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

Juan Gómez Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, Onur Mutlu

https://github.com/CMU-SAFARI/prim-benchmarks
Executive Summary

• **Data movement** between memory/storage units and compute units is a major contributor to execution time and energy consumption.

• **Processing-in-Memory (PIM)** is a paradigm that can tackle the *data movement bottleneck*.
  - Though explored for +50 years, technology challenges prevented the successful materialization.

• UPMEM has designed and fabricated the **first publicly-available real-world PIM architecture**.
  - DDR4 chips embedding in-order multithreaded DRAM Processing Units (DPUs).

• **Our work:**
  - Introduction to UPMEM programming model and PIM architecture.
  - Microbenchmark-based characterization of the DPU.
  - Benchmarking and workload suitability study.

• **Main contributions:**
  - Comprehensive characterization and analysis of the first commercially-available PIM architecture.
  - **PrIM (Processing-In-Memory)** benchmarks:
    - 16 workloads that are memory-bound in conventional processor-centric systems.
    - Strong and weak scaling characteristics.
  - Comparison to *state-of-the-art CPU and GPU*.

• **Takeaways:**
  - Workload characteristics for PIM suitability.
  - Programming recommendations.
  - Suggestions and hints for hardware and architecture designers of future PIM systems.
  - **PrIM**: (a) programming samples, (b) evaluation and comparison of current and future PIM systems.
Data Movement in Computing Systems

• **Data movement** dominates **performance** and is a major system energy bottleneck

• **Total system energy:** data movement accounts for
  - 62% in consumer applications*,
  - 40% in scientific applications*,
  - 35% in mobile applications*

---

* Boroumand et al., “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” ASPLOS 2018
* Kestor et al., “Quantifying the Energy Cost of Data Movement in Scientific Applications,” IISWC 2013
* Pandiy and Wu, “Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms,” IISWC 2014
Data Movement in Computing Systems

- Data movement dominates performance and is a major system energy bottleneck.

- Total system energy: data movement accounts for
  - 62% in consumer applications,*
  - 40% in scientific applications,**
  - 35% in mobile applications.***

* Boroumand et al., “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” ASPLOS 2018
** Kestor et al., “Quantifying the Energy Cost of Data Movement in Scientific Applications,” IISWC 2013
*** Pandiyan and Wu, “Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms,” IISWC 2014

Compute systems should be more data-centric.

Processing-In-Memory proposes computing where it makes sense (where data resides).
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

---

Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware

Juan Gómez-Luna  
ETH Zürich

Izzat El Hajj  
American University of Beirut

Ivan Fernandez  
University of Malaga

Christina Giannoula  
National Technical University of Athens

Geraldo F. Oliveira  
ETH Zürich

Onur Mutlu  
ETH Zürich

https://doi.org/10.1109/IGSC54211.2021.9651614
https://github.com/CMU-SAFARI/prim-benchmarks
Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

Juan Gómez-Luna¹  Izzat El Hajj²  Ivan Fernandez¹,³  Christina Giannoula¹,⁴  Geraldo F. Oliveira¹  Onur Mutlu¹

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

https://doi.org/10.1109/ACCESS.2022.3174101
https://github.com/CMU-SAFARI/prim-benchmarks
Outline

• Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview

• UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

• DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

• PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity

• Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

• Key Takeaways
Accelerator Model

• UPMEM DIMMs coexist with conventional DIMMs

• Integration of UPMEM DIMMs in a system follows an accelerator model

• UPMEM DIMMs can be seen as a loosely coupled accelerator
  - Explicit data movement between the main processor (host CPU) and the accelerator (UPMEM)
  - Explicit kernel launch onto the UPMEM processors

• This resembles GPU computing
System Organization (I)

• In a UPMEM-based PIM system UPMEM DIMMs coexist with regular DDR4 DIMMs
System Organization (II)

• A UPMEM DIMM contains **8 or 16 chips**
  - Thus, **1 or 2 ranks** of 8 chips each

• Inside each PIM chip there are:
  - **8 64MB banks** per chip: **Main RAM (MRAM)** banks
  - **8 DRAM Processing Units (DPUs)** in each chip, **64 DPUs** per rank
2,560-DPU UPMEM PIM System

- 20 UPMEM DIMMs of 16 chips each (40 ranks)
- Dual x86 socket
- UPMEM DIMMs coexist with regular DDR4 DIMMs
  - 2 memory controllers/socket (3 channels each)
  - 2 conventional DDR4 DIMMs on one channel of one controller

* There are some faulty DPUs in the system that we use in our experiments. Thus, the maximum number of DPUs we can use is 2,524
# Outline

- **Introduction**
  - Accelerator Model
  - UPMEM-based PIM System Overview

- **UPMEM PIM Programming**
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

- **DRAM Processing Unit**
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

- **PrIM Benchmarks**
  - Roofline Model
  - Benchmark Diversity

- **Evaluation**
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

- **Key Takeaways**
Vector Addition (VA)

• Our first programming example
• We partition the input arrays across:
  - DPUs
  - Tasklets, i.e., software threads running on a DPU
CPU-DPU/DPU-CPU Data Transfers

- CPU-DPU and DPU-CPU transfers
  - Between host CPU’s main memory and DPUs’ MRAM banks

- Serial CPU-DPU/DPU-CPU transfers:
  - A single DPU (i.e., 1 MRAM bank)

- Parallel CPU-DPU/DPU-CPU transfers:
  - Multiple DPUs (i.e., many MRAM banks)

- Broadcast CPU-DPU transfers:
  - Multiple DPUs with a single buffer
Inter-DPU Communication

- There is no direct communication channel between DPUs
- Inter-DPU communication takes place via the host CPU using CPU-DPU and DPU-CPU transfers
- Example communication patterns:
  - Merging of partial results to obtain the final result
    - Only DPU-CPU transfers
  - Redistribution of intermediate results for further computation
    - DPU-CPU transfers and CPU-DPU transfers
How Fast are these Data Transfers?

• With a microbenchmark, we obtain the sustained bandwidth of all types of CPU-DPU and DPU-CPU transfers

• Two experiments:
  - 1 DPU: variable CPU-DPU and DPU-CPU transfer size (8 bytes to 32 MB)
  - 1 rank: 32 MB CPU-DPU and DPU-CPU transfers to/from a set of 1 to 64 MRAM banks within the same rank

• Experiments with more than one rank
  - Channel-level parallelism

SAFARI
Outline

• Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview

• UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

• DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

• PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity

• Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

• Key Takeaways
DRAM Processing Unit

PIM Chip

Control/Status Interface

DDR4 Interface

DISPATCH
FETCH1
FETCH2
FETCH3
READOP1
READOP2
READOP3
FORMAT
ALU1
ALU2
ALU3
ALU4
MERGE1
MERGE2

24-KB IRAM

64-KB WRAM

DMA Engine

64-MB DRAM Bank (MRAM)

Pipeline

Register File

64 bits

x8

SAFARI
DPU Pipeline

- In-order pipeline
  - Up to 425 MHz
- Fine-grain multithreaded
  - 24 hardware threads
- 14 pipeline stages
  - DISPATCH: Thread selection
  - FETCH: Instruction fetch
  - READOP: Register file
  - FORMAT: Operand formatting
  - ALU: Operation and WRAM
  - MERGE: Result formatting
Arithmetic Throughput: Microbenchmark

• Goal
  - Measure the maximum arithmetic throughput for different datatypes and operations

• Microbenchmark
  - We stream over an array in WRAM and perform read-modify-write operations
  - Experiments on one DPU
  - We vary the number of tasklets from 1 to 24
  - Arithmetic operations: add, subtract, multiply, divide
  - Datatypes: int32, int64, float, double

• We measure cycles with an accurate cycle counter that the SDK provides
  - We include WRAM accesses (including address calculation) and arithmetic operation
The arithmetic throughput of a DRAM Processing Unit saturates at 11 or more tasklets. This observation is consistent for different datatypes (INT32, INT64, UINT32, UINT64, FLOAT, DOUBLE) and operations (ADD, SUB, MUL, DIV).
Arithmetic Throughput: Native Support

Key Observation 2

• DPUs provide native hardware support for 32- and 64-bit integer addition and subtraction, leading to high throughput for these operations.

• DPUs do not natively support 32- and 64-bit multiplication and division, and floating point operations. These operations are emulated by the UPMEM runtime library, leading to much lower throughput.
DPU: WRAM Bandwidth

**PIM Chip**

- Control/Status Interface
- DDR4 Interface
- 24-KB IRAM
- DMA Engine
- 64-KB WRAM
- 64-MB DRAM Bank (MRAM)
- 64 bits
- x8

**Pipeline**
- DISPATCH
- FETCH1
- FETCH2
- FETCH3
- READOP1
- READOP2
- READOP3
- FORMAT
- ALU1
- ALU2
- ALU3
- ALU4
- MERGE1
- MERGE2

**Register File**
DPU: MRAM Latency and Bandwidth

**PIM Chip**

- Control/Status Interface
- DDR4 Interface
- 24-KB IRAM
- 64-KB WRAM
- DMA Engine
- 64-MB DRAM Bank (MRAM)

Pipeline:
- DISPATCH
- FETCH1
- FETCH2
- FETCH3
- READOP1
- READOP2
- READOP3
- FORMAT
- ALU1
- ALU2
- ALU3
- ALU4
- MERGE1
- MERGE2

64 bits
MRAM Bandwidth

• Goal
  - Measure MRAM bandwidth for different access patterns

• Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    • mram_read(); // MRAM-WRAM DMA transfer
    • mram_write(); // WRAM-MRAM DMA transfer
  - STREAM benchmark
    • COPY, COPY-DMA
    • ADD, SCALE, TRIAD
  - Strided access pattern
    • Coarse-grain strided access
    • Fine-grain strided access
  - Random access pattern (GUPS)

• We do include accesses to MRAM
MRAM Read and Write Latency (I)

We can model the MRAM latency with a linear expression:

\[ \text{MRAM Latency (in cycles)} = \alpha + \beta \times \text{size} \]

In our measurements, \( \beta \) equals 0.5 cycles/byte.

Theoretical maximum MRAM bandwidth = 700 MB/s at 350 MHz.
**KEY OBSERVATION 4**

- The DPU’s **Main memory (MRAM)** bank access latency increases **linearly** with the transfer size.
- The maximum theoretical MRAM **bandwidth** is 2 bytes per cycle.
MRAM Bandwidth

- Goal
  - Measure **MRAM bandwidth** for different access patterns

- Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    - `mram_read();` // MRAM-WRAM DMA transfer
    - `mram_write();` // WRAM-MRAM DMA transfer
  - **STREAM** benchmark
    - COPY, COPY-DMA
    - ADD, SCALE, TRIAD
  - Strided access pattern
    - Coarse-grain strided access
    - Fine-grain strided access
  - Random access pattern (GUPS)

- We do include accesses to MRAM
KEY OBSERVATION 5

- **When the access latency to an MRAM bank** for a streaming benchmark (COPY-DMA, COPY, ADD) is larger than the pipeline latency (i.e., execution latency of arithmetic operations and WRAM accesses), the performance of the DPU saturates at a number of tasklets smaller than 11. This is a memory-bound workload.

- **When the pipeline latency** for a streaming benchmark (SCALE, TRIAD) is larger than the MRAM access latency, the performance of a DPU saturates at 11 tasklets. This is a compute-bound workload.
MRAM Bandwidth

• Goal
  - Measure MRAM bandwidth for different access patterns

• Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    • mram_read(); // MRAM-WRAM DMA transfer
    • mram_write(); // WRAM-MRAM DMA transfer
  - STREAM benchmark
    • COPY, COPY-DMA
    • ADD, SCALE, TRIAD
  - Strided access pattern
    • Coarse-grain strided access
    • Fine-grain strided access
  - Random access pattern (GUPS)

• We do include accesses to MRAM
DPU: Arithmetic Throughput vs. Operational Intensity

**PIM Chip**

- Control/Status Interface
- DDR4 Interface
- Register File
- Pipeline
- Dispatch
- Fetch1
- Fetch2
- Fetch3
- Readop1
- Readop2
- Readop3
- Format
- ALU1
- ALU2
- ALU3
- ALU4
- Merge1
- Merge2
- DMA Engine
- 64-KB WRAM
- 24-KB IRAM
- 64-MB DRAM Bank (MRAM)
**Arithmetic Throughput vs. Operational Intensity (I)**

**Goal**
- Characterize memory-bound regions and compute-bound regions for different datatypes and operations

**Microbenchmark**
- We load one chunk of an MRAM array into WRAM
- Perform a variable number of operations on the data
- Write back to MRAM

**The experiment is inspired by the Roofline model**

**We define operational intensity** (OI) as the number of arithmetic operations performed per byte accessed from MRAM (OP/B)

**The pipeline latency changes with the operational intensity, but the MRAM access latency is fixed**

---

Arithmetic Throughput vs. Operational Intensity (II)

In the memory-bound region, the arithmetic throughput increases with the operational intensity.

In the compute-bound region, the arithmetic throughput is flat at its maximum.

The throughput saturation point is the operational intensity where the transition between the memory-bound region and the compute-bound region happens.

The throughput saturation point is as low as 1/4 OP/B, i.e., 1 integer addition per every 32-bit element fetched.
Outline

• Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview

• UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

• DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

• PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity

• Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

• Key Takeaways
PrIM Benchmarks

• Goal
  - A common set of workloads that can be used to
    • evaluate the UPMEM PIM architecture,
    • compare software improvements and compilers,
    • compare future PIM architectures and hardware

• Two key selection criteria:
  - Selected workloads from different application domains
  - Memory-bound workloads on processor-centric architectures

• 14 different workloads, 16 different benchmarks*

*There are two versions for two of the workloads (HST, SCAN).
## PrIM Benchmarks: Application Domains

<table>
<thead>
<tr>
<th>Domain</th>
<th>Benchmark</th>
<th>Short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense linear algebra</td>
<td>Vector Addition</td>
<td>VA</td>
</tr>
<tr>
<td></td>
<td>Matrix-Vector Multiply</td>
<td>GEMV</td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Sparse Matrix-Vector Multiply</td>
<td>SpMV</td>
</tr>
<tr>
<td>Databases</td>
<td>Select</td>
<td>SEL</td>
</tr>
<tr>
<td></td>
<td>Unique</td>
<td>UNI</td>
</tr>
<tr>
<td>Data analytics</td>
<td>Binary Search</td>
<td>BS</td>
</tr>
<tr>
<td></td>
<td>Time Series Analysis</td>
<td>TS</td>
</tr>
<tr>
<td>Graph processing</td>
<td>Breadth-First Search</td>
<td>BFS</td>
</tr>
<tr>
<td>Neural networks</td>
<td>Multilayer Perceptron</td>
<td>MLP</td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>Needleman-Wunsch</td>
<td>NW</td>
</tr>
<tr>
<td>Image processing</td>
<td>Image histogram (short)</td>
<td>HST-S</td>
</tr>
<tr>
<td></td>
<td>Image histogram (large)</td>
<td>HST-L</td>
</tr>
<tr>
<td>Parallel primitives</td>
<td>Reduction</td>
<td>RED</td>
</tr>
<tr>
<td></td>
<td>Prefix sum (scan-scan-add)</td>
<td>SCAN-SSA</td>
</tr>
<tr>
<td></td>
<td>Prefix sum (reduce-scan-scan)</td>
<td>SCAN-RSS</td>
</tr>
<tr>
<td></td>
<td>Matrix transposition</td>
<td>TRNS</td>
</tr>
</tbody>
</table>
Roofline Model

- Intel Advisor on an Intel Xeon E3-1225 v6 CPU

All workloads fall in the memory-bound area of the Roofline
Outline

• Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview

• UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

• DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

• PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity

• Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

• Key Takeaways
Evaluation Methodology

• We evaluate the 16 PrIM benchmarks on two UPMEM-based systems:
  - 2,556-DPU system
  - 640-DPU system

• Strong and weak scaling experiments on the 2,556-DPU system
  - 1 DPU with different numbers of tasklets
  - 1 rank (strong and weak)
  - Up to 32 ranks

• Comparison of both UPMEM-based PIM systems to state-of-the-art CPU and GPU
  - Intel Xeon E3-1240 CPU
  - NVIDIA Titan V GPU
Strong Scaling: 1 DPU (I)

- Strong scaling experiments on 1 DPU
  - We set the number of tasklets to 1, 2, 4, 8, and 16
  - We show the breakdown of execution time:
    - **DPU**: Execution time on the DPU
    - **Inter-DPU**: Time for inter-DPU communication via the host CPU
    - **CPU-DPU**: Time for CPU to DPU transfer of input data
    - **DPU-CPU**: Time for DPU to CPU transfer of final results
  - Speedup over 1 tasklet
Strong Scaling: 1 DPU (II)

VA, GEMV, SpMV, SEL, UNI, TS, MLP, NW, HST-S, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), and TRNS (Step 2 kernel), the best performing number of tasklets is 16

Speedups 1.5-2.0x as we double the number of tasklets from 1 to 8. Speedups 1.2-1.5x from 8 to 16, since the pipeline throughput saturates at 11 tasklets

KEY OBSERVATION 10
A number of tasklets greater than 11 is a good choice for most real-world workloads we tested (16 kernels out of 19 kernels from 16 benchmarks), as it fully utilizes the DPU’s pipeline.
Strong Scaling: 1 Rank

- Strong scaling experiments on 1 rank
  - We set the number of tasklets to the best performing one
  - The number of DPUs is 1, 4, 16, 64
  - We show the breakdown of execution time:
    - **DPU**: Execution time on the DPU
    - **Inter-DPU**: Time for inter-DPU communication via the host CPU
    - **CPU-DPU**: Time for CPU to DPU transfer of input data
    - **DPU-CPU**: Time for DPU to CPU transfer of final results
  - Speedup over 1 DPU
**Weak Scaling: 1 Rank**

**KEY OBSERVATION 17**

Equally-sized problems assigned to different DPUs and little/no inter-DPU synchronization lead to linear weak scaling of the execution time spent on the DPUs (i.e., constant execution time when we increase the number of DPUs and the dataset size accordingly).

**KEY OBSERVATION 18**

Sustained bandwidth of parallel CPU-DPU/DPU-CPU transfers inside a rank of DPUs increases sublinearly with the number of DPUs.
**Strong Scaling: 32 Ranks**

- **Strong scaling experiments on 32 rank**
  - We set the number of tasklets to the best performing one
  - The number of DPUs is 256, 512, 1024, 2048
  - We show the breakdown of execution time:
    - **DPU**: Execution time on the DPU
    - **Inter-DPU**: Time for inter-DPU communication via the host CPU
    - We do not show CPU-DPU/DPU-CPU transfer times
  - Speedup over 256 DPUs

---

**Table:**

<table>
<thead>
<tr>
<th>#DPUs</th>
<th>Speedup (Add)</th>
<th>Speedup (Scan)</th>
<th>Speedup (Red.)</th>
<th>Speedup (Step 1)</th>
<th>Speedup (Step 2)</th>
<th>Speedup (Step 3)</th>
<th>Speedup (Step 4)</th>
<th>Speedup (Step 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>512</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1024</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2048</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
CPU/GPU: Evaluation Methodology

- Comparison of both UPMEM-based PIM systems to state-of-the-art CPU and GPU
  - Intel Xeon E3-1240 CPU
  - NVIDIA Titan V GPU
- We use state-of-the-art CPU and GPU counterparts of PrIM benchmarks
  - https://github.com/CMU-SAFARI/prim-benchmarks
- We use the largest dataset that we can fit in the GPU memory
- We show overall execution time, including DPU kernel time and inter DPU communication
The 2,556-DPU and the 640-DPU systems outperform the CPU for all benchmarks except SpMV, BFS, and NW.

The 2,556-DPU and the 640-DPU are, respectively, 93.0x and 27.9x faster than the CPU for 13 of the PrIM benchmarks.
The 2,556-DPU outperforms the GPU for 10 PrIM benchmarks with an average of 2.54x.

The performance of the 640-DPU is within 65% the performance of the GPU for the same 10 PrIM benchmarks.
The UPMEM-based PIM system can outperform a state-of-the-art GPU on workloads with three key characteristics:

1. Streaming memory accesses
2. No or little inter-DPU synchronization
3. No or little use of integer multiplication, integer division, or floating point operations

These three key characteristics make a workload potentially suitable to the UPMEM PIM architecture.
Outline

• Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview

• UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth

• DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth

• PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity

• Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU

• Key Takeaways
**Key Takeaway 1**

The UPMEM PIM architecture is fundamentally compute bound. As a result, the most suitable workloads are memory-bound.

The throughput saturation point is as low as $\frac{1}{4}$ OP/B, i.e., 1 integer addition per every 32-bit element fetched.
Key Takeaway 2

The most well-suited workloads for the UPMEM PIM architecture use no arithmetic operations or use only simple operations (e.g., bitwise operations and integer addition/subtraction).
**Key Takeaway 3**

The most well-suited workloads for the UPMEM PIM architecture require little or no communication across DPUs (inter-DPU communication).
• UPMEM-based PIM systems **outperform state-of-the-art CPUs in terms of performance** (by $23.2 \times$ on 2,556 DPUs for 16 PrIM benchmarks) and **energy efficiency on most of PrIM benchmarks**.

• UPMEM-based PIM systems **outperform state-of-the-art GPUs on a majority of PrIM benchmarks** (by $2.54 \times$ on 2,556 DPUs for 10 PrIM benchmarks), and the outlook is even more positive for future PIM systems.

• UPMEM-based PIM systems are **more energy-efficient than state-of-the-art CPUs and GPUs on workloads that they provide performance improvements** over the CPUs and the GPUs.
Executive Summary

• **Data movement** between memory/storage units and compute units is a major contributor to execution time and energy consumption

• **Processing-in-Memory (PIM)** is a paradigm that can tackle the *data movement bottleneck*
  - Though explored for +50 years, technology challenges prevented the successful materialization

• UPMEM has designed and fabricated the *first publicly-available real-world PIM architecture*
  - DDR4 chips embedding in-order multithreaded DRAM Processing Units (DPUs)

• Our work:
  - Introduction to UPMEM programming model and PIM architecture
  - Microbenchmark-based characterization of the DPU
  - Benchmarking and *workload suitability* study

• Main contributions:
  - Comprehensive characterization and analysis of the first commercially-available PIM architecture
  - **PrIM (Processing-In-Memory) benchmarks:**
    • 16 workloads that are memory-bound in conventional processor-centric systems
    • Strong and weak scaling characteristics
  - Comparison to *state-of-the-art CPU and GPU*

• Takeaways:
  - Workload characteristics for PIM suitability
  - Programming recommendations
  - Suggestions and hints for hardware and architecture designers of future PIM systems
  - PrIM: (a) programming samples, (b) evaluation and comparison of current and future PIM systems
Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware

Juan Gómez-Luna
ETH Zürich

Izzat El Hajj
American University of Beirut

Ivan Fernandez
University of Malaga

Christina Giannoula
National Technical University of Athens

Geraldo F. Oliveira
ETH Zürich

Onur Mutlu
ETH Zürich

https://doi.org/10.1109/IGSC54211.2021.9651614
https://github.com/CMU-SAFARI/prim-benchmarks
Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

Juan Gómez-Luna\(^1\)  Izzat El Hajj\(^2\)  Ivan Fernandez\(^1,3\)  Christina Giannoula\(^1,4\)
Geraldo F. Oliveira\(^1\)  Onur Mutlu\(^1\)

\(^1\)ETH Zürich  \(^2\)American University of Beirut  \(^3\)University of Malaga  \(^4\)National Technical University of Athens

https://doi.org/10.1109/ACCESS.2022.3174101
https://github.com/CMU-SAFARI/prim-benchmarks
Understanding a Modern PIM Architecture

https://www.youtube.com/watch?v=D8Hjy2iU9l4&list=PL5Q2soXY2Zi_tOTAYm--dYByNPL7JhwR9
PrIM Repository

- All microbenchmarks, benchmarks, and scripts
- [https://github.com/CMU-SAFARI/prim-benchmarks](https://github.com/CMU-SAFARI/prim-benchmarks)

PrIM (Processing-In-Memory Benchmarks)

PrIM is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PrIM is developed to evaluate, analyze, and characterize the first publicly-available real-world processing-in-memory (PIM) architecture, the UPMEM PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called DRAM Processing Units (DPUs), integrated in the same chip.

PrIM provides a common set of workloads to evaluate the UPMEM PIM architecture with and can be useful for programming, architecture and system researchers all alike to improve multiple aspects of future PIM hardware and software. The workloads have different characteristics, exhibiting heterogeneity in their memory access patterns, operations and data types, and communication patterns. This repository also contains baseline CPU and GPU implementations of PrIM benchmarks for comparison purposes.

PrIM also includes a set of microbenchmarks can be used to assess various architecture limits such as compute throughput and memory bandwidth.
Processing-in-Memory (PIM)

• PIM is a computing paradigm that advocates for memory-centric computing systems, where processing elements are placed near or inside the memory arrays.

• Real-world PIM architectures are becoming a reality
  - UPMEM PIM, Samsung HBM-PIM, Samsung AxDIMM, SK Hynix AiM, Alibaba HB-PNM

• These PIM systems have some common characteristics:
  1. There is a host processor (CPU or GPU) with access to (1) standard main memory, and (2) PIM-enabled memory.
  2. PIM-enabled memory contains multiple PIM processing elements (PEs) with high bandwidth and low latency memory access.
  3. PIM PEs run only at a few hundred MHz and have a small number of registers and small (or no) cache/scratchpad.
  4. PIM PEs may need to communicate via the host processor.
Processing-in-Memory Course (Spring 2023)

- Short weekly lectures
- Hands-on projects

https://safari.ethz.ch/projects_and_seminars/spring2023/doku.php?id=processing_in_memory

https://www.youtube.com/playlist?list=PL5Q2soXY2Zi_EOбуoAZVSq_o6UySWQHyZ
Benchmarking a New Paradigm: Analysis of a Real Processing-in-Memory System

Juan Gómez Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, Onur Mutlu

https://github.com/CMU-SAFARI/prim-benchmarks