Machine Learning Training
on a Real Processing-in-Memory System

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

https://arxiv.org/pdf/2206.06022.pdf
juang@ethz.ch

ETH Zürich SAFARI up mem

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 accuracy, performance, and scaling
  - Comparison to their counterpart implementations on processor-centric systems (CPU and GPU)
- Experimental evaluation on a real-world PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM memory
- New observations and insights:
  - ML training in PIM systems benefits from (1) fixed-point representation, (2) quantization, and (3) hybrid precision implementations
  - Complex activation functions (e.g., sigmoid) can take advantage of LUTs in PIM systems without native support for those activation functions
  - Data can be placed and laid out for PIM cores to access nearby memory banks in streaming, thus maximizing PIM memory bandwidth
  - ML training benefits from scaling the size of PIM-enabled memory with PIM cores attached to memory banks

Outline

Machine learning workloads

Processing-in-memory

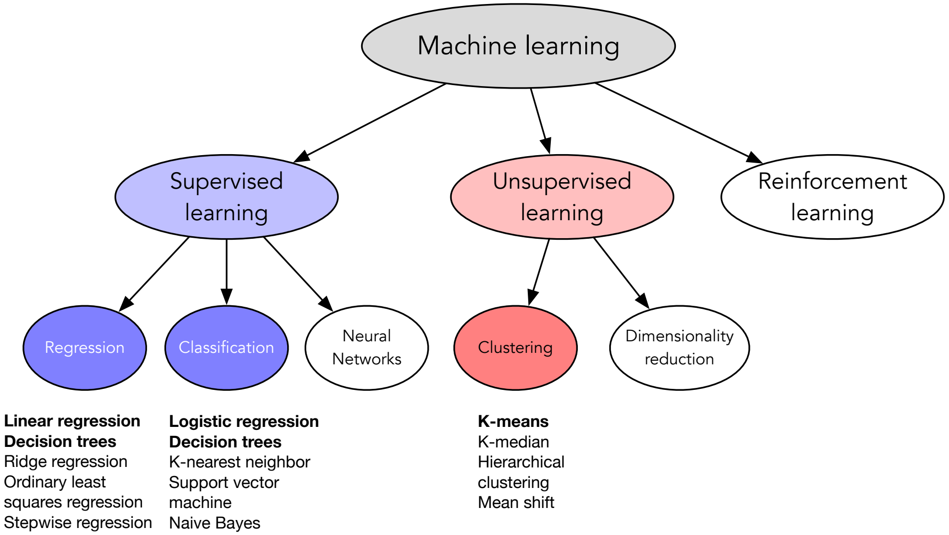
PIM implementation of ML workloads

Evaluation

Key observations and insights

Machine Learning Workloads

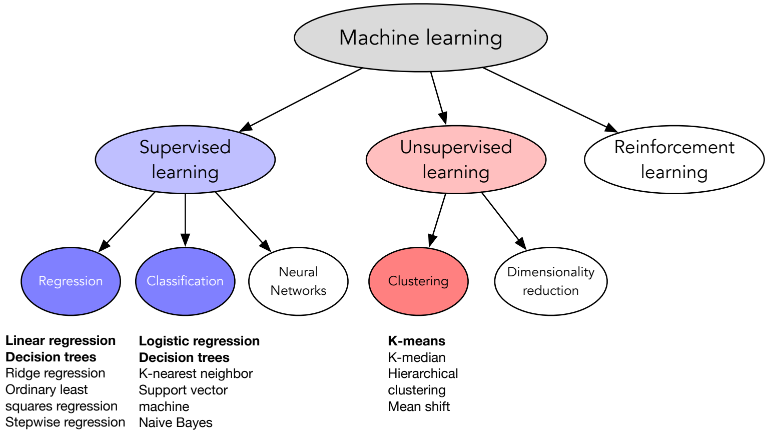
- Machine learning training with large amounts of data is a computationally expensive process, which requires many iterations to update an ML model's parameters



- 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



Machine learning

Supervised learning

Unsupervised learning

Reinforcement learning

Regression

Classification

Neural Networks

Clustering

Dimensionality reduction

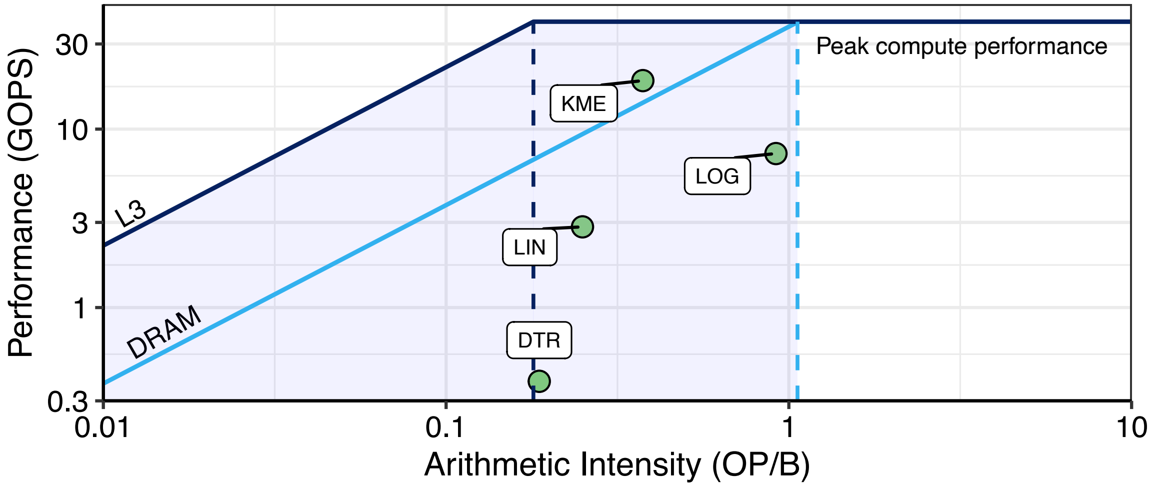
Linear regression
Decision trees
Ridge regression
Ordinary least squares regression
Stepwise regression

Logistic regression
Decision trees
K-nearest neighbor
Support vector machine
Naive Bayes

K-means
K-median
Hierarchical clustering
Mean shift

- 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

Outline

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

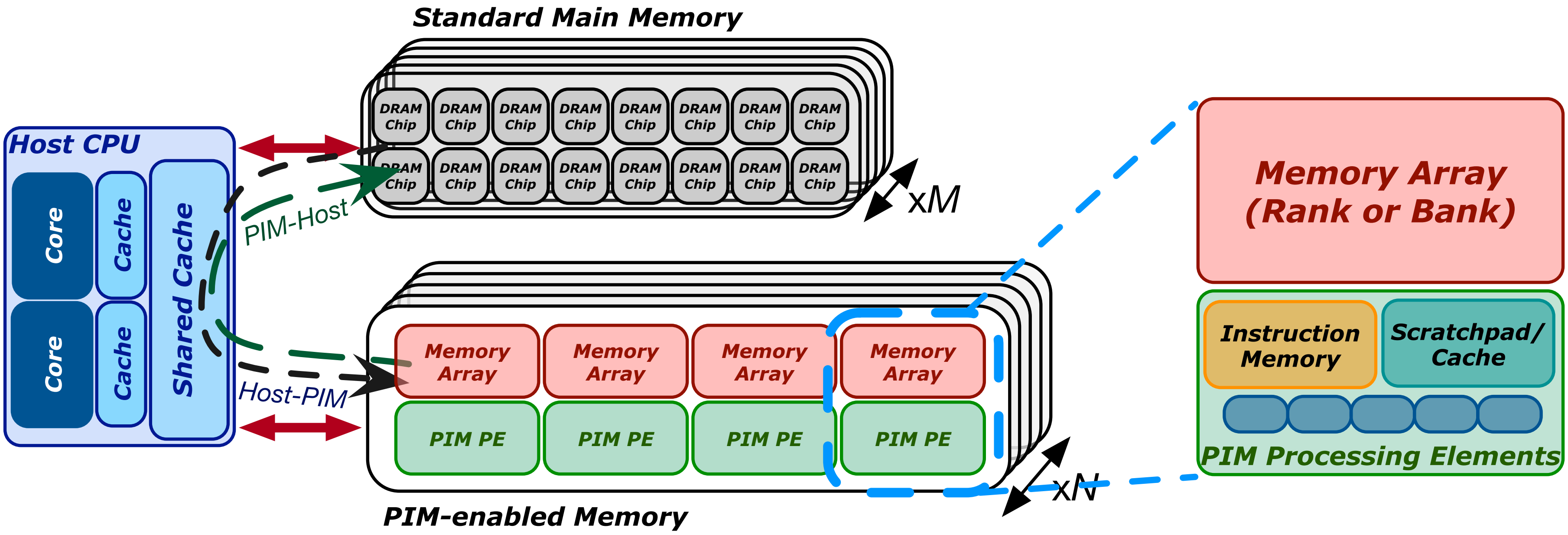
Evaluation

Key observations and insights

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. 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*
    - 32-bit integer arithmetic, but multiplication/division are emulated, as well as floating-point operations
  - 64-MB DRAM bank (*MRAM*), 64-KB scratchpad (*WRAM*)

Outline

Machine learning workloads

Processing-in-memory

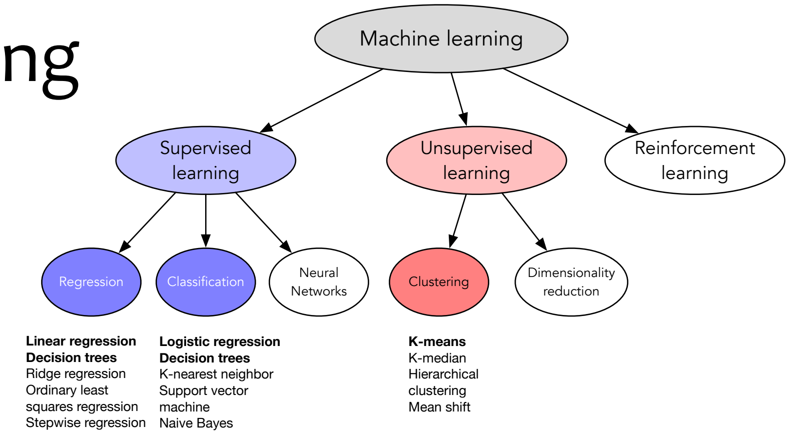
PIM implementation of ML workloads

Evaluation

Key observations and insights

ML Training Workloads

- Four widely-used machine learning workloads:
  - Linear regression (`LIN`)
  - Logistic regression (`LOG`)
  - Decision tree (`DTR`)
  - K-means clustering (`KME`)



Linear regression
Decision trees
Ridge regression
Ordinary least squares regression
Stepwise regression

Logistic regression
Decision trees
K-nearest neighbor
Support vector machine
Naive Bayes

K-means
K-median
Hierarchical clustering
Mean shift

- Diversity of our ML training workloads:
  - Memory access patterns
  - Operations and datatypes
  - Communication/synchronization

Learning approach	Application	Algorithm	Short name	Memory access pattern			Computation pattern		Communication/synchronization	
				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 32-bit real values
  - `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

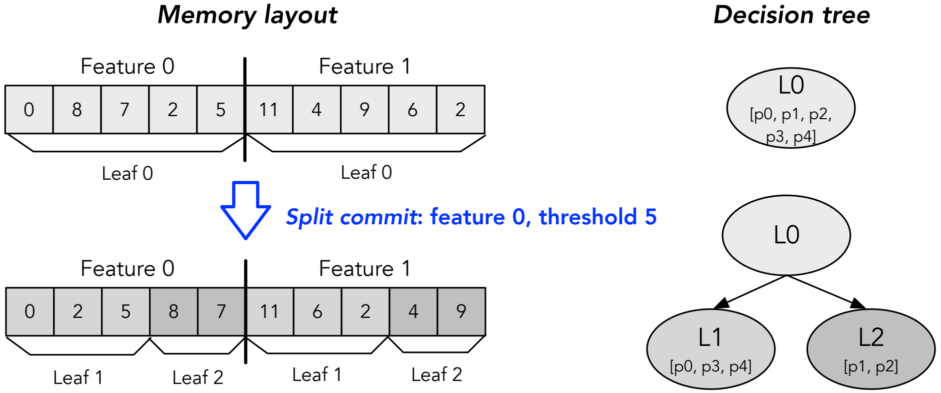
Logistic Regression

- 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 32-bit real values, 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 boxes, with a simple prediction model in each box
- Our PIM implementation partitions the training set among PIM cores, which compute partial *Gini* scores to evaluate *split* decisions done by the host
- 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 the minimum and maximum values of a feature in a tree leaf
- PIM threads work on different batches of feature values, compare them to a threshold, and update the partial Gini score
- Data layout in split commit to maximize memory bandwidth with streaming accesses

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*)

Outline

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Key observations and insights

Evaluation Methodology

- Synthetic and real datasets

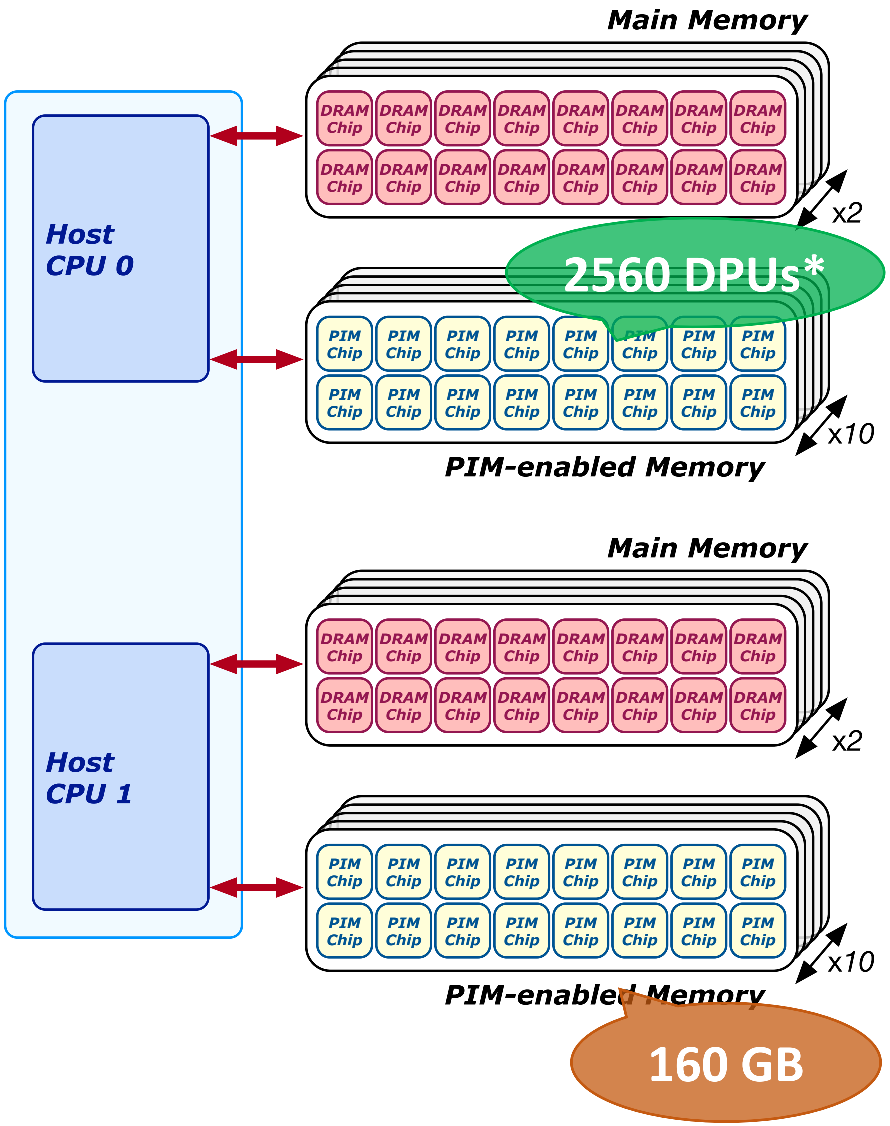
ML Workload	**Synthetic Datasets**		**Real Dataset**
	Strong Scaling (1 PIM core | 256-2048 PIM cores)	**Weak Scaling (per PIM core)**	
Linear regression	2,048 samples, 16 attr. (0.125 MB) | 6,291,456 samples, 16 attr. (384 MB)	2,048 samples, 16 attr. (0.125 MB)	SUSY [223, 224]
Logistic regression	2,048 samples, 16 attr. (0.125 MB) | 6,291,456 samples, 16 attr. (384 MB)	2,048 samples, 16 attr. (0.125 MB)	Skin segmentation [225]
Decision tree	60,000 samples, 16 attr. (3.84 MB) | 153,600,000 samples, 16 attr. (9830 MB)	600,000 samples, 16 attr. (38.4 MB)	Higgs boson [223, 226]
K-Means	10,000 samples, 16 attr. (0.64 MB) | 25,600,000 samples, 16 attr. (1640 MB)	100,000 samples, 16 attr. (6.4 MB)	Higgs boson [223, 226]

- Evaluated systems
  - UPMEM PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM
  - Intel Xeon Silver 4215 CPU (16 hardware threads)
  - NVIDIA A100 GPU

- We evaluate:
  - Metrics
  - Performance of PIM kernels
  - Performance scaling
  - Comparison to CPU and GPU

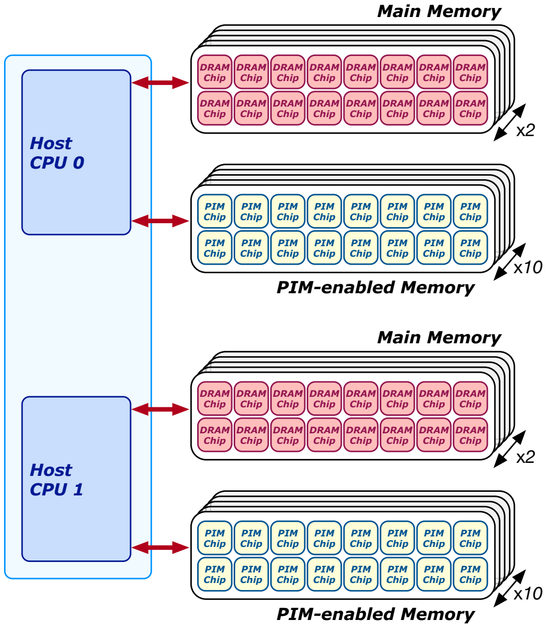
2,560-DPU System (I)

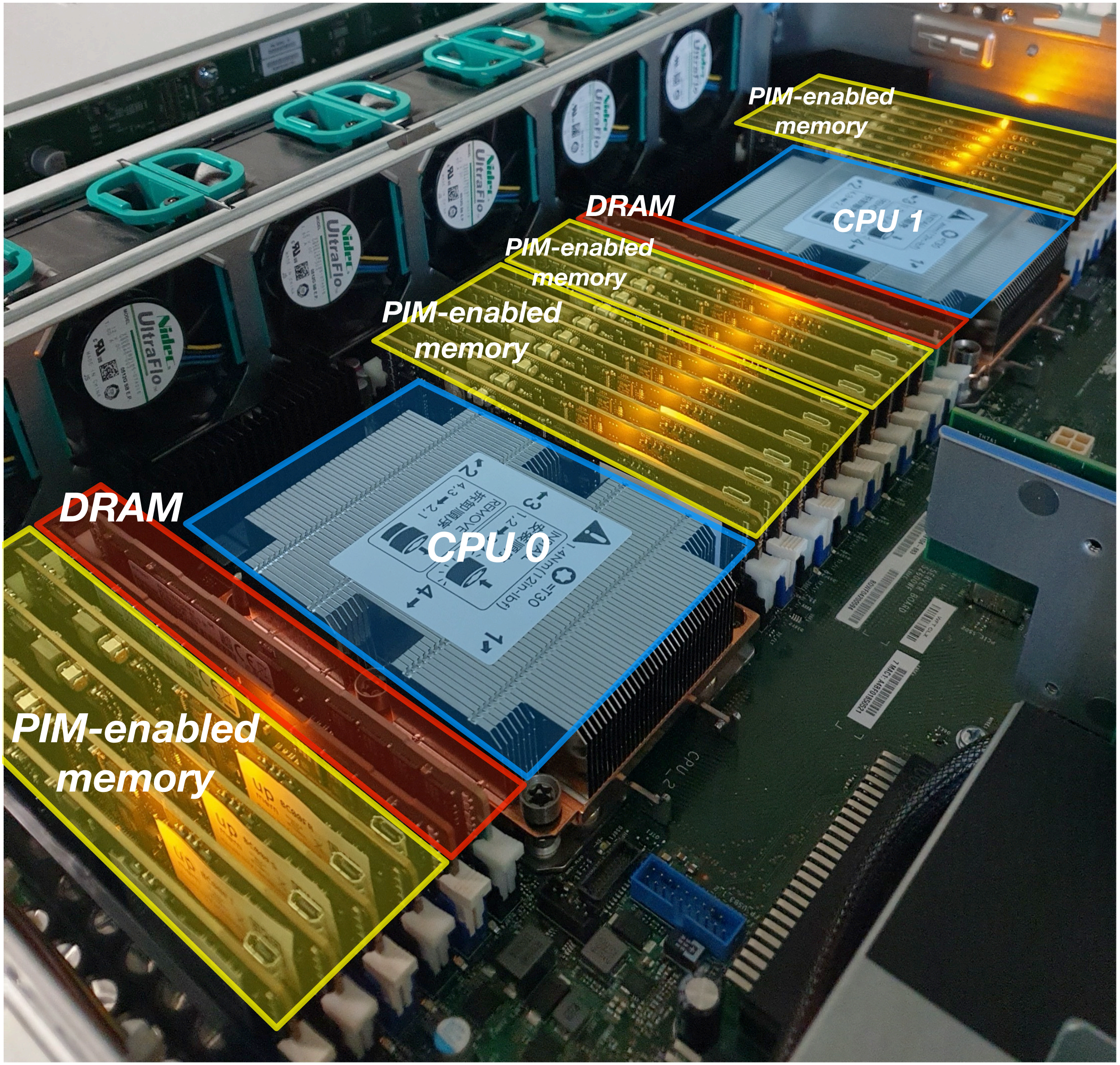
- UPMEM-based PIM system with 20 UPMEM DIMMs of 16 chips each (40 ranks)
  - P21 DIMMs
  - Dual x86 socket
    - UPMEM DIMMs coexist with regular DDR4 DIMMs
    - 2 memory controllers/socket (3 channels each)
    - 2 conventional DDR4 DIMMs on one channel of one controller



* 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

2,560-DPU System (II)



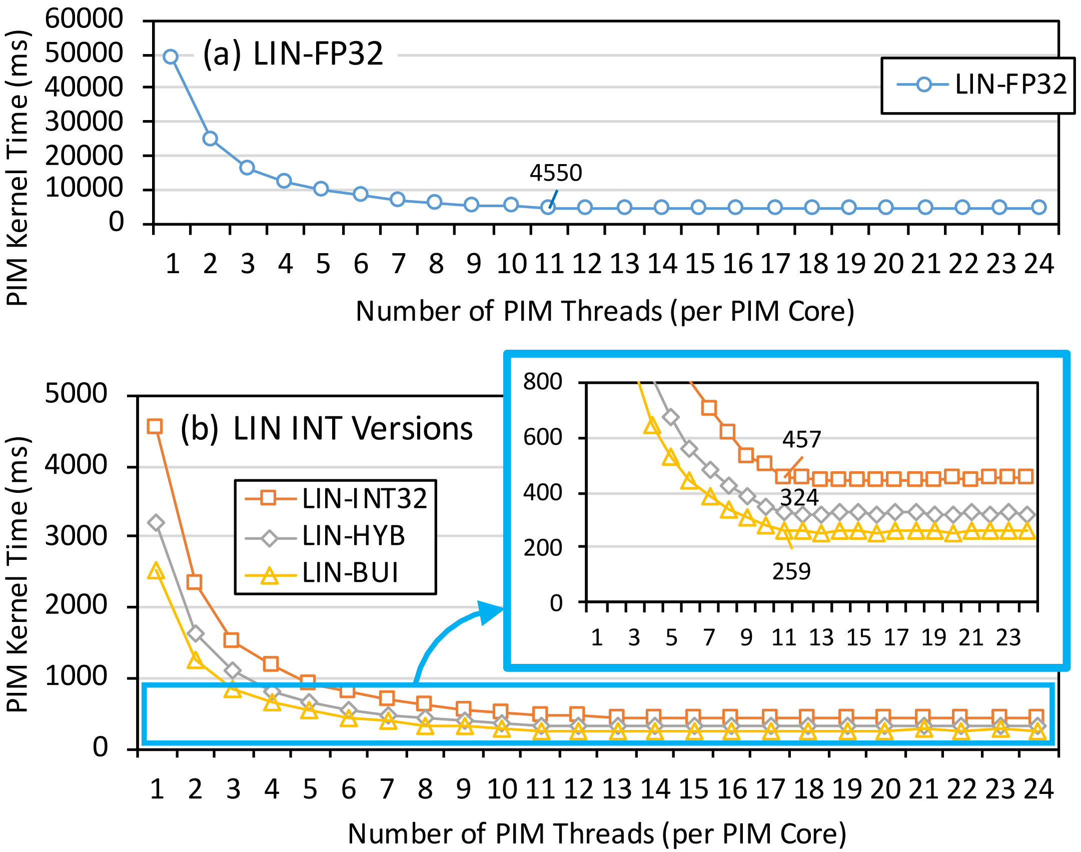


Evaluation: Metrics

- 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
  - Same *Calinski-Harabasz score* and *adjusted Rand index* of PIM and CPU versions

Evaluation: Analysis of PIM Kernels (I)

- Linear regression



All versions saturate at 11 or more PIM threads

Fixed point accelerates the kernel by an order of magnitude

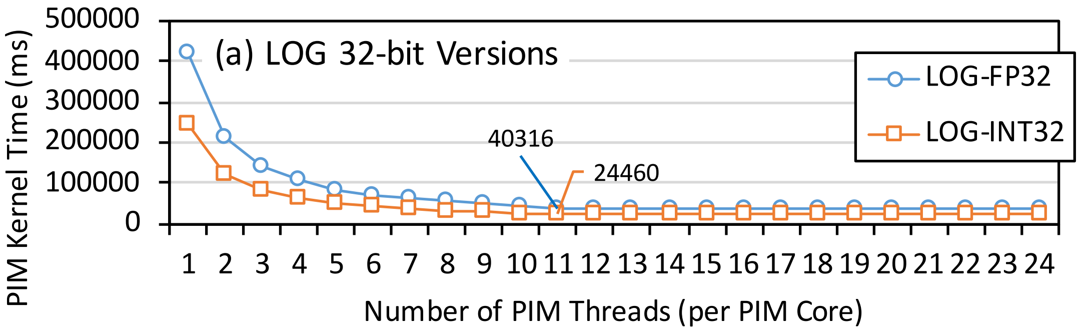
`LIN-HYB` is 41% faster than `LIN-INT32`

`LIN-BUI` provides an additional 25% speedup

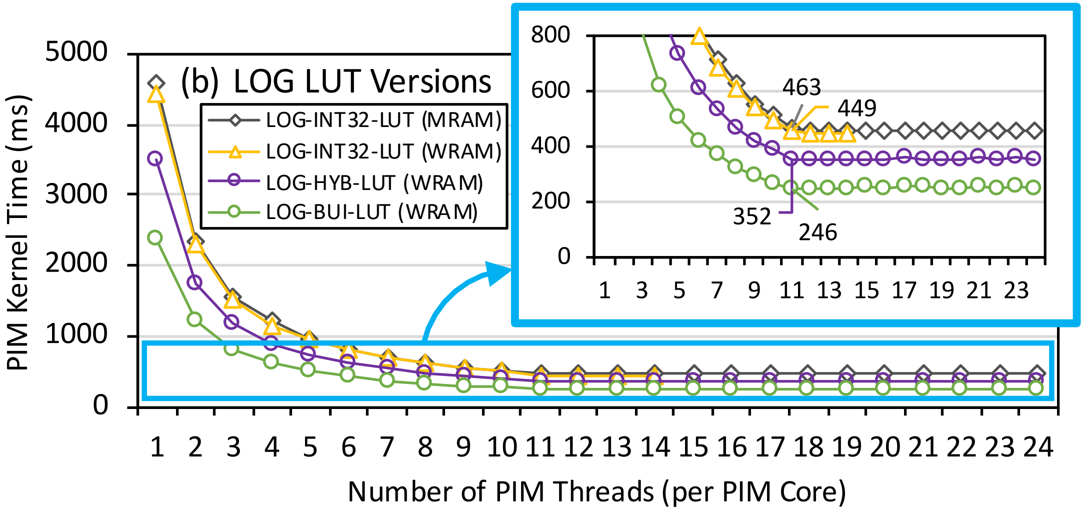
Evaluation: Analysis of PIM Kernels (II)

- 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`



`LOG-HYB-LUT` is 28% faster than `LOG-INT32-LUT`

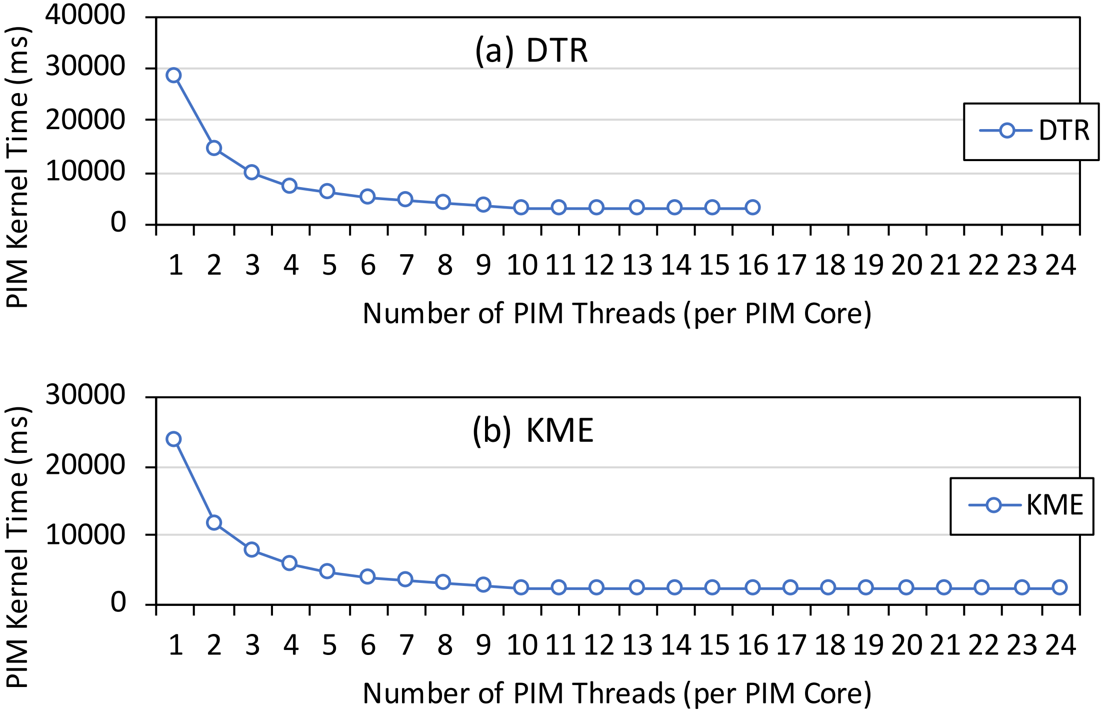
`LOG-BUI-LUT` provides an additional 43% speedup

Evaluation: Analysis of PIM Kernels (III)

- Decision tree & K-means

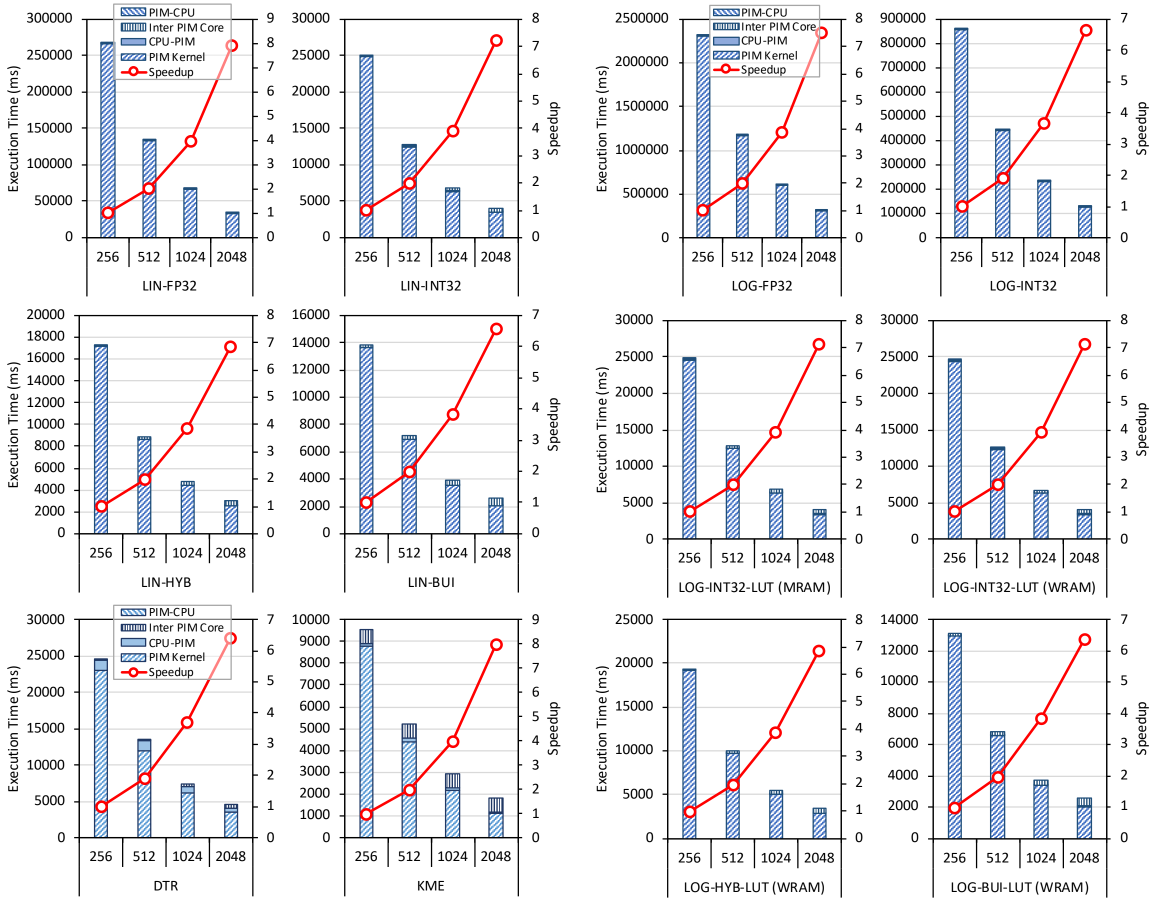
Both workloads saturate at 11 or more PIM threads

Maximum number of PIM threads in DTR is 16 due to the usage of local scratchpad memory



Evaluation: Performance Scaling

- Strong scaling: 256 to 2,048 PIM cores

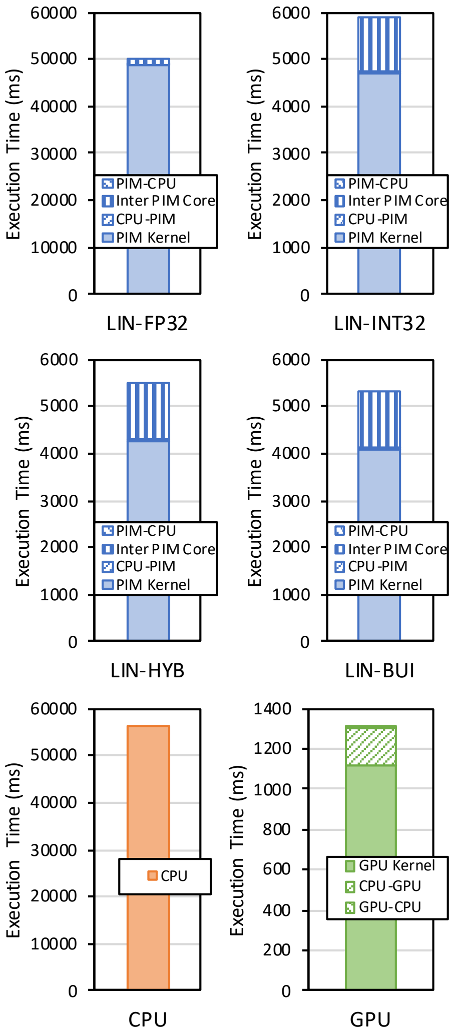


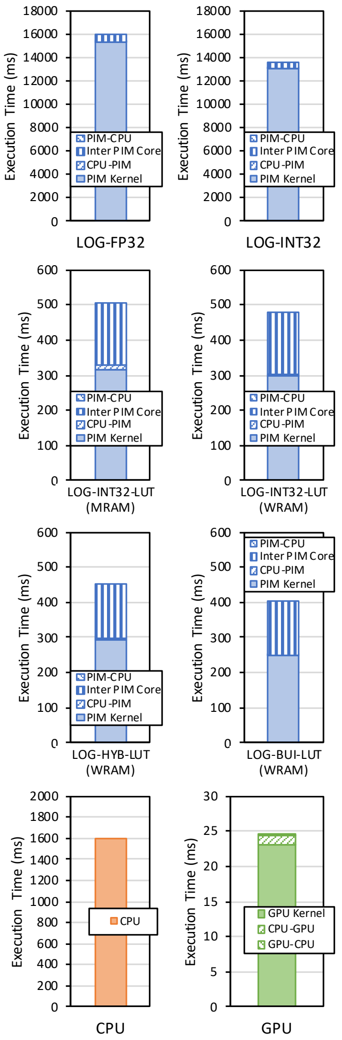
PIM kernel time scales linearly with the number of PIM cores

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

Comparison to CPU and GPU (I)

- Linear regression and logistic regression



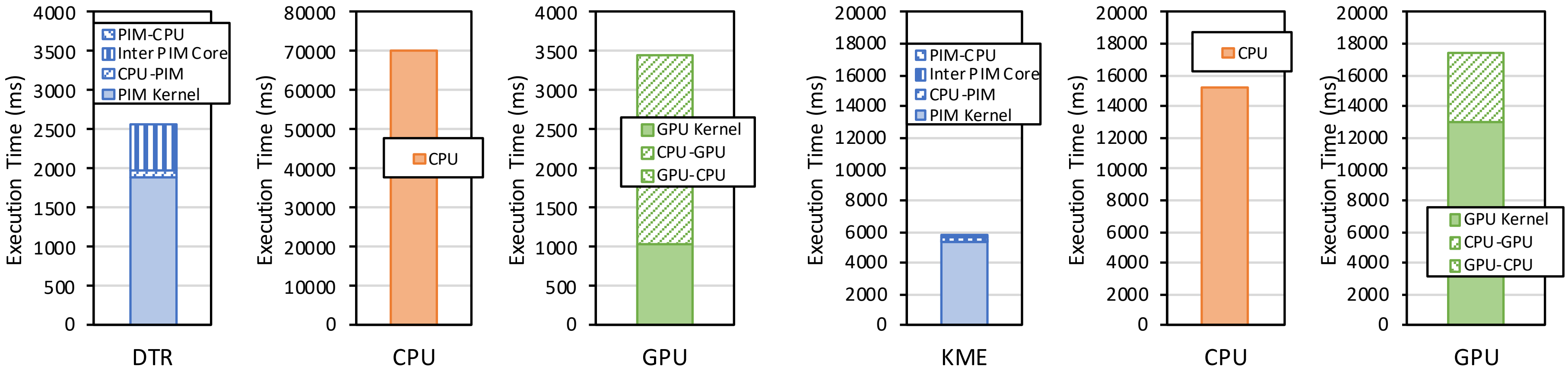


PIM versions are heavily burdened when they use operations that are not natively supported by the hardware

Several optimizations reduce the execution time considerably and close the gap with GPU performance

Comparison to CPU and GPU (II)

- Decision tree and K-means



(a) Decision Tree

(b) K-means

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

Outline

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Key observations and insights

Key Observations and Insights

- ML training workloads can greatly benefit from (1) fixed-point data representation, (2) quantization, and (3) hybrid precision implementation in PIM systems
- ML training workloads that require complex activation functions (e.g., sigmoid) can take advantage of lookup tables (LUTs) in PIM systems instead of function approximation
- Data can be placed and laid out such that memory accesses of PIM cores are streaming
- ML training workloads with large training datasets benefit from scaling the size of PIM-enabled memory with PIM cores attached to memory arrays

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 accuracy, performance, and scaling
  - Comparison to their counterpart implementations on processor-centric systems (CPU and GPU)
- Experimental evaluation on a real-world PIM system with 2,524 PIM cores @ 425 MHz and 158 GB of DRAM memory
- New observations and insights:
  - ML training in PIM systems benefits from (1) fixed-point representation, (2) quantization, and (3) hybrid precision implementations
  - Complex activation functions (e.g., sigmoid) can take advantage of LUTs in PIM systems without native support for those activation functions
  - Data can be placed and laid out for PIM cores to access nearby memory banks in streaming, thus maximizing PIM memory bandwidth
  - ML training benefits from scaling the size of PIM-enabled memory with PIM cores attached to memory banks

ML Training on a Real PIM System

Machine Learning Training on a Real Processing-in-Memory System

Juan Gómez-Luna[1] Yuxin Guo[1] Sylvan Brocard[2] Julien Legriel[2]
Remy Cimadomo[2] Geraldo F. Oliveira[1] Gagandeep Singh[1] Onur Mutlu[1]

[1]ETH Zürich [2]UPMEM

https://arxiv.org/pdf/2206.06022.pdf

Analysis of Real PIM Hardware

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://arxiv.org/pdf/2110.01709.pdf

https://doi.org/10.1109/IGSC54211.2021.9651614

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

Understanding a Modern PIM Architecture

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

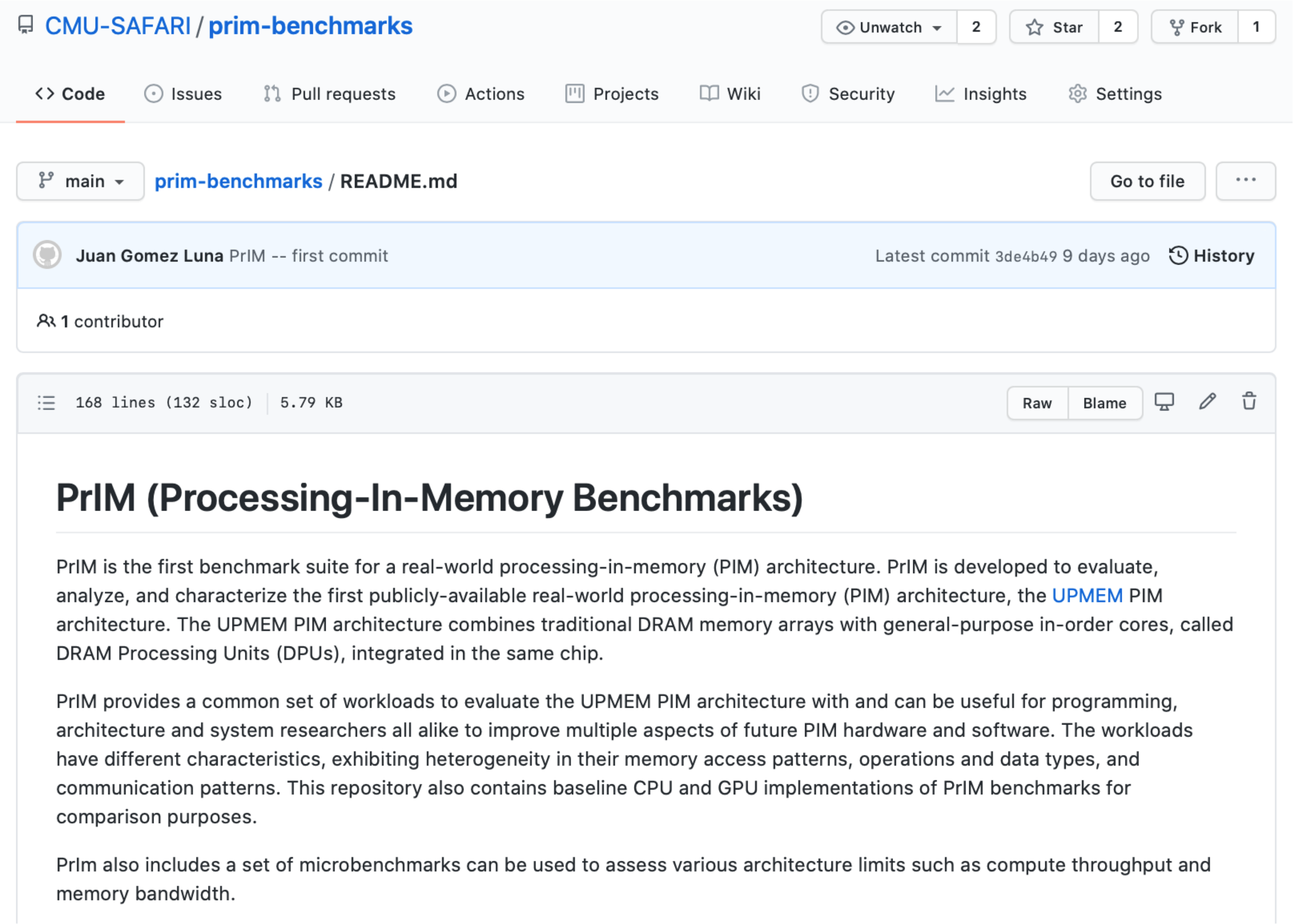
https://arxiv.org/pdf/2105.03814.pdf

https://doi.org/10.1109/ACCESS.2022.3174101

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

PrIM Repository

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



CMU-SAFARI / prim-benchmarks
Unwatch 2
Star 2
Fork 1
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
prim-benchmarks / README.md
Go to file
Juan Gomez Luna PrIM -- first commit
Latest commit 3de4b49 9 days ago
History
1 contributor
168 lines (132 sloc) | 5.79 KB
Raw
Blame

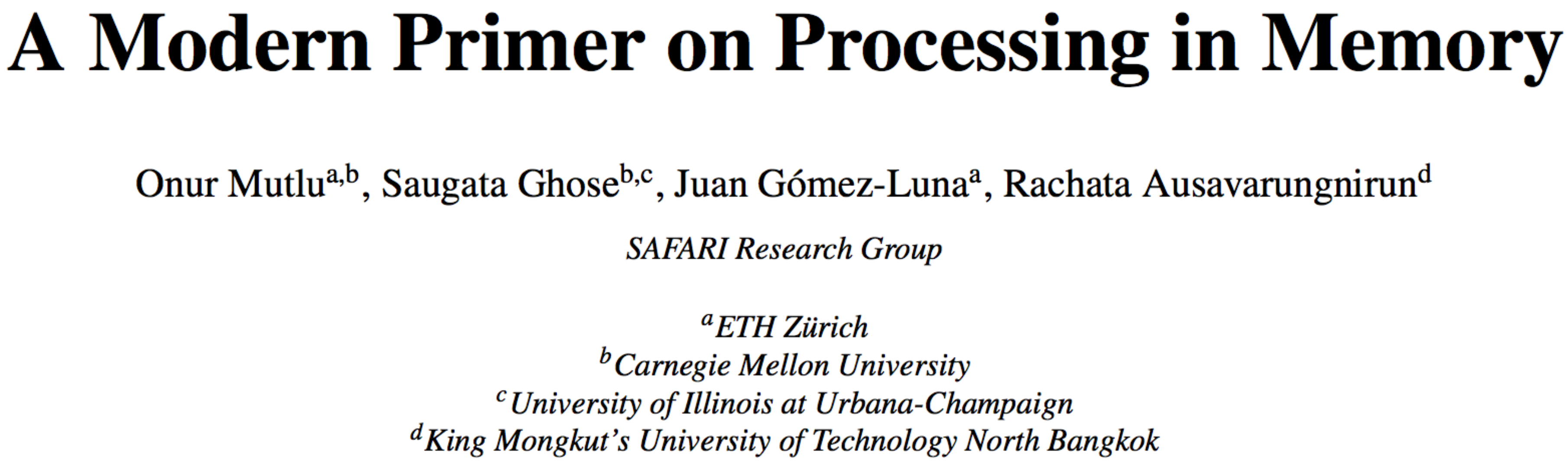
PrIM (Processing-In-Memory Benchmarks)

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.

Prlm also includes a set of microbenchmarks can be used to assess various architecture limits such as compute throughput and memory bandwidth.

PIM Review and Open Problems



A Modern Primer on Processing in Memory

Onur Mutlu[a,b], Saugata Ghose[b,c], Juan Gómez-Luna[a], Rachata Ausavarungnirun[d]

SAFARI Research Group

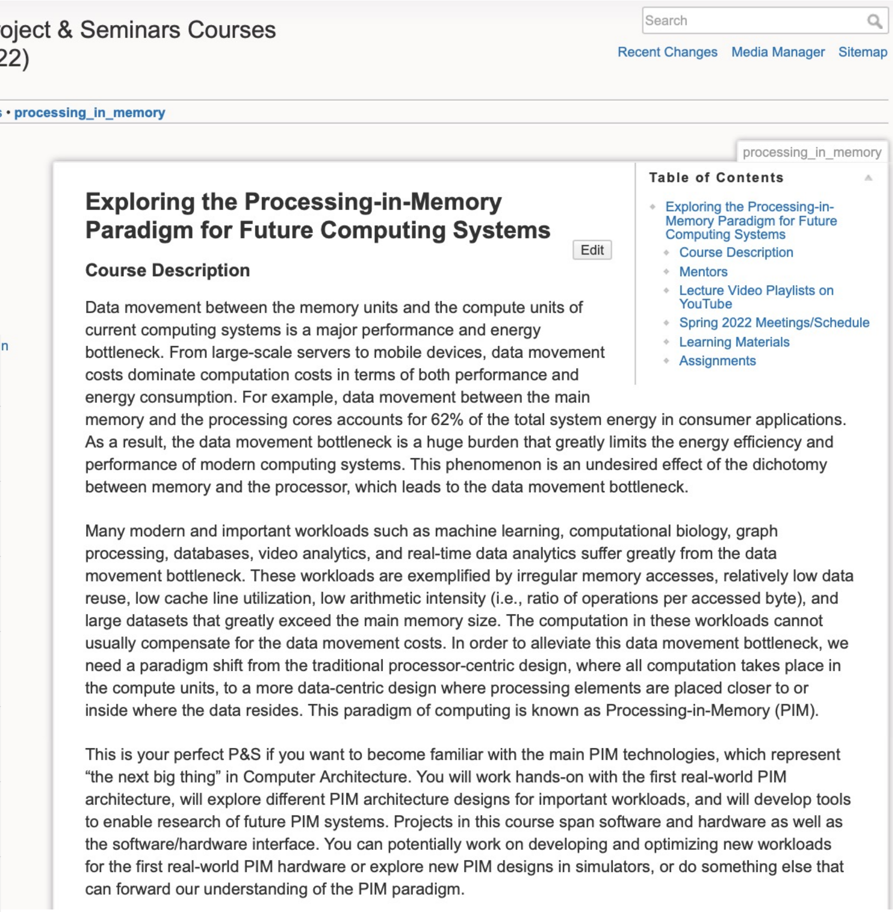
[a]ETH Zürich
[b]Carnegie Mellon University
[c]University of Illinois at Urbana-Champaign
[d]King Mongkut's University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann***, Springer, to be published in 2021.

https://arxiv.org/pdf/1903.03988.pdf

Processing-in-Memory Course (Spring 2022)

- Short weekly lectures
- Hands-on projects



SAFARI Project & Seminars Courses (Spring 2022)

Trace: • heterogeneous_systems • **processing_in_memory**

Home

Courses

- SoftMC
- Ramulator
- Accelerating Genomics
- Mobile Genomics
- **Processing-in-Memory**
- Heterogeneous Systems
- Modern SSDs

Table of Contents

- Exploring the Processing-in-Memory Paradigm for Future Computing Systems
  - Course Description
  - Mentors
  - Lecture Video Playlists on YouTube
  - Spring 2022 Meetings/Schedule
  - Learning Materials
  - Assignments

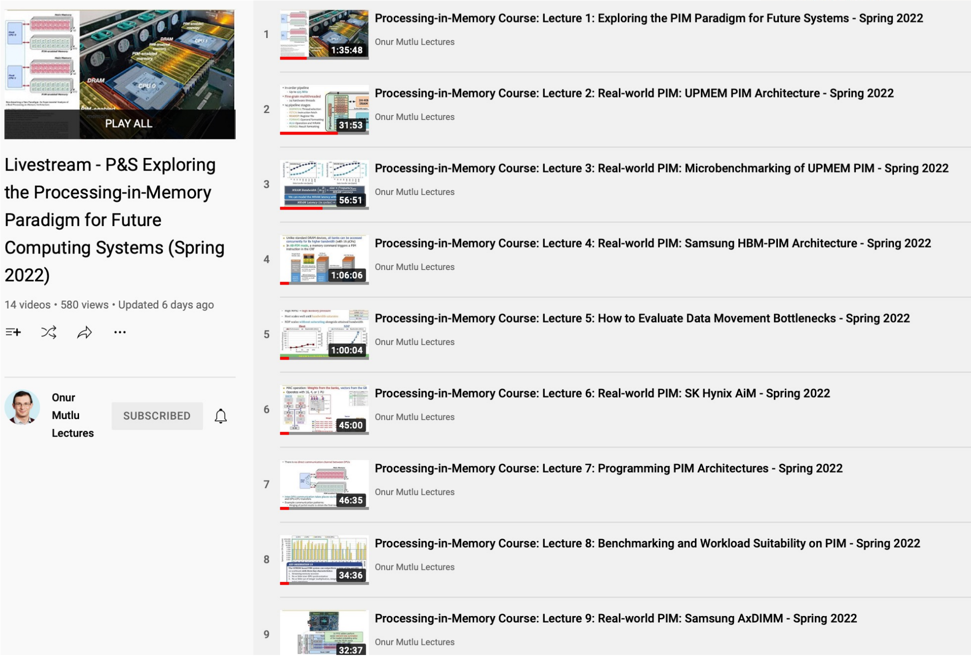
Exploring the Processing-in-Memory Paradigm for Future Computing Systems

Course Description

Data movement between the memory units and the compute units of current computing systems is a major performance and energy bottleneck. From large-scale servers to mobile devices, data movement costs dominate computation costs in terms of both performance and energy consumption. For example, data movement between the main memory and the processing cores accounts for 62% of the total system energy in consumer applications. As a result, the data movement bottleneck is a huge burden that greatly limits the energy efficiency and performance of modern computing systems. This phenomenon is an undesired effect of the dichotomy between memory and the processor, which leads to the data movement bottleneck.

Many modern and important workloads such as machine learning, computational biology, graph processing, databases, video analytics, and real-time data analytics suffer greatly from the data movement bottleneck. These workloads are exemplified by irregular memory accesses, relatively low data reuse, low cache line utilization, low arithmetic intensity (i.e., ratio of operations per accessed byte), and large datasets that greatly exceed the main memory size. The computation in these workloads cannot usually compensate for the data movement costs. In order to alleviate this data movement bottleneck, we need a paradigm shift from the traditional processor-centric design, where all computation takes place in the compute units, to a more data-centric design where processing elements are placed closer to or inside where the data resides. This paradigm of computing is known as Processing-in-Memory (PIM).

This is your perfect P&S if you want to become familiar with the main PIM technologies, which represent “the next big thing” in Computer Architecture. You will work hands-on with the first real-world PIM architecture, will explore different PIM architecture designs for important workloads, and will develop tools to enable research of future PIM systems. Projects in this course span software and hardware as well as the software/hardware interface. You can potentially work on developing and optimizing new workloads for the first real-world PIM hardware or explore new PIM designs in simulators, or do something else that can forward our understanding of the PIM paradigm.



Livestream - P&S Exploring the Processing-in-Memory Paradigm for Future Computing Systems (Spring 2022)

14 videos • 580 views • Updated 6 days ago

Onur Mutlu Lectures

1 Processing-in-Memory Course: Lecture 1: Exploring the PIM Paradigm for Future Systems - Spring 2022 (1:35:48)
Onur Mutlu Lectures

2 Processing-in-Memory Course: Lecture 2: Real-world PIM: UPMEM PIM Architecture - Spring 2022 (31:53)
Onur Mutlu Lectures

3 Processing-in-Memory Course: Lecture 3: Real-world PIM: Microbenchmarking of UPMEM PIM - Spring 2022 (56:51)
Onur Mutlu Lectures

4 Processing-in-Memory Course: Lecture 4: Real-world PIM: Samsung HBM-PIM Architecture - Spring 2022 (1:06:06)
Onur Mutlu Lectures

5 Processing-in-Memory Course: Lecture 5: How to Evaluate Data Movement Bottlenecks - Spring 2022 (1:00:04)
Onur Mutlu Lectures

6 Processing-in-Memory Course: Lecture 6: Real-world PIM: SK Hynix AiM - Spring 2022 (45:00)
Onur Mutlu Lectures

7 Processing-in-Memory Course: Lecture 7: Programming PIM Architectures - Spring 2022 (46:35)
Onur Mutlu Lectures

8 Processing-in-Memory Course: Lecture 8: Benchmarking and Workload Suitability on PIM - Spring 2022 (34:36)
Onur Mutlu Lectures

9 Processing-in-Memory Course: Lecture 9: Real-world PIM: Samsung AxDIMM - Spring 2022 (32:37)
Onur Mutlu Lectures

https://safari.ethz.ch/projects_and_seminars/spring2022/doku.php?id=processing_in_memory

https://youtube.com/playlist?list=PL5Q2soXY2Zi-0NK1C5vi2Zx9nmE_3-cKN

Machine Learning Training

on a Real Processing-in-Memory System

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

https://arxiv.org/pdf/2206.06022.pdf

juang@ethz.ch

ETH Zürich SAFARI upmem