PIM-Opt

Demystifying Distributed Optimization Algorithms on a Real-World Processing-In-Memory System

Steve Rhyner Haocong Luo Juan Gómez-Luna Mohammad Sadrosadati

Jiawei Jiang Ataberk Olgun Harshita Gupta Ce Zhang Onur Mutlu

PIM-Opt: Summary

Problem: Modern machine learning (ML) training is a data-intensive workload and processor-centric architectures commonly used for ML training suffer from the data movement bottleneck

Goal: Understand the capabilities of popular distributed Stochastic Gradient Descent (SGD) algorithms on real-world Processing-In-Memory (PIM) systems to accelerate distributed ML training workloads

Contributions:

- Implementation, analysis, and training of linear ML models on two large datasets using distributed SGD algorithms on a real-world PIM system (i.e., UPMEM)
- Demonstrate scalability challenges of the UPMEM PIM system
- Discuss implications for future PIM hardware design
- Highlight the need for a shift towards an algorithm-hardware codesign

Evaluation:

- Comparison of the UPMEM PIM to state-of-the-art CPU and GPU
  - YFCC100M-HNfc6 dataset: UPMEM PIM is up to 1.9x/3.2x faster than the CPU/GPU
  - Criteo 1TB Click Logs dataset: UPMEM PIM is up to 9.3x/10.7x faster than the CPU/GPU
- Scalability challenges of the UPMEM PIM
  - YFCC100M-HNfc6 (Criteo 1TB Click Logs) dataset: Speedup of 7.4x (3.9x) while the achieved test accuracy (AUC score) decreases from 95.5% (0.74) to 92.2% (0.72)

Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

Evaluation & Key Results

Implications for PIM Hardware Design

Conclusion

Models & ML Training

- Two of the **most commonly trained linear binary classification models:**
  - **Logistic Regression (LR)**
  - **Support Vector Machines (SVM)**

- **The goal of machine learning (ML) training** is to **find an optimal ML model** by minimizing an objective function over a training dataset

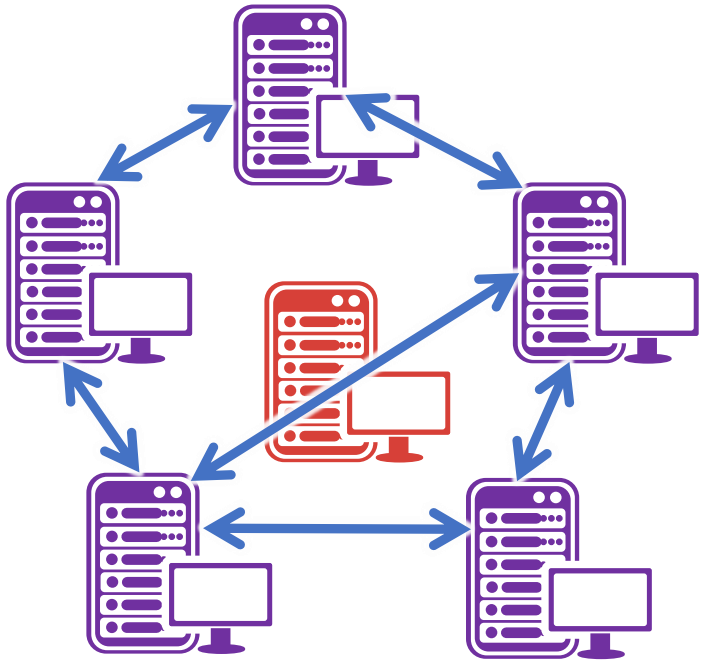
- **Regularization techniques** are used
  - **Prevent overfitting on the training dataset**
  - **Control the model complexity**

Algorithms

- **Stochastic Gradient Descent (SGD)** is perhaps the **most important** and **commonly deployed optimization algorithm** for modern ML training
- **SGD** is the **main building block of most distributed optimization algorithms**
- Variants of SGD such as **mini-batch SGD** allow for **parallelization** by batching the training samples in each iteration

Distributed Optimization Algorithms

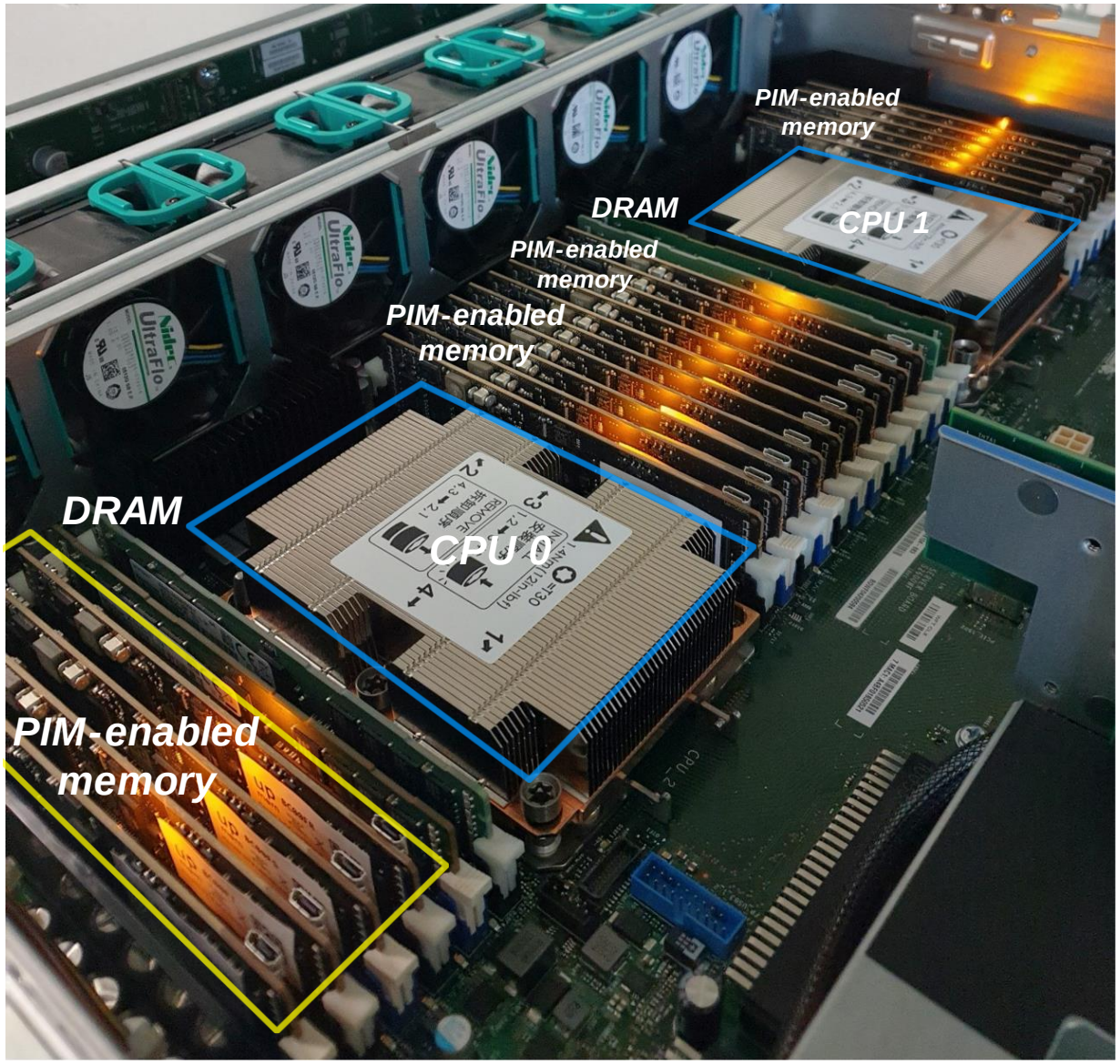
Decentralized Topology



Popular centralized optimization algorithms:

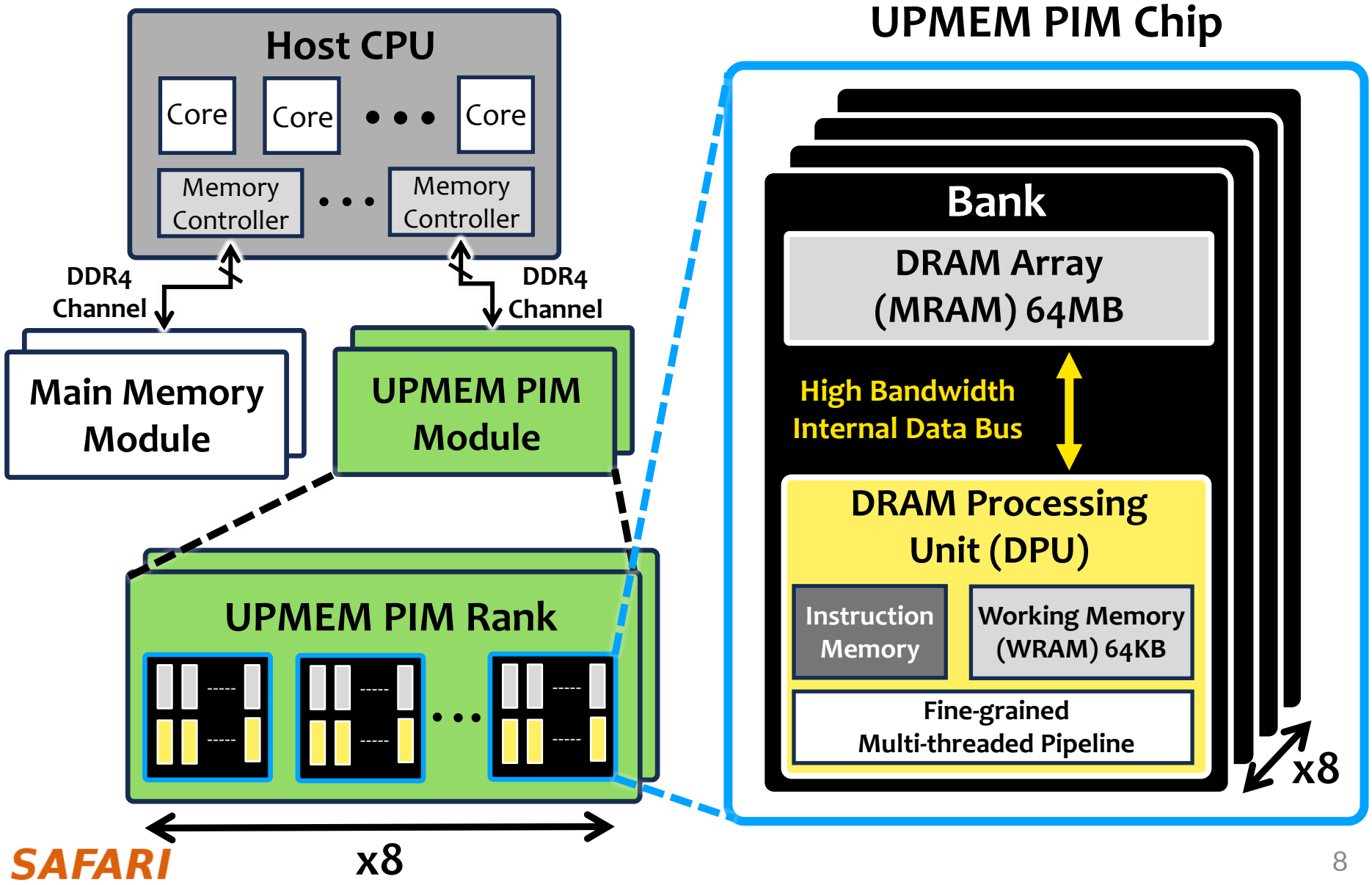
- **Mini-batch SGD with Model Averaging (MA-SGD)**
- **Mini-batch SGD with Gradient Averaging (GA-SGD)**
- **Distributed Alternating Direction Method of Multipliers (ADMM)**

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

UPMEM PIM System Architecture



Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

Evaluation & Key Results

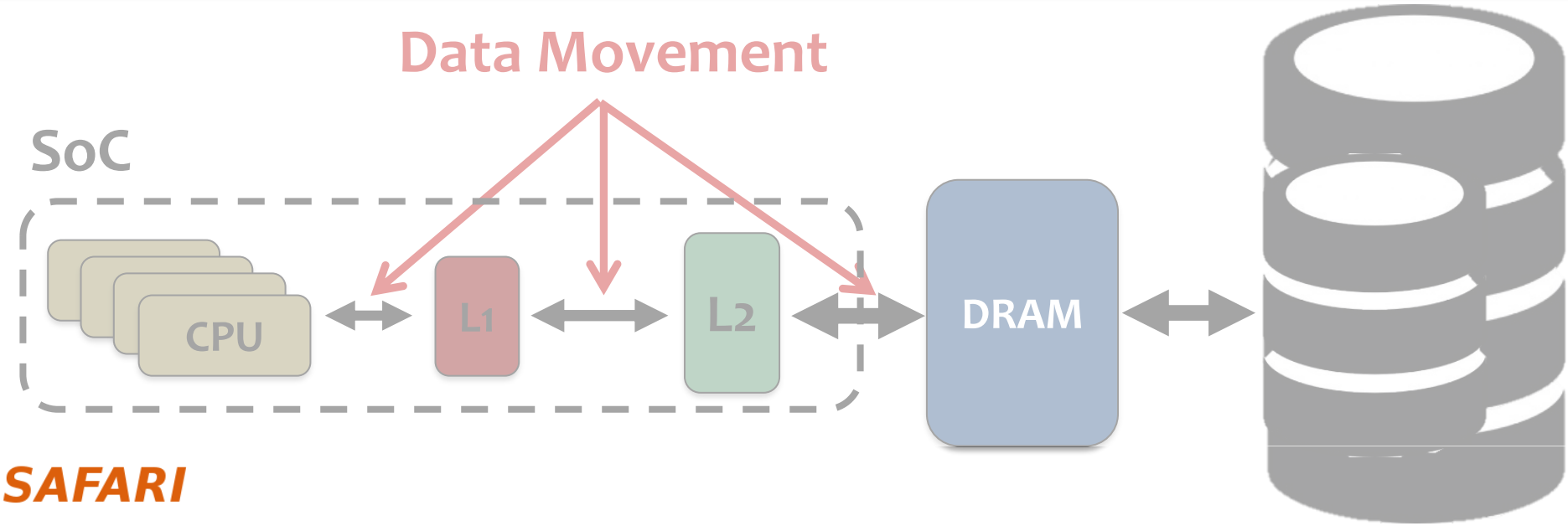
Implications for PIM Hardware Design

Conclusion

Key Problem

Data Movement Bottleneck

The **data movement bottleneck** is a **key limiter** of **large-scale Machine Learning training**



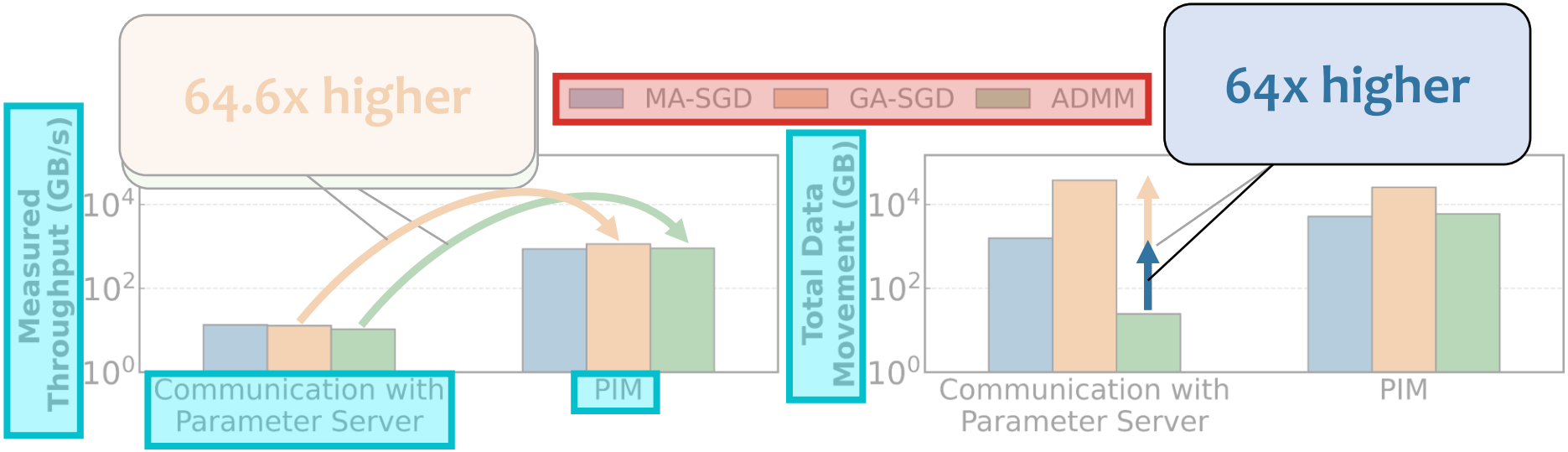
Motivation

Processing-In-Memory is a promising solution to perform large-scale ML Training

How do we start?

By identifying key characteristics covering the design space of both hardware and optimization algorithms

Motivation



→ Minimize the communication between the parameter server and PIM

→ Minimize the total data movement between the parameter server and PIM

The *communication-efficient* **ADMM optimization algorithm** is **attractive for distributed ML training** on the UPMEM PIM system

Outline

Background

Motivation

UPMEM PIM System Implementation

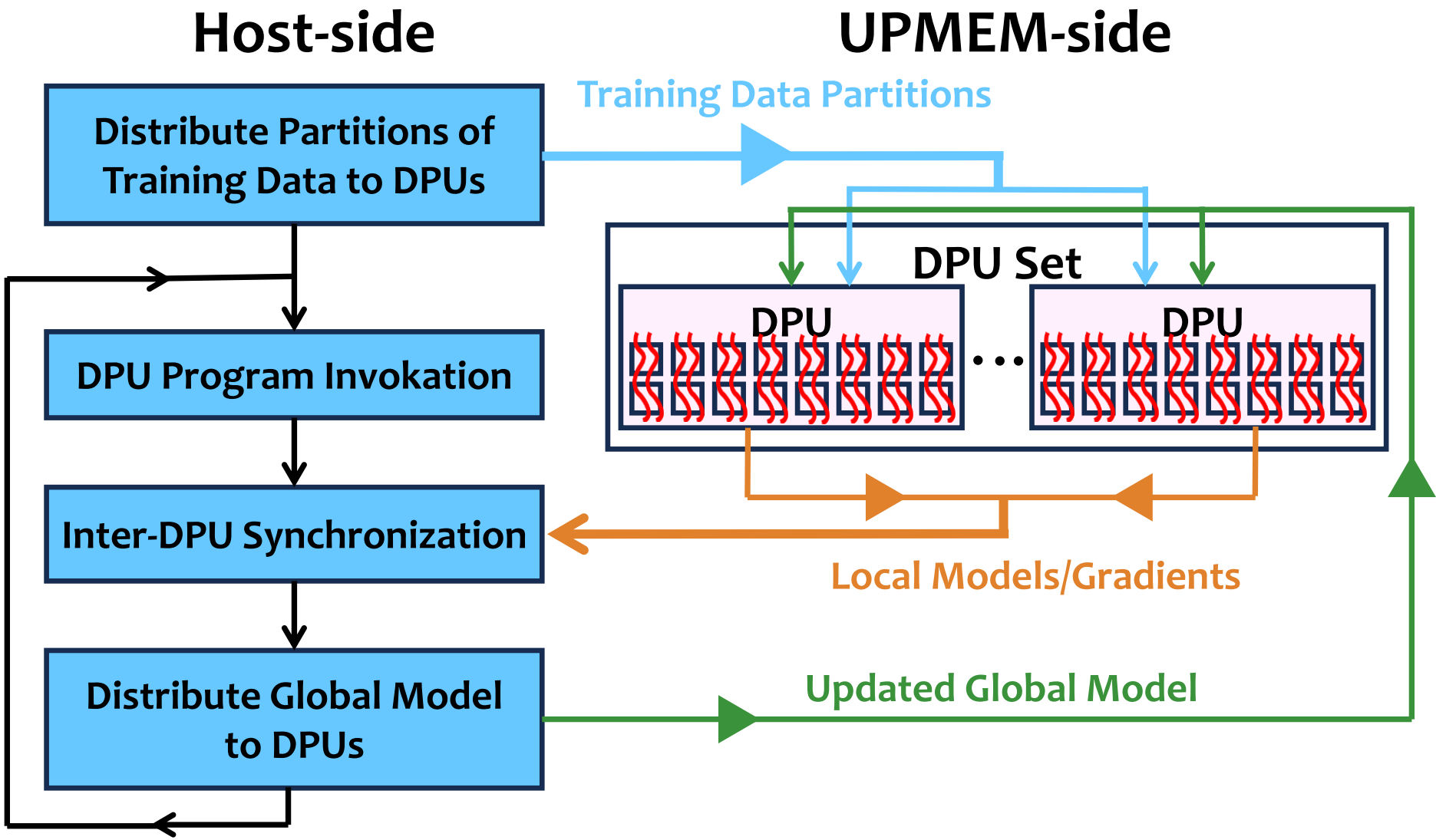
Methodology

Evaluation & Key Results

Implications for PIM Hardware Design

Conclusion

UPMEM PIM System Implementation



UPMEM PIM System Implementation

- **Task Parallelism:**
  - **Every DPU** is a ***worker***
  - **Every DPU** uses **16 tasklets collaboratively** to implement the **mini-batch SGD optimizer**
  - **Features** of the **training samples** & **model parameters** are **evenly distributed among tasklets**

- **LUT-based Methods:**
  - Training of **Logistic Regression** involves computing the exponential function **to evaluate the sigmoid**
  - **UPMEM PIM system does not support transcendental functions**

Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

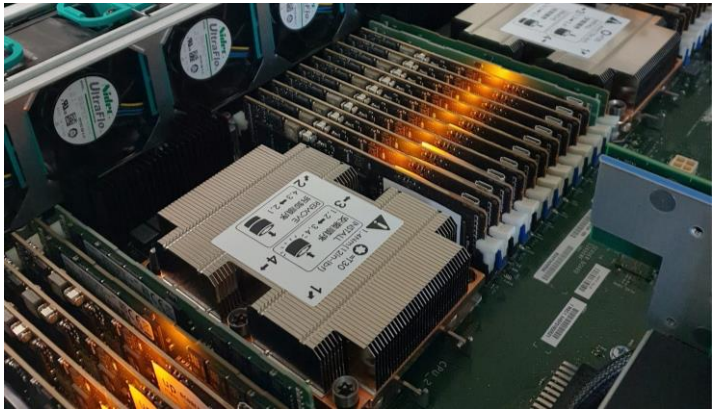
Evaluation & Key Results

Implications for PIM Hardware Design

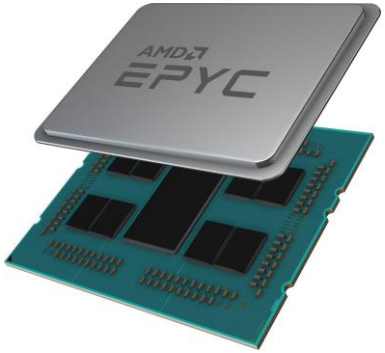
Conclusion

System Configurations

- **UPMEM PIM System:**
  - 2x Intel Xeon Silver 4215
    8-core processor @ 2.50GHz
  - 2560 DPUs @ 350 MHz
  - 20×8 GB UPMEM PIM modules



- **CPU Baseline System:**
  - 2x AMD EPYC 7742
    64-core processor @ 2.25GHz



- **GPU Baseline System:**
  - 2x Intel Xeon Gold 5118
    12-core processor @ 2.30GHz
  - 1× NVIDIA A100 (PCIe, 80 GB)

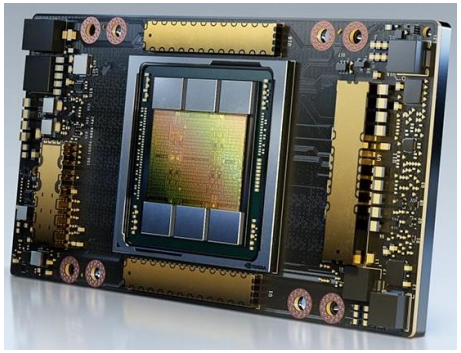


Image source: https://gzhls.at/i/35/19/2113519-l3.jpg (AMD EPYC), https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/nvidia-a100-80-gb-og-social-1200x630.jpg (NVIDIA A100)

Baseline Implementations

- **CPU Baseline Implementation:**
  - Implementations use **PyTorch**
  - We implement **MA-SGD**, **GA-SGD**, and **ADMM**, to train **LR** and **SVM models**, using the optimizers and communication libraries provided by PyTorch
  - **Each CPU thread** is a ***worker***

- **GPU Baseline Implementation:**
  - Implementations use **PyTorch**
  - We only implement **mini-batch SGD on the GPU**

  - **For fair comparison**, **we do not use a cluster of GPUs for our baseline** because the **UPMEM PIM system** is a **single-server node**

Experiment Implementation Details

PIM-Opt: Demystifying Distributed Optimization Algorithms on a Real-World Processing-In-Memory System

Steve Rhyner[1] Haocong Luo[1] Juan Gómez-Luna[2] Mohammad Sadrosadati[1]
Jiawei Jiang[3] Ataberk Olgun[1] Harshita Gupta[1] Ce Zhang[4] Onur Mutlu[1]

[1]ETH Zurich [2]NVIDIA [3]Wuhan University [4]University of Chicago

Abstract

Modern *Machine Learning* (ML) training on large-scale datasets is a very time-consuming workload. It relies on the optimization algorithm *Stochastic Gradient Descent* (SGD) due to its effectiveness, simplicity, and generalization performance (i.e., test performance on unseen data). Processor-centric architectures (e.g., CPUs, GPUs) commonly used for modern ML training workloads based on SGD are bottlenecked by data movement between the processor and memory units due to the poor data locality in accessing large training datasets. As a result, processor-centric architectures suffer from low performance and high energy consumption while executing ML training workloads. *Processing-In-Memory* (PIM) is a promising solution to alleviate the data movement bottleneck by placing the main building block of most centralized and decentralized optimization algorithms that have been introduced to accommodate the continuously increasing demand for scalability and high-performance training of ML models on large-scale datasets.

Training ML models on growing datasets [55, 190, 193] is a time-consuming task that demands both high computational power and memory bandwidth [42, 74, 75, 81]. The low data reuse during ML training on large-scale datasets leads to poor data locality. As a result, processor-centric architectures (e.g., CPU, GPU) commonly used by the ML community repeatedly need to move training samples between the processor and off-chip memory. This not only degrades performance [96] but is also a major source of the overall system's energy consumption [17]. This phenomenon is referred to

https://arxiv.org/pdf/2404.07164v2

Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

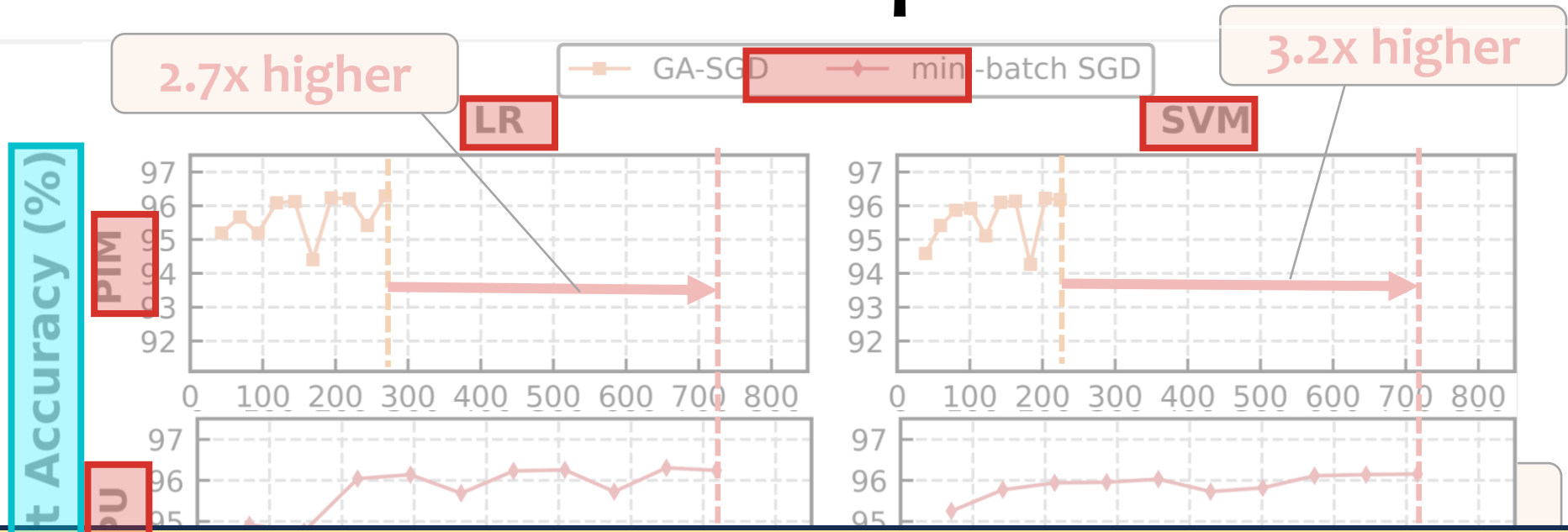
Evaluation & Key Results

Implications for PIM Hardware Design

Conclusion

YFCC100M-HNfc6 Dataset

PIM Performance Comparison

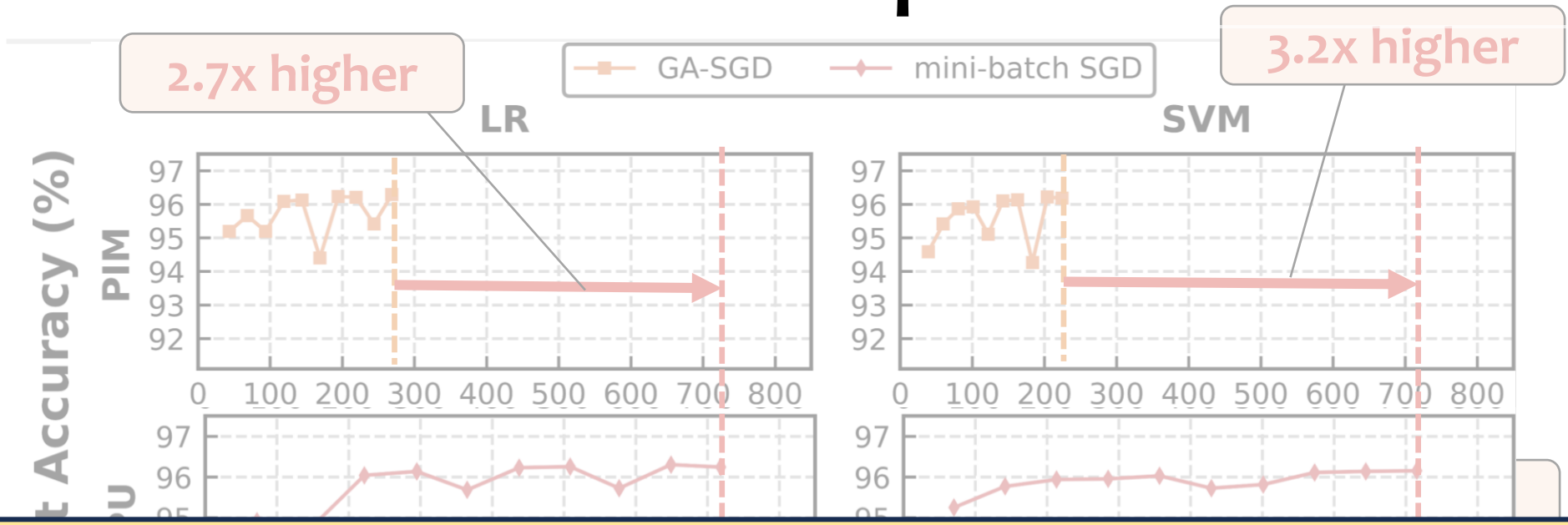


- MA-SGD on the UPMEM PIM significantly outperforms MA-SGD on the CPU for both LR and SVM
- ADMM on the UPMEM PIM and on the CPU exhibits comparable performance for both LR and SVM

both LR and SVM

- GA-SGD on the UPMEM PIM outperforms mini-batch SGD on the GPU for both LR and SVM

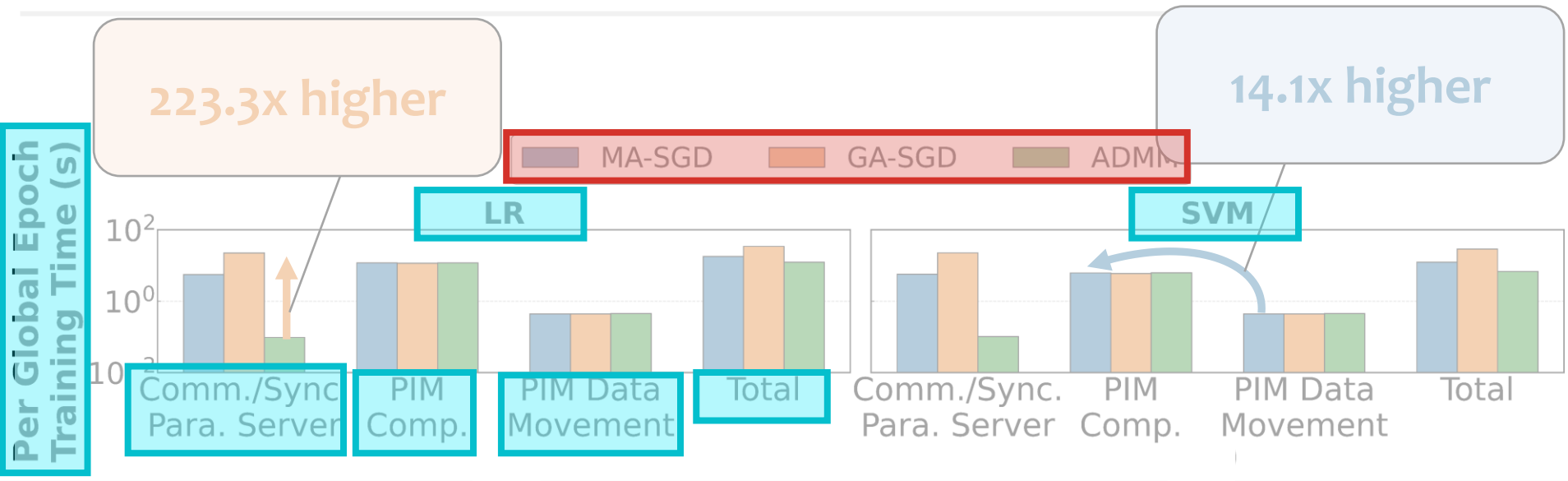
PIM Performance Comparison



The UPMEM PIM is a viable alternative to the CPU and the GPU for training small dense models on large-scale datasets

- **GA-SGD** on the **UPMEM PIM outperforms GA-SGD** on the **CPU** for both LR and SVM
- **GA-SGD** on the **UPMEM PIM outperforms mini-batch SGD** on the **GPU** for both LR and SVM

PIM Performance Breakdown

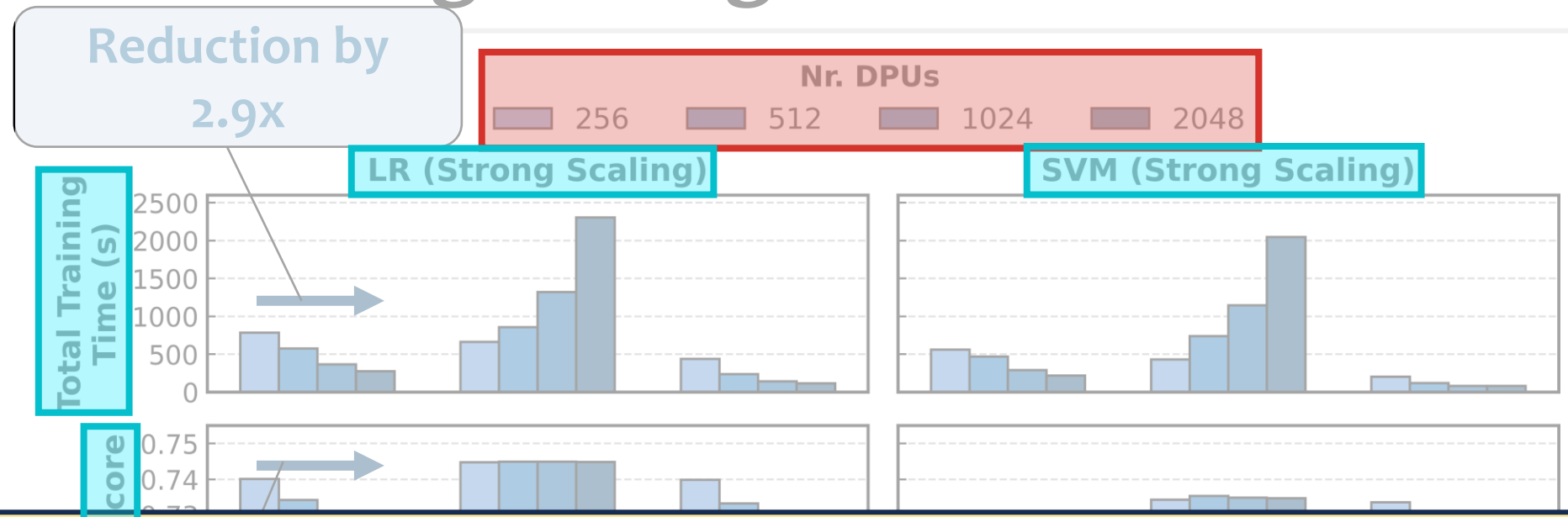


The UPMEM PIM is *less* suitable for ML models and optimization algorithms that require frequent communication and synchronization between PIM and the parameter server

- For *all* combinations of optimization algorithms and models, PIM computation takes more time than PIM data movement on the UPMEM PIM

Criteo Dataset

PIM Strong Scaling



The scalability potential of the UPMEM PIM for training high-dimensional sparse models is limited by its lack of direct inter-DPU communication

- For high-dimensional sparse models, for **MA-SGD/ADMM** on the UPMEM PIM we observe **good strong scalability** in terms of **total training time**, but **poor in AUC Score**

More in the Paper

- Rigorous analysis of many combinations of algorithms, models, arc
- For we exa
  -
  -
  -
  -
- Extend future PIM hardware design

PIM-Opt: Demystifying Distributed Optimization Algorithms on a Real-World Processing-In-Memory System

Steve Rhyner[1] Haocong Luo[1] Juan Gómez-Luna[2] Mohammad Sadrosadati[1]
Jiawei Jiang[3] Ataberk Olgun[1] Harshita Gupta[1] Ce Zhang[4] Onur Mutlu[1]

[1]ETH Zurich [2]NVIDIA [3]Wuhan University [4]University of Chicago

Abstract

Modern *Machine Learning* (ML) training on large-scale datasets is a very time-consuming workload. It relies on the optimization algorithm *Stochastic Gradient Descent* (SGD) due to its effectiveness, simplicity, and generalization performance (i.e., test performance on unseen data). Processor-centric architectures (e.g., CPUs, GPUs) commonly used for modern ML training workloads based on SGD are bottlenecked by data movement between the processor and memory units due to the poor data locality in accessing large training datasets. As a result, processor-centric architectures suffer from low performance and high energy consumption while executing ML training workloads. *Processing-In-Memory* (PIM) is a promising solution to alleviate the data movement bottleneck by placing the main building block of most centralized and decentralized optimization algorithms that have been introduced to accommodate the continuously increasing demand for scalability and high-performance training of ML models on large-scale datasets.

Training ML models on growing datasets [55, 190, 193] is a time-consuming task that demands both high computational power and memory bandwidth [42, 74, 75, 81]. The low data reuse during ML training on large-scale datasets leads to poor data locality. As a result, processor-centric architectures (e.g., CPU, GPU) commonly used by the ML community repeatedly need to move training samples between the processor and off-chip memory. This not only degrades performance [96] but is also a major source of the overall system's energy consumption [17]. This phenomenon is referred to

https://arxiv.org/pdf/2404.07164v2

Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

Evaluation & Key Results

Implications for PIM Hardware Design

Conclusion

Implications for PIM Hardware Design

- Our evaluation demonstrates that a **real-world PIM system** (i.e., **UPMEM**) can be a **viable alternative** to state-of-the-art **processor-centric architectures** for many **distributed ML training** workloads

- We argue that **future PIM architectures** should **add interconnects** and/or **shared memory among PIM processing units**
  - Enables implementation of decentralized optimization algorithms
  - **Decentralized parallel SGD algorithms** are a promising solution to overcome scalability challenges of the real-world PIM system

- We posit that a **shift towards an algorithm-hardware codesign** perspective is **necessary in the context of ML training using PIM** due to the high complexity of the design space

Outline

Background

Motivation

UPMEM PIM System Implementation

Methodology

Evaluation & Key Results

Implications for PIM Hardware Design

Conclusion

Conclusion

- We evaluate and train **ML models** on **large-scale datasets** with **centralized optimization algorithms** on a **real-world PIM system** (i.e., **UPMEM**)

- We show that it is important to **carefully choose** the **distributed optimization algorithm** that **best fits the real-world PIM system** and analyze tradeoffs

- We demonstrate that commercial general-purpose **PIM systems** can be a **viable alternative** to state-of-the-art **processor-centric architectures** (e.g., **CPU**, **GPU**) for many **distributed ML training** workloads on large-scale datasets

- Our results demonstrate the necessity of **adjust PIM architectures** to **enable decentralized parallel SGD algorithms** to overcome scalability challenges for many distributed ML training workloads

- **Future work:**
  - **Larger models:** Deep neural networks, large language models, ...
  - **New compute paradigms and accelerators**
    - Rethinking the full stack
    - Algorithm-hardware codesign

https://github.com/CMU-SAFARI/PIM-Opt

PIM-Opt

Demystifying Distributed Optimization Algorithms on a Real-World Processing-In-Memory System

arXiv

GitHub

SAFARI
SAFARI Research Group
safari.ethz.ch

WUHAN UNIVERSITY 1893

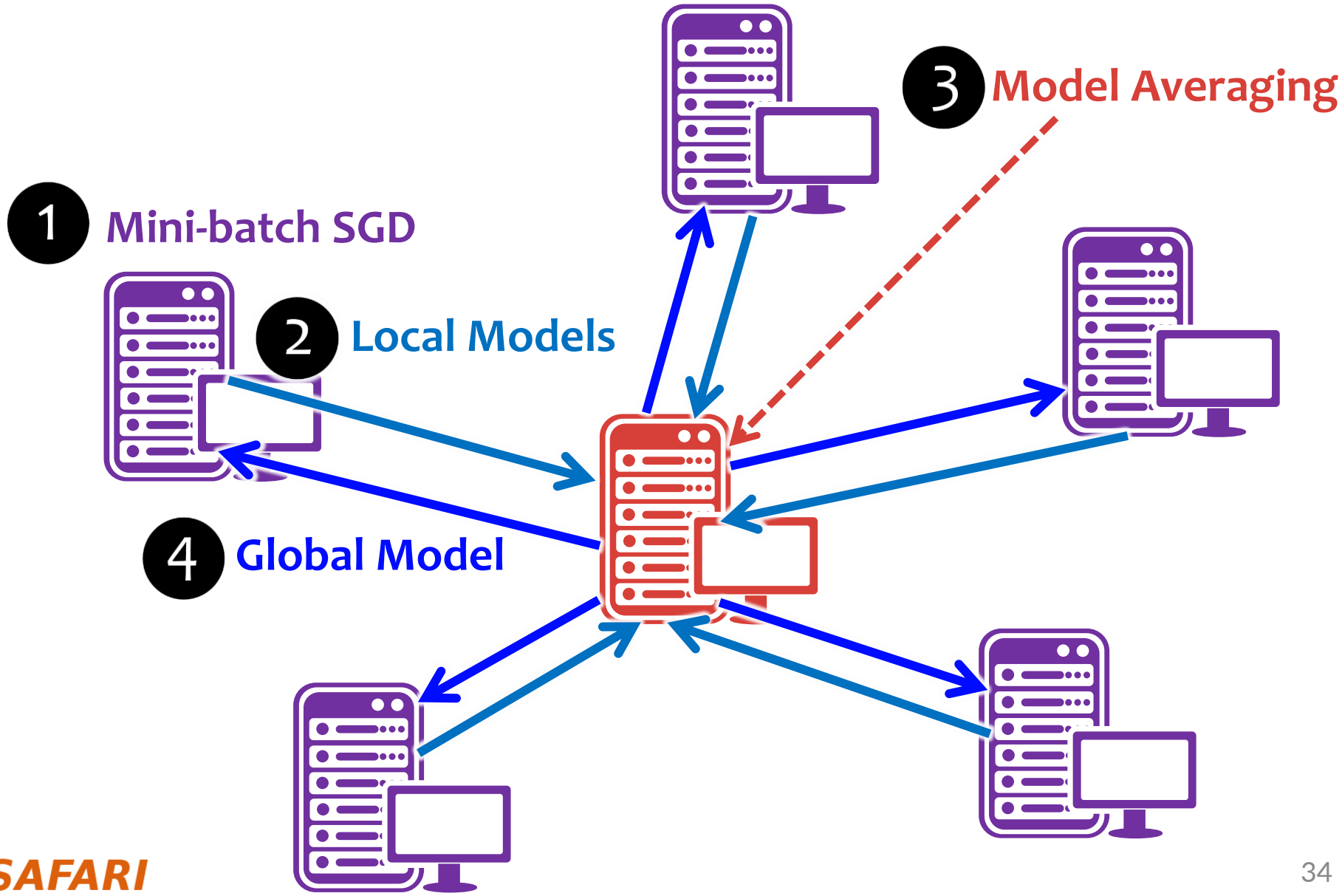
ETH zürich

THE UNIVERSITY OF CHICAGO

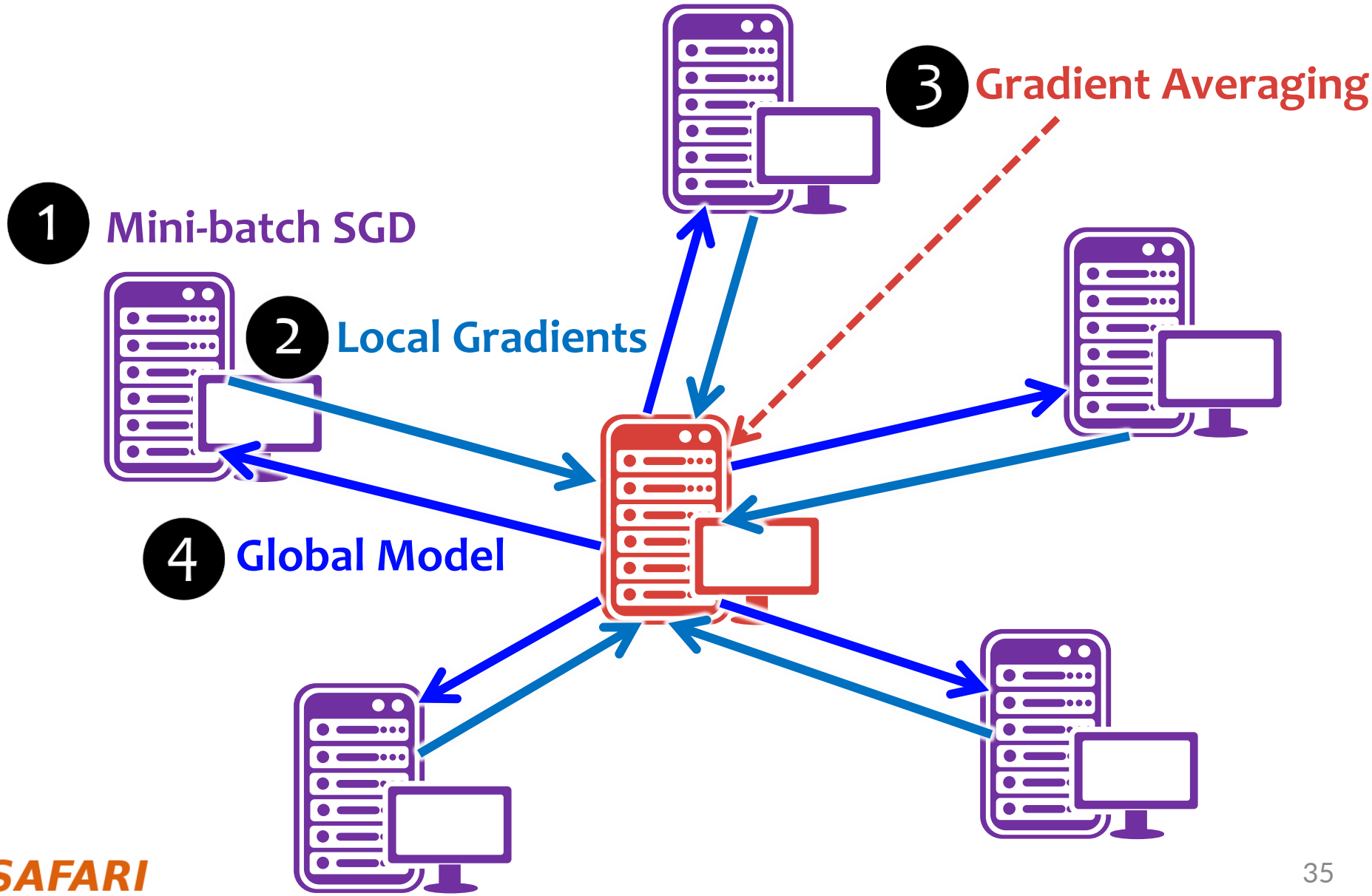
NVIDIA

Backup Slides

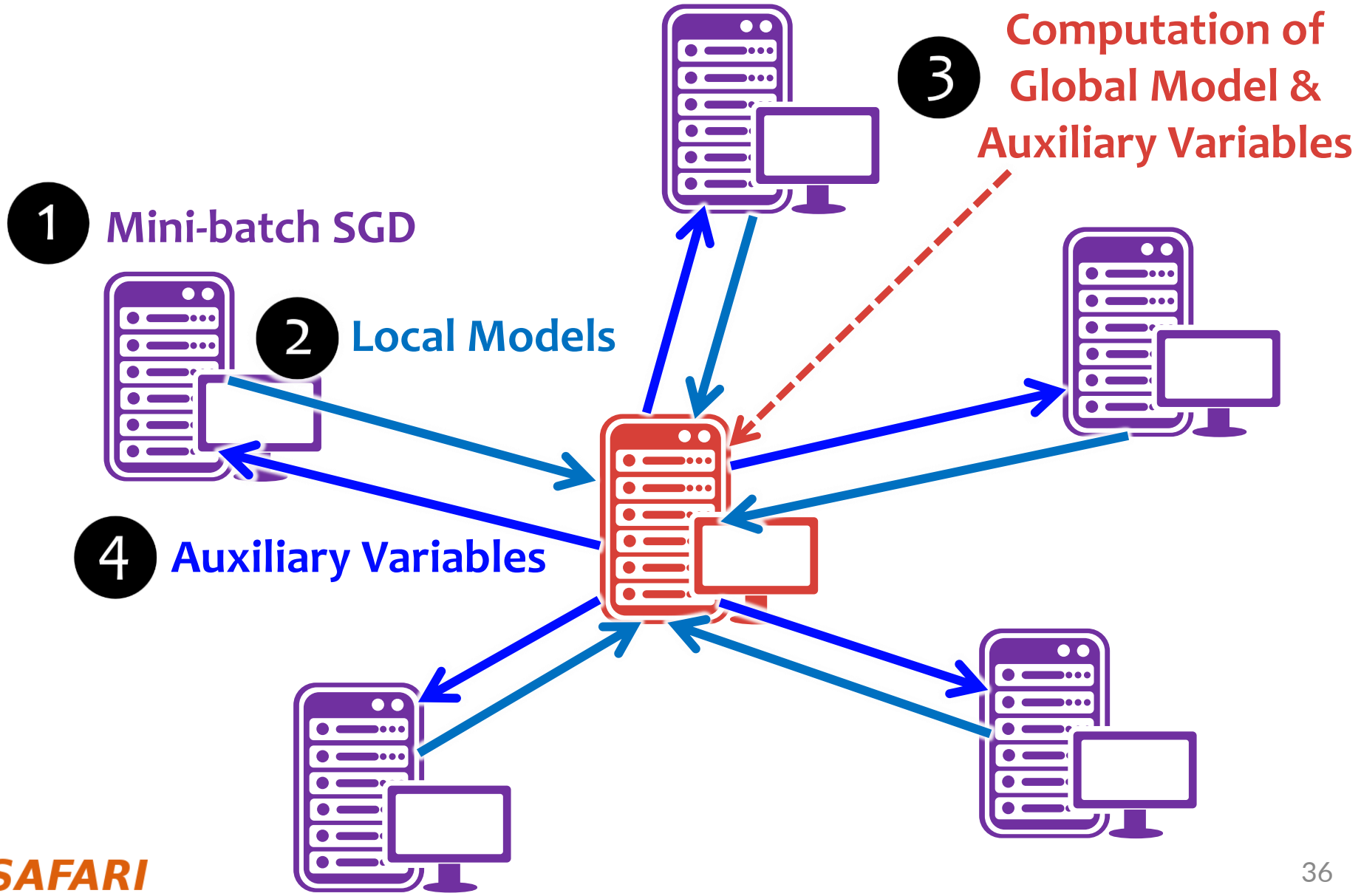
Mini-batch SGD with Model Averaging (MA-SGD)



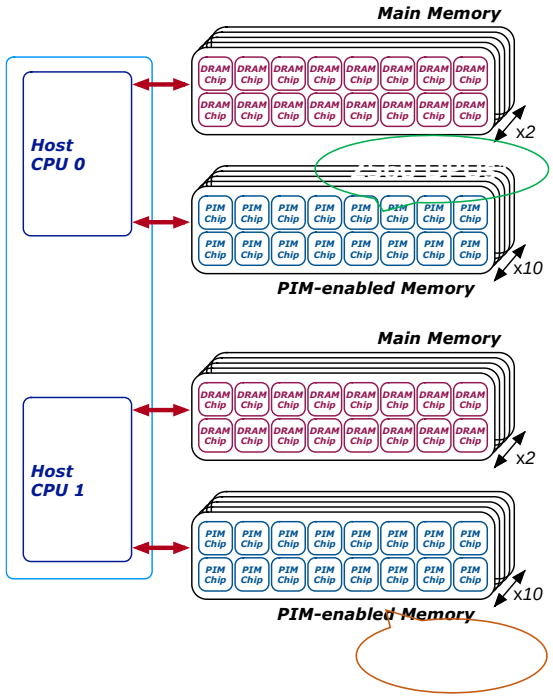
Mini-batch SGD with Gradient Averaging (GA-SGD)

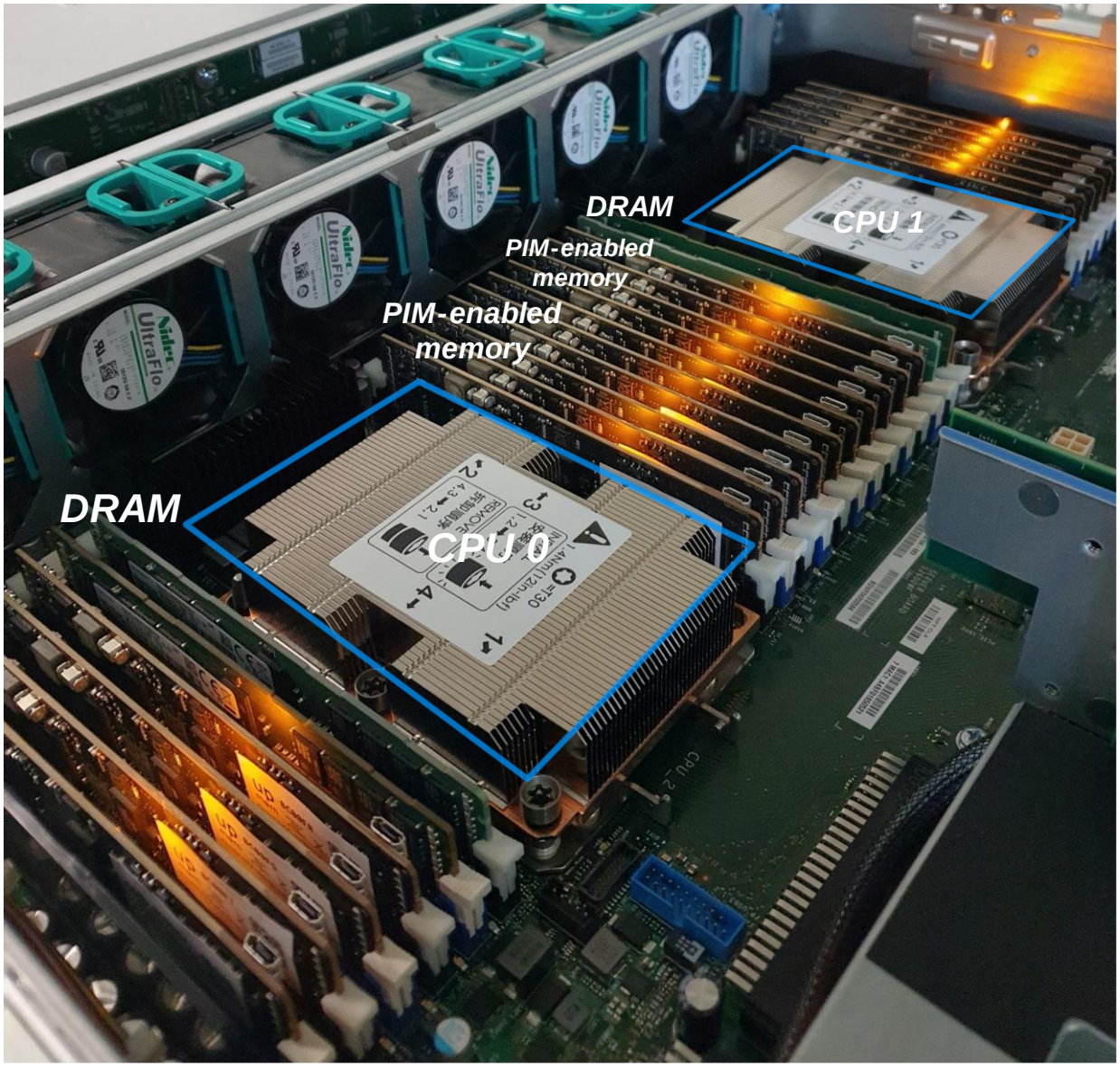


Alternating Direction Method of Multipliers (ADMM)



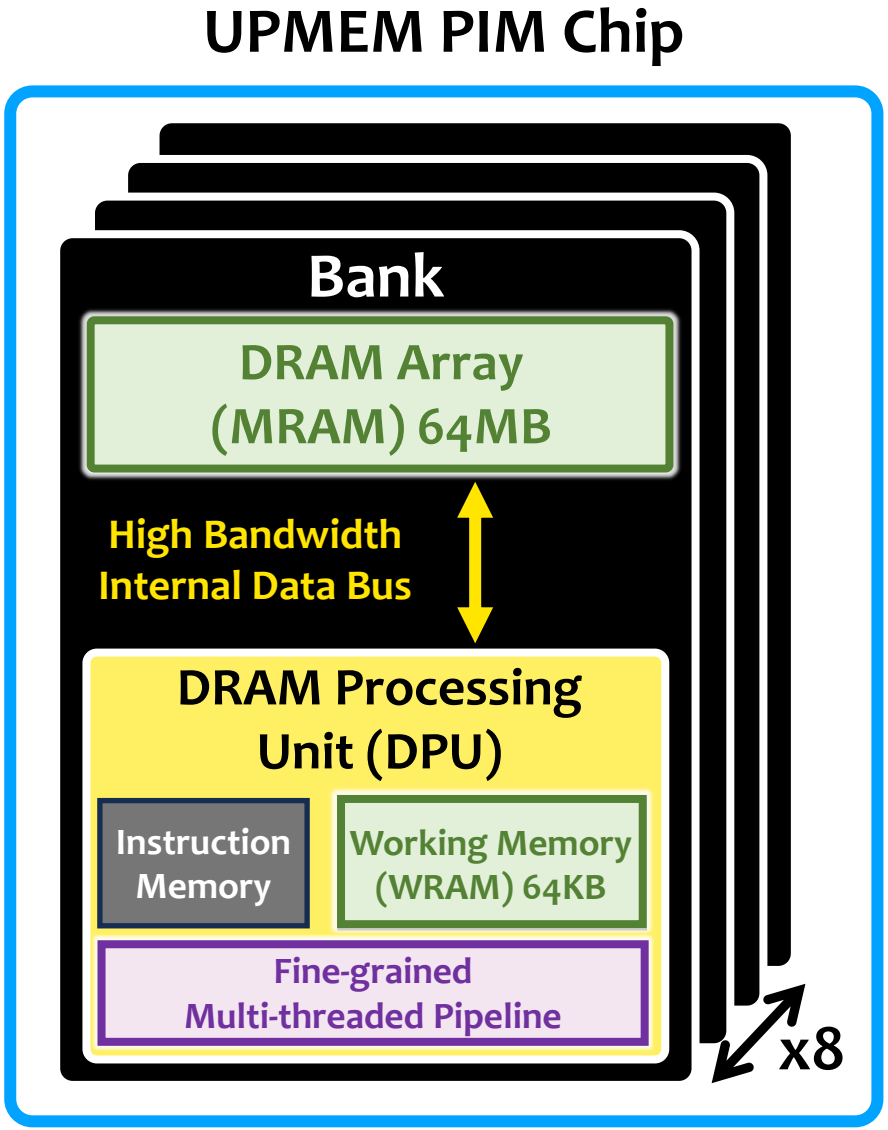
2,560-DPU UPMEM PIM System





- 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

PIM Programming and Execution Model



- **DPU programs** are written in **C**
  - UPMEM SDK
  - Runtime libraries
- **Execution model** of a DPU is based on the **Single-Program-Multiple-Data (SPMD)** paradigm
- Each DPU can run up to 24 **tasklets**
  - Assigned ***statically*** at **compile-time**
- Tasklets assigned to the same DPU **share MRAM** and **WRAM**

UPMEM PIM System Implementation

- **Data Partitioning:**
  - For both **MA-SGD** and **ADMM**, each DPU's partition consists of **multiple mini-batches** of the training data
  - For **GA-SGD**, each partition consists of a **fraction of all the mini-batches** of the training data

- **Synchronization:**
  - For **MA-SGD**, each DPU only processes **one mini-batch** from its assigned training data partition and updates its local model before synchronization on the host
  - For **GA-SGD**, each DPU **computes intermediate gradients** from its assigned fraction of one mini-batch before synchronization on the host
  - For **ADMM**, each DPU processes **all assigned mini-batches** and updates its local model for every mini-batch

Experiment Implementation Details

- **Data Format:**
  - **UPMEM PIM system** uses quantized **training data** and **models** both represented **32-bit fixed-point format**
    → **Floating-point operations** are **not natively supported**
    → **Quantization** is necessary to **enable fixed-point operations**
  - **Baseline implementations** use the **FP32 floating-point format**
    → **Natively supported**
    → **Higher accuracy**

Experiment Implementation Details

- **Regularization:**
  - Apply standard **regularization techniques**
    → **Achieve lower generalization errors**

- **Batch Size:**
  - For each experiment, **we tune the batch size** to ensure
    → **High accuracy**
    → **High performance** in terms of total training time
    → **Fair comparison** of algorithms & architectures

Experiment Implementation Details

- **Hyperparameter Tuning:**
  - For **all** evaluated workloads, we **tune** the **learning rates** and **regularization terms**
  - **All** tested **hyperparameters** along with our **complete codebase** are **open source** at https://github.com/CMU-SAFARI/PIM-Opt

- **Datasets:**
  - **YFCC100M-HNfc6:** Small and dense model
  - **Criteo 1 TB Click Logs:** Large and sparse model

System Configurations

UPMEM PIM System	
Processor	2x Intel Xeon Silver 4215 8-core processor @ 2.50GHz
Main Memory	256 GB total capacity 4×64 GB DDR4 (RDIMMs)
PIM-Enabled Memory	160 GB total capacity 20×8 GB UPMEM PIM modules, 2560 DPUs, 2 ranks per module, 8 chips per rank, 8 DPUs per chip 350 MHz DPU clock frequency
CPU Baseline System	
Processor	2x AMD EPYC 7742 64-core processor @ 2.25GHz
Main Memory	1TB total capacity 32×32 GB DDR4 (RDIMMs)
GPU Baseline System	
Processor	2x Intel Xeon Gold 5118 12-core processor @ 2.30GHz
Main Memory	512 GB total capacity 16×32 GB DDR4 (RDIMMs)
GPU	1× NVIDIA A100 (PCIe, 80 GB)

Experiment Implementation Details

- **Hyperparameter Tuning:**
  - For **all** evaluated workloads, we **tune** the **learning rates** and **regularization terms**
  - **All** tested **hyperparameters** along with our **complete codebase** are **open source** at https://github.com/CMU-SAFARI/PIM-Opt

- **Initialization:**
  - For ***all*** implementations, the **training data & model parameters *initially* reside in main memory**
  - For **UPMEM PIM system** and **GPU baseline** experiments, the **initialization phase includes transferring the data from the main memory** to the **PIM DRAM bank** and the **GPU global memory**

Datasets

- **YFCC100M-HNfc6:**
  - Popular multimedia dataset that consists of 97M samples
  - Each sample has 4096 floating-point dense features and a collection of tags
  - We randomly sample and shuffle data points and turn this subset into a binary classification task
  - The total size of model parameters is 4 KB

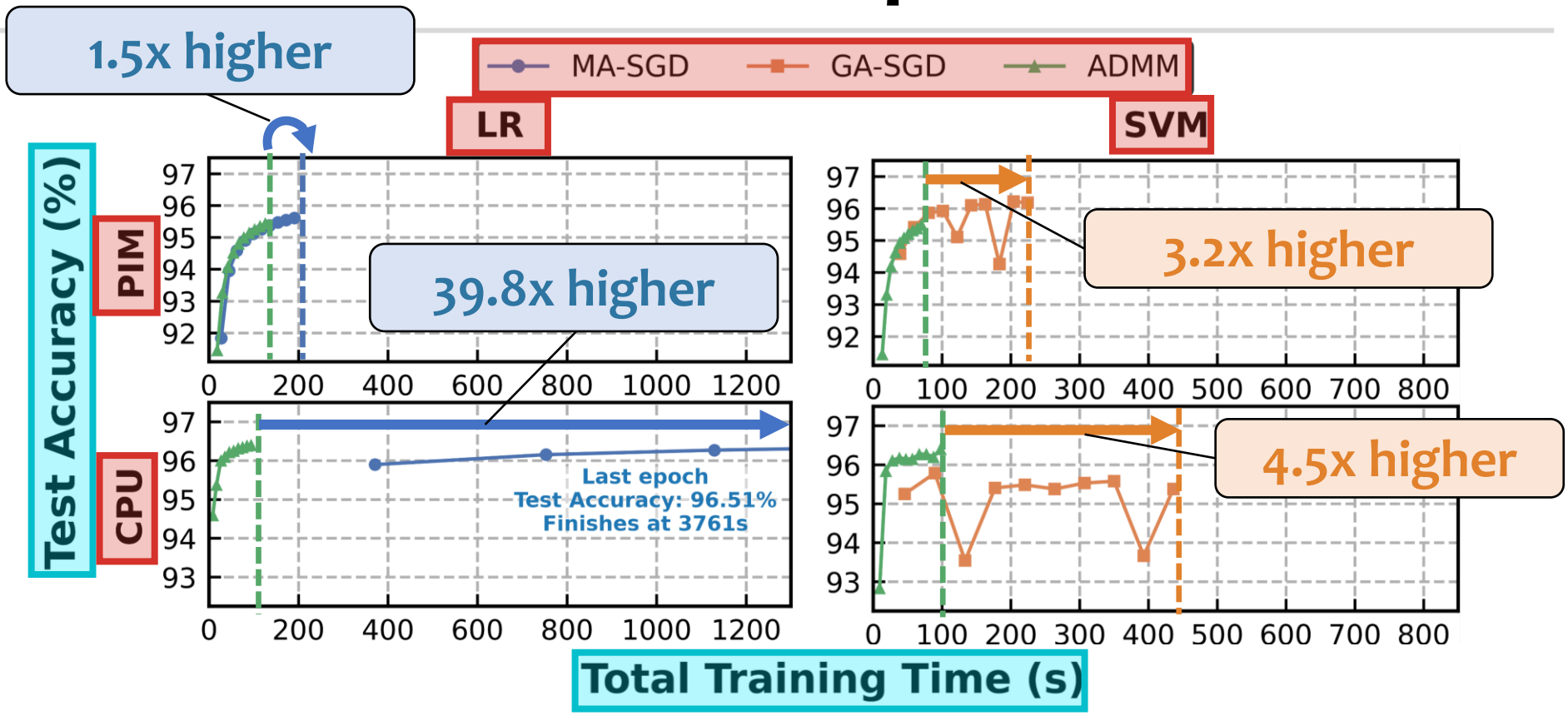
Datasets

- **Criteo 1TB Click Logs (Criteo):**
  - Criteo is a popular click-through rate prediction dataset that consists of 4.37 b high-dimensional sparse samples with 1M features
  - Each data point consists of label and 39 categorical
  - While data points only consist of 40 parameters, the models/gradients consist of 1M variables
  - The dataset is highly imbalanced
  - We randomly sample and shuffle the dataset
  - We use the area under the receiver operating characteristics curve (AUC score) to assess the generalization capabilities of models trained on Criteo
  - The total size of the model parameters is 4MB

Datasets Configurations

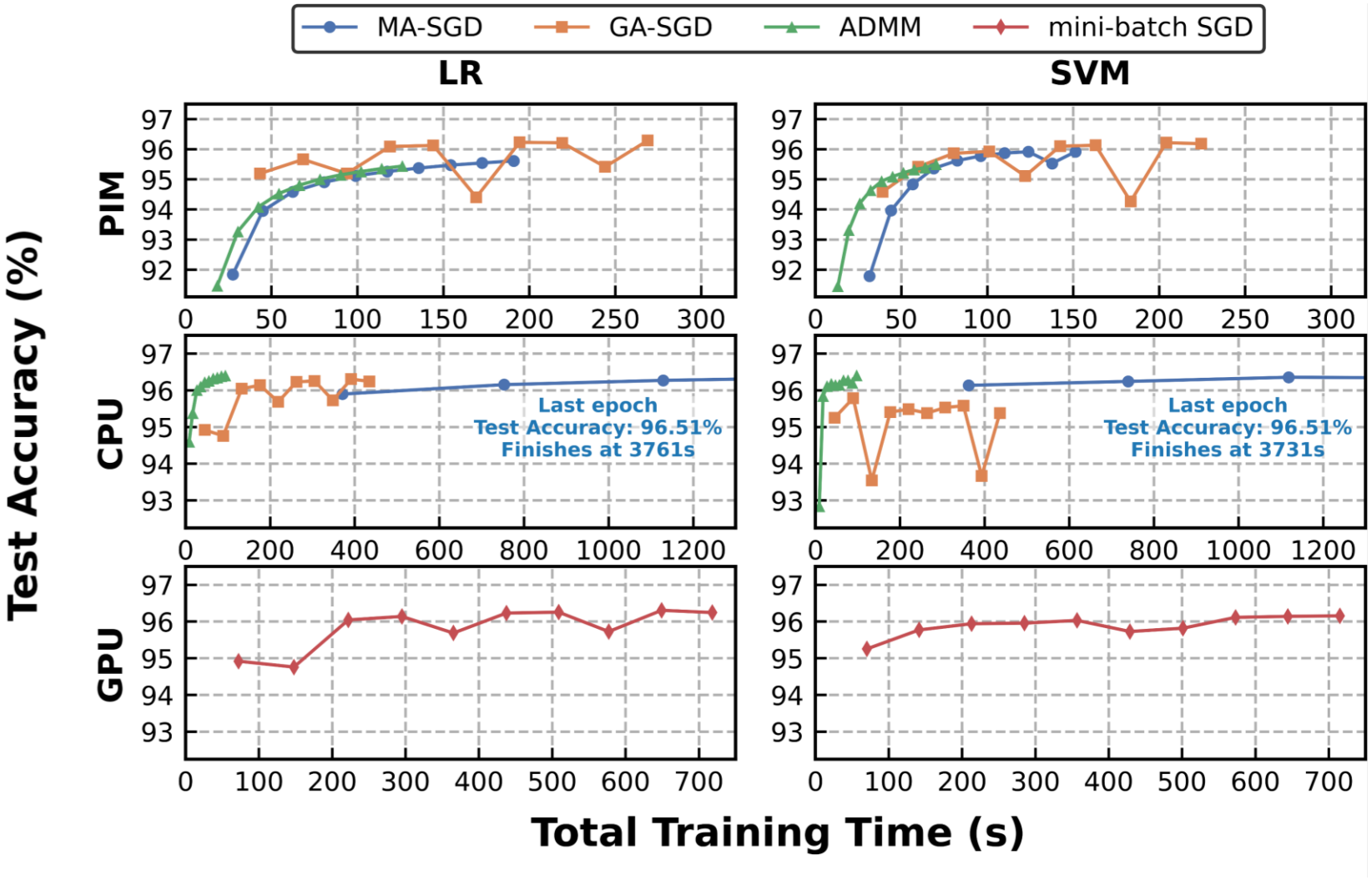
YFCC100M-HNfc6				
# Workers	# Training samples	Training size (GB)	# Test samples	Test size (GB)
256 DPUs	851'968	13.96	212'992	3.49
512 DPUs	1'703'936	27.92	425'984	6.98
1024 DPUs	3'407'872	55.83	851'968	13.96
2'048 DPUs	6'815'744	111.67	1'703'936	27.92
128 CPU threads	6'815'744	111.67	1'703'936	27.92
1 GPU	6'815'744	111.67	1'703'936	27.92
Criteo				
# Workers	# Training samples	Training size (GB)	# Test samples	Test size (GB)
256 DPUs	50'331'648	8.05	178'236'537	28.52
512 DPUs	100'663'296	16.11	178'236'537	28.52
1'024 DPUs	201'326'592	32.21	178'236'537	28.52
2'048 DPUs	402'653'184	64.42	178'236'537	28.52
128 CPU threads	402'653'184	64.42	178'236'537	28.52
1 GPU	402'653'184	64.42	178'236'537	28.52

PIM Performance Comparison

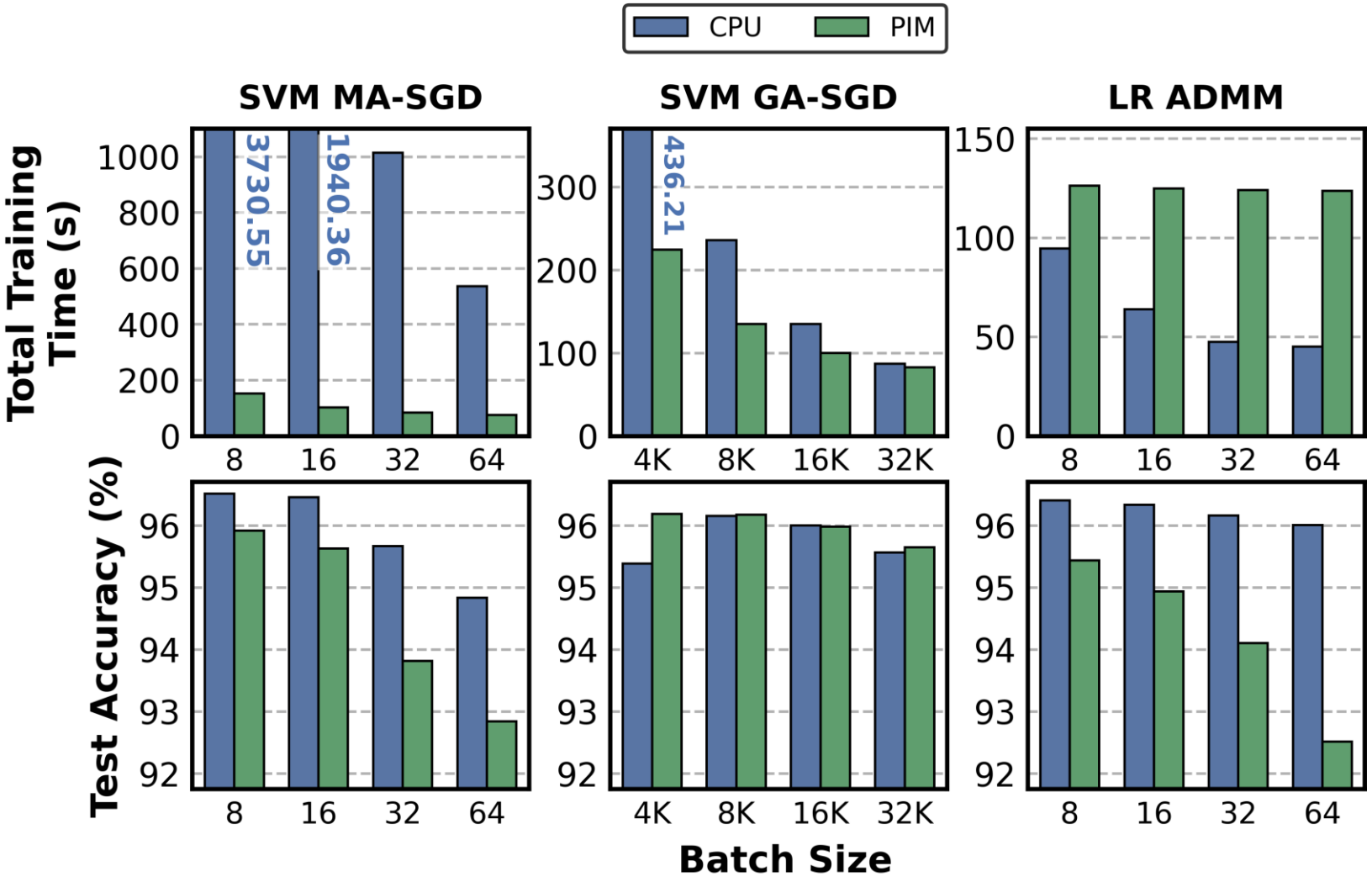


- **Difference** in total training time between **MA-SGD** and **ADMM** is **significantly lower** on **the UPMEM PIM** compared to the **CPU**
- **GA-SGD is slower** than **ADMM** for **all configurations** of LR, SVM, the UPMEM PIM, and the CPU

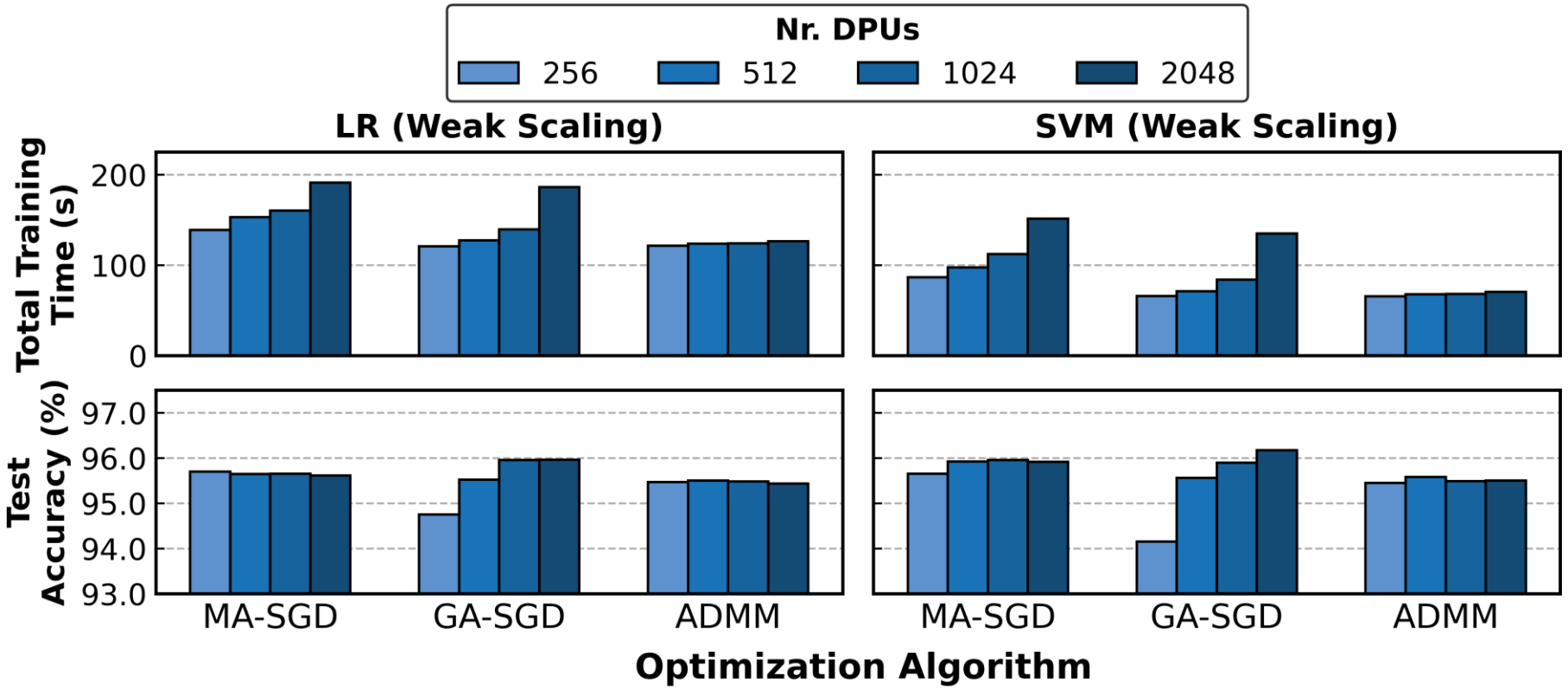
YFCC100M: Performance Comparison



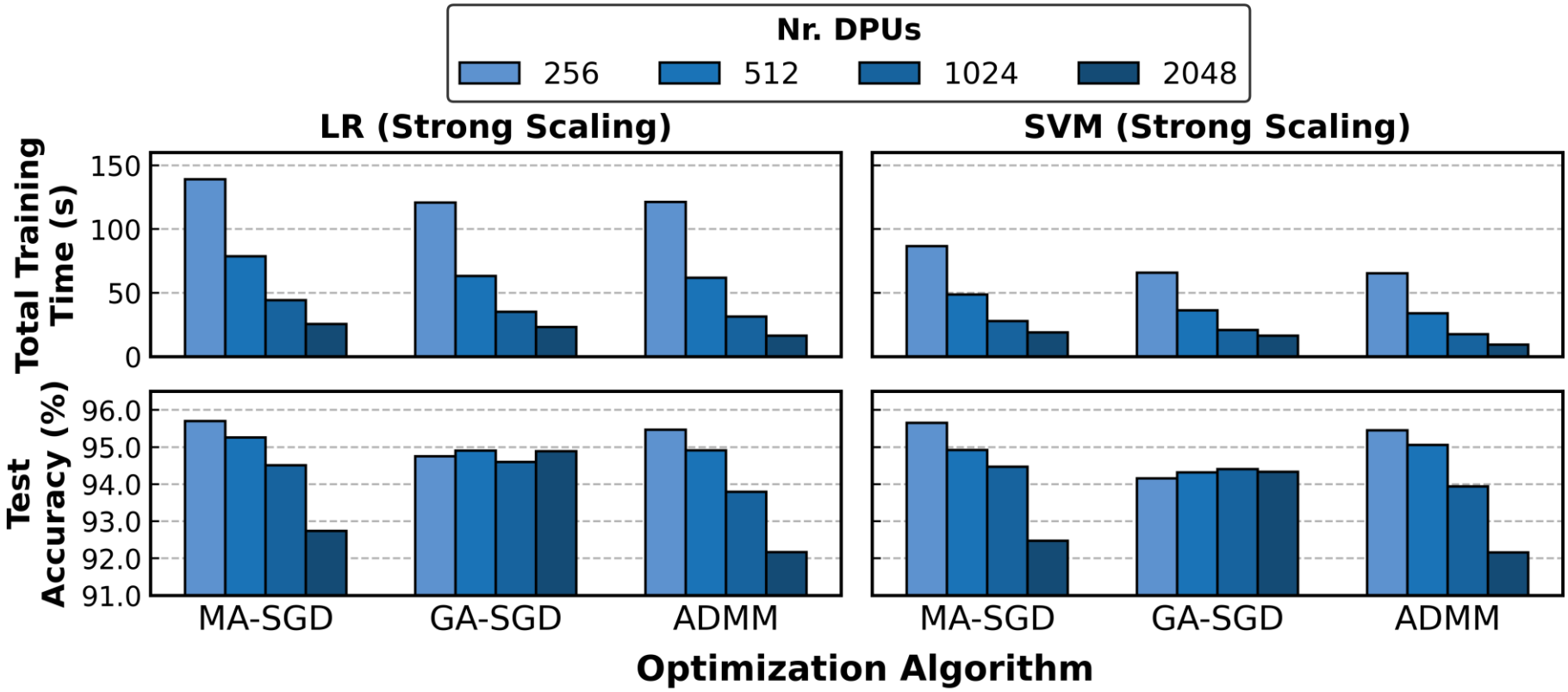
YFCC100M: Batch Size



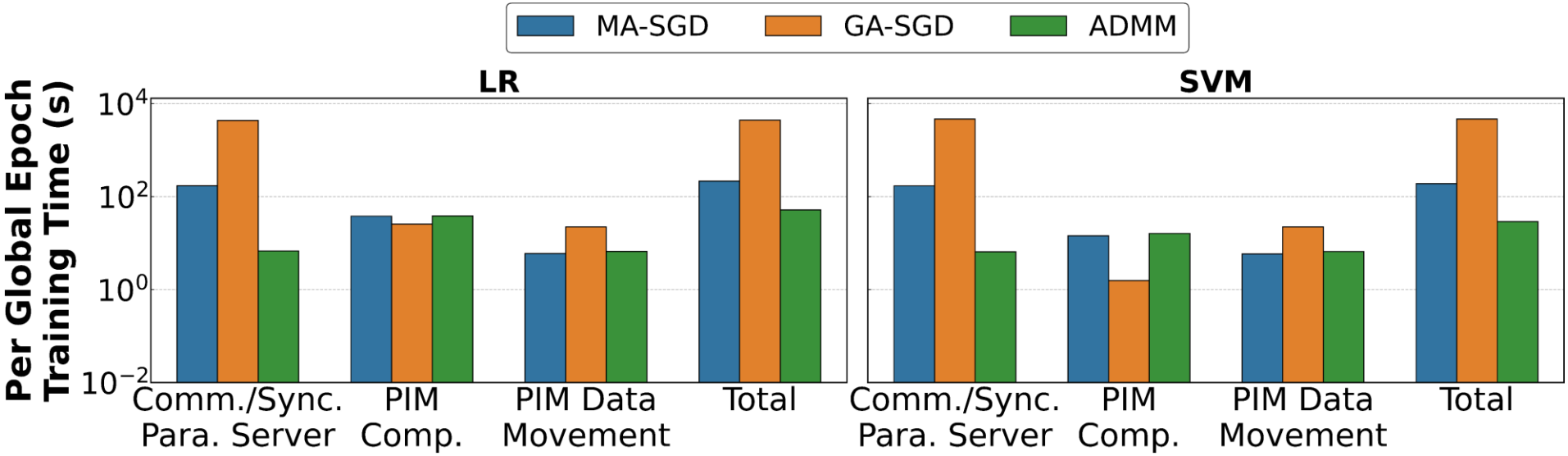
YFCC100M: Weak Scaling



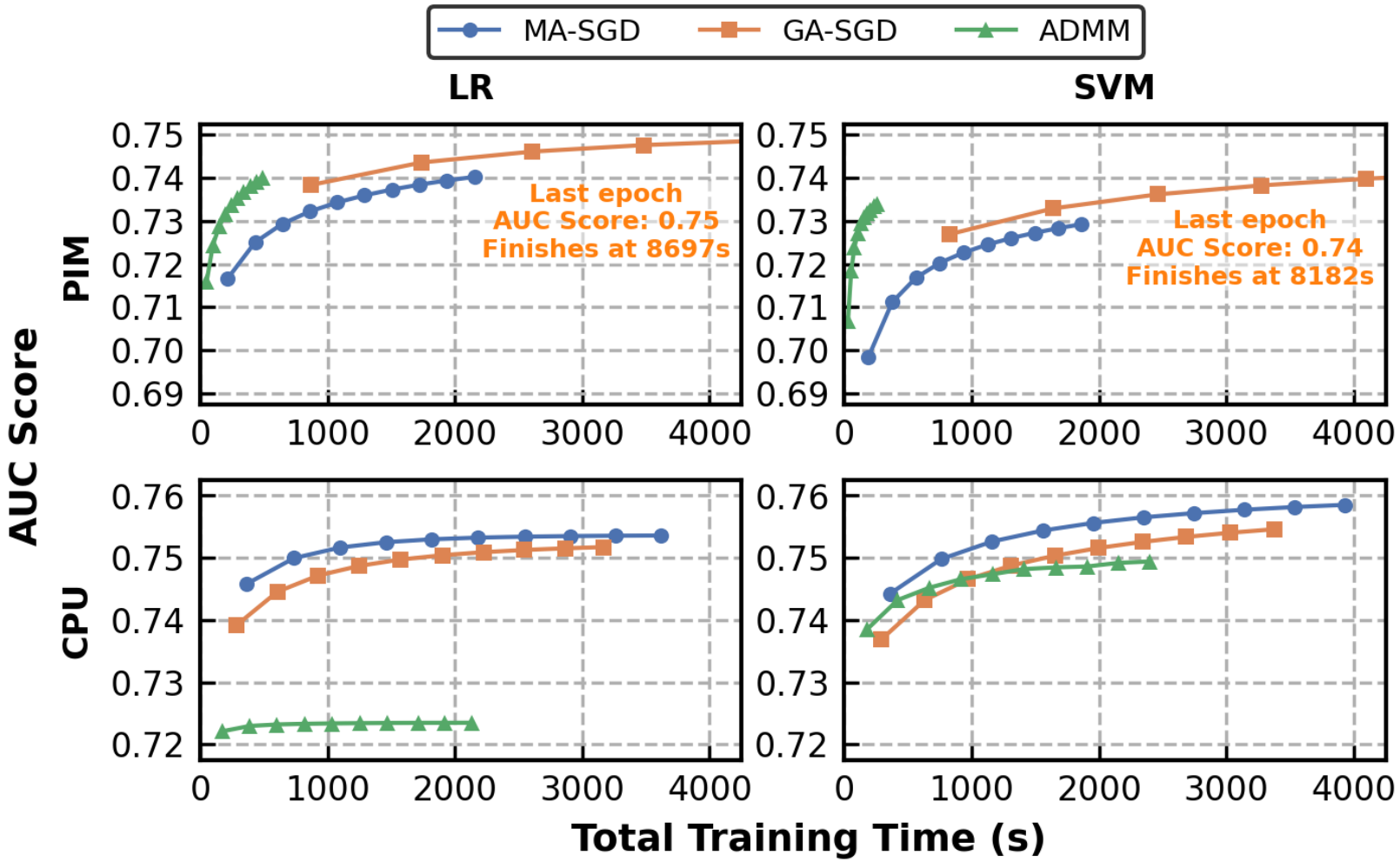
YFCC100M: Strong Scaling



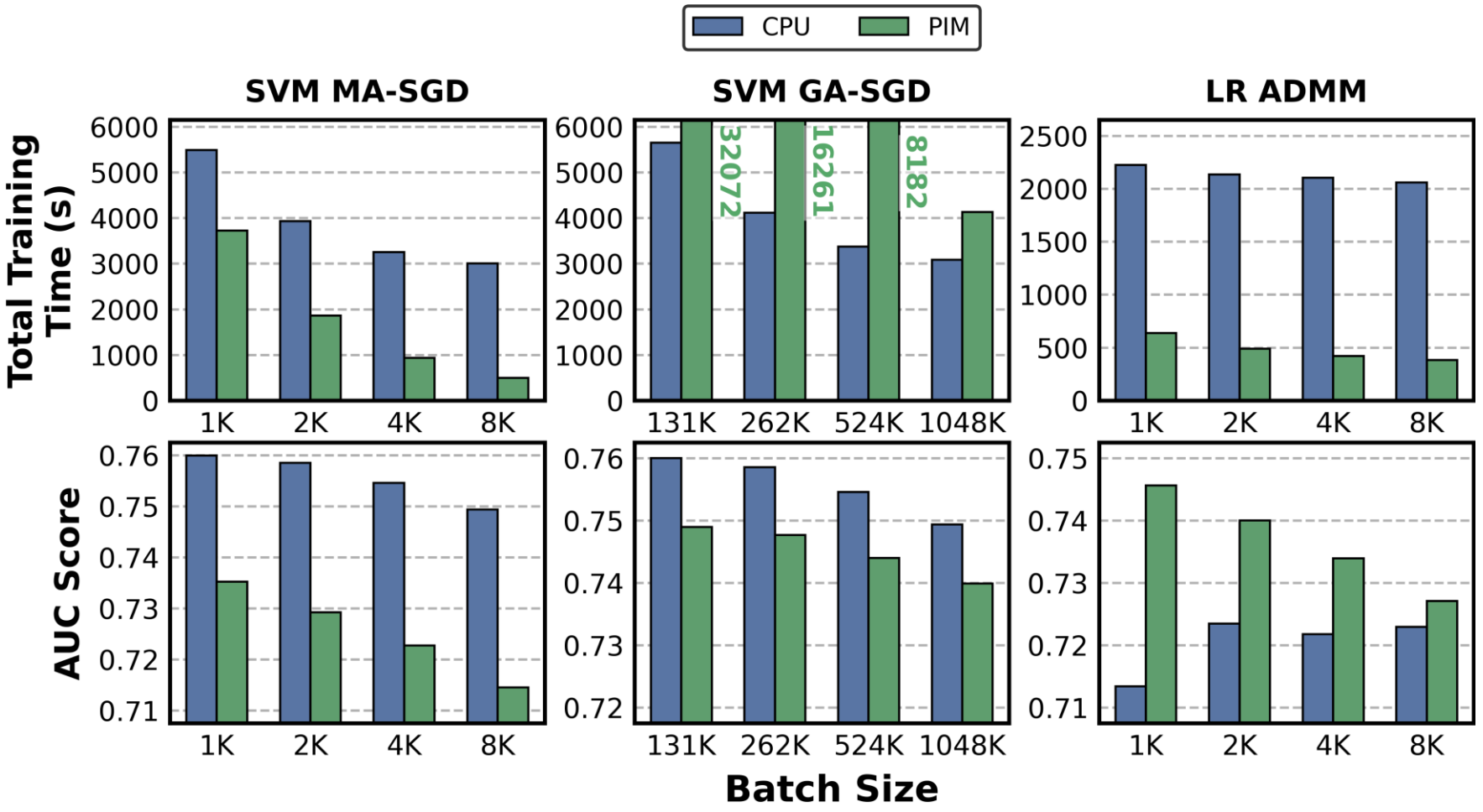
Criteo: PIM Performance Breakdown



Criteo: PIM Performance Comparison



Criteo: Batch Size



Criteo: Weak Scaling