Tutorial on Memory-Centric Computing: Processing-Using-Memory

Geraldo F. Oliveira
Prof. Onur Mutlu

ISCA 2024
29 June 2024
Agenda

- Introduction to Memory-Centric Computing Systems
- Invited Talk by Prof. Minsoo Rhu:
  “Memory-Centric Computing Systems – For AI and Beyond”
- Coffee Break
- Real-World Processing-Near-Memory Systems
- Invited Talk by Prof. Saugata Ghose:
  “RACER and ReRAM PUM”
- PIM Programming & Infrastructure for PIM Research
- Closing Remarks
Processing in Memory: Two Approaches

1. Processing near Memory
2. Processing using Memory
Starting Simple: Data Copy and Initialization

\textit{memmove} \& \textit{memcpy}: 5\% cycles in Google’s datacenter [Kanev+ ISCA’15]

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

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

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

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

1046ns, 3.6uJ $\rightarrow$ 90ns, 0.04uJ
Brief Review:
Inside A DRAM Chip
Inside a DRAM Chip

- Subarray (2D Array of DRAM Cells)
- Sense Amplifiers
- Row Buffer
- DRAM Bank
- DRAM Chips
- DRAM Module
- Bitline
- Wordline
- Access Transistor
- Storage Capacitor

SAFARI
Inside a DRAM Chip: Another View

Chip I/O

Bank

Memory Channel

Subarray

Bank I/O

Row of DRAM Cells

Row Buffer
DRAM Cell Operation

1. ACTIVATE (ACT)

2. READ/ WRITE

3. PRECHARGE (PRE)
DRAM Cell Operation (1/3)

1. ACTIVATE (ACT)
   - raise wordline

2. READ/WRITE
   - capacitor charge transfers to bitline

3. PRECHARGE (PRE)
   - amplify deviation in the bitline
   - enable sense amplifier

Diagram highlights:
- Wordline
- Storage capacitor
- Access transistor
- Bitline
- Sense amplifier
- Half of V_DD + delta
DRAM Cell Operation (2/3)

1. ACTIVATE (ACT)

2. READ/WRITE

3. PRECHARGE (PRE)

read/write charge latched in sense amplifier
DRAM Cell Operation (3/3)

1. lower wordline

2. precharge bitline for next access

3. disable sense amplifier

SAFARI
Future Systems: In-Memory Copy

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

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

Idea: Two consecutive ACTivates

Negligible HW cost

Step 1: Activate row A

Transfer row

Step 2: Activate row B

DRAM subarray

Row Buffer (4 Kbytes)

4 Kbytes

8 bits

Data Bus
RowClone: Intra-Subarray

Data gets copied

V_{DD}/2 \delta

0

src

V_{DD}/2 + \delta

0

dst

S sense Amplifier (Row Buffer)

Amplify the difference

V_{DD}/2

0
RowClone: Latency and Energy Savings

More on RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,

"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"

Proceedings of the 46th International Symposium on Microarchitecture (MICRO), Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
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
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 Mansourí Ghiasi, and Onur Mutlu,

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

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"

(Truly) In-Memory Computation

- We can support in-DRAM AND, OR, NOT, MAJ
- At low cost
- Using analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement

- New memory technologies enable even more opportunities
  - Memristors, resistive RAM, phase change mem, STT-MRAM, ...
  - Can operate on data with minimal movement
In-DRAM AND/OR: Triple Row Activation

Final State
\[ AB + BC + AC \]

\[ C(A + B) + \sim C(AB) \]

Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM", IEEE CAL 2015.
In-DRAM Bulk Bitwise AND/OR Operation

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

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

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

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

"Fast Bulk Bitwise AND and OR in DRAM"


Fast Bulk Bitwise AND and OR in DRAM

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

*Carnegie Mellon University †Intel Pittsburgh
In-DRAM NOT: Dual Contact Cell

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

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

In-DRAM NOT Operation

Performance: In-DRAM Bitwise Operations

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

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

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

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

Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving
Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
"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.
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
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 Mansouri Ghiasi$^1$  

*Geraldo F. Oliveira$^1$  
Minesh Patel$^1$  
Juan Gómez-Luna$^1$  

Sven Gregorio$^1$  
Mohammed Alser$^1$  
Onur Mutlu$^1$  
João Dinis Ferreira$^1$  
Saugata Ghose$^3$

$^1$ETH Zürich  
$^2$Simon Fraser University  
$^3$University of Illinois at Urbana–Champaign
SIMDRAM Framework: Overview

**User Input**

Desired operation

AND/OR/NOT logic

**Step 1: Generate MAJ logic**

MAJ

MAJ/NOT logic

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

<table>
<thead>
<tr>
<th>ACT/PRE</th>
<th>ACT/PRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT/PRE</td>
<td>ACT/ACT/PRE</td>
</tr>
<tr>
<td>done</td>
<td></td>
</tr>
</tbody>
</table>

**SIMDRAM Output**

New SIMDRAM µProgram

µProgram

Main memory

ISA

bbop_new

New SIMDRAM instruction

**User Input**

SIMDRAM-enabled application

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

**Step 3: Execution according to µProgram**

Control Unit

µProgram

Memory Controller

Instruction result in memory

ACT/PRE

ACT/PRE

ACT/PRE

ACT/PRE/PRE

done
**SIMDRAM Framework: Step 1**

**User Input**
- Desired operation
  - AND/OR/NOT logic

**Step 1: Generate MAJ logic**
- MAJ
- 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
- µProgram
- µProgram
- done

**Main memory**
- ISA

**User Input**
- SIMDRAM-enabled application
  ```
  foo () {
    bbop_new
  }
  ```

**Step 3: Execution according to µProgram**
- Control Unit

**SIMDRAM Output**
- Instruction result in memory
  ```
  ACT/PRE
  ACT/PRE
  ACT/PRE
  ACT/PRE/PRE
  done
  ```

**SAFARI**
Step 1: Naïve MAJ/NOT Implementation

output is “1” only when \( A = B = \text{“1”} \)

output is “0” only when \( A = B = \text{“0”} \)

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

Greedy optimization algorithm

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**

New SIMDRAM μProgram

<table>
<thead>
<tr>
<th>μProgram</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbop_new</td>
</tr>
</tbody>
</table>

Main memory

**SIMDRAM Framework:**

**Step 1:** Generate MAJ logic

**MAJ**

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

<table>
<thead>
<tr>
<th>ACT/PRE</th>
<th>ACT/PRE</th>
<th>ACT/PRE</th>
<th>ACT/ACT/PRE</th>
<th>done</th>
</tr>
</thead>
</table>

**Step 3:** Execution according to μProgram

**Control Unit**

**μProgram**

- bbop_new

**Memory Controller**

Instruction result in memory

- ACT/PRE

**User Input**

SIMDRAM-enabled application

```c
foo () {
    bbop_new
}
```
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
SIMDRAM Framework: Step 3

**Step 1: Generate MAJ logic**

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

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

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

**Step 3: Execution according to μProgram**

- SIMDRAM-enabled application
- foo () {
  bbop_new
}

- Control Unit
- μProgram

**SIMDRAM Output**

- New SIMDRAM μProgram
- μProgram
- Main memory
- ISA
- bbop_new

- Memory Controller

**Instruction result in memory**

- ACT/PRE
- ACT/PRE
- ACT/PRE
- ACT/ACT/PRE
- done
Step 3: \( \mu \)Program Execution

- **SIMDRAM control unit**: handles the execution of the \( \mu \)Program at runtime

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

```c
foo () {
  bbop_new
}
```
More in the Paper


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

*Nastaran Hajinazar1, 2 *Geraldo F. Oliveira1  
Nika Mansouri Ghiasi1  Minesh Patel1  Sven Gregorio1  João Dinis Ferreira1  
Juan Gómez-Luna1  Mohammed Alser1  Onur Mutlu1  Saugata Ghose3

1ETH Zürich  2Simon Fraser University  3University 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

SAFARI
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

SAFARI
More on SIMDRAM

Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu, "SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"

[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\textsuperscript{1,2}  Geraldo F. Oliveira\textsuperscript{1}  Sven Gregorio\textsuperscript{1}  João Dinis Ferreira\textsuperscript{1}
Nika Mansouri Ghiasi\textsuperscript{1}  Minesh Patel\textsuperscript{1}  Mohammed Alser\textsuperscript{1}  Saugata Ghose\textsuperscript{3}
Juan Gómez-Luna\textsuperscript{1}  Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \quad \textsuperscript{2}Simon Fraser University \quad \textsuperscript{3}University of Illinois at Urbana–Champaign
SIMDRAM: Follow-Ups

- Limitations of current substrate?
  - Computing granularity
  - Data layout conversion
  - High-latency bit-serial operations
  - Assembly-like programming model
  - Application scope
  - ...

- We are working on even better processing-using-memory substrates
  - One step at a time!
Limitations of PUD Systems: Overview

PUD systems suffer from **three sources of inefficiency** due to the **large and rigid** DRAM access granularity

<table>
<thead>
<tr>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
</table>
| **1** | **SIMD Underutilization**  
  - due to data parallelism variation within and across applications  
  - leads to throughput and energy waste |
| **2** | **Limited Computation Support**  
  - due to a lack of low-cost interconnects across columns  
  - limits PUD operations to only parallel map constructs |
| **3** | **Challenging Programming Model**  
  - due to a lack of compiler support for PUD systems  
  - creates a burden on programmers, limiting PUD adoption |
Problem & Goal

**Problem**

Processing-Using-DRAM’s large and rigid granularity limits its applicability and efficiency for different applications.

**Goal**

Design a flexible PUD system that overcomes the three limitations caused by large and rigid DRAM access granularity.
MIMDRAM: Key Idea (I)

DRAM’s hierarchical organization can enable fine-grained access

Key Issue:
On a DRAM access, the global wordline propagates across all DRAM mats

Fine-Grained DRAM:
Segments the global wordline to access individual DRAM mats
MIMDRAM:
Key Idea (II)

Fine-Grained DRAM:
segments the global wordline to access individual DRAM mats

Fine-grained DRAM for energy-efficient DRAM access:

[Cooper-Balis+, 2010]: Fine-Grained Activation for Power Reduction in DRAM
[Udipi+, 2010]: Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores
[Zhang+, 2014]: Half-DRAM
[Ha+, 2016]: Improving Energy Efficiency of DRAM by Exploiting Half Page Row Access
[O’Connor+, 2017]: Fine-Grained DRAM
[Olgun+, 2024]: Sectored DRAM
**MIMDRAM: Key Idea (III)**

**Fine-grained DRAM for processing-using-DRAM:**

1. Improves SIMD utilization
   - for a single PUD operation, only access the DRAM mats with target data
Fine-grained DRAM for processing-using-DRAM:

1. Improves SIMD utilization
   - for a single PUD operation, only access the DRAM mats with target data
   - for multiple PUD operations, execute independent operations concurrently
     → multiple instruction, multiple data (MIMD) execution model
MIMDRAM: Key Idea (III)

1. **Improves SIMD utilization**
   - For a single PUD operation, only access the DRAM mats with target data.
   - For multiple PUD operations, execute independent operations concurrently.
   → Multiple instruction, multiple data (MIMD) execution model.

2. **Enables low-cost interconnects for vector reduction**
   - Global and local data buses can be used for inter-/intra-mat communication.

Fine-grained DRAM for processing-using-DRAM:
Fine-grained DRAM for processing-using-DRAM:

1. Improves SIMD utilization
   - for a single PUD operation, only access the DRAM mats with target data
   - for multiple PUD operations, execute independent operations concurrently
     → multiple instruction, multiple data (MIMD) execution model

2. Enables low-cost interconnects for vector reduction
   - global and local data buses can be used for inter-/intra-mat communication

3. Eases programmability
   - SIMD parallelism in a DRAM mat is on par with vector ISAs’ SIMD width
MIMDRAM is a hardware/software co-designed PUD system that enables fine-grained PUD computation at low cost and programming effort.

Main components of MIMDRAM:

1. **Hardware**
   - DRAM array modification to enable fine-grained PUD computation
   - inter- and intra-mat interconnects to enable PUD vector reduction
   - control unit design to orchestrate PUD execution

2. **Software**
   - compiler support to transparently generate PUD instructions
   - system support to map and execute PUD instructions
MIMDRAM: Modifications to DRAM Chip

- row decoder
- mat selector
- mat range
- global wordline
- isolation transistors
- row decoder latch
- helper FFs
- global sense amplifier
- inter-mat interconnect
- IO interface
The control unit schedules and orchestrates the execution of multiple PUD operations transparently.
MIMDRAM: Compiler Support

**Goal**
Transparently: extract SIMD parallelism from an application, and schedule PUD instructions while maximizing utilization

**Three new LLVM-based passes targeting PUD execution**

---

**source code**

```c
for(i; i<1024;i++)
{
    C[i]=A[i]+B[i];
    F[i]=D[i]*E[i];
    G[i]=C[i]-F[i];
}
for(){
```

---

**loop auto-vectorization**

```c
%1=load<1024 x i32*> %A ... %3=add<1024 x i32> %1,%2 store %3,<1024 x i32*> %C ... %6=mul<1024 x i32> %4,%5 ... %7=sub<1024 x i32> %3,%6 ...
```

---

**code scheduling & data mapping**

**source code**

```c
for(i; i<1024;i++)
{
    C[i]=A[i]+B[i];
    F[i]=D[i]*E[i];
    G[i]=C[i]-F[i];
}
for(){
```

---

**code generation**

```c
*A=pim_malloc(s,mati) *D=pim_malloc(s,matj) *t=pim_malloc(s,mati) ... bbop_add(C,A,B,mati) bbop_mul(F,D,E,matj) bbop_mov(t,F) bbop_sub(G,C,t,mati)
```
MIMDRAM significantly improves energy efficiency compared to CPU (30.6x), GPU (6.8x), and SIMDREAM (14.3x)
More on MIMDRAM

- Geraldo F. Oliveira, Ataberk Olgun, Abdullah Giray Yağlıkçı, F. Nisa Bostancı, Juan Gómez-Luna, Saugata Ghose, and Onur Mutlu

"MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing"


MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Processing

Geraldo F. Oliveira† Ataberk Olgun† Abdullah Giray Yağlıkçı† F. Nisa Bostancı†
Juan Gómez-Luna† Saugata Ghose‡ Onur Mutlu†

† ETH Zürich ‡ Univ. of Illinois Urbana-Champaign

SAFARI

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
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
In-DRAM True Random Number Generation

- Ataberk Olgun, Minesh Patel, A. Giray Yağlıkçı, Haocong Luo, Jeremie S. Kim, F. Nisa Bostancı, Nandita Vijaykumar, Oğuz 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$\S\circ$
Oğuz Ergin$\dagger$
Onur Mutlu$\S$

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

- F. Nisa Bostancı, 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‡§
Jeremy S. Kim§

Ataberk Olgun‡§
Hasan Hassan§

Lois Orosa§
Oğuz Ergin†

A. Giray Yağlıkçı§
Onur Mutlu§

‡TOBB University of Economics and Technology
§ETH Zürich

In-DRAM Lookup-Table Based Execution


[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (26 minutes)]
[arXiv version]
[Source Code (Officially Artifact Evaluated with All Badges)]

Officially artifact evaluated as available, reusable and reproducible.

pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira§
Lois Orosa$\dagger$
Mohammad Sadrosadati§
Taha Shahroodi‡
$\dagger$ETH Zürich

Gabriel Falcao†
Juan Gómez-Luna§
Jeremie S. Kim§
Galicia Supercomputing Center

Mohammed Alser§
Geraldo F. Oliveira§
Anant Nori*
Onur Mutlu§
†TU Delft

$ETH$ Zürich
$\dagger$ IT, University of Coimbra
-$\dagger$TU Delft

SAFARI

<table>
<thead>
<tr>
<th>Category</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Movement</td>
<td>RowClone, Seshadri+ 2013</td>
</tr>
<tr>
<td></td>
<td>LISA, Chang+ 2013</td>
</tr>
<tr>
<td>Bitwise Operations</td>
<td>Ambit, Seshadri+ 2017</td>
</tr>
<tr>
<td>Bit Shifting</td>
<td>DRISA, Li+ 2017</td>
</tr>
<tr>
<td>Arithmetic Operations</td>
<td>SIMDGRAM, Hajinazar &amp; Oliveira+ 2021</td>
</tr>
</tbody>
</table>

Existing Processing using-DRAM architectures only support a **limited range** of operations.
The Goal of pLUTo

Extend Processing-using-DRAM to support the execution of arbitrarily complex operations
pLUTo: Key Idea

\[ f(x) = \text{ALU}(x) \]

input

output
pLUTTo: Key Idea

\[ f(x) \]

\[ X \rightarrow \text{ALU} \rightarrow \text{LUT} \rightarrow ? \]

\[ X \rightarrow \text{input} \]

\[ f(x) \rightarrow \text{output} \]
pLUTo: Key Idea

Replace computation with memory accesses
→ *pLUTo LUT Query* operation
System Integration

C-Like Code with
\texttt{pLUTo API} calls

\texttt{pLUTo Compiler}

Assembly Code with
\texttt{pLUTo ISA Extensions}

\texttt{pLUTo Controller}

Execution in the DRAM Substrate

\texttt{api\_pluto\_mul}

\texttt{pluto\_subarray\_alloc}
\texttt{pluto\_bit\_shift\_l}
\texttt{pluto\_or}
\texttt{pluto\_op}

\texttt{ACT}
\texttt{PRE}
\texttt{ACT}
\texttt{ACT}
\texttt{PRE}
\texttt{...}
Performance (normalized to area)

Average speedup normalized to area across 7 real-world workloads

pLUTo provides substantially higher performance per unit area than both the CPU and the GPU.
Energy Consumption

Average energy consumption across 7 real-world workloads

pLUTo significantly reduces energy consumption compared to processor-centric architectures for various workloads.
More Results in the Paper

- Comparison with FPGA
- Area Overhead Analysis
- Circuit-Level Reliability & Correctness
- Subarray-Level Parallelism
- LUT Loading Overhead
- Range of Supported Operations

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

Geraldo F. Oliveira

- pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

[YouTube Video](https://youtu.be/9t1FJQ6nNw4?si=bhylWCLZde2DC7os)
Bulk Bitwise Operations in Real DRAM Chips


Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis

İsmail Emir Yüksel  Yahya Can Tuğrul  Ataberk Olgun  F. Nisa Bostancı  A. Giray Yağlıkçı
Geraldo F. Oliveira  Haocong Luo  Juan Gómez-Luna  Mohammad Sadrosadati  Onur Mutlu

ETH Zürich

https://arxiv.org/pdf/2402.18736
The Capability of COTS DRAM Chips

We demonstrate that COTS DRAM chips:

1. Can simultaneously activate up to 48 rows in two neighboring subarrays

2. Can perform **NOT** operation with up to **32 output operands**

3. Can perform up to **16-input AND, NAND, OR, and NOR** operations
DRAM Testing Infrastructure

• Developed from DRAM Bender [Olgun+, TCAD’23]*
• Fine-grained control over DRAM commands, timings, and temperature

### DRAM Chips Tested

- **256 DDR4 chips** from **two major DRAM manufacturers**
- Covers **different die revisions and chip densities**

<table>
<thead>
<tr>
<th>Chip Mfr.</th>
<th>#Modules (#Chips)</th>
<th>Die Rev.</th>
<th>Mfr. Date&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Chip Density</th>
<th>Chip Org.</th>
<th>Speed Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK Hynix</td>
<td>9 (72)</td>
<td>M</td>
<td>N/A</td>
<td>4Gb</td>
<td>x8</td>
<td>2666MT/s</td>
</tr>
<tr>
<td></td>
<td>5 (40)</td>
<td>A</td>
<td>N/A</td>
<td>4Gb</td>
<td>x8</td>
<td>2133MT/s</td>
</tr>
<tr>
<td></td>
<td>1 (16)</td>
<td>A</td>
<td>N/A</td>
<td>8Gb</td>
<td>x8</td>
<td>2666MT/s</td>
</tr>
<tr>
<td></td>
<td>1 (32)</td>
<td>A</td>
<td>18-14</td>
<td>4Gb</td>
<td>x4</td>
<td>2400MT/s</td>
</tr>
<tr>
<td></td>
<td>1 (32)</td>
<td>A</td>
<td>16-49</td>
<td>8Gb</td>
<td>x4</td>
<td>2400MT/s</td>
</tr>
<tr>
<td></td>
<td>1 (32)</td>
<td>M</td>
<td>16-22</td>
<td>8Gb</td>
<td>x4</td>
<td>2666MT/s</td>
</tr>
<tr>
<td>Samsung</td>
<td>1 (8)</td>
<td>F</td>
<td>21-02</td>
<td>4Gb</td>
<td>x8</td>
<td>2666MT/s</td>
</tr>
<tr>
<td></td>
<td>2 (16)</td>
<td>D</td>
<td>21-10</td>
<td>8Gb</td>
<td>x8</td>
<td>2133MT/s</td>
</tr>
<tr>
<td></td>
<td>1 (8)</td>
<td>A</td>
<td>22-12</td>
<td>8Gb</td>
<td>x8</td>
<td>3200MT/s</td>
</tr>
</tbody>
</table>
The Capability of COTS DRAM Chips

We demonstrate that COTS DRAM chips:

1. Can simultaneously activate up to 48 rows in two neighboring subarrays

2. Can perform NOT operation with up to 32 output operands

3. Can perform up to 16-input AND, NAND, OR, and NOR operations
Characterization Methodology

- To understand **which and how many** rows are simultaneously activated

  - **Sweep** Row A and Row B addresses

  ![Diagram](image)

  - All rows in Subarray X
  - All rows in Subarray Y

  - ACT Row A $\rightarrow$ PRE $\rightarrow$ ACT Row B
  - <3ns

  - DRAM Bank
  - Subarray X
    - Row A
  - Shared Sense Amplifiers
  - Subarray Y
    - Row B

  - To understand which and how many rows are simultaneously activated.
COTS DRAM chips have \textbf{two distinct} sets of activation patterns in \textbf{neighboring subarrays} when two rows are activated with \textbf{violated timings}.

- \textbf{Exactly the same number} of rows in each subarray are activated.
- \textbf{Twice as many} rows in one subarray compared to its neighbor subarray are activated.

\begin{itemize}
  \item Subarray $X$: Up to 16 rows
  \item Subarray $Y$: Up to 16 rows
  \item A total of \textbf{32 rows}
  \item Shared Sense Amplifiers
  \item Subarray $X$: Up to 16 rows
  \item Subarray $Y$: Up to 32 rows
  \item A total of \textbf{48 rows}
\end{itemize}
The Capability of COTS DRAM Chips

We demonstrate that COTS DRAM chips:

1. Can simultaneously activate up to 48 rows in two neighboring subarrays

2. Can perform **NOT** operation with **up to 32** output operands

3. Can perform **up to 16-input** AND, NAND, OR, and NOR operations
Characterization Methodology

- **Sweep** Row A and Row B addresses

  ![Diagram showing ACT Row A, PRE, ACT Row B, and Nominal states](image)

- **Sweep** DRAM chip temperature

  ![Temperature graph showing 50°C and 95°C](image)
## Key Takeaways from In-DRAM NOT Operation

### Key Takeaway 1

COTS DRAM chips can perform NOT operations with up to 32 destination rows.

### Key Takeaway 2

Temperature has a small effect on the reliability of NOT operations.
The Capability of COTS DRAM Chips

We demonstrate that COTS DRAM chips:

1. Can simultaneously activate up to 48 rows in two neighboring subarrays.

2. Can perform NOT operation with up to 32 output operands.

3. Can perform up to 16-input AND, NAND, OR, and NOR operations.
Key Idea

Manipulate the bitline voltage to express a wide variety of functions using multiple-row activation in neighboring subarrays.

SAFARI
Two-Input AND and NAND Operations

V_{DD} = 1 & GND = 0

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>COM</th>
<th>REF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

AVG(V_{DD}, V_{DD}/2)

X

V_{DD}/2

Y

AVG(X, Y)

Reference Subarray (REF)

Compute Subarray (COM)

## Key Takeaways from In-DRAM Operations

<table>
<thead>
<tr>
<th>Key Takeaway 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>COTS DRAM chips can perform {2, 4, 8, 16}-input AND, NAND, OR, and NOR operations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Takeaway 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>COTS DRAM chips can perform AND, NAND, OR, and NOR operations with very high reliability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Takeaway 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data pattern slightly affects the reliability of AND, NAND, OR, and NOR operations</td>
</tr>
</tbody>
</table>
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§, Hasan Hassan§, Konstantinos Kanellopoulos§, Oğuz Ergin†, Onur Mutlu§, Behzad Salami§*

§ETH Zürich, †TOBB ETÜ, *BSC

https://github.com/cmu-safari/pidram
https://www.youtube.com/watch?v=qeukNs5XI3g&t=4192s
**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=qeukNs5XI3g&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 **STORE instructions** that target the memory mapped registers of the **PuM Operations Controller (POC)**

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**
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. 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 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
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.

Prototype on a RISC-V rocket chip system implemented on an FPGA. Described in our preprint:

https://arxiv.org/abs/2111.00082
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

Processing-using-memory (PuM) techniques leverage the analog operation of memory cells to perform computation. Several recent works have demonstrated PuM techniques in off-the-shelf DRAM devices. Since DRAM is the dominant memory technology as main memory in current computing systems, these PuM techniques represent an opportunity for alleviating the data movement bottleneck at very low cost. However, system integration of PuM techniques imposes non-trivial challenges that are yet to be solved. Design space exploration of potential solutions to the PuM 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 PuM integration challenges.

We design and develop PiDRAM, the first flexible end-to-end framework that enables system integration studies and evaluation of real PuM techniques. PiDRAM provides software and hardware components to rapidly integrate PuM 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 PuM 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
In-DRAM Physical Unclonable Functions


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

Jeremie S. Kim$\dagger$§ Minesh Patel§ Hasan Hassan§ Onur Mutlu§$\dagger$

$\dagger$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
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$§$, A. Giray Yağılkçı$§$, Haocong Luo$§$, Jeremie S. Kim$§$, F. Nisa Bostanci$§\dagger$, Nandita Vijaykumar$§\circ$, Oğuz Ergin$\dagger$, Onur Mutlu$§$

$§$ETH Zürich $\dagger$TOBB University of Economics and Technology $\circ$University of Toronto
In-DRAM True Random Number Generation

- F. Nisa Bostanci, Ataberk Olgun, Lois Orosa, A. Giray Yaglikci, Jeremie S. Kim, Hasan Hassan, Oguz 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§, Oguz Ergin†, Onur Mutlu§

†TOBB University of Economics and Technology
§ETH Zürich

Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

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

University of California, Santa Barbara¹, Hewlett Packard Labs²
University of California, Santa Cruz³, Qualcomm Inc.⁴, Huawei Technologies Inc.⁵
{shuangchenli, yuanxie}@ece.ucsb.edu¹
Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs $n$-row bitwise operations inside NVM in one step.
In-Flash Bulk Bitwise Execution

- Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez-Luna, Myungsuk Kim, and Onur Mutlu,
"Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory"
Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.
[Slides (pptx) (pdf)]
[Longer Lecture Slides (pptx) (pdf)]
[Lecture Video (44 minutes)]
[arXiv version]

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

Jisung Park$\nabla$  Roknoddin Azizi$\S$  Geraldo F. Oliveira$\S$  Mohammad Sadrosadati$\S$
Rakesh Nadig$\S$  David Novo$\dagger$  Juan Gómez-Luna$\S$  Myungsuk Kim$\dagger$  Onur Mutlu$\S$

$\S$ETH Zürich  $\nabla$POSTECH  $\dagger$LIRMM, Univ. Montpellier, CNRS  $\dagger$Kyungpook National University

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

\[
\begin{pmatrix}
i_1 & i_2 & i_3 & i_4
\end{pmatrix}
\begin{pmatrix}
w_{11} & w_{12} & w_{13} & w_{14} \\
w_{22} & w_{23} & w_{24} & w_{34} \\
w_{33} & w_{34} & w_{35} & w_{44} \\
w_{44} & w_{45} & w_{46} & w_{47} \\
\end{pmatrix}
= \begin{pmatrix}
o_1 & o_2 & o_3 & o_4
\end{pmatrix}
\]

\[
I = \frac{1}{R_{11}} O_1 + \frac{1}{R_{22}} O_2 + \frac{1}{R_{33}} O_3 + \frac{1}{R_{44}} O_4
\]
Tutorial on Memory-Centric Computing: Processing Using Memory

Geraldo F. Oliveira
Prof. Onur Mutlu

ISCA 2024
29 June 2024
Agenda

- Introduction to Memory-Centric Computing Systems
- Invited Talk by Prof. Minsoo Rhu: “Memory-Centric Computing Systems – For AI and Beyond”
- Coffee Break
- Real-World Processing-Near-Memory Systems
- Invited Talk by Prof. Saugata Ghose: “RACER and ReRAM PUM”
- PIM Programming & Infrastructure for PIM Research
- Closing Remarks