Evaluating Machine Learning Workloads on Memory-Centric Computing Systems

Juan Gómez Luna, 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

ETH Zürich SAFARI upmem

Friday, December 22, 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: https://github.com/CMU-SAFARI/pim-ml
- 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

Outline

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

Outline

Machine learning workloads

Processing-in-memory

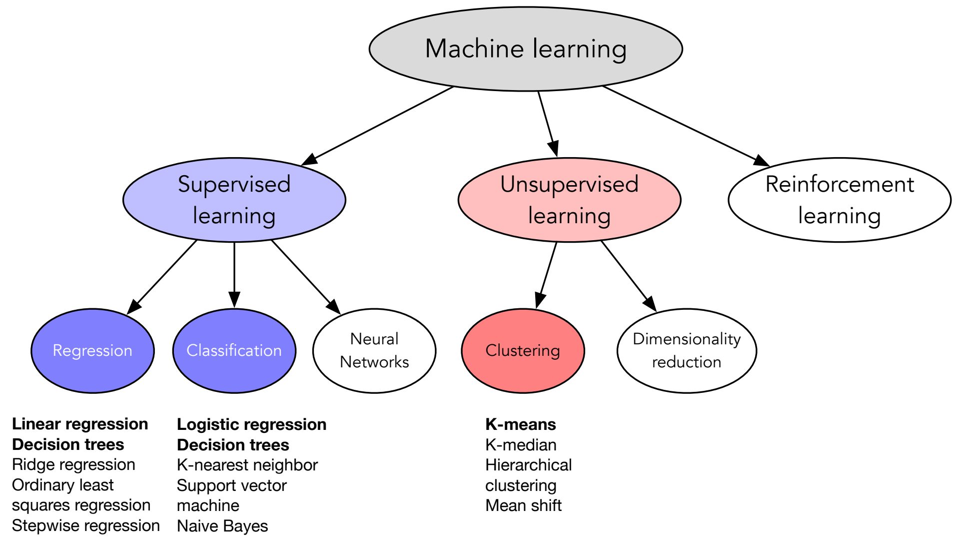
PIM implementation of ML workloads

Evaluation

- Quality Metrics
- Analysis of PIM Kernels
- Performance Scaling
- Comparison to CPU and GPU

Machine Learning Workloads

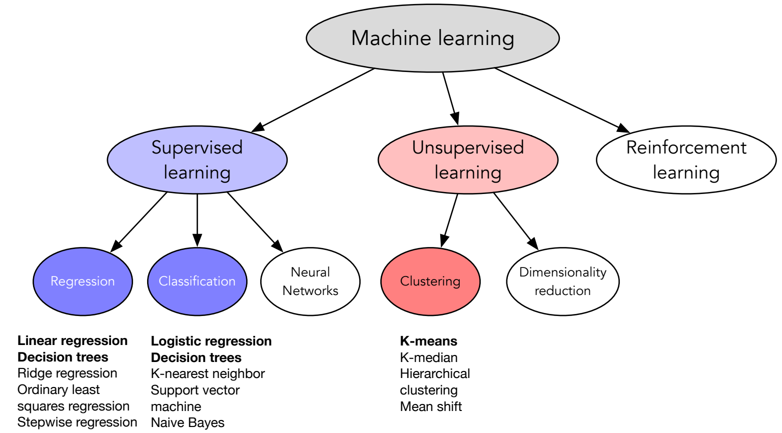
- Machine learning training with large amounts of data is a computationally expensive process, which requires many iterations to update an ML model's parameters

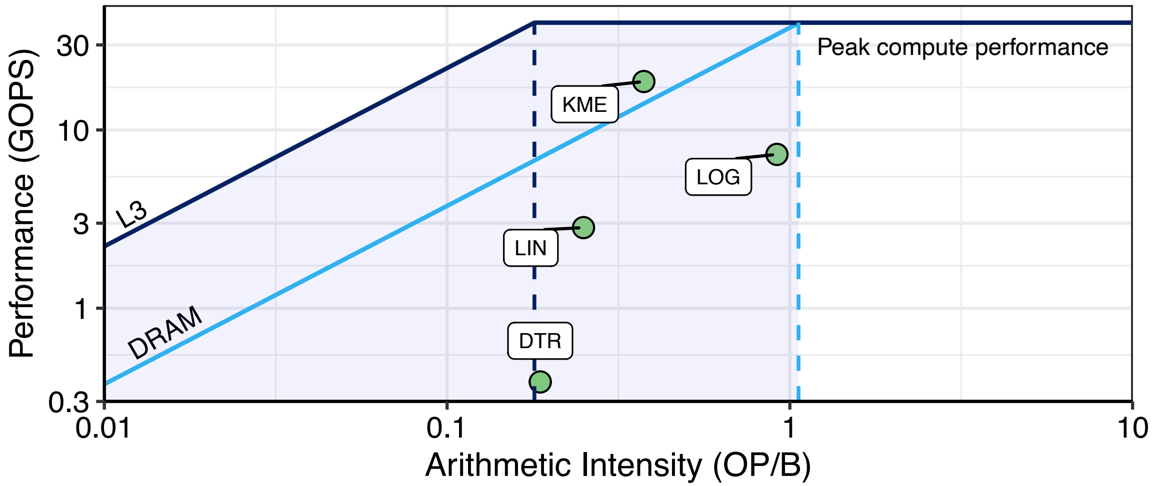


- Frequent data movement between memory and processing elements to access training data
- The amount of computation is not enough to amortize the cost of moving training data to the processing elements
  - Low arithmetic intensity
  - Low temporal locality
  - Irregular memory accesses

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

Outline

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

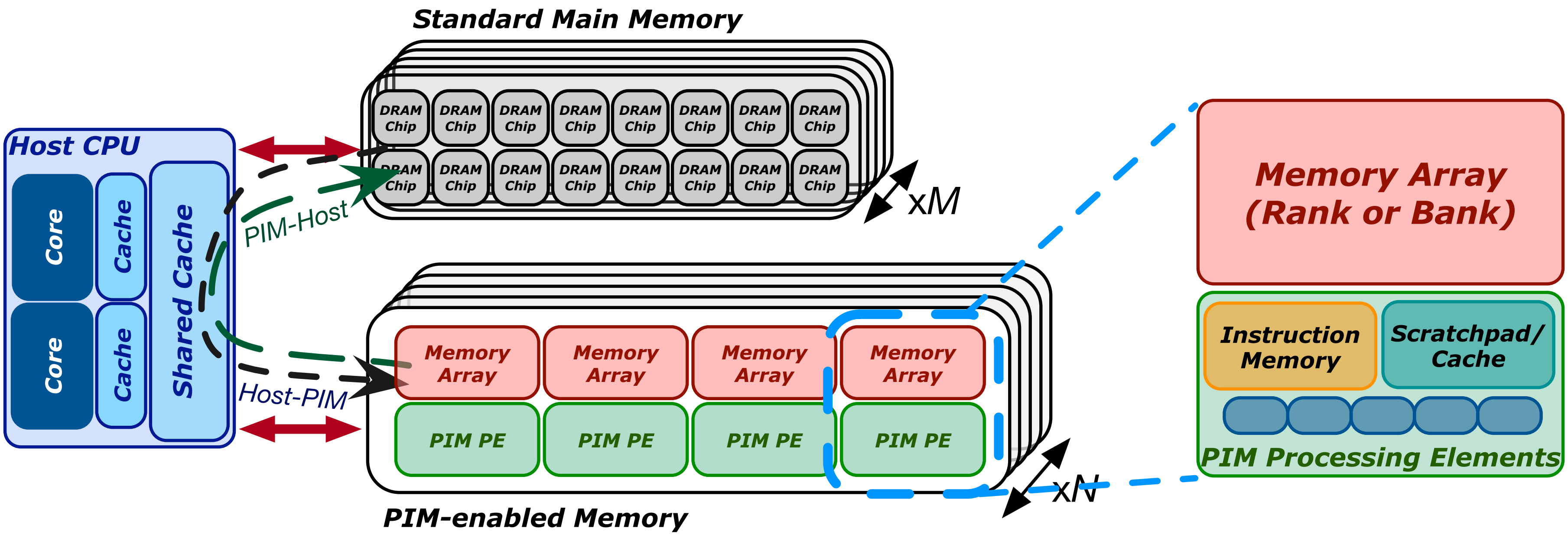
Evaluation

- Quality Metrics
- Analysis of PIM Kernels
- Performance Scaling
- Comparison to CPU and GPU

Processing-in-Memory (PIM)

- PIM is a computing paradigm that advocates for memory-centric computing systems, where processing elements are placed near or inside the memory arrays
- Real-world PIM architectures are becoming a reality
  - UPMEM PIM, Samsung HBM-PIM, Samsung AxDIMM, SK Hynix AiM, Alibaba HB-PNM
- These PIM systems have some common characteristics:
  1. There is a host processor (CPU or GPU) with access to (1) standard main memory, and (2) PIM-enabled memory
  2. PIM-enabled memory contains multiple PIM processing elements (PEs) with high bandwidth and low latency memory access
  3. PIM PEs run only at a few hundred MHz and have a small number of registers and small (or no) cache/scratchpad
  4. PIM PEs may need to communicate via the host processor

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*)

* 8-bit integer multiplication is natively supported

Outline

Machine learning workloads

Processing-in-memory

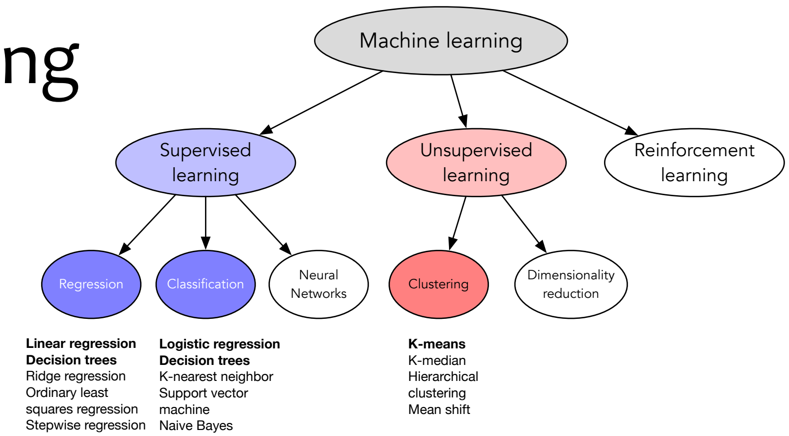
PIM implementation of ML workloads

Evaluation

- Quality Metrics
- Analysis of PIM Kernels
- Performance Scaling
- Comparison to CPU and GPU

ML Training Workloads

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



- 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

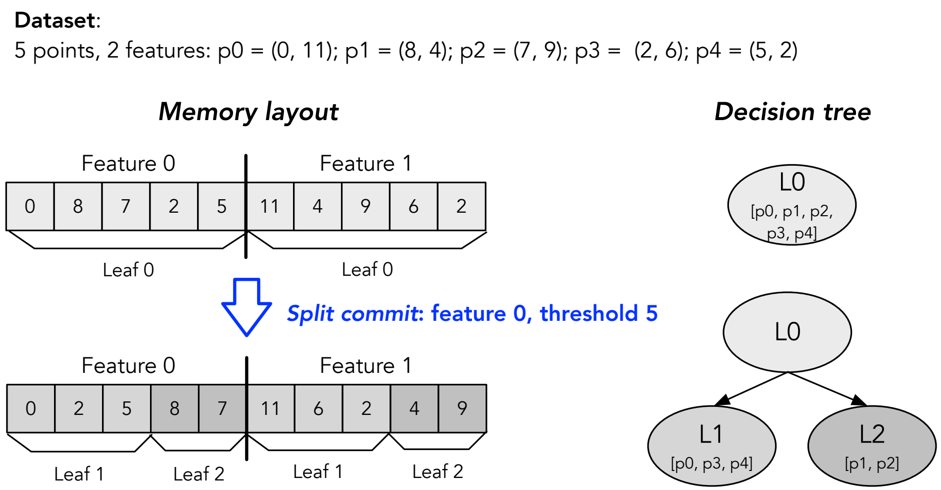
Logistic Regression

- 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 *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



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

- Quality Metrics
- Analysis of PIM Kernels
- Performance Scaling
- Comparison to CPU and GPU

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
  - NVIDIA A100 GPU
- We evaluate:
  - Quality metrics
  - Performance of PIM kernels
  - Performance scaling
  - Comparison to CPU and GPU

Outline

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

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)

Outline

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

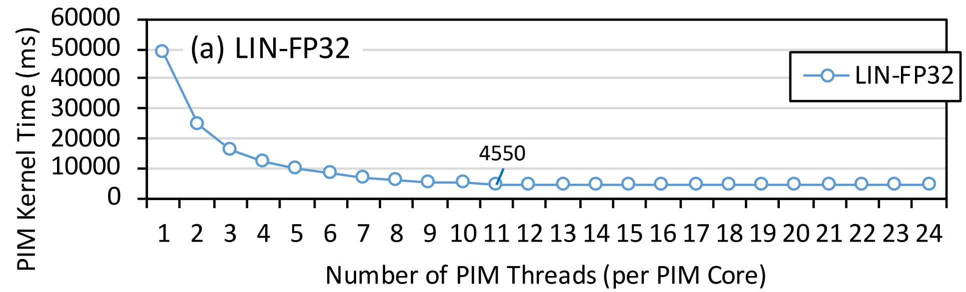
Evaluation: Analysis of PIM Kernels (I)

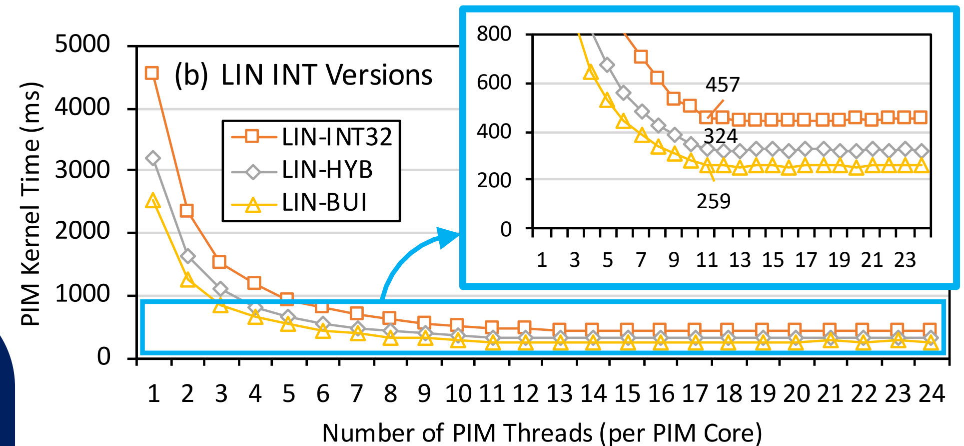
- Linear regression

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).





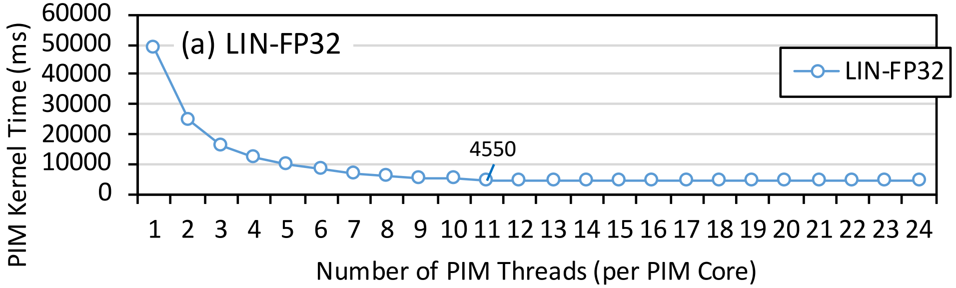
Recommendation 1. Use **fixed-point representation,** without much accuracy loss, if PIM cores do not support FP.

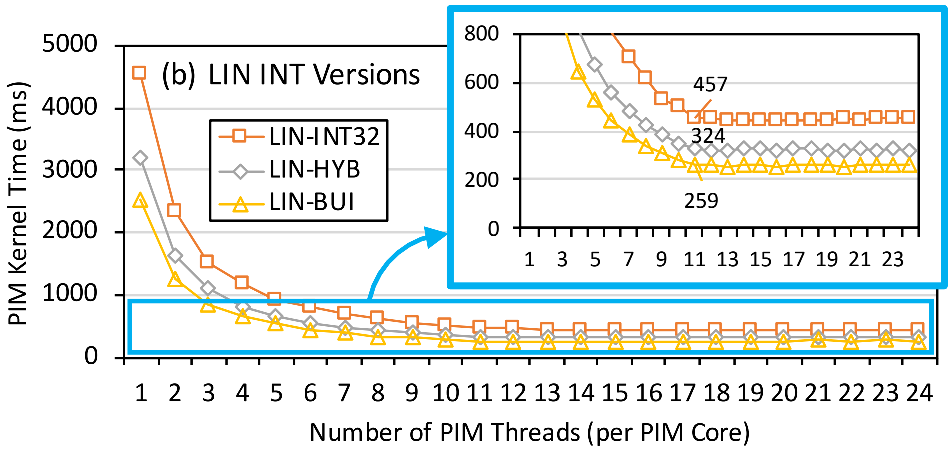
Evaluation: Analysis of PIM Kernels (II)

- 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) .

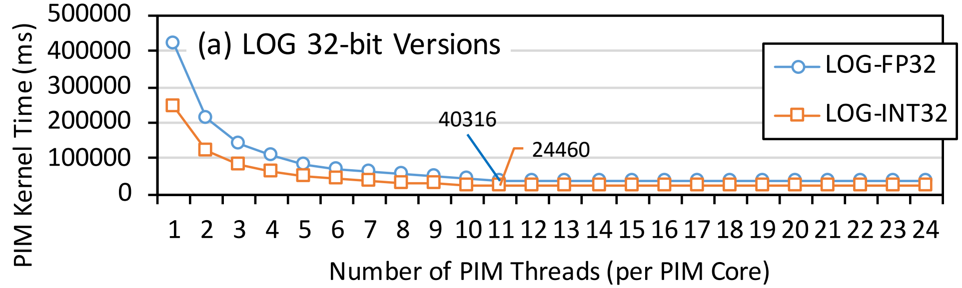
Evaluation: Analysis of PIM Kernels (III)

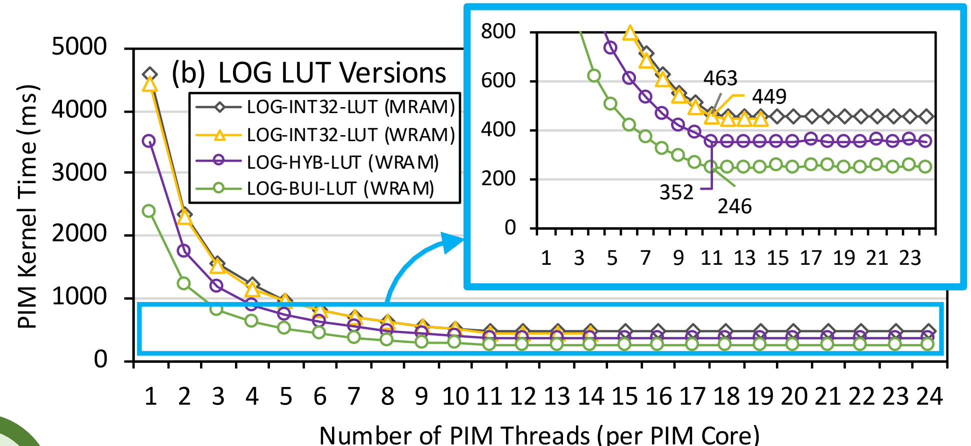
- Logistic regression

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**.



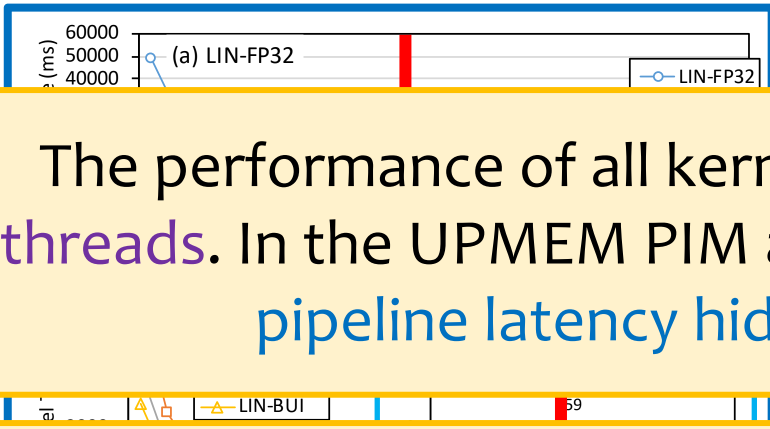


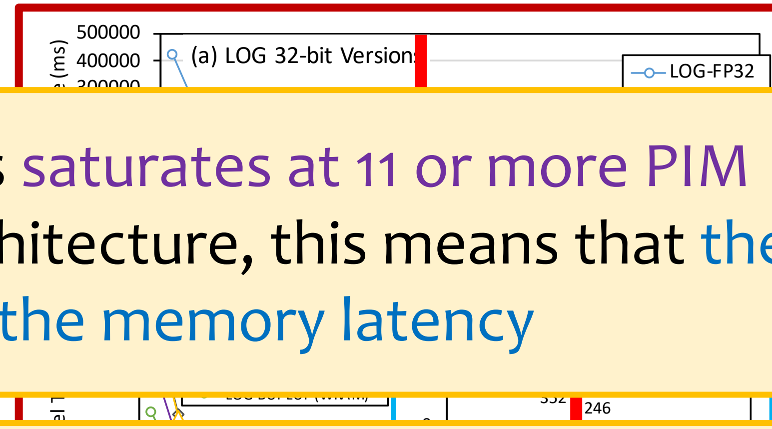
`LOG-HYB-LUT` is 28% faster than `LOG-INT32-LUT`

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

Evaluation: Analysis of PIM Kernels (IV)

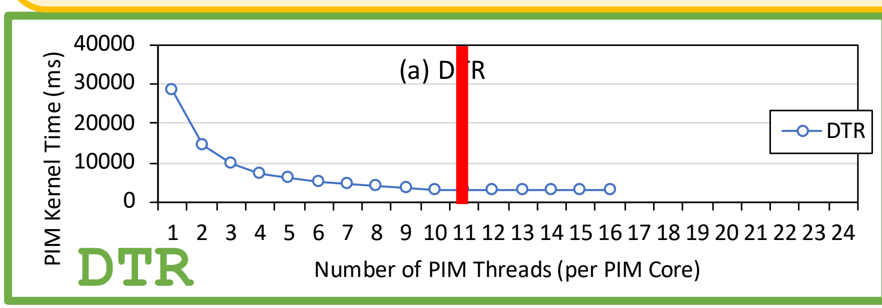
- Linear regression, logistic regression, decision tree, K-means clustering

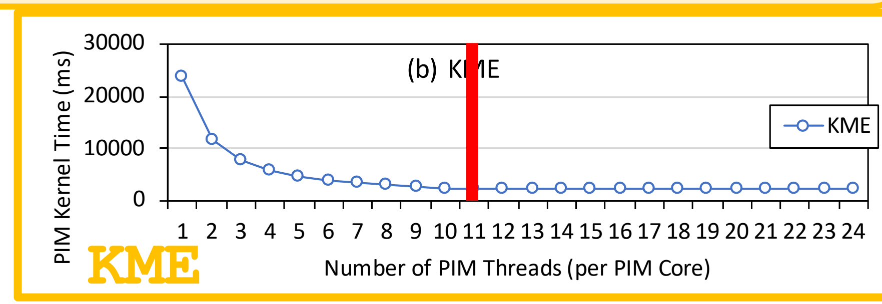




The performance of all kernels saturates at 11 or more PIM threads. In the UPMEM PIM architecture, this means that the pipeline latency hides the memory latency

As a result, these kernels are compute-bound on the UPMEM PIM architecture



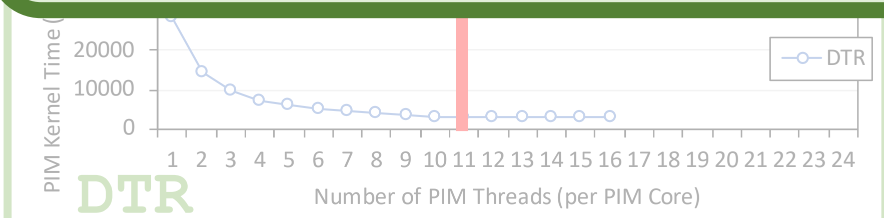


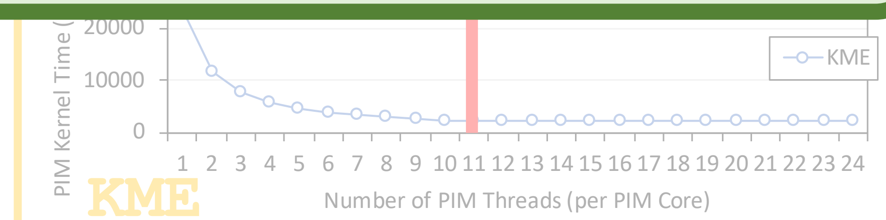
Evaluation: Analysis of PIM Kernels (V)

- 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.**





Outline

Machine learning workloads

Processing-in-memory

PIM implementation of ML workloads

Evaluation

Quality Metrics

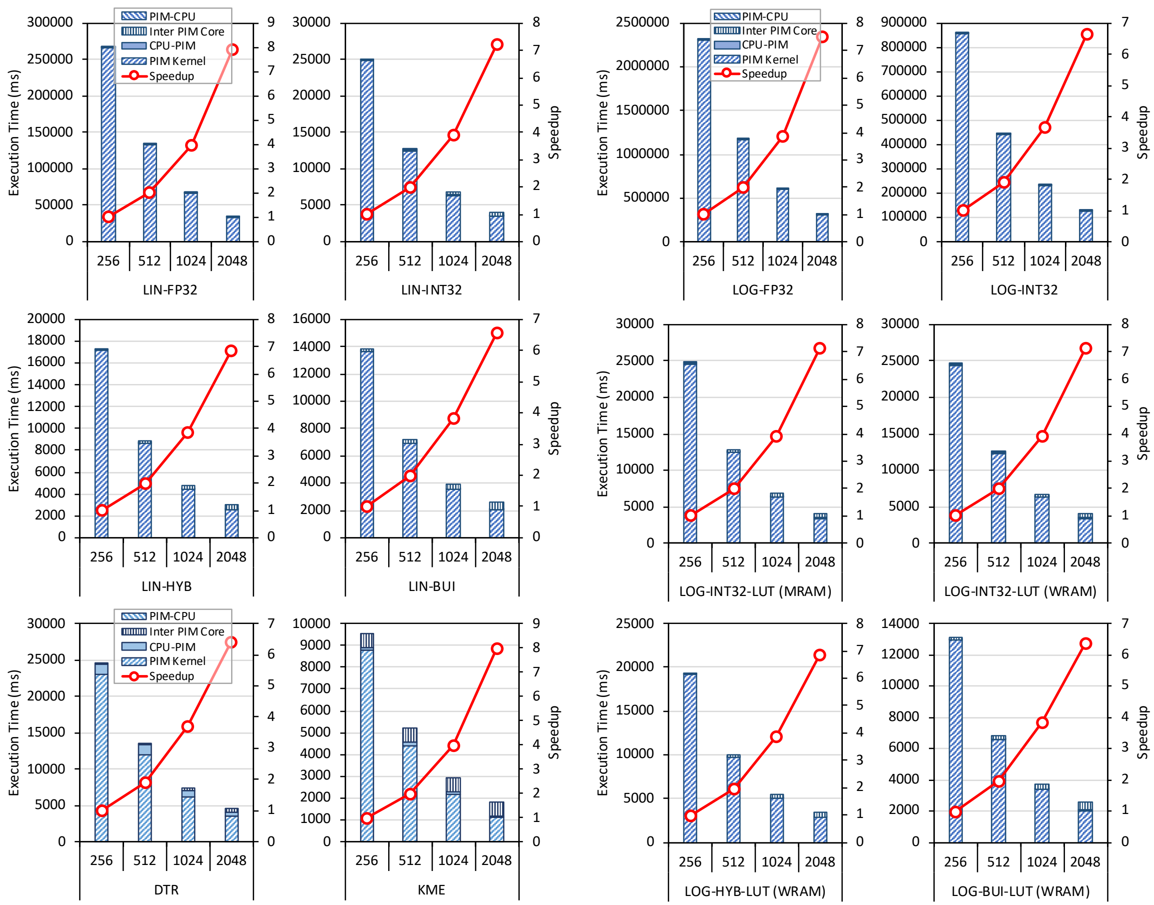
Analysis of PIM Kernels

Performance Scaling

Comparison to CPU and GPU

Evaluation: Performance Scaling (I)

- Strong scaling: 256 to 2,048 PIM cores

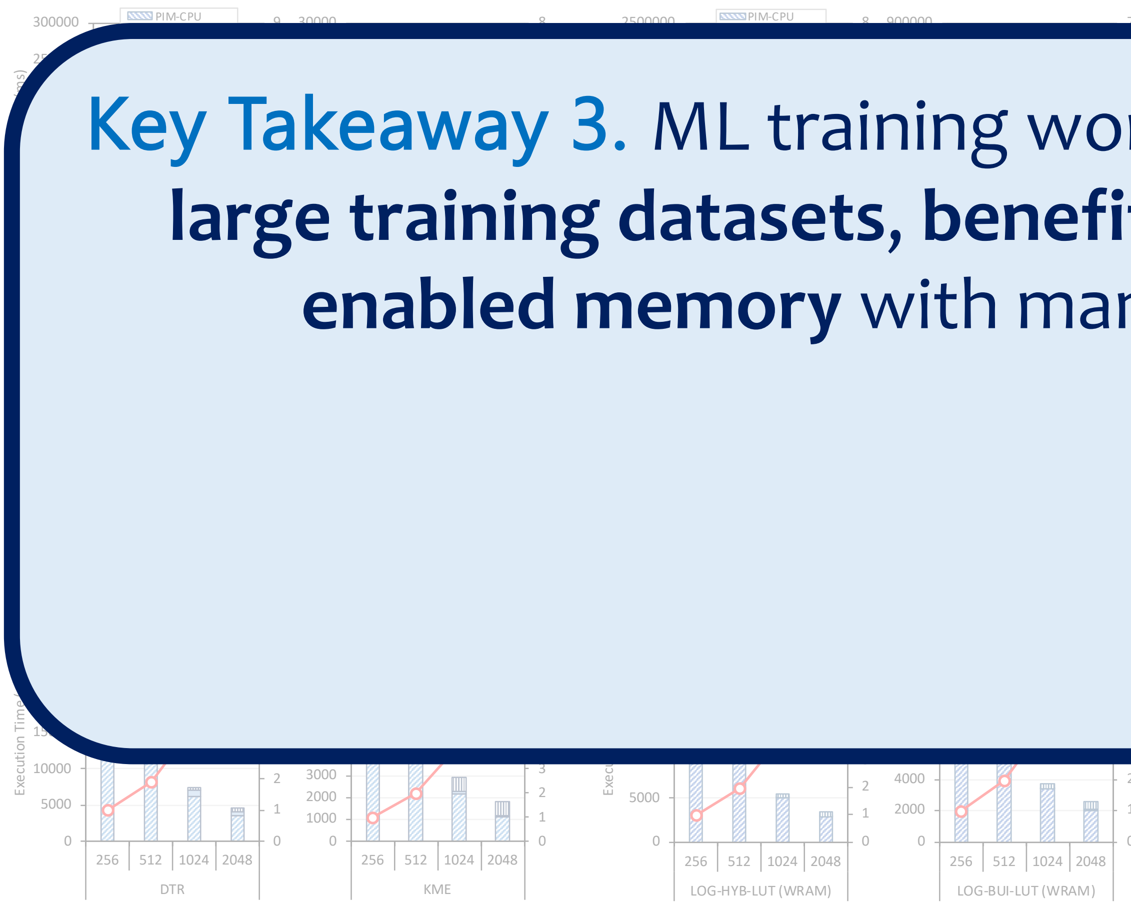


PIM kernel time scales linearly with the number of PIM cores

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



between host and
PIM cores

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

Outline

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**

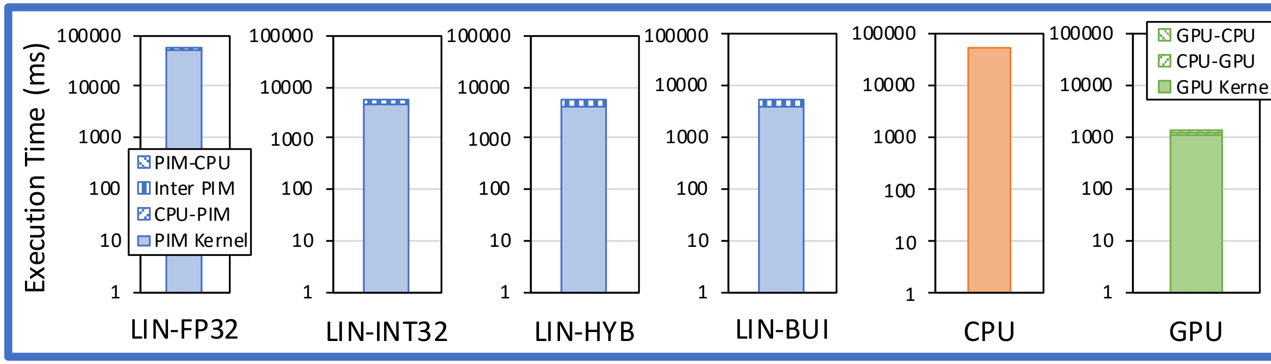
Comparison to CPU and GPU (I)

- Linear regression and logistic regression

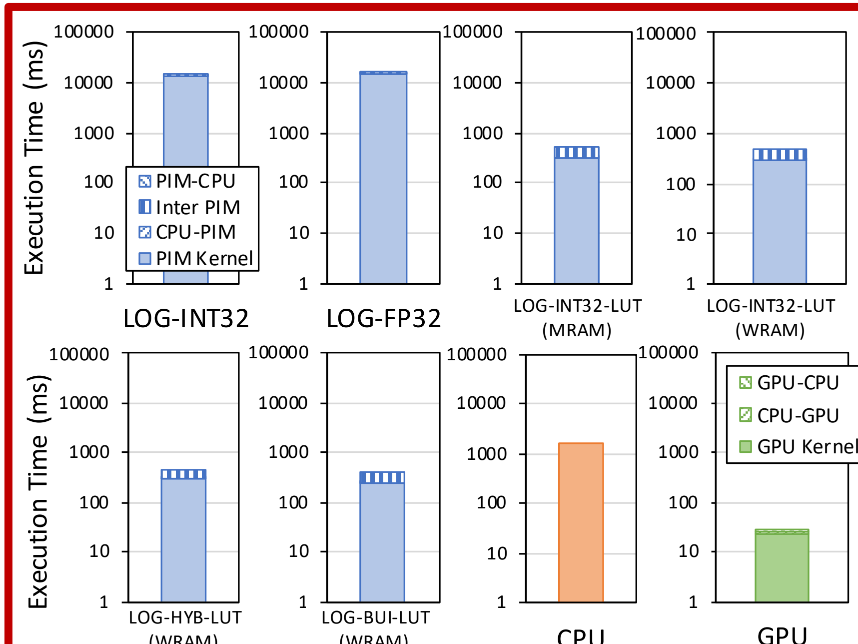
PIM versions are heavily burdened when they use operations that are not natively supported by the hardware

Several optimizations reduce the execution time considerably (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)

LIN

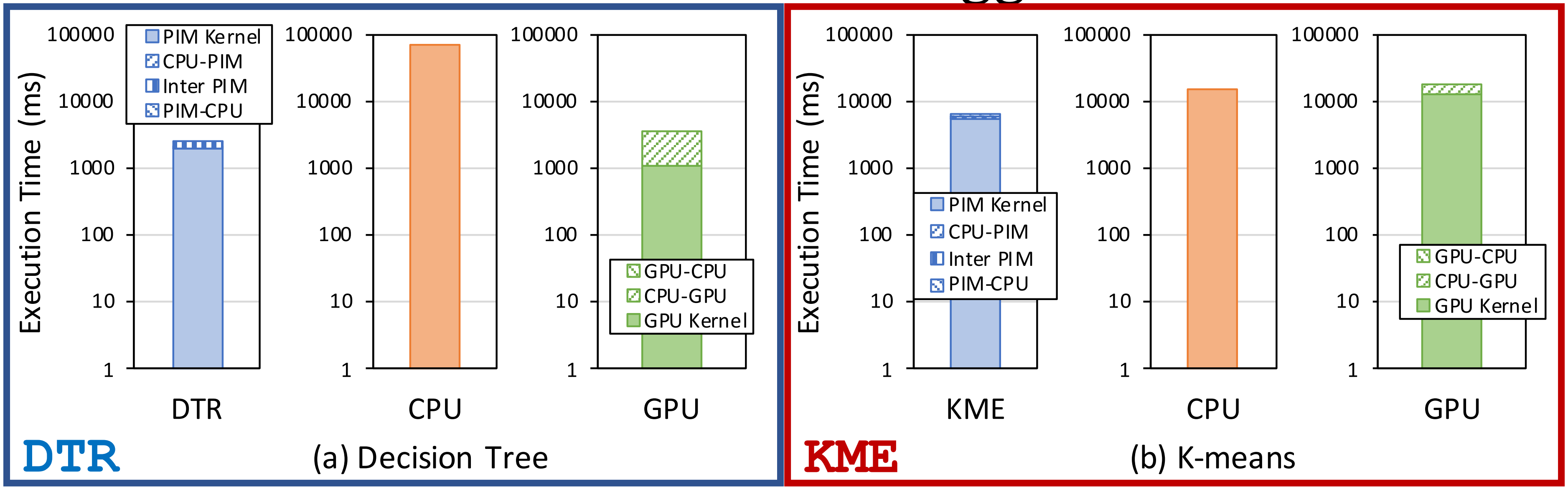


LOG



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

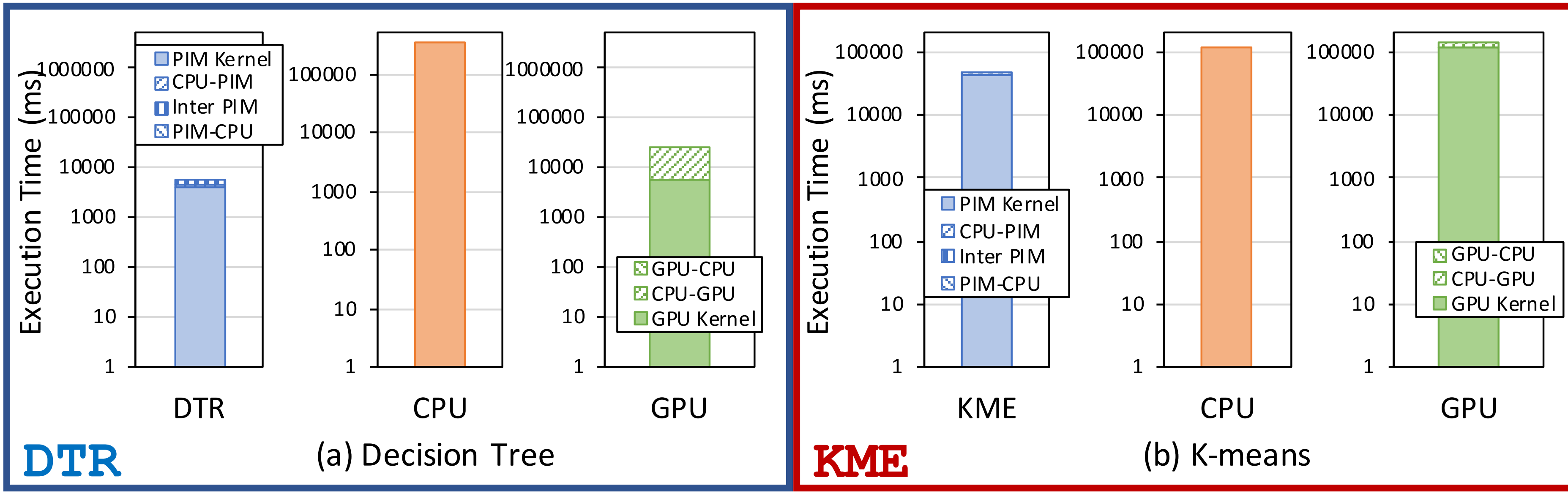
Juan Gómez-Luna[1] Yuxin Guo[1] Sylvan Brocard[2] Julien Legriel[2]
Remy Cimadomo[2] Geraldo F. Oliveira[1] Gagandeep Singh[1] Onur Mutlu[1]
[1]ETH Zürich [2]UPMEM

https://arxiv.org/pdf/2207.07886.pdf

Source code: https://github.com/CMU-SAFARI/pim-ml

Comparison to CPU and GPU (III)

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

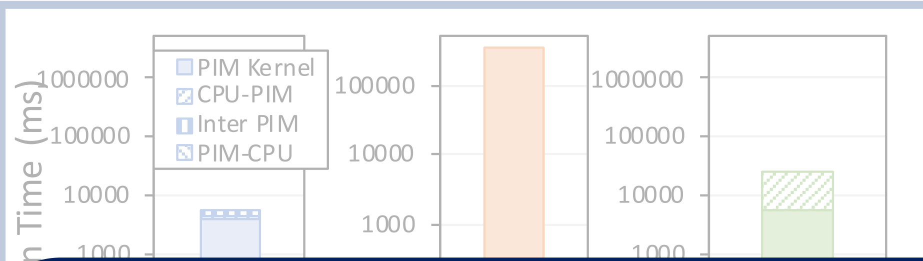


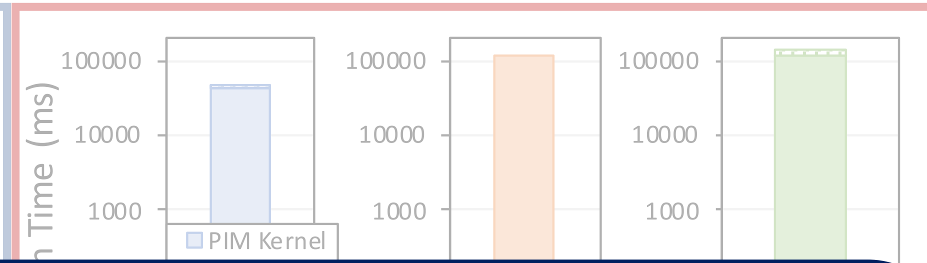
PIM version of DTR is **62x** faster than the CPU version and **4.5x** faster than the GPU version

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

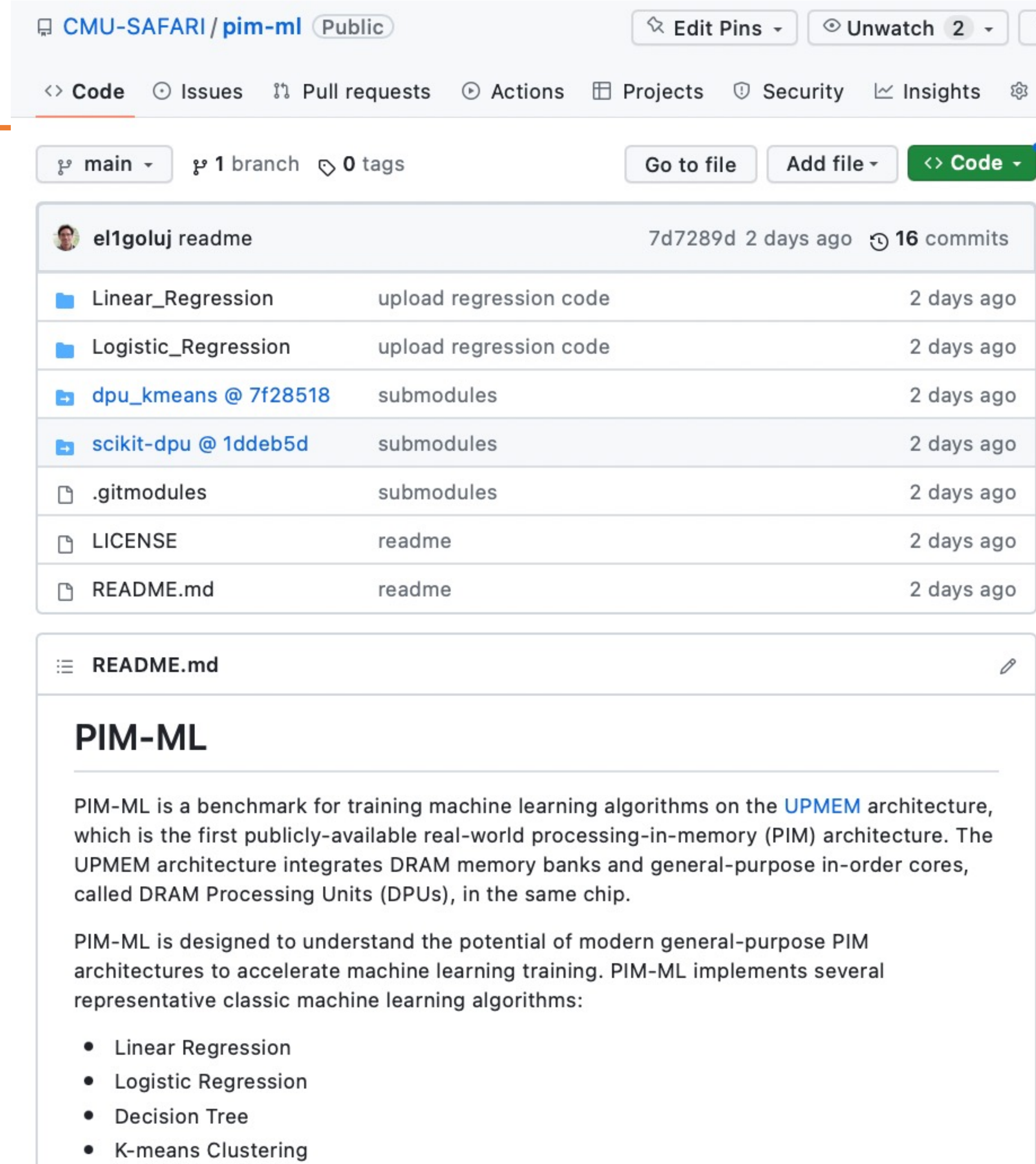
[1]ETH Zürich [2]UPMEM

https://arxiv.org/pdf/2207.07886.pdf

[Source code: https://github.com/CMU-SAFARI/pim-ml](https://github.com/CMU-SAFARI/pim-ml)

Source Code

- https://github.com/CMU-SAFARI/pim-ml



Conclusion

- 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: https://github.com/CMU-SAFARI/pim-ml
- 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

Evaluating Machine Learning Workloads on Memory-Centric Computing Systems

Juan Gómez Luna, 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

ETH Zürich SAFARI upmem

Friday, December 22, 2023