2023 IEEE International Symposium on Performance Analysis of Systems and Software

Evaluating Machine Learning Workloads on Memory-Centric Computing Systems

<u>Juan Gómez Luna</u>, Yuxin Guo, Sylvan Brocard, Julien Legriel, Remy Cimadomo, Geraldo F. Oliveira, Gagandeep Singh, Onur Mutlu

> https://arxiv.org/pdf/2207.07886.pdf https://github.com/CMU-SAFARI/pim-ml

juang@ethz.ch





Monday, April 24, 2023



Executive Summary

- Training machine learning (ML) algorithms is a computationally expensive process, frequently memory-bound due to repeatedly accessing large training datasets
- Memory-centric computing systems, i.e., with Processing-in-Memory (PIM) capabilities, can alleviate this data movement bottleneck
- Real-world PIM systems have only recently been manufactured and commercialized
 - UPMEM has designed and fabricated the first publicly-available real-world PIM architecture
- Our goal is to understand the potential of modern general-purpose PIM architectures to accelerate machine learning training
- Our main contributions:
 - PIM implementation of several classic machine learning algorithms: linear regression, logistic regression, decision tree, K-means clustering
 - Workload characterization in terms of quality, performance, and scaling
 - Comparison to their counterpart implementations on processor-centric systems (CPU and GPU)
 - PIM version of DTR is 27x / 1.34x faster than the CPU / GPU version, respectively
 - PIM version of KME is 2.8x / 3.2x faster than the CPU / GPU version, respectively
 - Source code: <u>https://github.com/CMU-SAFARI/pim-ml</u>
- Experimental evaluation on a real-world PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM memory
- Key observations, takeaways, and recommendations for ML workloads on general-purpose PIM systems

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Machine Learning Workloads

 Machine learning training with large amounts of data is a computationally expensive process, which requires many iterations to update an ML model's parameters



- Frequent data movement between memory and processing elements to access training data
- The amount of computation is not enough to amortize the cost of moving training data to the processing elements
 - Low arithmetic intensity
 - Low temporal locality
 - Irregular memory accesses

Machine Learning Workloads: Our Goal

- Our goal is to study and analyze how real-world general-purpose PIM can accelerate ML training
- Four representative ML algorithms: linear regression, logistic regression, decision tree, K-means



 Roofline model to quantify the memory boundedness of CPU versions of the four workloads



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

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Processing-in-Memory (PIM)

- PIM is a computing paradigm that advocates for memorycentric 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

A State-of-the-Art PIM System



- In our work, we use the UPMEM PIM architecture
 - General-purpose processing cores called DRAM Processing Units (DPUs)
 - Up to 24 PIM threads, called *tasklets*
 - <u>32-bit integer arithmetic</u>, but multiplication/division are emulated*, as well as floating-point operations
 - 64-MB DRAM bank (MRAM), 64-KB scratchpad (WRAM)

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



Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

ML Training Workloads



- Memory access patterns
- Operations and datatypes
- Communication/synchronization

Learning	Application	Algorithm	Short name	Memory access pattern			Computation pattern		Communication/synchronization	
approach				Sequential	Strided	Random	Operations	Datatype	Intra PIM Core	Inter PIM Core
Supervised	Regression	Linear Regression	LIN	Yes	No	No	mul, add	float, int32_t	barrier	Yes
	Classification	Logistic Regression	LOG	Yes	No	No	mul, add, exp, div	float, int32_t	barrier	Yes
		Decision Tree	DTR	Yes	No	No	compare, add	float	barrier, mutex	Yes
Unsupervised	Clustering	K-Means	KME	Yes	No	No	mul, compare, add	int16_t, int64_t	barrier, mutex	Yes

Linear Regression

- Linear regression (LIN) is a supervised learning algorithm where the predicted output variable has a linear relation with the input variable
 - We use *gradient descent* as the optimization algorithm to find the minimum of the loss function
- Our PIM implementation divides the training dataset (X) equally among PIM cores
 - PIM threads compute dot products of row vectors and weights
 - Each dot product is compared to the observed value y to compute a partial gradient value
 - Partial gradient values are reduced and sent to the host
- Four versions of LIN:
 - LIN-FP32: training datasets of <u>32-bit real values</u>
 - LIN-INT32: 32-bit fixed-point representation
 - LIN-HYB: hybrid precision (8-bit, 16-bit, 32-bit)
 - LIN-BUI: custom multiplication based on 8-bit built-in multiplication

Logistic Regression

- Logistic regression (LOG) is a supervised learning algorithm used for classification, which outputs probability values for each input observation variable or vector
 - Sigmoid function to map predicted values to probabilities
- Our PIM implementation follows the same workload distribution pattern as our linear regression implementation
- Six versions of LOG:
 - LOG-FP32: training datasets of <u>32-bit real values</u>, Sigmoid approximated with Taylor series
 - LOG-INT32: 32-bit fixed-point representation, Taylor series
 - LOG-INT32-LUT: Sigmoid calculation with a lookup table (LUT)
 - LOG-INT32-LUT(MRAM): LUT in MRAM
 - LOG-INT32-LUT(WRAM): LUT in WRAM
 - LOG-HYB-LUT: hybrid precision (8-bit, 16-bit, 32-bit), LUT in WRAM
 - LOG-BUI-LUT: custom multiplication based on 8-bit built-in multiplication, LUT in WRAM

Decision Tree

- Decision trees (DTR) are tree-based methods used for classification and regression, which partition the feature space into *leaves*, with a simple prediction model in each leaf
- Our PIM implementation partitions the training set among PIM cores, which compute partial *Gini* scores to evaluate the host's *split* decisions
- The host sends commands to the PIM cores:
 - Split commit to split a tree leaf
 - Split evaluate to evaluate a split
 - Min-max to query minimum/maximum values of a feature in a tree leaf
- Data layout in split commit to maximize memory bandwidth with streaming accesses
- This data layout also ensures memory accesses in streaming in split evaluate

Dataset:

5 points, 2 features: p0 = (0, 11); p1 = (8, 4); p2 = (7, 9); p3 = (2, 6); p4 = (5, 2)



K-Means Clustering

- K-means (KME) is an iterative clustering method used to find groups in a dataset which have not been explicitly labeled
- Our PIM implementation distributes the dataset evenly over the PIM cores
- PIM threads evaluate which centroid is the closest one to each point of the training set
 - Counter and accumulator per coordinate (per centroid)
- Then, the host recalculates the centroids
- Convergence to a local optimum when the updated centroid's coordinates are within a threshold (*Frobenius norm*)

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Evaluation Methodology

• Syr thetic and real datasets

d			Synthetic Datasets	Real Dataset		
u	Strong Sc	aling (1 PIM core	256-2048 PIM cores)	Weak Scaling (per PIM core)		
sion	2,048 sam	les, 16 attr. (0.125 I	IB) 6,291,456 samples, 16 attr. (384 MB)	2,048 samples, 16 attr. (0.125 MB)	SUSY [223, 224]	
ssion	2,048 sam	les, 16 attr. (0.125 I	IB) 6,291,456 samples, 16 attr. (384 MB)	2,048 samples, 16 attr. (0.125 MB)	Skin segmentation [225]	
	60,000 san	ples, 16 attr. (3.84 I	IB) 153,600,000 samples, 16 attr. (9830 MB)	600,000 samples, 16 attr. (38.4 MB)	Higgs boson [223, 226]	
	10,000 san	ples, 16 attr. (0.64 I	IB) 25,600,000 samples, 16 attr. (1640 MB)	100,000 samples, 16 attr. (6.4 MB)	Higgs boson [223, 226]	
	ı d sion ssion	Id Strong Sc sion 2,048 sam ssion 2,048 sam 60,000 san 10,000 san	Id Strong Sc ling (1 PIM core sion 2,048 sam les, 16 attr. (0.125 M ssion 2,048 sam les, 16 attr. (0.125 M 60,000 san ples, 16 attr. (3.84 M 10,000 san ples, 16 attr. (0.64 M	Id Synthetic Datasets sion 2,048 sam les, 16 attr. (0.125 N B) 6,291,456 samples, 16 attr. (384 MB) ssion 2,048 sam les, 16 attr. (0.125 N B) 6,291,456 samples, 16 attr. (384 MB) 60,000 san ples, 16 attr. (3.84 N B) 153,600,000 samples, 16 attr. (9830 MB) 10,000 san ples, 16 attr. (0.64 N B) 25,600,000 samples, 16 attr. (1640 MB)	Ind Strong Sc Stro	





- Quality metrics
- Performance of PIM kernels
- Performance scaling
- Comparison to CPU and GPU

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



Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Evaluation: Quality Metrics

- Linear regression
 - Training error rate of LIN-FP32 is the same as the CPU version
 - For integer versions, it remains low and close to that of LIN-FP32
- Logistic regression
 - LUT-based versions obtain lower training error rates that LOG-INT32, since they use exact values, not approximations
- Decision tree
 - Training accuracy only slightly lower than that of the CPU version
- K-means clustering
 - Same Calinski-Harabasz score and adjusted Rand index of PIM and CPU versions

We maintain the accuracy of all workloads (or keep it close to the CPU baseline)

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Evaluation: Analysis of PIM Kernels (I)

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 **fixedpoint representation**, without much accuracy loss, if PIM cores do not support FP.

Evaluation: Analysis of PIM Kernels (II)

Linear regression LIN-HYB is 41% faster than LIN-INT32

LIN-BUI provides an additional 25% speedup

Recommendation 2. Quantization can take advantage of native hardware support. Hybrid precision can significantly improve performance.



Recommendation 3. Programmers/better compilers can optimize code by leveraging native instructions (e.g., 8-bit integer multiplication in UPMEM).

Evaluation: Analysis of PIM Kernels (III)

Logistic regression

Very high kernel time of LOG–FP32 and LOG– INT32 due to Sigmoid approximation

LOG-INT32-LUT(MRAM) is 53x faster than LOG-INT32

Recommendation 4. Convert computation to memory accesses by keeping pre-calculated operation results (e.g., LUTs, memoization) in memory.



LOG–HYB–LUT is 28% faster than LOG–INT32–LUT

LOG-BUI-LUT provides an additional 43% speedup

Evaluation: Analysis of PIM Kernels (IV)

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



KNAE

Evaluation: Analysis of PIM Kernels (V)

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

Key Takeaway 2. ML workloads that are memorybound 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**.



(b) KME

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Evaluation: Performance Scaling (I)

• Strong scaling: 256 to 2,048 PIM cores



PIM kernel time scales linearly with the number of PIM cores

Little overhead from inter PIM core communication and communication between host and PIM cores

Evaluation: Performance Scaling (II)

• Strong scaling: 256 to 2,048 PIM cores

Key Takeaway 3. ML training workloads, which need large training datasets, benefit from large PIMenabled memory with many PIM cores.



Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

Analysis of PIM Kernels

Performance Scaling

Comparison to CPU and GPU (I)

Linear regression and logistic regression

PIM versions are heavily burdened when they use operations that are not natively supported by the hardware

SAFARI



Execution Time (ms)

Several optimizations reduce the execution time considerably (LIN/LOG up to 10x/3.9x faster than CPU) and close the gap with GPU performance (LIN/LOG still 4x/16x slower than GPU)

1 1 1 LOG-INT32-LUT LOG-INT32-LUT LOG-FP32 LOG-INT32 (MRAM) (WRAM) 100000 100000 100000 100000 ☑ GPU-CPU 10000 10000 10000 10000 CPU-GPU GPU Kernel 1000 1000 1000 1000 100 100 100 100 10 10 10 10 1 1 1 LOG-HYB-LUT LOG-BUI-LUT CPU GPU (WRAM) (WRAM)

T . T N

Comparison to CPU and GPU (II)

• 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

Long arXiv Version

- Additional implementation details
- More evaluation results
- Extended observations, takeaways, and recommendations

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

<u>https://arxiv.org/pdf/2207.07886.pdf</u> <u>Source code: https://github.com/CMU-SAFARI/pim-ml</u>

Comparison to CPU and GPU (III)

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



PIM version of DTR 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

Comparison to CPU and GPU (IV)

• 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.

faster than the CPU version and **4.5x** faster than the GPU version faster than the CPU version and **3.2x** faster than the GPU version

Long arXiv Version

- Additional implementation details
- More evaluation results
- Extended observations, takeaways, and recommendations

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

<u>https://arxiv.org/pdf/2207.07886.pdf</u> <u>Source code: https://github.com/CMU-SAFARI/pim-ml</u>

Source Code

• <u>https://github.com/</u> <u>CMU-SAFARI/pim-ml</u>

CMU-SAFARI/pim-ml Pul	blic	☆ Edit Pins → ③ Unwatch 2 →				
> Code 📀 Issues 🏗 Pull requests 🕑 Actions 🖽 Projects 🙂 Security 🗠 Insights 🕸 S						
ះ main - ្រ 1 branch 🕤 🕻	tags	Go to file Add file - <> Code -				
🐒 el1goluj readme		7d7289d 2 days ago 🕤 16 commits				
Linear_Regression	upload regression code	2 days ago				
Logistic_Regression	upload regression code	2 days ago				
dpu_kmeans @ 7f28518	submodules	2 days ago				
🕤 scikit-dpu @ 1ddeb5d	submodules	2 days ago				
🗅 .gitmodules	submodules	2 days ago				
	readme	2 days ago				
🗅 README.md	readme	2 days ago				

i README.md

PIM-ML

PIM-ML is a benchmark for training machine learning algorithms on the UPMEM architecture, which is the first publicly-available real-world processing-in-memory (PIM) architecture. The UPMEM architecture integrates DRAM memory banks and general-purpose in-order cores, called DRAM Processing Units (DPUs), in the same chip.

PIM-ML is designed to understand the potential of modern general-purpose PIM architectures to accelerate machine learning training. PIM-ML implements several representative classic machine learning algorithms:

- Linear Regression
- Logistic Regression
- Decision Tree
- K-means Clustering

SAFARI

0

Executive Summary

- Training machine learning (ML) algorithms is a computationally expensive process, frequently memory-bound due to repeatedly accessing large training datasets
- Memory-centric computing systems, i.e., with Processing-in-Memory (PIM) capabilities, can alleviate this data movement bottleneck
- Real-world PIM systems have only recently been manufactured and commercialized
 - UPMEM has designed and fabricated the first publicly-available real-world PIM architecture
- Our goal is to understand the potential of modern general-purpose PIM architectures to accelerate machine learning training
- Our main contributions:
 - PIM implementation of several classic machine learning algorithms: linear regression, logistic regression, decision tree, K-means clustering
 - Workload characterization in terms of quality, performance, and scaling
 - Comparison to their counterpart implementations on processor-centric systems (CPU and GPU)
 - PIM version of DTR is 27x / 1.34x faster than the CPU / GPU version, respectively
 - PIM version of KME is 2.8x / 3.2x faster than the CPU / GPU version, respectively
 - Source code: <u>https://github.com/CMU-SAFARI/pim-ml</u>
- Experimental evaluation on a real-world PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM memory
- Key observations, takeaways, and recommendations for ML workloads on general-purpose PIM systems

Real PIM Tutorial (ISCA 2023)

• June 18th: Lectures + Hands-on labs + Invited lectures



Trace: • start

ISCA 2023 Real-World PIM Tutorial



Real-world Processing-in-Memory Systems for Modern Workloads

Tutorial Description

Processing-in-Memory (PIM) is a computing paradigm that aims at overcoming the data movement bottleneck (i.e., the waste of execution cycles and energy resulting from the back-and-forth data movement between memory units and compute units) by making memory compute-capable.

Explored over several decades since the 1960s, PIM systems are becoming a reality with the advent of the first commercial products and prototypes.

A number of startups (e.g., UPMEM, Neuroblade) are already commercializing real PIM hardware, each with its own design approach and target applications. Several major vendors (e.g., Samsung, SK Hynix, Alibaba) have presented real PIM chip prototypes in the last two years. Most of these architectures have in common that they place compute units near the memory arrays. This type of PIM is called processing near memory (PNM).

2,560-DPU Processing-in-Memory System



PIM can provide large improvements in both performance and energy consumption for many modern applications, thereby enabling a commercially viable way of dealing with huge amounts of data that is bottlenecking our computing systems. Yet, it is critical to (1) study and understand the characteristics that make a workload suitable for a PIM architecture, (2) propose optimization strategies for PIM kernels, and (3) develop programming frameworks and tools that can lower the learning curve and ease the adoption of PIM.

This tutorial focuses on the latest advances in PIM technology, workload characterization for PIM, and programming and optimizing PIM kernels. We will (1) provide an introduction to PIM and taxonomy of PIM systems, (2) give an overview and a rigorous analysis of existing real-world PIM hardware, (3) conduct hand-on labs about important workloads (machine learning, sparse linear algebra, bioinformatics, etc.) using real PIM systems,

and (4) shed light on how to improve future PIM systems for such workloads.

Table of Contents

Real-world Processing-in-Memory Systems for Modern Workloads

start

- Tutorial Description
- Organizers
- Agenda (June 18, 2023)
- Lectures (tentative)
- Hands-on Labs (tentative)
- Learning Materials

SAFARI

https://events.safari.ethz.ch/isca-pim-tutorial/doku.php?id=start

2023 IEEE International Symposium on Performance Analysis of Systems and Software

Evaluating Machine Learning Workloads on Memory-Centric Computing Systems

<u>Juan Gómez Luna</u>, Yuxin Guo, Sylvan Brocard, Julien Legriel, Remy Cimadomo, Geraldo F. Oliveira, Gagandeep Singh, Onur Mutlu

> https://arxiv.org/pdf/2207.07886.pdf https://github.com/CMU-SAFARI/pim-ml

juang@ethz.ch





Monday, April 24, 2023

