P&S Processing-in-Memory

Benchmarking and Workload Suitability on a Real-World PIM Architecture

Dr. Juan Gómez Luna
Prof. Onur Mutlu
ETH Zürich
Spring 2023
11 May 2023
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.

Fig 6

1. SOC LOADS DATA TO BE PROCESSED TO DRAM MEMORY BANK (601)
2. SOC TRANSMITS DATA PROCESSING COMMAND TO DRAM PROCESSOR(S) (602)
3. DATA PROCESSING BY DRAM PROCESSOR(S) (603)
4. DATA PROCESSING COMPLETE? (604)
   - N (604): CONTINUE
   - Y (605): MEMORY BANK ACCESSIBLE BY SOC

Vector Addition (VA)

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

Getting started

- The UPMEM DPU toolchain
  - Notes before starting
  - The toolchain purpose
  - dpu-upmem-dpurtcl-lang
  - Limitations
  - The DPU Runtime Library
  - The Host Library
  - dpu-ildb
- 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
    - About dpu-pkg-config
  - Conclusion
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
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
# 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>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>Redaction</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></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
**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 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
Strong Scaling: 1 DPU (IV)

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.

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 Rank (II)

VA, GEMV, SpMV, 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.

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: 32 Ranks (II)**

VA, GEMV, SEL, UNI, BS, TS, MLP, HST-S, HSTS-L, RED, SCAN-SSA (both kernels), 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).
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 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.
Key Takeaways
**Key Takeaway 1**

The UPMEM PIM architecture is fundamentally compute bound. As a result, the most suitable workloads are memory-bound.
Analysis of ML Training Workloads (I)

• Linear regression, logistic regression, decision tree, K-means clustering

The performance of all kernels saturates at 11 or more PIM threads. In the UPMEM PIM architecture, this means that the pipeline latency hides the memory latency

As a result, these kernels are compute-bound on the UPMEM PIM architecture
Analysis of ML Training Workloads (II)

- Linear regression, logistic regression, decision tree, K-means clustering

Key Takeaway 2. ML workloads that are memory-bound due to low arithmetic intensity in CPU/GPU become compute-bound when running on PIM.

Recommendation 6. Maximize the utilization of PIM cores by keeping their pipeline fully busy.
ML Training on a Real PIM System

• Presented at ISPASS 2023

An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System

Juan Gómez-Luna\textsuperscript{1} Yuxin Guo\textsuperscript{1} Sylvan Brocard\textsuperscript{2} Julien Legriel\textsuperscript{2} Remy Cimadomo\textsuperscript{2} Geraldo F. Oliveira\textsuperscript{1} Gagandeep Singh\textsuperscript{1} Onur Mutlu\textsuperscript{1}

\textsuperscript{1}ETH Zürich \textsuperscript{2}UPMEM

Source code: https://github.com/CMU-SAFARI/pim-ml
**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** (by 23.2 × 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 × 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.
Comparison to CPU and GPU: ML Training (I)

- Decision tree and K-means with Higgs boson dataset

**PIM version of DTR is 27x faster than the CPU version and 1.34x faster than the GPU version**

**PIM version of KME is 2.8x faster than the CPU version and 3.2x faster than the GPU version**
Comparison to CPU and GPU: ML Training (II)

- Decision tree and K-means with Criteo 1TB dataset

**DTR**

- PIM version of DTR is **62x faster** than the CPU version and **4.5x faster** than the GPU version

**KME**

- PIM version of KME is **2.7x faster** than the CPU version and **3.2x faster** than the GPU version
Comparison to CPU and GPU: ML Training (III)

• Decision tree and K-means with Criteo 1TB dataset

Key Takeaway 4. ML workloads that require mainly operations natively supported by the PIM architecture, such as decision tree and K-means clustering, outperform their CPU and GPU counterparts.

- PIM version of Decision Tree is 62x faster than the CPU version and 4.5x faster than the GPU version
- PIM version of KME is 2.7x faster than the CPU version and 3.2x faster than the GPU version
Longer Lecture

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.

PIM Course: Lecture 10: Benchmarking and Workload Suitability on PIM - Fall 2022

Onur Mutlu Lectures
32.7K subscribers

161 views 4 months ago  Livestream - P&S Data-Centric Architectures: Fundamentally Improving Performance and Energy (Fall 2022)
Projects & Seminars, ETH Zürich, Fall 2022
Data-Centric Architectures: Fundamentally Improving Performance and Energy

https://youtu.be/H_xvB_O-bWM
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-SAFARI/prim-benchmarks](https://github.com/CMU-SAFARI/prim-benchmarks)
Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems

Christina Giannoula\textsuperscript{1,2}  Ivan Fernandez\textsuperscript{1,3}  Juan Gómez-Luna\textsuperscript{1}
Nectarios Koziris\textsuperscript{2}  Georgios Goumas\textsuperscript{2}  Onur Mutlu\textsuperscript{1}
\textsuperscript{1}ETH Zürich  \textsuperscript{2}National Technical University of Athens  \textsuperscript{3}University of Malaga

https://doi.org/10.1145/3489048.3522661
Source code: https://github.com/CMU-SAFARI/SparseP
https://youtu.be/5kaOsJkIGrE
ML Training on a Real PIM System

• Presented at ISPASS 2023

An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System

Juan Gómez-Luna¹ Yuxin Guo¹ Sylvan Brocard² Julien Legriel²
Remy Cimadomo² Geraldo F. Oliveira¹ Gagandeep Singh¹ Onur Mutlu¹
¹ETH Zürich ²UPMEM

Source code: https://github.com/CMU-SAFARI/pim-ml
ML Training on a Real PIM System: ISPASS 2023 Talk

Evaluation: Analysis of PIM Kernels

- Linear regression

  All versions saturate at 11 or more PIM threads

  Fixed-point representation accelerates the kernel by an order of magnitude over FP32

Key Takeaway 1. Workloads with arithmetic operations or datatypes not natively supported by PIM cores run at low performance due to instruction emulation (e.g., FP in UPMEM PIM).

Recommendation 1. Use fixed-point representation, without much accuracy loss, if PIM cores do not support FP.

Machine Learning Training on Memory-centric Computing Systems, Juan Gómez-Luna for ISPASS 2023

180 views 1 day ago  Livestream - Data-Centric Architectures: Fundamentally Improving Performance and Energy (Spring 2023)
Evaluating Machine Learning Workloads on Memory-centric Computing Systems
Speaker: Dr. Juan Gómez Luna (https://safari.ethz.ch/juan-gomez-luna/)
ISPASS 2023 Show more

https://youtu.be/60pkaI5AeM4
A Framework for High-throughput Sequence Alignment using Real Processing-in-Memory Systems

A Framework for High-throughput Sequence Alignment using Real Processing-in-Memory Systems

Safaa Diab¹, Amir Nassereldine¹, Mohammed Alser², Juan Gómez Luna², Onur Mutlu², Izzat El Hajj¹

¹American University of Beirut, Lebanon  ²ETH Zürich, Switzerland

Source code: https://github.com/safaad/aim
A Library for Transcendental Functions

- Presented at ISPASS 2023

TransPimLib: A Library for Efficient Transcendental Functions on Processing-in-Memory Systems

Maurus Item
Geraldo F. Oliveira

Juan Gómez-Luna
Mohammad Sadrosadati

Yuxin Guo
Onur Mutlu

ETH Zürich

Source code: https://github.com/CMU-SAFARI/transpimlib
A Library for Transcendental Functions: ISPASS 2023 Talk

TransPimLib: CORDIC-based Method

- TransPimLib contains CORDIC implementations of trigonometric (sin, cos, tan) and hyperbolic (sinh, cosh, tanh) functions, exponentiation, logarithm, and square root

- Example: Sine function

TransPimLib: Efficient Transcendental Functions for PIM, Juan Gómez-Luna for ISPASS 2023

60 views 3 hours ago Livestream - Data-Centric Architectures: Fundamentally Improving Performance and Energy (Spring 2023)

TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems

Speaker: Dr. Juan Gómez Luna (https://safari.ethz.ch/juan-gomez-luna/)

ISPASS 2023 talk: Show more

https://youtu.be/lqqf4eaaEE4
Upcoming Lectures

- Case studies
  - SpMV on a real PIM system
  - ML training on a real PIM system

- PUM architectures and prototypes
  - SIMDRA: 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
Spring 2023
11 May 2023