM Output**

**Instruction result in memory**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/PRE/PRE
- done
Step 2: μProgram Generation

• **μProgram**: A series of microarchitectural operations (e.g., ACT/PRE) that SIMDRAM uses to execute SIMDRAM operation in DRAM

• **Goal of Step 2**: To generate the μProgram that executes the desired SIMDRAM operation in DRAM

---

Task 1: Allocate DRAM rows to the operands

Task 2: Generate μProgram
Step 2: μProgram Generation

• **μProgram**: A series of microarchitectural operations (e.g., ACT/PRE) that SIMDGRAM uses to execute SIMDGRAM operation in DRAM

• **Goal of Step 2**: To generate the μProgram that executes the desired SIMDGRAM operation in DRAM

---

Task 1: Allocate DRAM rows to the operands

Task 2: Generate μProgram
Task 1: Allocating DRAM Rows to Operands

• Allocation algorithm considers **two constraints** specific to processing-using-DRAM

![subarray organization]

**Constraint 1:** Limited number of rows reserved for computation

Compute rows
Task 1: Allocating DRAM Rows to Operands

- Allocation algorithm considers two constraints specific to processing-using-DRAM

Constraint 2:
Destructive behavior of triple-row activation

Overwritten with MAJ output

subarray organization
Task 1: Allocating DRAM Rows to Operands

- Allocation algorithm:
  - Assigns as many inputs as the number of free compute rows
  - All three input rows contain the MAJ output and can be reused
Step 2: µProgram Generation

- **µProgram**: A series of microarchitectural operations (e.g., ACT/PRE) that SIMDRAAM uses to execute SIMDRAAM operation in DRAM

- **Goal of Step 2**: To generate the µProgram that executes the desired SIMDRAAM operation in DRAM

Task 1: Allocate DRAM rows to the operands

Task 2: Generate µProgram
Task 2: Generate an initial μProgram

1. Generate μProgram

1. Generate μProgram

1. Generate μProgram
Task 2: Optimize the μProgram

1. Generate μProgram
   - 1. Copy A to reserved row (ACT/ACT/PRE)
   - 2. Copy B to reserved row (ACT/ACT/PRE)
   - 3. Copy C<sub>in</sub> to reserved row (ACT/ACT/PRE)
   - 4. Execute MAJ (ACT/PRE)
   - 5. Copy C<sub>out</sub> to destination row (ACT/PRE)

2. Optimize
Task 2: Optimize the µProgram

1. Generate µProgram
2. Optimize

**Initial µProgram**

1. **Copy** A to reserved row (ACT/ACT/PRE)
2. **Copy** B to reserved row (ACT/ACT/PRE)
3. **Copy** $C_{in}$ to reserved row (ACT/ACT/PRE)
4. Execute MAJ (ACT/PRE)
5. Copy $C_{out}$ to destination row (ACT/PRE)

**Coalesce row copies**
Task 2: Optimize the µProgram

Initial µProgram

1. Copy A to reserved row (ACT/ACT/PRE)
2. Copy B to reserved row (ACT/ACT/PRE)
3. Copy C<sub>in</sub> to reserved row (ACT/ACT/PRE)
4. Execute MAJ (ACT/PRE)
5. **Copy** C<sub>out</sub> to destination row (ACT/PRE)

Merge MAJ + row copy

1. Generate µProgram
2. Optimize

SAFARI
Task 2: Optimize the µProgram

1. Generate µProgram
2. Optimize

Optimized µProgram

1. Copy A, B, C_in to reserved rows (ACT/ACT/PRE)
2. Execute MAJ and copy C_out to destination row (ACT/ACT/PRE)

Coalesce row copies
Merge MAJ + row copy
Task 2: Generate N-bit Computation

- **Final µProgram** is optimized and computes the desired operation for operands of N-bit size in a bit-serial fashion.

1. **Generate µProgram**

   - Copy A, B, C_in to reserved rows (ACT/ACT/PRE)

   - Execute MAJ and copy C_out to destination row (ACT/ACT/PRE)

2. **Optimize**

3. **Generate N-bit computation**

- **Optimized µProgram**

Repeat N times:

   1. Copy A, B, C_in to reserved rows (ACT/ACT/PRE)

   2. Execute MAJ and copy C_out to destination row (ACT/ACT/PRE)
Task 2: Generate µProgram

- **Final µProgram** is optimized and computes the desired operation for operands of N-bit size in a bit-serial fashion.

  - **Final µProgram**
  - Stored in a reserved DRAM region for future use
  - A new SIMD-RAM instruction (**called** `bbop_new`) added to CPU ISA
SIMDRAM Framework: Step 3

**User Input**

Desired operation

AND/OR/NOT logic

**Step 1: Generate MAJ logic**

MAJ/NOT logic

**Step 2: Generate sequence of DRAM commands**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/ACT/PRE
- done

**SIMDRAM Output**

New SIMDRAM μProgram

μProgram

Main memory

bbop_new

New SIMDRAM instruction

ISA

**User Input**

SIMDRAM-enabled application

```foo () {
    bbop_new
}
```

**Control Unit**

**Memory Controller**

**Step 3: Execution according to μProgram**

ACT/PRE

ACT/PRE

ACT/PRE

ACT/PRE/PRE

done

**SIMDRAM Output**

Instruction result in memory

ACT/PRE
Step 3: \textit{μ}Program Execution

- **SIMDRAM control unit:** handles the execution of the \textit{μ}Program at runtime

- Upon receiving a **\textit{bbop} instruction**, the control unit:
  
  1. Loads the \textit{μ}Program corresponding to SIMDRAM operation
  2. Issues the sequence of DRAM commands (\textit{ACT/PRE}) stored in the \textit{μ}Program to SIMDRAM subarrays to perform the in-DRAM operation

```c
SIMDRAM-enabled application

foo () {
  \textit{bbop\_new}
}
```

![Diagram of the execution process](diagram.png)
More in the Paper

- Detailed reference implementation and microarchitecture of the SIMDRAM control unit
System Integration

- Efficiently transposing data
- Programming interface
- Handling page faults, address translation, coherence, and interrupts
- Handling limited subarray size
- Security implications
- Limitations of our framework
Transposing Data

- **SIMDRAM** operates on _vertically-laid-out_ data

- **Other system components** expect data to be laid out _horizontally_

**Challenging** to share data between SIMDRAM and CPU
Transposition Unit

Transforms the data layout from **horizontal** to **vertical**, and vice versa
Efficiently Transposing Data

Low impact on the throughput of SIMD RAM operations

Low area cost (0.06 mm² in 22nm tech. node)
More in the Paper

SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM

*Nastaran Hajinazar\textsuperscript{1,2} \quad *Geraldo F. Oliveira\textsuperscript{1} \quad Sven Gregorio\textsuperscript{1} \quad João Dinis Ferreira\textsuperscript{1}
Nika Mansouri Ghiasi\textsuperscript{1} \quad Minesh Patel\textsuperscript{1} \quad Mohammed Alser\textsuperscript{1} \quad Saugata Ghose\textsuperscript{3}
Juan Gómez-Luna\textsuperscript{1} \quad Onur Mutlu\textsuperscript{1} \quad \textsuperscript{1}ETH Zürich \quad \textsuperscript{2}Simon Fraser University \quad \textsuperscript{3}University of Illinois at Urbana–Champaign

- Efficiently transposing data
- Programming interface
- Handling page faults, address translation, coherence, and interrupts
- Handling limited subarray size
- Security implications
- Limitations of our framework
Methodology: Experimental Setup

• Simulator: gem5

• Baselines:
  - A multi-core CPU (Intel Skylake)
  - A high-end GPU (NVidia Titan V)
  - Ambit: a state-of-the-art in-memory computing mechanism

• Evaluated SIMDARAM configurations (all using a DDR4_2400_x64 device):
  - 1-bank: SIMDARAM exploits 65’536 SIMD lanes (an 8 kB row buffer)
  - 4-banks: SIMDARAM exploits 262’144 SIMD lanes
  - 16-banks: SIMDARAM exploits 1’048’576 SIMD lanes
Methodology: Workloads

Evaluated:

• 16 complex in-DRAM operations:
  - Absolute
  - Addition/Subtraction
  - BitCount
  - Equality/ Greater/Greater Equal
  - Absolute
  - Predication
  - ReLU
  - AND-/OR-/XOR-Reduction
  - Division/Multiplication

• 7 real-world applications
  - BitWeaving (databases)
  - TPH-H (databases)
  - kNN (machine learning)
  - LeNET (neural networks)
  - VGG-13/VGG-16 (neural networks)
  - Brightness (graphics)
Throughput Analysis

Average normalized throughput across all 16 SIMDGRAM operations

SIMDRAM significantly outperforms all state-of-the-art baselines for a wide range of operations.
Energy Analysis

Average normalized energy efficiency across all 16 SIMDRAM operations

SIMDRAM is more energy-efficient than all state-of-the-art baselines for a wide range of operations
Real-World Applications
Average speedup across 7 real-world applications

SIMDRAM effectively and efficiently accelerates many commonly-used real-world applications
SIMDRAM Key Results

Evaluated on:
- 16 complex in-DRAM operations
- 7 commonly-used real-world applications

SIMDRAM provides:

• 88× and 5.8× the throughput of a CPU and a high-end GPU, respectively, over 16 operations

• 257× and 31× the energy efficiency of a CPU and a high-end GPU, respectively, over 16 operations

• 21× and 2.1× the performance of a CPU and a high-end GPU, over seven real-world applications
SIMDRAM Conclusion

• **SIMDRAM:**
  - Enables efficient computation of a flexible set and wide range of operations in a PuM massively parallel SIMD substrate
  - Provides the hardware, programming, and ISA support, to:
    • Address key system integration challenges
    • Allow programmers to define and employ new operations without hardware changes

**SIMDRAM is a promising PuM framework**
• Can ease the adoption of processing-using-DRAM architectures
• Improves the performance and efficiency of processing-using-memory architectures
Lecture on SIMDRAAM

Step 3: μProgram Execution

- **SIMDRAM control unit**: handles the execution of the μProgram at runtime

- Upon receiving a **bbop instruction**, the control unit:
  1. Loads the μProgram corresponding to SIMDRAAM operation
  2. Issues the sequence of DRAM commands (ACT/PRE) stored in the μProgram to SIMDRAAM subarrays to perform the in-DRAM operation

Processing-in-Memory Course: Lecture 13: Bit-Serial SIMD Processing using DRAM - Spring 2022

512 views • Streamed live on Jun 2, 2022

Onur Mutlu Lectures
25.9K subscribers

[https://youtu.be/pmZoAAhvkJQ](https://youtu.be/pmZoAAhvkJQ)
In-DRAM Physical Unclonable Functions

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu,
  "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices"
  [Lightning Talk Video] [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Full Talk Lecture Video (28 minutes)]

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

Jeremie S. Kim$†$ Minesh Patel$§$ Hasan Hassan$§$ Onur Mutlu$§†$
†Carnegie Mellon University §ETH Zürich

SAFARI
In-DRAM True Random Number Generation

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput" 
  [Slides (pptx) (pdf)] 
  [Full Talk Video (21 minutes)] 
  [Full Talk Lecture Video (27 minutes)] 
  Top Picks Honorable Mention by IEEE Micro.

D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim‡$ Minesh Patel§ Hasan Hassan§ Lois Orosa§ Onur Mutlu‡$
‡Carnegie Mellon University §ETH Zürich

SAFARI
In-DRAM True Random Number Generation

- Ataberk Olgun, Minesh Patel, A. Giray Yaglikci, Haocong Luo, Jeremie S. Kim, F. Nisa Bostanci, Nandita Vijaykumar, Oguz Ergin, and Onur Mutlu,

"QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips"
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]
[Talk Video (25 minutes)]
[SAFARI Live Seminar Video (1 hr 26 mins)]

QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips

Ataberk Olgun$^\dagger$
Minesh Patel$^\S$
A. Giray Yağlıkçı$^\S$
Haocong Luo$^\S$
Jeremie S. Kim$^\S$
F. Nisa Bostancı$^\dagger$
Nandita Vijaykumar$^\S^{\odot}$
Oğuz Ergin$^\dagger$
Onur Mutlu$^\S$

$^\S$ETH Zürich
$^\dagger$TOBB University of Economics and Technology
$^{\odot}$University of Toronto
In-DRAM True Random Number Generation

- F. Nisa Bostanci, Ataberk Olgun, Lois Orosa, A. Giray Yağlıkçı, Jeremie S. Kim, Hasan Hassan, Oğuz Ergin, and Onur Mutlu,

"DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators"
Proceedings of the 28th International Symposium on High-Performance Computer Architecture (HPCA), Virtual, April 2022.
[Slides (pptx) (pdf)]
[Short Talk Slides (pptx) (pdf)]

DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators

F. Nisa Bostancı†§  Ataberk Olgun†§  Lois Orosa§  A. Giray Yağlıkçı§
Jeremie S. Kim§  Hasan Hassan§  Oğuz Ergin†  Onur Mutlu§

†TOBB University of Economics and Technology  §ETH Zürich

In-DRAM Lookup-Table Based Execution

- To appear at MICRO 2022

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira§  Gabriel Falcao†  Juan Gómez-Luna§  Mohammed Alser§
Lois Orosa§  Mohammad Sadrosadati§  Jeremie S. Kim§  Geraldo F. Oliveira§
Taha Shahroodi‡  Anant Nori*  Onur Mutlu§

§ETH Zürich  †IT, University of Coimbra  "Galicia Supercomputing Center  ‡TU Delft  *Intel

In-Flash Bulk Bitwise Execution

■ To appear at MICRO 2022

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park§ V Roknoddin Azizi§ Geraldo F. Oliveira§ Mohammad Sadrosadati§ Rakesh Nadig§ David Novo† Juan Gómez-Luna§ Myungsuk Kim‡ Onur Mutlu§

§ETH Zürich  ∨ POSTECH  †LIRMM, Univ. Montpellier, CNRS  ‡Kyungpook National University

RowClone in Off-the-Shelf DRAM Chips

- Idea: Violate DRAM timing parameters to mimic RowClone

**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

Real Processing Using Memory Prototype

- End-to-end RowClone & TRNG using off-the-shelf DRAM chips
- Idea: Violate DRAM timing parameters to mimic RowClone

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun§†  Juan Gómez Luna§  Konstantinos Kanellopoulos§  Behzad Salami§*  Hasan Hassan§  Oğuz Ergin†  Onur Mutlu§  §ETH Zürich  †TOBB ETÜ  *BSC

https://github.com/cmu-safari/pidram  
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Real Processing Using Memory Prototype

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Real Processing Using Memory Prototype

Building a PiDRAM Prototype

To build PiDRAM’s prototype on Xilinx ZC706 boards, developers need to use the two sub-projects in this directory. The `fpga-zynq` is a repository branched off of UCB-BAR’s fpga-zynq repository. We use fpga-zynq to generate rocket chip designs that support end-to-end DRAM PuM execution. controller-hardware is where we keep the main Vivado project and Verilog sources for PiDRAM’s memory controller and the top level system design.

Rebuilding Steps

1. Navigate into `fpga-zynq` and read the README file to understand the overall workflow of the repository
   - Follow the readme in `fpga-zynq/rocket-chip/riscv-tools` to install dependencies
2. Create the Verilog source of the rocket chip design using the ZynqCopyFPGAConfig
   - Navigate into zc706, then run `make rocket CONFIG=ZynqCopyFPGAConfig -j number of cores`
3. Copy the generated Verilog file (should be under zc706/src) and overwrite the same file in controller-hardware/source/hdl/impl/rocket-chip
4. Open the Vivado project in controller-hardware/Vivado_Project using Vivado 2016.2
5. Generate a bitstream
6. Copy the bitstream (system_top.bit) to `fpga-zynq/zc706`
7. Use the `./build_script.sh` to generate the new `boot.bin` under fpga-images-zc706, you can use this file to program the FPGA using the SD-Card
   - For details, follow the relevant instructions in `fpga-zynq/README.md`

You can run programs compiled with the RISC-V Toolchain supplied within the fpga-zynq repository. To install the toolchain, follow the instructions under fpga-zynq/rocket-chip/riscv-tools.

Generating DDR3 Controller IP sources

We cannot provide the sources for the Xilinx Phv IP we use in PiDRAM’s memory controller due to licensing issues. We describe here how to regenerate them using Vivado 2016.2. First, you need to generate the IP RTL files:

1- Open IP Catalog
2- Find “Memory Interface Generator (MIG 7 Series)” IP and double click

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li\textsuperscript{1,*}, Cong Xu\textsuperscript{2}, Qiaosha Zou\textsuperscript{1,5}, Jishen Zhao\textsuperscript{3}, Yu Lu\textsuperscript{4}, and Yuan Xie\textsuperscript{1}

University of California, Santa Barbara\textsuperscript{1}, Hewlett Packard Labs\textsuperscript{2}
University of California, Santa Cruz\textsuperscript{3}, Qualcomm Inc.\textsuperscript{4}, Huawei Technologies Inc.\textsuperscript{5}
{shuangchenli, yuanxie}@ece.ucsb.edu\textsuperscript{1}
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.
In-Memory Crossbar Array Operations

- Some emerging NVM technologies have crossbar array structure
  - Memristors, resistive RAM, phase change mem, STT-MRAM, ...

- Crossbar arrays can be used to perform dot product operations using “analog computation capability”
  - Can operate on multiple pieces of data using Kirchoff’s laws
    - Bitline current is a sum of products of wordline V x (1 / cell R)
    - Computation is in analog domain inside the crossbar array

- Need peripheral circuitry for D→A and A→D conversion of inputs and outputs
Aside: In-Memory Crossbar Computation

(a) Multiply-Accumulate operation

(b) Vector-Matrix Multiplier

Fig. 1. (a) Using a bitline to perform an analog sum of products operation. (b) A memristor crossbar used as a vector-matrix multiplier.
Aside: In-Memory Crossbar Computation

\[
\begin{pmatrix}
  i_1 & i_2 & i_3 & i_4 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
  o_1 & o_2 & o_3 & o_4 \\
\end{pmatrix}
\]

\[
\frac{1}{R_{11}} \cdot i_1 + \frac{1}{R_{21}} \cdot i_2 + \frac{1}{R_{31}} \cdot i_3 + \frac{1}{R_{41}} \cdot i_4
\]
Readings on Processing using NVM


Challenge: Intelligent Memory Device

Does memory have to be dumb?
Challenge and Opportunity for Future Computing Architectures with Minimal Data Movement
Historical Perspective &
A Detour
on the Review Process
Ambit and RowClone
Sound Great!
No?
Some History: RowClone
RowClone: Historical Perspective

- This work is likely the first example of “minimally changing DRAM chips” to perform data movement and computation
  - Surprising that it was done as late as 2013!

- It led to a body of work on in-DRAM (and in-NVM) computation with “hopefully small” changes

- Work building on RowClone still continues

- Initially, it was dismissed by some reviewers
  - Rejected from ISCA 2013 conference
**Paper Strengths**
The paper includes a well written background on DRAM organization/operation. The proposed technique is simple and elegant; it nicely exploits key circuit-level characteristics of DRAM designs and minimizes the changes necessary to commodity DRAM chips.

**Paper Weaknesses**
I am concerned on the applicability of the technique and found the evaluation to be uncompelling in terms of motivating the work as well as quantifying the potential benefit. Details on how to efficiently manage the coherence between the cache hierarchy and DRAM to enable the proposed technique are glossed over, but in my opinion are critical to the narrative.
Another Review and Rebuttal

**Detailed Comments**

The paper proposes a simple and not new idea, block copy in a DRAM, and the creates a complete

Reviewer B mentions that our idea is "not new". An explicit reference by the reviewer would be helpful here. While the reviewer may be referring to one of the patents that we cite in our paper (citations 2, 6, 25, 26, 27 in the paper), these patents are at a superficial level and do *not* provide a concrete mechanism. In contrast, we propose three concrete mechanisms and provide details on the most important architectural and microarchitectural modifications required at the DRAM chip, the memory controller, and the CPU to enable a system that supports the mechanisms. We also analyze their latency, hardware overhead, power, and performance in detail. We are not aware of any prior work that achieves this.
RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data

You are an author of this paper.

Many programs initialize or copy large amounts of memory data. Initialization and copying are

OveMer Nov WriQua RevConAnd
Review #295A 3 4 5 3
Review #295B 4 3 4 3
Review #295C 3 4 4 3
Yet Later… in ISCA 2015…

Profiling a warehouse-scale computer

Svilen Kanev†
Harvard University

Juan Pablo Darago†
Universidad de Buenos Aires

Kim Hazelwood†
Yahoo Labs

Parthasarathy Ranganathan
Google

Tipp Moseley
Google

Gu-Yeon Wei
Harvard University

David Brooks
Harvard University

Figure 4: 22-27% of WSC cycles are spent in different components of “datacenter tax”.

In this section, we quantify the performance impact of the datacenter tax, and argue that its components are prime candidates for hardware acceleration in future datacenter SoCs.

Data movement In fact, RPCs are by far not the only code portions that do data movement. We also tracked all calls to the memcpy() and memmove() library functions to estimate the amount of time spent on explicit data movement (i.e., exposed through a simple API). This is a conservative estimate because it does not track inlined or explicit copies. Just the variants of these two library functions represent 4-5% of datacenter cycles. Recent work in performing data movement in DRAM [45] could optimize away this piece of tax.
#206 RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Accepted 1947kB Friday 31 May 2013 1:48:46pm PDT | fd8423acdd9a222280302355899340083e5a40b1

You are an author of this paper.

+ Abstract
Bulk data copy and initialization operations are frequently triggered by several system level operations in modern systems. Despite the fact that these operations do not require [more]

+ Authors

+ Topics

<table>
<thead>
<tr>
<th>OveMer</th>
<th>Nov</th>
<th>WriQua</th>
<th>RevExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review #206A</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Review #206B</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Review #206C</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Review #206D</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Review #206E</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
More History: Ambit
Ambit

- **First work on performing bulk bitwise operations in DRAM**
  - By exploiting analog computation capability of bitlines
  - Extends and completes our IEEE CAL 2015 paper

- **Disruptive** -- spans algorithms to circuits/devices
  - Requires hardware/software cooperation for adoption

- Led to a large amount of work in similar approaches in DRAM and NVM
  - The work continues to build

- Initially, it was dismissed by many reviewers
  - Rejected from 4 conferences!
Buddy RAM: Fast and Efficient Bulk Bitwise Operations Using DRAM

You are an author of this paper.

Abstract
Many data structures (e.g., database bitmap indices) rely on fast bitwise operations on large bit vectors to achieve high performance. Unfortunately, the throughput of such bulk [more]

Authors
V. Seshadri, D. Lee, T. Mullins, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

Topics and Options

<table>
<thead>
<tr>
<th>Review #171A</th>
<th>OveMerPos</th>
<th>RevConAnd</th>
<th>Nov</th>
<th>WriQua</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Submission (1662kB)  10 Apr 2016 9:32:31pm EDT  
e518c6a8916109492574858db80a6184fe61ca0c

Abstract
Certain widely-used data structures (e.g., bitmap indices) rely on ...

Authors
Vivek Seshadri (CMU)  
<vseshadr@cs.cmu.edu>
Donghyuk Lee (NVIDIA Research)  
<donghyuk1@cmu.edu>
Thomas Mullins (Intel)  
<thomas.p.mullins@int>
Amirali Boroumand (CMU)  
Jeremie Kim (CMU)  
Michael A. Kozuch (Intel)  
<michael.a.kozuch@in>
Onur Mutlu (CMU/ETH)  
<omutlu@gmail.com>
Phillip B. Gibbons (CMU)  
<gibbons@cs.cmu.edu>
Todd C. Mowry (CMU)  <tc>

Topics

Rejected  You are an author of this paper.
Review #249A  2  2  4  3  3
Review #249B  4  4  3  3  5
Review #249C  2  3  4  2  3
Review #249D  5  5  2  3  3
Review #249E  5  5  2  2  3
Review #249F  3  3  3  3  4
1) significantly improves the performance of queries in applications that use bitmap indices for fast analytics, and 2) makes bit vectors more attractive than red-black trees to represent sets. We believe Buddy can trigger programmers to redesign applications to use bitwise operations with the goal of achieving high performance and efficiency.

Rejected · You are an author of this paper.

Review #119A 1 2 3 2 2
Review #119B 4 1 4 4 3
Review #119C 4 4 4 4 4
Review #119D 3 1 4 4 3
Review #119E 3 2 5 4 4

1 Comment: Response (V. Seshadri)
Rejected

Submission  🕒 19 Nov 2016 12:03:02am EST   
舐 3eea263e35e53552851cabc5225162776f809eaa

▷ Abstract
Bitwise operations are an important component of modern processors. [more]

▷ Authors
V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

▷ Topics and Options

<table>
<thead>
<tr>
<th>PosRebOve</th>
<th>OveMer</th>
<th>Nov</th>
<th>WriQua</th>
<th>RevExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review #162A</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Review #162B</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Review #162C</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Review #162D</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Review #162E</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Ambit Sounds Good, No?

**Review from ISCA 2016**

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.
Very Interesting and Novel, ….. BUT …

Comments for the authors
I found this idea very interesting and novel. In particular, while there have been many works proposing moving computation closer to memory, I'm not aware of any work which proposes to leverage the DRAM rows themselves to implement the computation. The benefits to this approach are large in that no actual logic is used to implement the logical functions. Further the operation occurs in parallel across the whole row, a huge degree of data parallelism.
... This Will Never Get Implemented

- The biggest problem with the work is that it underestimates the difficulty in modifying DRAM process for benefit in only a subset of applications which do bulk bitwise operations. In particular, I find it hard to believe that the commodity DRAM industry will incorporate this into their standard DRAM process. DRAM process is, at this point, a highly optimized, extremely tuned endeavor. Adding this kind of functionally will have a big impact on DRAM cost. The performance benefit on the subset of applications isn't enough to justify the higher costs this will incur and this will never get implemented.
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.
Comments for the authors
This paper shows that DRAM could be modified to support bitwise operations directly within the DRAM itself. The performance advantages are compelling for situations in which bulk bitwise operations matter.

However, I am not really convinced that any DRAM manufacturer would really consider modifying the DRAM in this way. It benefits one specific type of operation, and while that is important for some applications, it is not really a general-purpose operation. It is not like the STL library would be changed to use this for its implementation of sets.
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.

This paper suits better to be peer-reviewed and published in a circuit conference or with a fabricated chip in ISSCC.
A Review from HPCA 2017: REJECT

Review #119A

Paper summary
Paper proposes DRAM technology changes (inverts, etc) to implement bit-wise operations directly on DRAM rows.

Overall merit
1. Reject

Reviewer expertise
2. I have passing familiarity with this area

Experimental methodology
2. Poor

Post-response overall merit
Unknown

Writing quality
3. Adequate

Novelty
2. Incremental improvement

Strengths
Seems like a new idea. Processor-in-Memory (PIM) ideas have resurfaced.

Weaknesses

Comments for author
I am skeptical this would benefit real-world computations. I've never seen real-world program profiles with hot functions or instructions that are bit-wise operations.

On the other hand, I have seen system profiles that show non-trivial time zeroing pages. Suggest re-tooling your work to support page zeroing and evaluating that with a full-system simulation. Take a look at when/why the Linux kernel zeroes pages. You might be surprised at the possible impact.

* Impractical. Too many implications on ISA, DRAM design, and coherence protocols.
* Unlikely to benefit real-world computations.
* Evaluation did not consider full-program performance.
A Review from ISCA 2017

Review #162A  Updated 28 Jan 2017 5:16:50am EST

Post rebuttal overall merit
1. Reject
2. Weak reject

Novelty
2. Incremental improvement

Reviewer expertise
5. This is my area

Overall merit
Writing quality
4. Well-written

Paper summary
This paper proposes in-DRAM bit-wise operations by activating more than one word lines (and cells connected to the wordiness). Basically, it's a charge-based computation where the difference in charge stored cells connected to the same bit line is used for the logic operation.

Strengths
- conceptually a very interesting proposal (but practically not sure).
- consider various aspects including the interaction between processors and RAM (although there isn't any new contribution and rather use the same proposal as prior work).

Weaknesses
- negative impact on the regularity of DRAM array design (and associated overhead evaluation seems to be very weak.
- significantly increase the testing cost

Comments to authors
This is an interesting proposal and well presented paper. However, I have some concerns regarding the evaluation (especially related to circuit level issues).

Especially, I feel that the variation related modeling and evaluation are weak as there are multiple sources of variations such as access transistors and sense-amp mismatches, minor defects in either access transistors and/or capacitor that can manifest in this particular proposed operation scenarios. That is, the authors oversimplify the variation modeling, which I believe failed to convince me this will work in practice. Also, the area overhead analysis sounds hand-waivy. I totally understand the difficulty of DRAM overhead analysis but also we must pursue more precise ways of evaluating the area impact as DRAM is very cost-sensitive.
Another Review from ISCA 2017

Review #162B  Updated 1 Feb 2017 6:50:31pm EST

Post rebuttal overall merit
2. Weak reject

Overall merit
2. Weak reject

Novelty
Writing quality
3. New contribution
3. Adequate

Reviewer expertise
3. I know the material, but am not an expert

Paper summary
This paper proposes performing bulk bit-wise operations at DRAM. They leverage analog operation of DRAM, and add some extra circuits to do bit-wise operations at row granularity.

Weaknesses
Not motivated well.
Not convinced the possible gains worth all the complexity.
Not convinced if the proposal is applicable in real world applications that do bit-wise operations on different data granularity.

Comments to authors
* The paper lacks motivation. The authors talk about how common bit-wise operations are. However, they do not provide any stat on how often these operations are being used, and more importantly, on what data granularity.

* Although bit-wise operations are common in some applications, they are not necessarily done at large granularity. For example, many applications do bit-wise operations at small 64-byte (or even smaller) entities. For such cases, this paper requires copying two whole rows to some temporary rows, and doing the operation on those rows. Please explain how you handle such cases, and what the benefits would be.

* What happens if the user does bit-wise operation on two 8-byte data, and want to store it in a third block?

* What happens if both operands are located in one row?

* The main issue with this work is that it requires flushing blocks out of caches to do the bit-wise operations. Imagine you have blocks A and B in the cache, as discussed in section 6.2.3., the proposal would flush them out of caches (not sure how?), writes
ISCA 2017 Summary

This paper was discussed both online and at the PC meeting. Reviewers were uniformly positive about the novelty of the proposed Buddy-RAM design. However, reviewers were also concerned about the feasibility of the design. During the post-rebuttal and PC discussion, the main concerns raised were (1) the impact of process variation on the design's functional correctness;

(2) the potential reliability issues that arise due to the lack of ECC/CRC mechanisms; and (3) the impact on DRAM testing cost.

Specifically on point (1), some reviewers raised concerns about the limitations of the simulations performed to address variability: "Monte-Carlo cannot capture tail distribution of cell failures. Also Monte-Carlo cannot capture random correlated WID process variation issues (only some random uncorrelated variations)."

Given these concerns, the PC ultimately decided to reject the paper. We hope that this feedback is useful in preparing a future version of the paper.
The Reviewer Accountability Problem

Acknowledgments

We thank the reviewers of ISCA 2016/2017, MICRO 2016/2017, and HPCA 2017 for their valuable comments. We
Accepted

**Submission (1837kB)** 4 Apr 2017 11:33:57pm EDT
7420f9f02c549bcca0dc6216a5e9887dfe0d422

**Revision (1852kB)** 14 Jun 2017 4:16am EDT
22f0d123a22cf04960928e0ac43d972b5a33a848

**Abstract**
Many important applications trigger bitwise operations on large bit vectors (bulk bitwise operations). In fact, recent

**Authors**
V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. Kozuch, O. Mutlu, P. Gibbons, T. Mowry [details]

<table>
<thead>
<tr>
<th>Review</th>
<th>PosResOve</th>
<th>OveMer</th>
<th>RevExp</th>
<th>Nov</th>
<th>PotImp</th>
<th>WriQua</th>
<th>ImpRev</th>
</tr>
</thead>
<tbody>
<tr>
<td>#347A</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>#347B</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>#347C</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>#347D</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>#347E</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>#347F</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**1 Comment:** Rebuttal Response (V. Seshadri)
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 it presented 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.
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 or writing papers
- Be constructive, not destructive
- Enable heterogeneity, but do not have double standards...

Do not block or delay scientific progress for non-reasons
Suggestion to Community

We Need to Fix the Reviewer Accountability Problem
Takeaway

Main Memory Needs

Intelligent Controllers
Research Community Needs Accountable Reviewers
An Interview on Research and Education

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

SAFARI
https://www.youtube.com/onurmutlulectures
RowClone in Off-the-Shelf DRAM Chips

- Idea: Violate DRAM timing parameters to mimic RowClone

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

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
- $R_1 = 01_2$
  
  - 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_1600</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SKhynix_4G_1600</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SKhynix_4G_1600</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SKhynix_4G_1600</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Samsung_4G_1600</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Samsung_4G_1600</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.
Real Processing Using Memory Prototype

- End-to-end RowClone & TRNG using off-the-shelf DRAM chips
- Idea: Violate DRAM timing parameters to mimic RowClone

PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun$^\dagger$, Juan Gómez Luna$, Hasan Hassan$, Konstantinos Kanellopoulos$, Behzad Salami$^\ast$

$^\dagger$ETH Zürich, $^\ast$TOBB ETÜ, $^\ast$BSC

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
PiDRAM

**Goal:** Develop a **flexible** platform to explore end-to-end implementations of PuM techniques  
- Enable rapid integration via key components

### Hardware
1. Easy-to-extend Memory Controller
2. ISA-transparent PuM Controller

### Software
1. Extensible Software Library
2. Custom Supervisor Software
Real Processing Using Memory Prototype

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=queukNs5XI3g&t=4192s
1- User application interfaces with the OS via system calls
2- OS uses PuM Operations Library (pumolib) to convey operation related information to the hardware using
3- STORE instructions that target the memory mapped registers of the PuM Operations Controller (POC)
4- POC oversees the execution of a PuM operation (e.g., RowClone, bulk bitwise operations)
5- Scheduler arbitrates between regular (load, store) and PuM operations and issues DRAM commands with custom timings
Building a PiDRAM Prototype

To build PiDRAM's prototype on Xilinx ZC706 boards, developers need to use the two sub-projects in this directory. `fpga-zynq` is a repository branched off of UCB-BAR's `fpga-zynq` repository. We use `fpga-zynq` to generate rocket chip designs that support end-to-end DRAM PuM execution. `controller-hardware` is where we keep the main Vivado project and Verilog sources for PiDRAM's memory controller and the top level system design.

**Rebuilding Steps**

1. Navigate into `fpga-zynq` and read the README file to understand the overall workflow of the repository
   - Follow the readme in `fpga-zynq/rocket-chip/riscv-tools` to install dependencies
2. Create the Verilog source of the rocket chip design using `ZynqCopyFGACfg`
   - Navigate into zc706, then run `make rocket CONFIG=ZynqCopyFGACfg -j number of cores`
3. Copy the generated Verilog file (should be under zc706/src) and overwrite the same file in `controller-hardware/source/hdl/impl/rocket-chip`
4. Open the Vivado project in `controller-hardware/Vivado_Project` using Vivado 2016.2
5. Generate a bitstream
6. Copy the bitstream (`system_top.bit`) to `fpga-zynq/zc706`
7. Use the `./build_script.sh` to generate the new `boot.bin` under `fpga-images-zc706`, you can use this file to program the FPGA using the SD-Card
   - For details, follow the relevant instructions in `fpga-zynq/README.md`

You can run programs compiled with the RISC-V Toolchain supplied within the `fpga-zynq` repository. To install the toolchain, follow the instructions under `fpga-zynq/rocket-chip/riscv-tools`.

**Generating DDR3 Controller IP sources**

We cannot provide the sources for the Xilinx PHY IP we use in PiDRAM's memory controller due to licensing issues. We describe here how to regenerate them using Vivado 2016.2. First, you need to generate the IP RTL files:

1. Open IP Catalog
2. Find "Memory Interface Generator (MIG 7 Series)" IP and double click


https://github.com/cmu-safari/pidram

https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
Microbenchmark Copy/Initialization Throughput

In-DRAM Copy and Initialization improve throughput by 119x and 89x
PiDRAM is the first flexible end-to-end framework that enables system integration studies and evaluation of real Processing-using-Memory (PuM) techniques. PiDRAM, at a high level, comprises a RISC-V system and a custom memory controller that can perform PuM operations in real DDR3 chips. This repository contains all sources required to build PiDRAM and develop its prototype on the Xilinx ZC706 FPGA boards.

https://github.com/CMU-SAFAI/PiDRAM
PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM

Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Öğuz Ergin, Onur Mutlu

Processing-in-memory (PiM) techniques leverage the analog operation of memory cells to perform computation. Several recent works have demonstrated PiM techniques in off-the-shelf DRAM devices. Since DRAM is the dominant memory technology as main memory in current computing systems, these PiM techniques represent an opportunity for alleviating the data movement bottleneck at very low cost. However, system integration of PiM techniques imposes non-trivial challenges that are yet to be solved. Design space exploration of potential solutions to the PiM integration challenges requires appropriate tools to develop necessary hardware and software components. Unfortunately, current specialized DRAM-testing platforms, or system simulators do not provide the flexibility and/or the holistic system view that is necessary to deal with PiM integration challenges.

We design and develop PiDRAM, the first flexible end-to-end framework that enables system integration studies and evaluation of real PiM techniques. PiDRAM provides software and hardware components to rapidly integrate PiM techniques across the whole system software and hardware stack (e.g., necessary modifications in the operating system, memory controller). We implement PiDRAM on an FPGA-based platform along with an open-source RISC-V system. Using PiDRAM, we implement and evaluate two state-of-the-art PiM techniques: in-DRAM (i) copy and initialization, (ii) true random number generation. Our results show that the in-memory copy and initialization techniques can improve the performance of bulk copy operations by 12.6x and bulk initialization operations by 14.6x on a real system. Implementing the true random number generator requires only 190 lines of Verilog and 74 lines of C code using PiDRAM’s software and hardware components.
Long Talk + Tutorial on Youtube

https://youtu.be/s_z_S6FYpC8
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li\textsuperscript{1,*}, Cong Xu\textsuperscript{2}, Qiaosha Zou\textsuperscript{1,5}, Jishen Zhao\textsuperscript{3}, Yu Lu\textsuperscript{4}, and Yuan Xie\textsuperscript{1}

University of California, Santa Barbara\textsuperscript{1}, Hewlett Packard Labs\textsuperscript{2}
University of California, Santa Cruz\textsuperscript{3}, Qualcomm Inc.\textsuperscript{4}, Huawei Technologies Inc.\textsuperscript{5}
{shuangchenli, yuanxie}@ece.ucsb.edu\textsuperscript{1}
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.
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
Principle: Work Hard

Work Hard to Enable Your Passion
You can make a good impact on the world.
Recommended Interview on Research & 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)

- **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"]
Recommended Interview

Recommended Interview

https://www.youtube.com/watch?v=8ffSEKZhmvo
A Talk on Impactful Research & Education

Applying to Grad School & Doing Impactful Research

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
13 June 2020
Undergraduate Architecture Mentoring Workshop @ ISCA 2021

SAFARI
ETH Zürich
Carnegie Mellon

Arch. Mentoring Workshop @ISCA'21 - Applying to Grad School & Doing Impactful Research - Onur Mutlu
1,563 views • Premiered Jun 16, 2021

Onur Mutlu Lectures
17.2K subscribers

Panel talk at Undergraduate Architecture Mentoring Workshop at ISCA 2021
(https://sites.google.com/wisc.edu/uar...)

https://www.youtube.com/watch?v=83tlorht7Mc&list=PL5Q2soXY2Zi8D_5MGV6EnXEJHnlV2YFBJl&index=54
Richard Hamming

``You and Your Research''

Transcription of the
Bell Communications Research Colloquium Seminar
7 March 1986

If you really want to be a first-class scientist you need to know yourself, your weaknesses, your strengths, and your bad faults, like my egotism. How can you convert a fault to an asset? How can you convert a situation where you haven't got enough manpower to move into a direction when that's exactly what you need to do? I say again that I have seen, as I studied the history, the successful scientist changed the viewpoint and what was a defect became an asset.

In summary, I claim that some of the reasons why so many people who have greatness within their grasp don't succeed are: they don't work on important problems, they don't become emotionally involved, they don't try and change what is difficult to some other situation which is easily done but is still important, and they keep giving themselves alibis why they don't. They keep saying that it is a matter of luck. I've told you how easy it is; furthermore I've told you how to reform. Therefore, go forth and become great scientists!
Computer Architecture
Lecture 3: Processing using Memory

Prof. Onur Mutlu
ETH Zürich
Fall 2022
6 October 2022