P&S Processing-in-Memory

Benchmarking and Workload Suitability on a Real-World PIM Architecture

Dr. Juan Gómez Luna
Prof. Onur Mutlu
ETH Zürich
Fall 2022
13 December 2022
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

System Organization

- 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**
Accelerator Model (II)

- **FIG. 6** is a flow diagram representing operations in a method of delegating a processing task to a DRAM processor according to an example embodiment.

![Flow Diagram](image-url)

**Fig 6**

- **Fabric Devaux, Jean-François Roy.** “Memory circuit with integrated processor.” US 10,324,870 B2.
Vector Addition (VA)

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

Getting started

- The UPMEM DPU toolchain
  - Notes before starting
  - The toolchain purpose
  - dpu-upmem-dpurte-clang
    - Limitations
  - The DPU Runtime Library
  - The Host Library
  - dpu-lldb
- Installing the UPMEM DPU toolchain
  - Dependencies
    - Python
  - Installation packages
    - Installation from tar.gz binary archive
  - Functional simulator
- Hello World! Example
  - Purpose
  - Writing and building the program
  - Running and testing hello world
  - Creating a host application to drive the program
General Programming Recommendations

- From UPMEM programming guide*, presentations★, and white papers☆

**GENERAL PROGRAMMING RECOMMENDATIONS**

1. Execute on the *DRAM Processing Units (DPUs) portions of parallel code* that are as long as possible.
2. Split the workload into **independent data blocks**, which the DPUs operate on independently.
3. Use **as many working DPUs** in the system as possible.
4. Launch at least **11 tasklets** (i.e., software threads) per DPU.

★ F. Devaux, "The true Processing In Memory accelerator," HotChips 2019. doi: 10.1109/HOTCHIPS.2019.8875680
☆ UPMEM, “Introduction to UPMEM PIM. Processing-in-memory (PIM) on DRAM Accelerator,” White paper
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
Parallel Reduction (II)

- Each tasklet computes a local sum
Benchmark Selection and Diversity
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
**PrIM Benchmarks: Diversity**

- **PrIM benchmarks are diverse:**
  - Memory access patterns
  - Operations and datatypes
  - Communication/synchronization

<table>
<thead>
<tr>
<th>Domain</th>
<th>Benchmark</th>
<th>Short name</th>
<th>Memory access pattern</th>
<th>Computation pattern</th>
<th>Communication/synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sequential</td>
<td>Strided</td>
<td>Random</td>
</tr>
<tr>
<td>Dense linear algebra</td>
<td>Vector Addition</td>
<td>VA</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Matrix-Vector Multiply</td>
<td>GEMV</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Sparse Matrix-Vector Multiply</td>
<td>SpMV</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Databases</td>
<td>Select</td>
<td>SEL</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unique</td>
<td>UNI</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data analytics</td>
<td>Binary Search</td>
<td>BS</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time Series Analysis</td>
<td>TS</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph processing</td>
<td>Breadth-First Search</td>
<td>BFS</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Neural networks</td>
<td>Multilayer Perceptron</td>
<td>MLP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>Needleman-Wunsch</td>
<td>NW</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Image processing</td>
<td>Image histogram (short)</td>
<td>HST-S</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image histogram (long)</td>
<td>HST-L</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Parallel primitives</td>
<td>Reduction</td>
<td>RED</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefix sum (scan-scan-add)</td>
<td>SCAN-SSA</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefix sum (reduce-scan-scan)</td>
<td>SCAN-RSS</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Matrix transposition</td>
<td>TRNS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
## PrIM Benchmarks: Inter-DPU Communication

<table>
<thead>
<tr>
<th>Domain</th>
<th>Benchmark</th>
<th>Short name</th>
<th>Memory access pattern</th>
<th>Computation pattern</th>
<th>Communication/synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sequential</td>
<td>Strided</td>
<td>Random</td>
</tr>
<tr>
<td>Dense linear algebra</td>
<td>Vector Addition</td>
<td>VA</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Matrix-Vector Multiply</td>
<td>GEMV</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse linear algebra</td>
<td>Sparse Matrix-Vector Multiply</td>
<td>SpMV</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Databases</td>
<td>Select</td>
<td>SEL</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unique</td>
<td>UNI</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data analytics</td>
<td>Binary Search</td>
<td>BS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Time Series Analysis</td>
<td>TS</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph processing</td>
<td>Breadth-First Search</td>
<td>BFS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Neural networks</td>
<td>Multilayer Perceptron</td>
<td>MLP</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>Needleman-Wunsch</td>
<td>NW</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Image processing</td>
<td>Image histogram (short)</td>
<td>HST-S</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Image histogram (long)</td>
<td>HST-L</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel primitives</td>
<td>Reduction</td>
<td>RED</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefix sum (scan-scan-add)</td>
<td>SCAN-SSA</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefix sum (reduce-scan-scan)</td>
<td>SCAN-RSS</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Matrix transposition</td>
<td>TRNS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
# PrIM Benchmarks: Inter-DPU Communication

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

- **Inter-DPU communication**
  - **Result merging:**
    - SEL, UNI, HST-S, HST-L, RED
  - Only DPU-CPU transfers
- **Redistribution of intermediate results:**
  - BFS, MLP, NW, SCAN-SSA, SCAN-RSS
  - DPU-CPU and CPU-DPU transfers
Benchmark Evaluation
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

*Strong scaling* refers to how the execution time of a program solving a particular problem varies with the number of processors for a fixed problem size.

*Weak scaling* refers to how the execution time of a program solving a particular problem varies with the number of processors for a fixed problem size per processor.
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
## Datasets

- **Strong and weak scaling experiments**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Strong Scaling Dataset</th>
<th>Weak Scaling Dataset</th>
<th>MRAM-WRAM Transfer Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>1 DPU-1 rank: 2.5M elem. (10 MB)</td>
<td>2.5M elem./DPU (10 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>GEMV</td>
<td>1 DPU-1 rank: 8192 x 1024 elem. (32 MB)</td>
<td>1024 x 2048 elem./DPU (8 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>SpMV</td>
<td>bcsstk30 [253] (12 MB)</td>
<td>bcsstk30 [253]</td>
<td>64 bytes</td>
</tr>
<tr>
<td>SEL</td>
<td>1 DPU-1 rank: 3.8M elem. (30 MB)</td>
<td>3.8M elem./DPU (30 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>UNI</td>
<td>1 DPU-1 rank: 3.8M elem. (30 MB)</td>
<td>3.8M elem./DPU (30 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>BS</td>
<td>2M elem. (16 MB). 1 DPU-1 rank: 256K queries. (2 MB)</td>
<td>2M elem. (16 MB). 256K queries./DPU (2 MB)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>TS</td>
<td>256 elem. query. 1 DPU-1 rank: 512K elem. (2 MB)</td>
<td>512K elem./DPU (2 MB)</td>
<td>256 bytes</td>
</tr>
<tr>
<td>BFS</td>
<td>loc-gowalla [254] (22 MB)</td>
<td>rMat [255] (=100K vertices and 1.2M edges per DPU)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>MLP</td>
<td>3 fully-connected layers. 1 DPU-1 rank: 2K neurons (32 MB)</td>
<td>3 fully-connected layers. 1K neur./DPU (4 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>NW</td>
<td>1 DPU-1 rank: 2560 bps (50 MB), large/small sub-block = ( \frac{2560 \ times \ # \ DPU}{2} )</td>
<td>512 bps/DPU (2MB), 1/s = 512/2</td>
<td>8, 16, 32, 40 bytes</td>
</tr>
<tr>
<td>HST-S</td>
<td>1 DPU-1 rank: 1536 x 1024 input image [256] (6 MB)</td>
<td>1536 x 1024 input image [256]/DPU (6 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>HST-L</td>
<td>1 DPU-1 rank: 1536 x 1024 input image [256] (6 MB)</td>
<td>1536 x 1024 input image [256]/DPU (6 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>RED</td>
<td>1 DPU-1 rank: 6.3M elem. (50 MB)</td>
<td>6.3M elem./DPU (50 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>SCAN-SSA</td>
<td>1 DPU-1 rank: 3.8M elem. (30 MB)</td>
<td>3.8M elem./DPU (30 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>SCAN-RSS</td>
<td>1 DPU-1 rank: 3.8M elem. (30 MB)</td>
<td>3.8M elem./DPU (30 MB)</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>TRNS</td>
<td>1 DPU-1 rank: 12288 x 16 x 64 x 8 (768 MB)</td>
<td>12288 x 16 x 1 x 8/DPU (12 MB)</td>
<td>128, 1024 bytes</td>
</tr>
</tbody>
</table>

The **PrIM benchmarks** repository includes all datasets and scripts used in our evaluation.

[https://github.com/CMU-SAFAI/prim-benchmarks](https://github.com/CMU-SAFAI/prim-benchmarks)
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)

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 DPU (III)

VA, GEMV, SpMV, BS, TS, MLP, HST-S do not use intra-DPU synchronization primitives.

In SEL, UNI, NW, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), synchronization is lightweight.

BFS, HST-L, TRNS (Step 3) use mutexes, which cause contention when accessing shared data structures.

synchronization is lightweight kernel), SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), synchronization is lightweight.
Strong Scaling: 1 DPU (IV)

VA, GEMV, SpMV, BS, TS, MLP, HST do not use intra-DPU synchronization primitives.

In SEL, UNI, NW, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), synchronization is lightweight.

BFS, HST-L, TRNS (Step 3) use mutexes, which cause contention when accessing shared data structures.

**KEY OBSERVATION 11**

Intensive use of intra-DPU synchronization across tasklets (e.g., mutexes, barriers, handshakes) may limit scalability, sometimes causing the best performing number of tasklets to be lower than 11.
**Strong Scaling: 1 DPU (V)**

SCAN-SSA (Add kernel) is not compute-intensive. Thus, performance saturates with less than 11 tasklets (recall STREAM ADD). BS shows similar behavior.

**KEY OBSERVATION 12**

Most real-world workloads are in the compute-bound region of the DPU (all kernels except SCAN-SSA (Add kernel) and BS), i.e., the pipeline latency dominates the MRAM access latency.
Strong Scaling: 1 DPU (VI)

The amount of time spent on CPU-DPU and DPU-CPU transfers is low compared to the time spent on DPU execution.

TRNS performs step 1 of the matrix transposition via the CPU-DPU transfer. Using small transfers (8 elements) does not exploit full CPU-DPU bandwidth.

KEY OBSERVATION 13

Transferring large data chunks from/to the host CPU is preferred for input data and output results due to higher sustained CPU-DPU/DPU-CPU bandwidths.
Strong Scaling: 1 Rank (I)

- 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
Strong Scaling: 1 Rank (II)

- VA, GEMV, SpMV, SEL, UNI, BS, TS, MLP, HST-S, HSTS-L, RED, SCAN-SSA (both kernels), and TRNS (both kernels) scale linearly with the number of DPUs.

- Scaling is sublinear for BFS and NW.

- BFS suffers load imbalance due to irregular graph topology.

- NW computes a diagonal of a 2D matrix in each iteration. More DPUs does not mean more parallelization in shorter diagonals.
Strong Scaling: 1 Rank (III)

VA, GEMV, SpMV, BS, TS, TRNS do not need inter-DPU synchronization

SEL, UNI, HST-S, HST-L, RED, SCAN-SSA, SCAN-RSS need inter-DPU synchronization but 64 DPUs still obtain the best performance

BFS, MLP, NW require heavy inter-DPU synchronization, involving DPU-CPU and CPU-DPU transfers
Strong Scaling: 1 Rank (IV)

- VA, GEMV, TS, MLP, HST-S, HST-L, RED, SCAN-SSA, SCAN-RSS, TRNS use parallel transfers.
- CPU-DPU and DPU-CPU transfer times decrease as we increase the number of DPU.

- BS, NW use parallel transfers but do not reduce transfer times:
  - BS transfers a complete array to all DPU.
  - NW does not use all DPU in all iterations.

- SpMV, SEL, UNI, BFS cannot use parallel transfers, as the transfer size per DPU is not fixed.

**PROGRAMMING RECOMMENDATION 5**
Parallel CPU-DPU/DPU-CPU transfers inside a rank of DPU are recommended for real-world workloads when all transferred buffers are of the same size.
Strong Scaling: 32 Ranks (I)

• **Strong scaling experiments on 32 ranks**
  - 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**
Strong Scaling: 32 Ranks (II)

VA, GEMV, SEL, UNI, BS, TS, MLP, HST-S, HSTS-L, RED, SCAN-SSA (both kernel), SCAN-RSS (both kernels), and TRNS (both kernels) scale linearly with the number of DPUs.

VA, GEMV, SEL, UNI, BS, TS, MLP, HST-S, HSTS-L, RED, SCAN-SSA (both kernel), SCAN-RSS (both kernels), and TRNS (both kernels) scale linearly with the number of DPUs.

SpMV, BFS, NW do not scale linearly due to load imbalance.

**KEY OBSERVATION 14**

Load balancing across DPUs ensures linear reduction of the execution time spent on the DPUs for a given problem size, when all available DPUs are used (as observed in strong scaling experiments).
Strong Scaling: 32 Ranks (III)

**KEY OBSERVATION 15**

The overhead of merging partial results from DPUs in the host CPU is tolerable across all PrIM benchmarks that need it.

**KEY OBSERVATION 16**

Complex synchronization across DPUs (i.e., inter-DPU synchronization involving two-way communication with the host CPU) imposes significant overhead, which limits scalability to more DPUs.

SEL, UNI, HST-S, HST-L, RED only need to merge final results.
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.
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](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.
CPU/GPU: Performance Comparison (II)

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.**
The 640-DPU system consumes on average 1.64x less energy than the CPU for all 16 PrIM benchmarks.

For 12 benchmarks, the 640-DPU system provides energy savings of 5.23x over the CPU.
The UPMEM-based PIM system provides large energy savings over a state-of-the-art CPU due to higher performance (thus, lower static energy) and less data movement between memory and processors.

The UPMEM-based PIM system provides energy savings over a state-of-the-art CPU/GPU on workloads where it outperforms the CPU/GPU. This is because the source of both performance improvement and energy savings is the same: the significant reduction in data movement between the memory and the processor cores, which the UPMEM-based PIM system can provide for PIM-suitable workloads.

### Key Observation 20

<table>
<thead>
<tr>
<th>Workload</th>
<th>Energy Savings over CPU (log scale)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td></td>
</tr>
<tr>
<td>640 DPUs</td>
<td></td>
</tr>
</tbody>
</table>

---

**Additional Observations**

- **GMEAN (1)**
- **GMEAN (2)**
- **GMEAN**
Key Takeaways
The UPMEM PIM architecture is fundamentally compute bound. As a result, the most suitable workloads are memory-bound.
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).
Key Takeaway 4

KEY TAKEAWAY 4

• UPMEM-based PIM systems outperform state-of-the-art CPUs in terms of performance and energy efficiency on most of PrIM benchmarks.

• UPMEM-based PIM systems outperform state-of-the-art GPUs on a majority of 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.
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://github.com/CMU-SAFARI/prim-benchmarks
Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

Juan Gómez-Luna\textsuperscript{1}  Izzat El Hajj\textsuperscript{2}  Ivan Fernandez\textsuperscript{1,3}  Christina Giannoula\textsuperscript{1,4}
Geraldo F. Oliveira\textsuperscript{1}  Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich  \textsuperscript{2}American University of Beirut  \textsuperscript{3}University of Malaga  \textsuperscript{4}National Technical University of Athens

https://github.com/CMU-SAFARI/prim-benchmarks
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 GIANNOULA¹,⁴, 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
PrIM Benchmarks

- 16 benchmarks and scripts for evaluation
- https://github.com/CMU-SAFAIR/prim-benchmarks
Upcoming Lectures

- More real-world PIM architectures
- PUM architectures and prototypes
  - SIMDGRAM: An end-to-end framework for bit-serial SIMD computing in DRAM
P&S Processing-in-Memory

Benchmarking and Workload Suitability on a Real-World PIM Architecture

Dr. Juan Gómez Luna
Prof. Onur Mutlu

ETH Zürich
Fall 2022
13 December 2022