Workshop on Computing with Unconventional Technologies (CUT 2021)

Benchmarking Memory-Centric Computing Systems:

Analysis of Real Processing-in-Memory Hardware

Juan Gómez Luna, Izzat El Hajj,
Ivan Fernandez, Christina Giannoula,
Geraldo F. Oliveira, Onur Mutlu

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

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

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

ETH Zürich

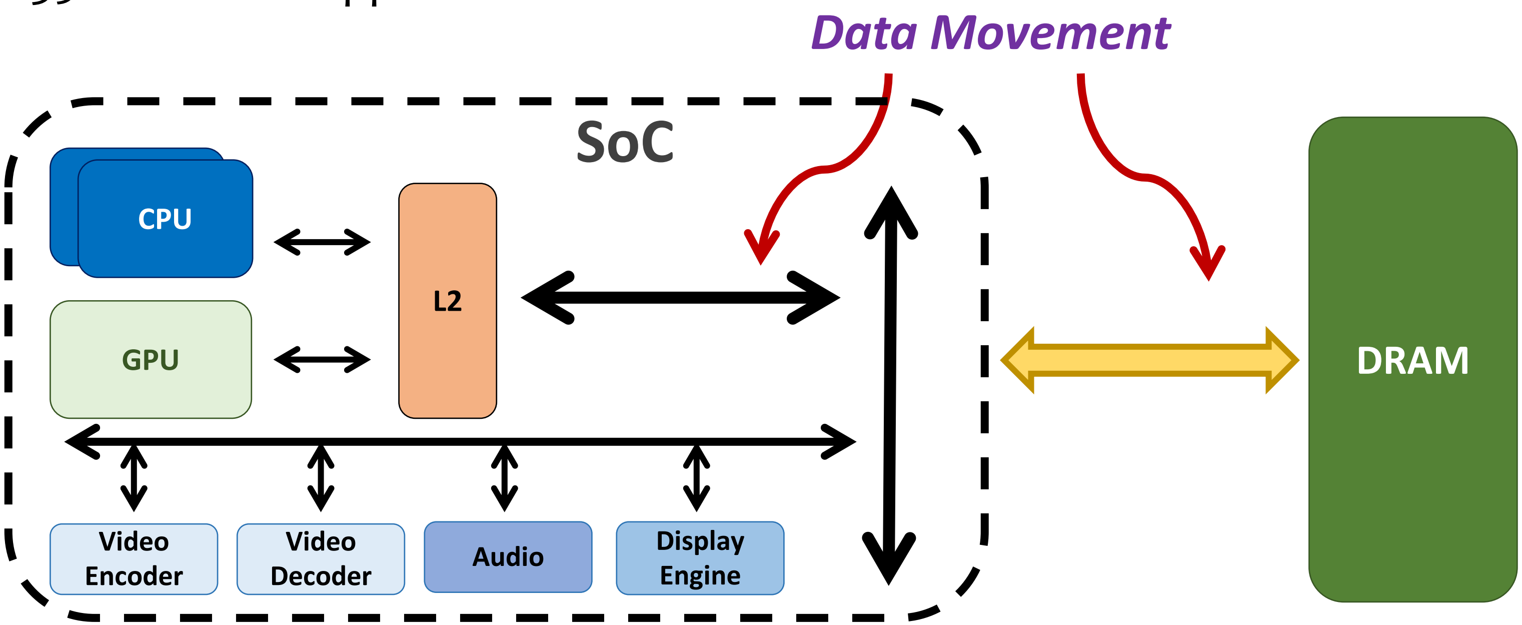
SAFARI

Executive Summary

- Data movement between memory/storage units and compute units is a major contributor to execution time and energy consumption
- Processing-in-Memory (PIM) is a paradigm that can tackle the *data movement bottleneck*
  - Though explored for +50 years, technology challenges prevented the successful materialization
- UPMEM has designed and fabricated the first publicly-available real-world PIM architecture
  - DDR4 chips embedding in-order multithreaded DRAM Processing Units (DPUs)
- Our work:
  - Introduction to UPMEM programming model and PIM architecture
  - Microbenchmark-based characterization of the DPU
  - Benchmarking and workload suitability study
- Main contributions:
  - Comprehensive characterization and analysis of the first commercially-available PIM architecture
  - **PrIM** (Processing-In-Memory) benchmarks:
    - 16 workloads that are memory-bound in conventional processor-centric systems
    - Strong and weak scaling characteristics
  - Comparison to state-of-the-art CPU and GPU
- Takeaways:
  - Workload characteristics for PIM suitability
  - Programming recommendations
  - Suggestions and hints for hardware and architecture designers of future PIM systems
  - PrIM: (a) programming samples, (b) evaluation and comparison of current and future PIM systems

Data Movement in Computing Systems

- Data movement dominates performance and is a major system energy bottleneck
- Total system energy: data movement accounts for
  - 62% in consumer applications✻,
  - 40% in scientific applications★,
  - 35% in mobile applications☆



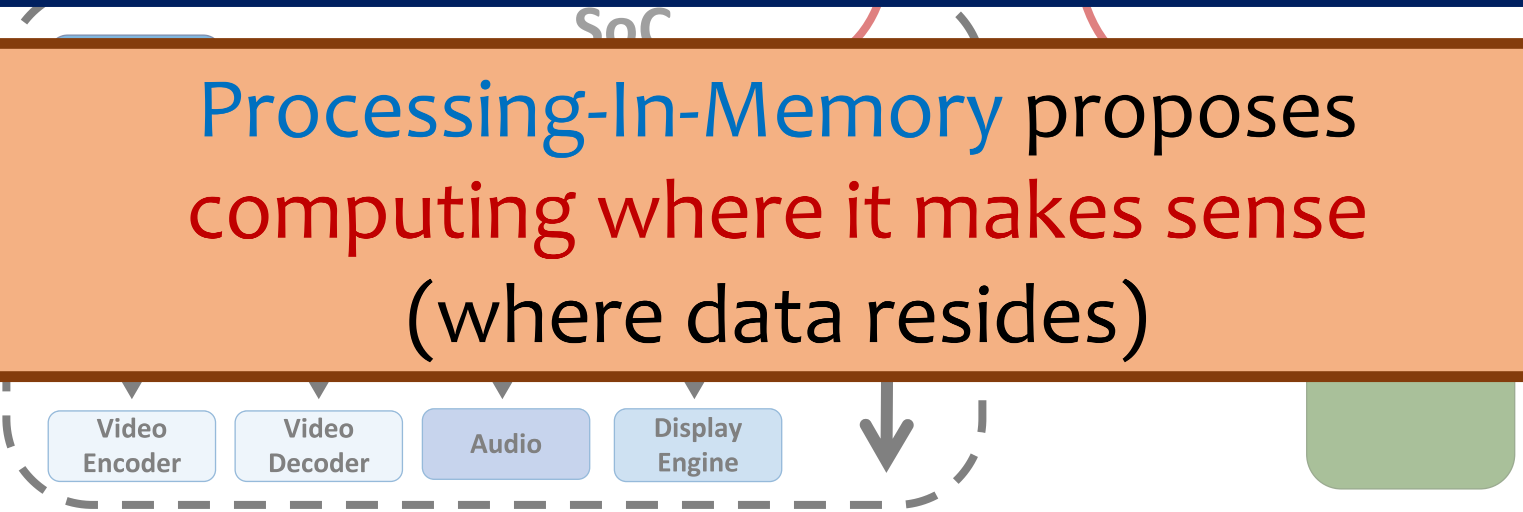
✻ Boroumand et al., “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” ASPLOS 2018
★ Kestor et al., “Quantifying the Energy Cost of Data Movement in Scientific Applications,” IISWC 2013
☆ Pandiyan and Wu, “Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms,” IISWC 2014

Data Movement in Computing Systems

- Data movement dominates performance and is a major system energy bottleneck
- Total system energy: data movement accounts for
  - 62% in consumer applications✻,

Compute systems should be more data-centric

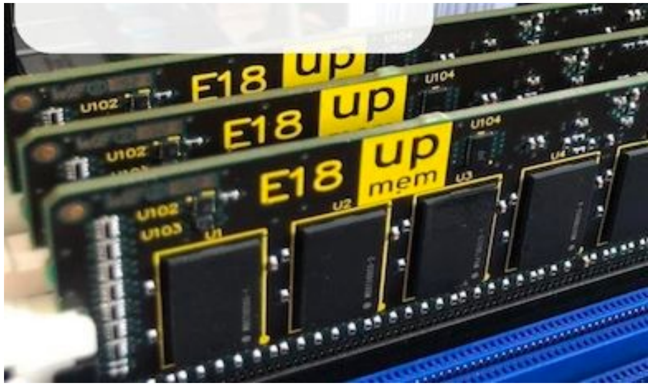
Processing-In-Memory proposes computing where it makes sense (where data resides)



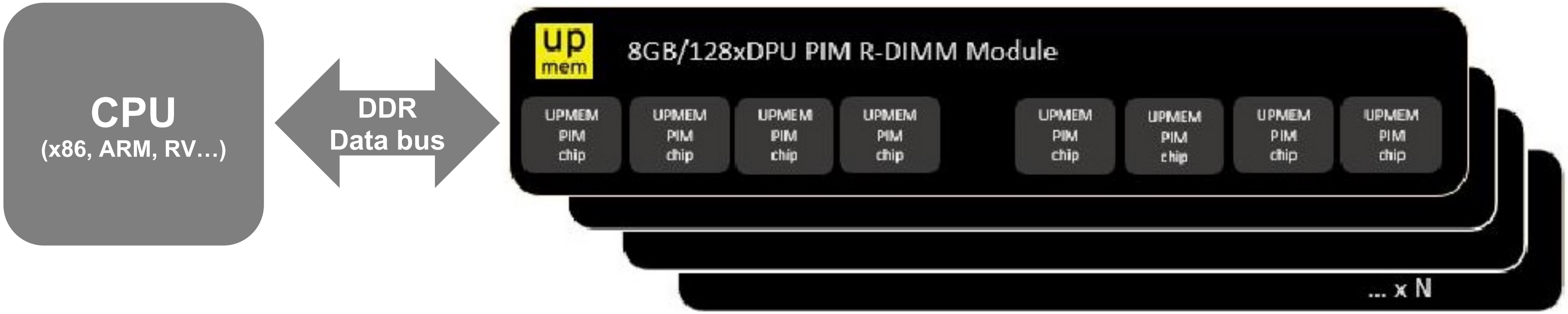
✻ Boroumand et al., “Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks,” ASPLOS 2018
★ Kestor et al., “Quantifying the Energy Cost of Data Movement in Scientific Applications,” IISWC 2013
☆ Pandiyan and Wu, “Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms,” IISWC 2014

UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine
- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.



- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - **Large amounts of** compute & memory bandwidth



https://www.anandtech.com/show/14750/hot-chips-31-analysis-inmemory-processing-by-upmem
https://www.upmem.com/video-upmem-presenting-its-true-processing-in-memory-solution-hot-chips-2019/

PIM Alleviates the Shortcomings of Processor-Centric Systems

KEY TAKEAWAY 4

- UPMEM-based PIM systems **outperform state-of-the-art CPUs in terms of performance** (by 23.2× on 2,556 DPUs for 16 PrIM benchmarks) **and energy efficiency on most of PrIM benchmarks.**

- UPMEM-based PIM systems **outperform state-of-the-art GPUs on a majority of PrIM benchmarks** (by 2.54× on 2,556 DPUs for 10 PrIM benchmarks), and the outlook is even more positive for future PIM systems.

- UPMEM-based PIM systems are **more energy-efficient than state-of-the-art CPUs and GPUs on workloads that they provide performance improvements** over the CPUs and the GPUs.

CUT 2021 Paper

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

Long arXiv Version

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

Observations, Recommendations, Takeaways

GENERAL PROGRAMMING RECOMMENDATIONS

1. Execute on the *DRAM Processing Units* (*DPUs*) **portions of parallel code** that are as long as possible.
2. Split the workload into **independent data blocks**, which the DPUs operate on independently.
3. Use **as many working DPUs** in the system as possible.
4. Launch at least **11 *tasklets* (i.e., software threads)** per DPU.

PROGRAMMING RECOMMENDATION 1

For data movement between the DPU's MRAM bank and the WRAM, **use large DMA transfer sizes when all the accessed data is going to be used**.

KEY OBSERVATION 7

Larger CPU-DPU and DPU-CPU transfers between the host main memory and the DRAM Processing Unit's Main memory (MRAM) banks **result in higher sustained bandwidth**.

KEY TAKEAWAY 1

The UPMEM PIM architecture is fundamentally compute bound. As a result, **the most suitable work- loads are memory-bound.**

Outline

- Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview
- UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth
- PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity
- Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU
- Key Takeaways

Accelerator Model

- UPMEM DIMMs coexist with conventional DIMMs

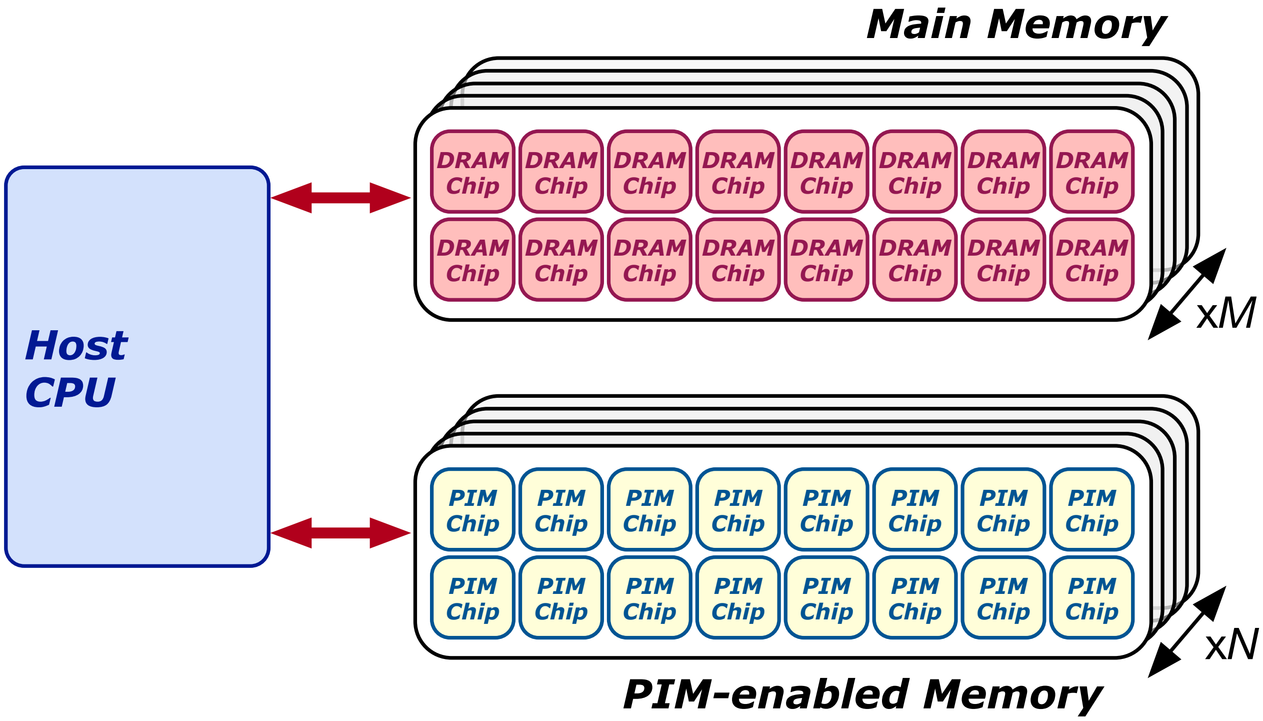
- Integration of UPMEM DIMMs in a system follows an accelerator model

- UPMEM DIMMs can be seen as a loosely coupled accelerator
  - Explicit data movement between the main processor (host CPU) and the accelerator (UPMEM)
  - Explicit kernel launch onto the UPMEM processors

- This resembles GPU computing

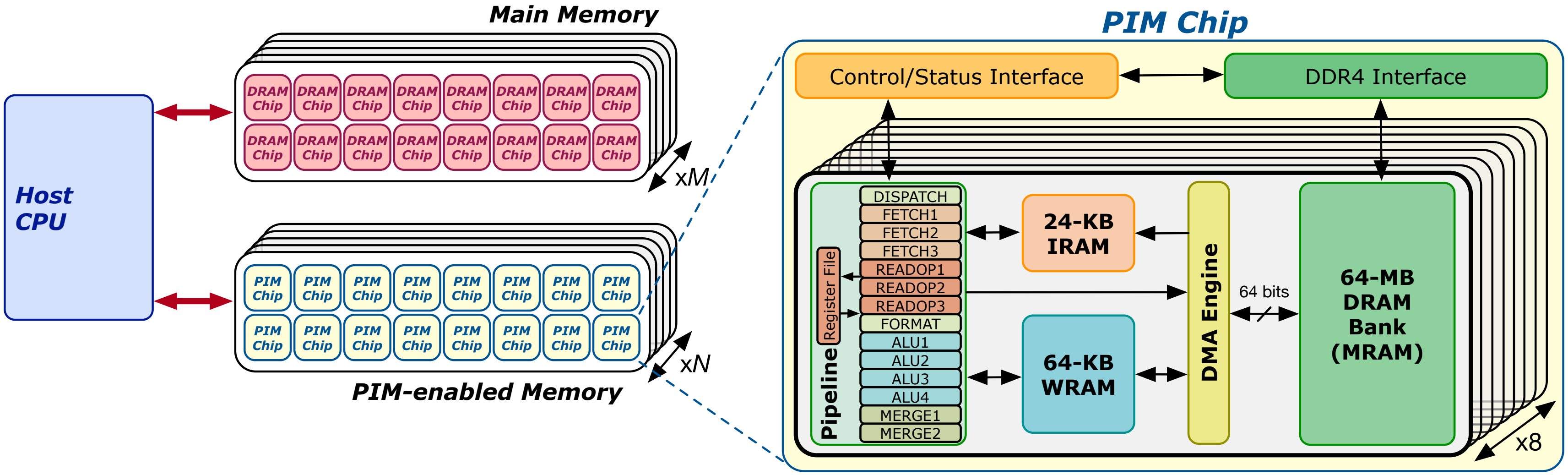
System Organization (I)

- In a UPMEM-based PIM system UPMEM DIMMs coexist with regular DDR4 DIMMs



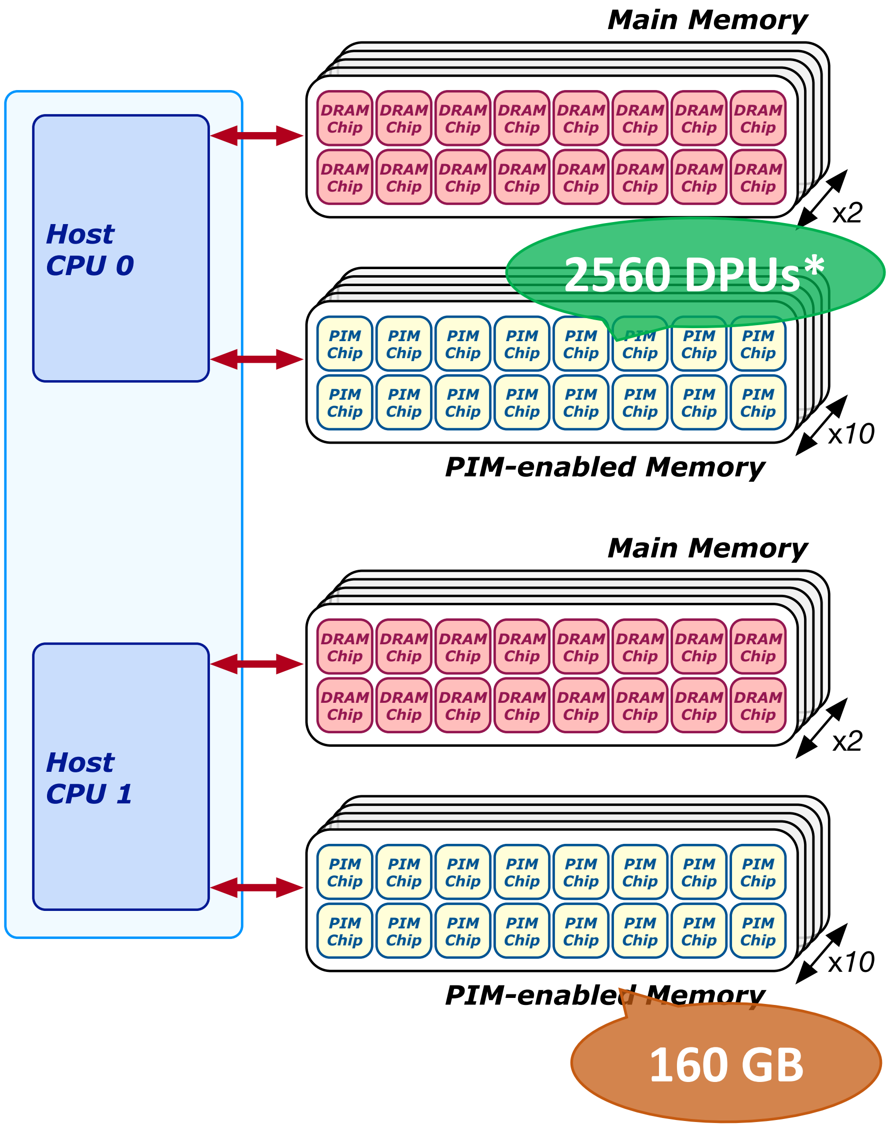
System Organization (II)

- A UPMEM DIMM contains 8 or 16 chips
  - Thus, 1 or 2 ranks of 8 chips each
- Inside each PIM chip there are:
  - 8 64MB banks per chip: Main RAM (MRAM) banks
  - 8 DRAM Processing Units (DPUs) in each chip, 64 DPUs per rank



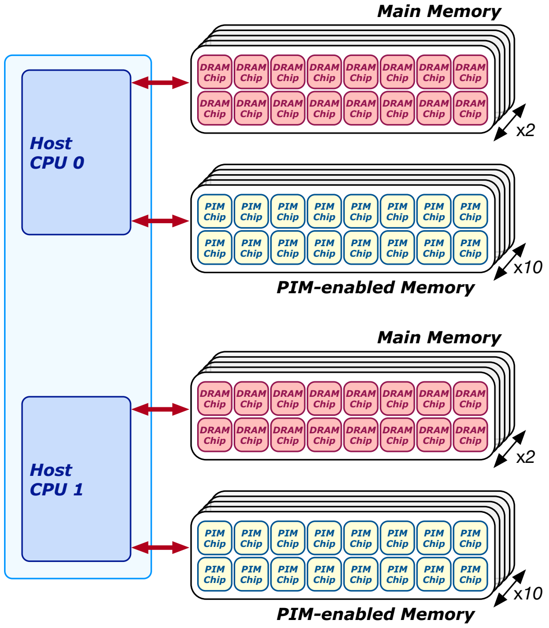
2,560-DPU System (I)

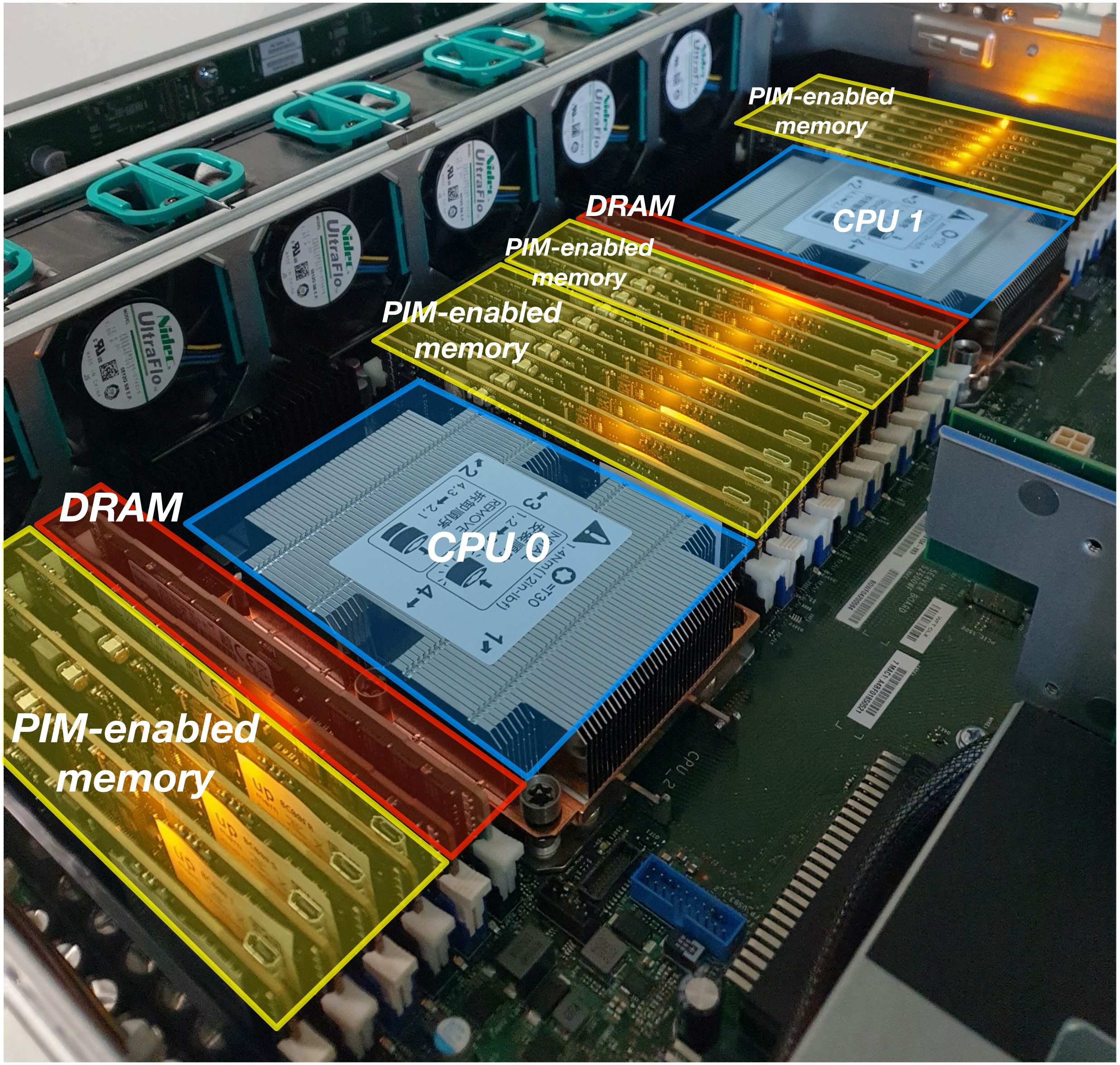
- UPMEM-based PIM system with 20 UPMEM DIMMs of 16 chips each (40 ranks)
  - P21 DIMMs
  - Dual x86 socket
    - UPMEM DIMMs coexist with regular DDR4 DIMMs
    - 2 memory controllers/socket (3 channels each)
    - 2 conventional DDR4 DIMMs on one channel of one controller



* There are 4 faulty DPUs in the system that we use in our experiments. Thus, the maximum number of DPUs we can use is 2,556.

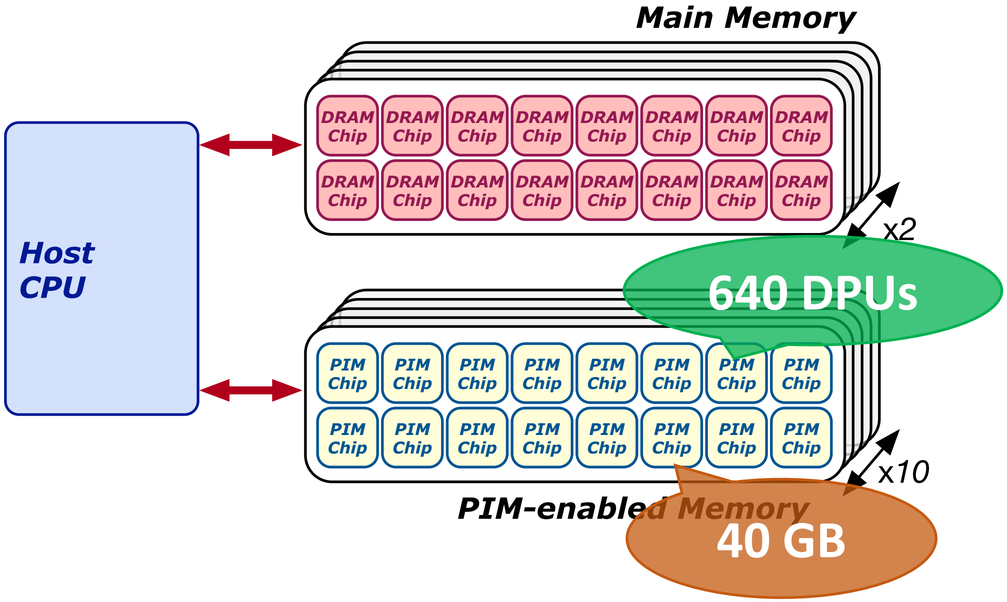
2,560-DPU System (II)





640-DPU System

- UPMEM-based PIM system with 10 UPMEM DIMMs of 8 chips each (10 ranks)
  - E19 DIMMs
  - x86 socket
    - 2 memory controllers (3 channels each)
    - 2 conventional DDR4 DIMMs on one channel of one controller

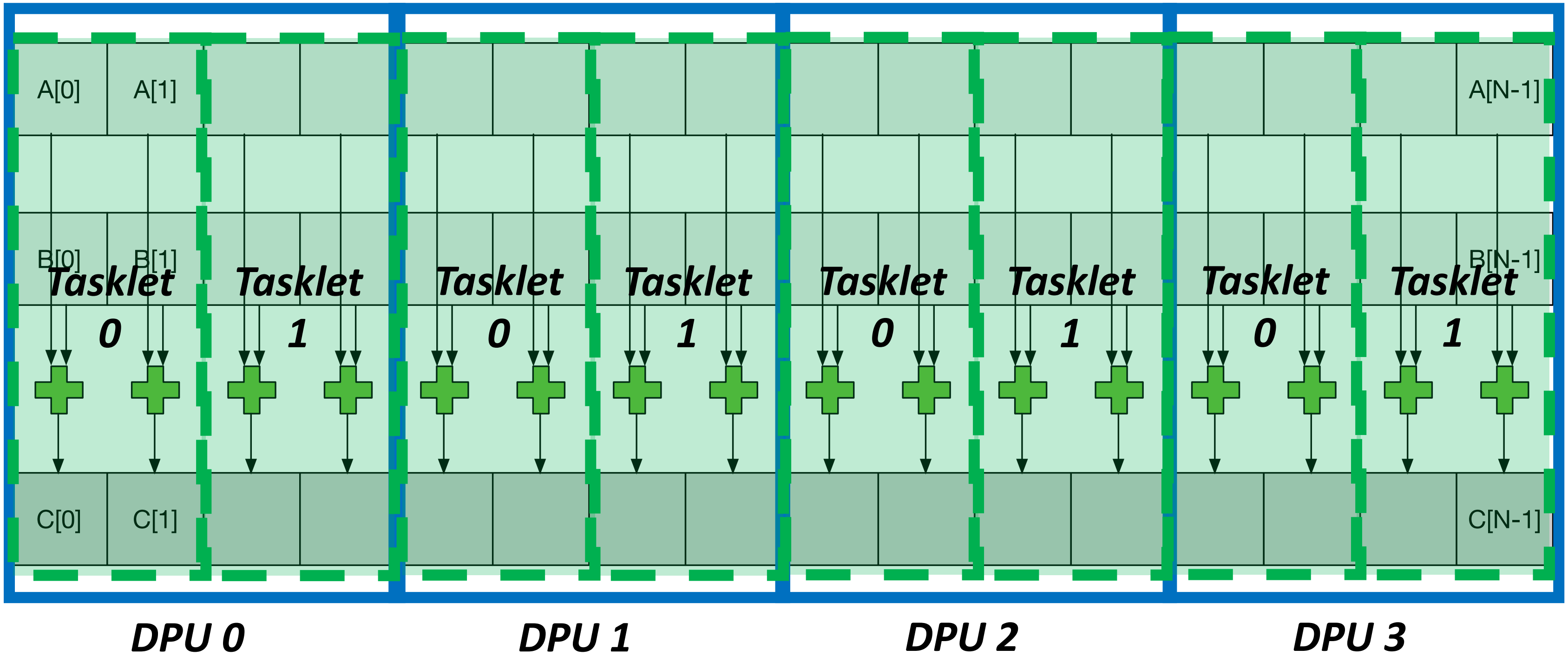


Outline

- Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview
- UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth
- PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity
- Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU
- Key Takeaways

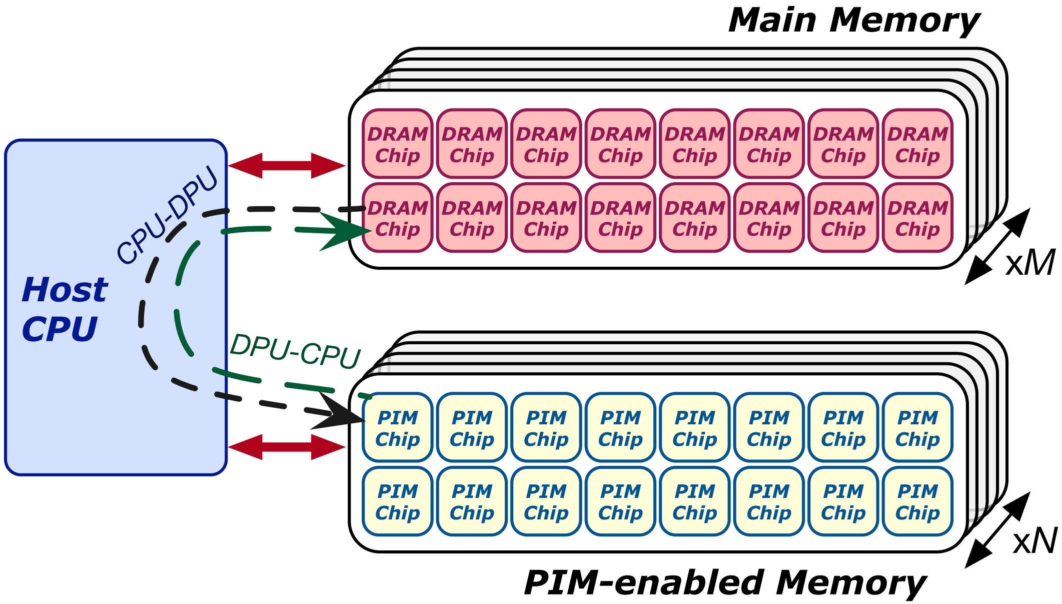
Vector Addition (VA)

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



CPU-DPU/DPU-CPU Data Transfers

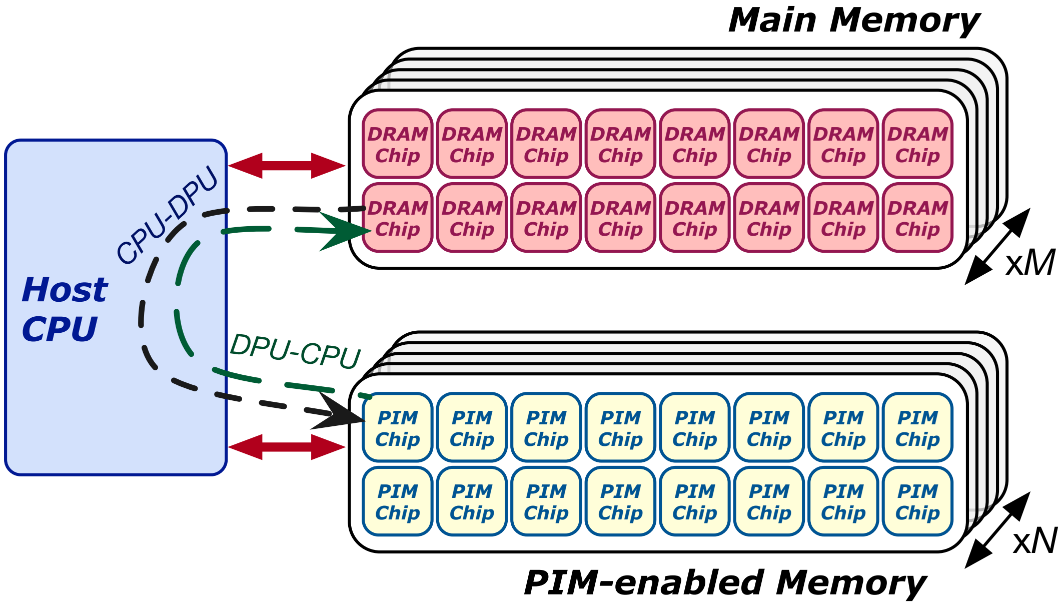
- CPU-DPU and DPU-CPU transfers
  - Between host CPU's main memory and DPUs' MRAM banks



- Serial CPU-DPU/DPU-CPU transfers:
  - A single DPU (i.e., 1 MRAM bank)
- Parallel CPU-DPU/DPU-CPU transfers:
  - Multiple DPUs (i.e., many MRAM banks)
- Broadcast CPU-DPU transfers:
  - Multiple DPUs with a single buffer

Inter-DPU Communication

- There is no direct communication channel between DPUs



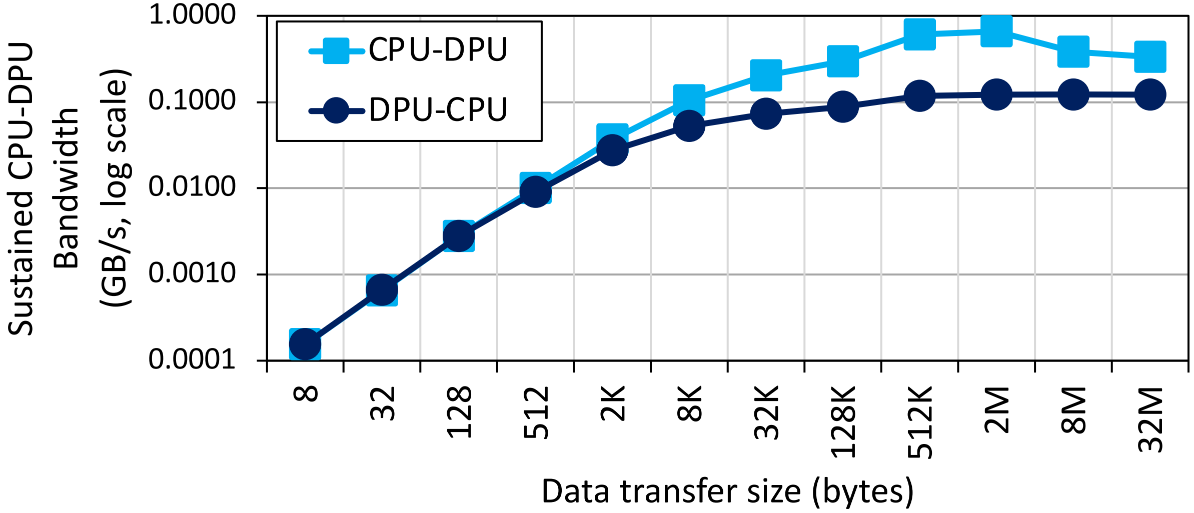
- Inter-DPU communication takes places via the host CPU using CPU-DPU and DPU-CPU transfers
- Example communication patterns:
  - Merging of partial results to obtain the final result
    - Only DPU-CPU transfers
  - Redistribution of intermediate results for further computation
    - DPU-CPU transfers and CPU-DPU transfers

How Fast are these Data Transfers?

- With a microbenchmark, we obtain the sustained bandwidth of all types of CPU-DPU and DPU-CPU transfers
- Two experiments:
  - 1 DPU: variable CPU-DPU and DPU-CPU transfer size (8 bytes to 32 MB)
  - 1 rank: 32 MB CPU-DPU and DPU-CPU transfers to/from a set of 1 to 64 MRAM banks within the same rank
- Preliminary experiments with more than one rank
  - Channel-level parallelism

CPU-DPU/DPU-CPU Transfers: 1 DPU

- Data transfer size varies between 8 bytes and 32 MB

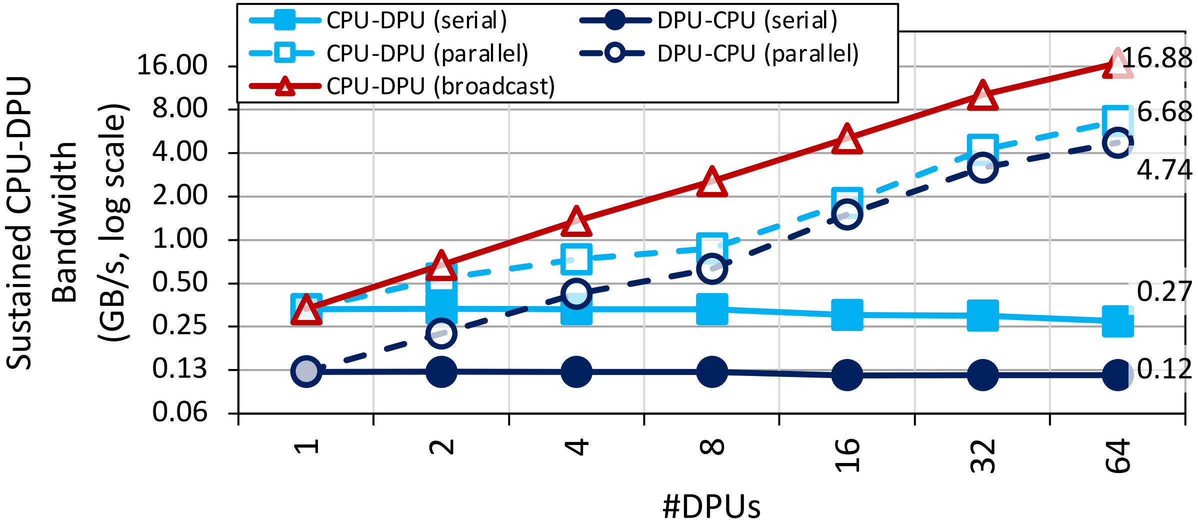


KEY OBSERVATION 7

Larger CPU-DPU and DPU-CPU transfers between the host main memory and the DRAM Processing Unit's Main memory (MRAM) banks **result in higher sustained bandwidth**.

CPU-DPU/DPU-CPU Transfers: 1 Rank

- CPU-DPU (serial/parallel/broadcast) and DPU-CPU (serial/parallel)
- The number of DPUs varies between 1 and 64



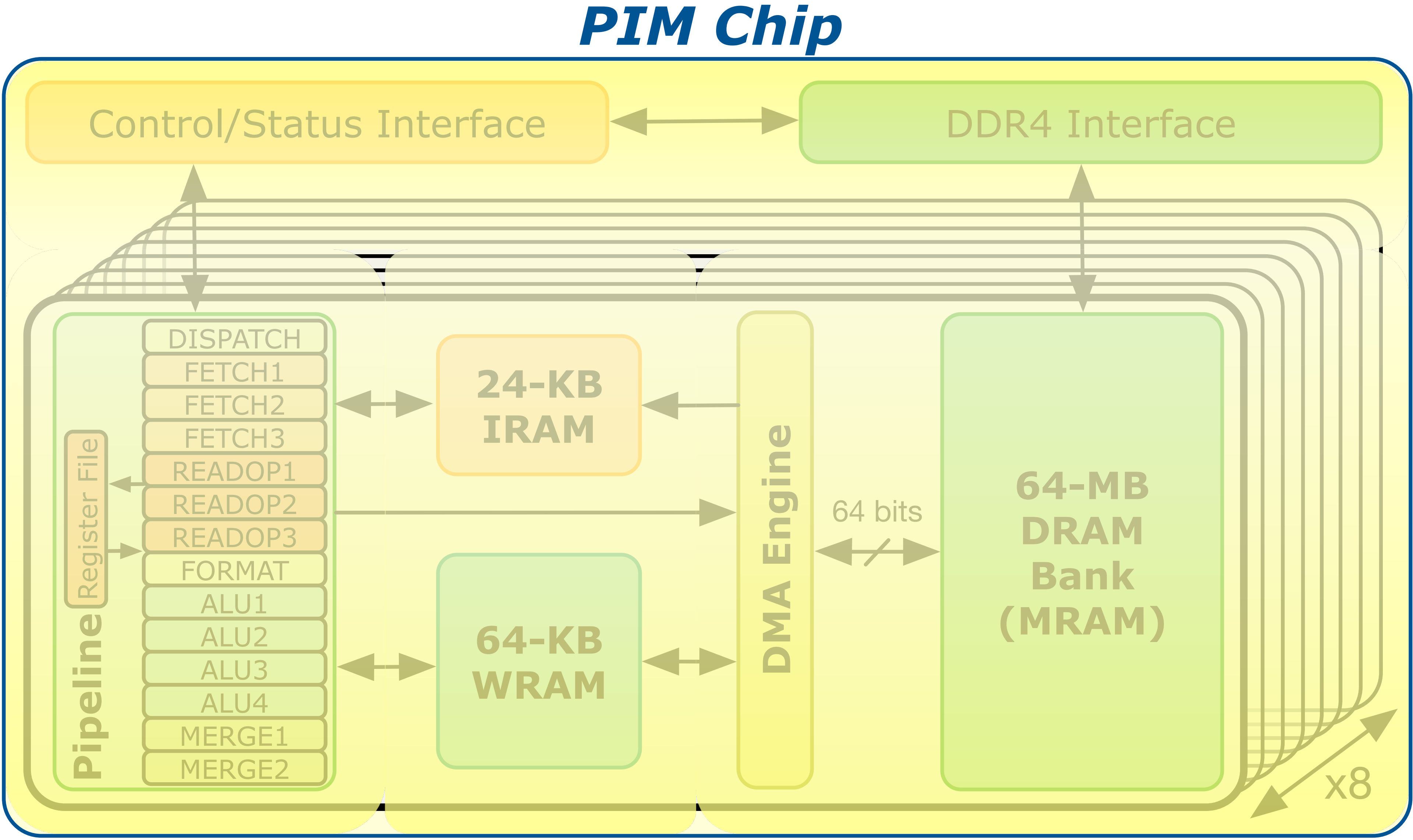
KEY OBSERVATION 8

The **sustained bandwidth of parallel CPU-DPU and DPU-CPU transfers** between the host main memory and the DRAM Processing Unit's Main memory (MRAM) banks **increases with the number of DRAM Processing Units inside a rank.**

Outline

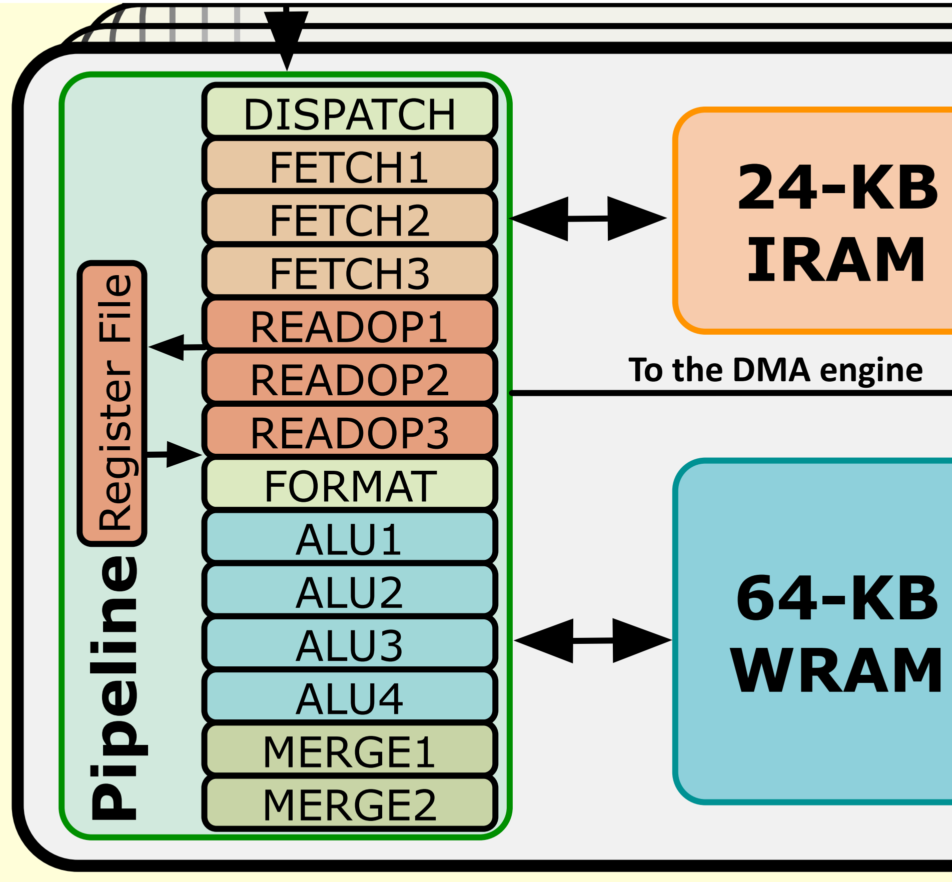
- Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview
- UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth
- PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity
- Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU
- Key Takeaways

DRAM Processing Unit



DPU Pipeline

- In-order pipeline
  - Up to 350 MHz
- Fine-grain multithreaded
  - 24 hardware threads
- 14 pipeline stages
  - DISPATCH: Thread selection
  - FETCH: Instruction fetch
  - READOP: Register file
  - FORMAT: Operand formatting
  - ALU: Operation and WRAM
  - MERGE: Result formatting



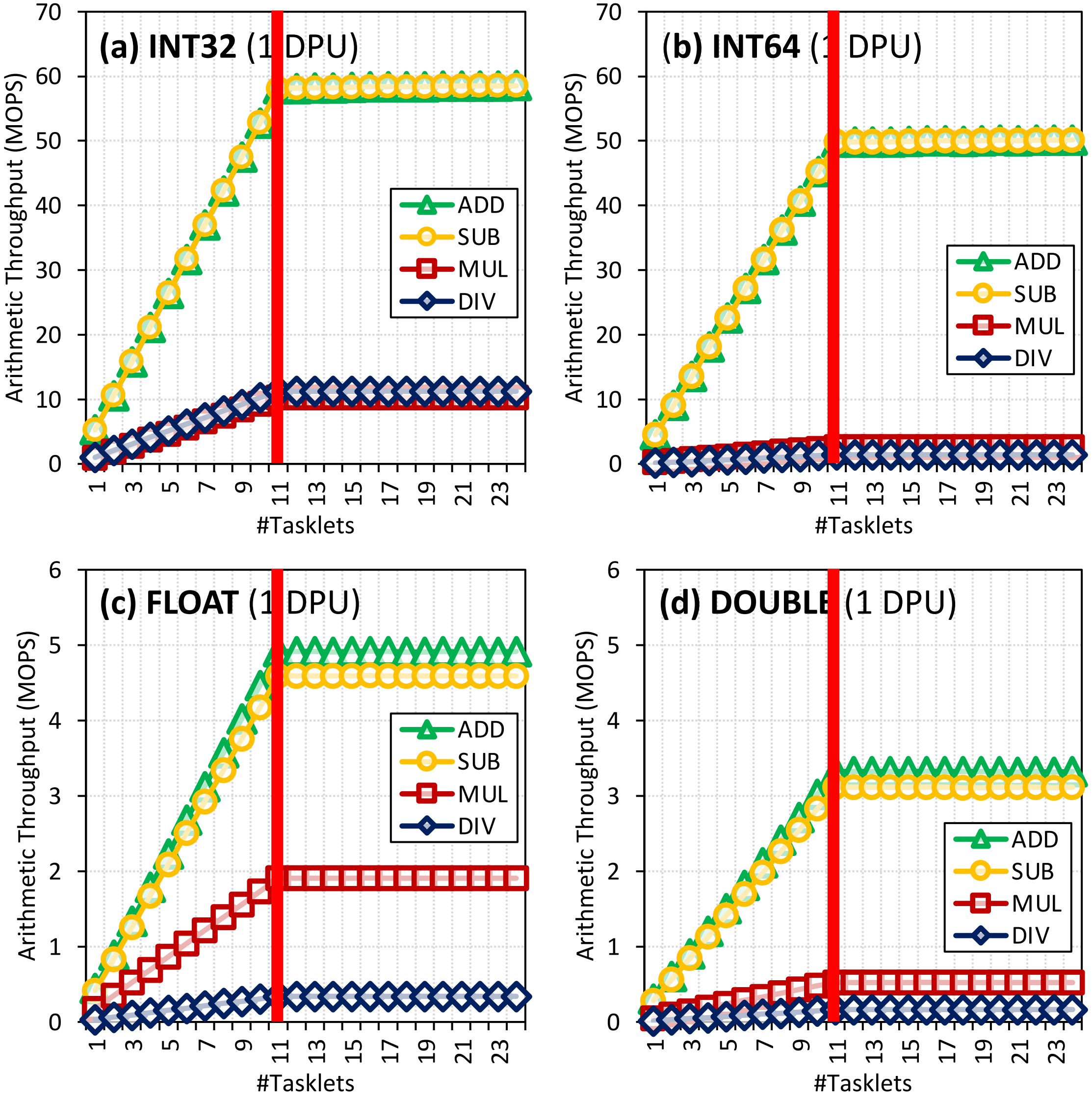
Arithmetic Throughput: Microbenchmark

- Goal
  - Measure the maximum arithmetic throughput for different datatypes and operations

- Microbenchmark
  - We stream over an array in WRAM and perform read-modify-write operations
  - Experiments on one DPU
  - We vary the number of tasklets from 1 to 24
  - Arithmetic operations: add, subtract, multiply, divide
  - Datatypes: int32, int64, float, double

- We measure cycles with an accurate cycle counter that the SDK provides
  - We include WRAM accesses (including address calculation) and arithmetic operation

Arithmetic Throughput: 11 Tasklets

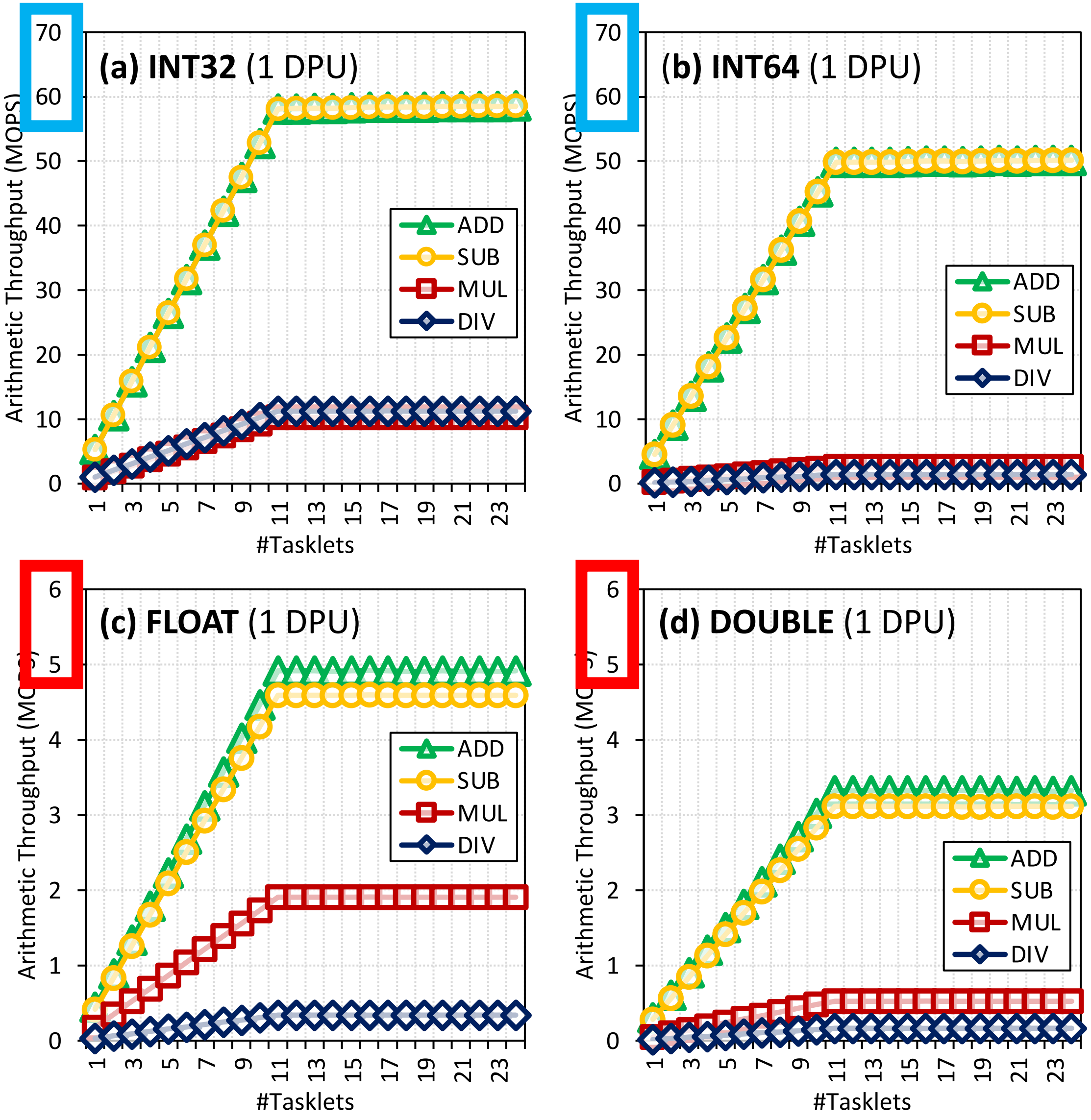


KEY OBSERVATION 1

The arithmetic throughput of a DRAM Processing Unit saturates at 11 or more tasklets.

This observation is consistent for different datatypes (INT32, INT64, UINT32, UINT64, FLOAT, DOUBLE) and operations (ADD, SUB, MUL, DIV).

Arithmetic Throughput: Native Support

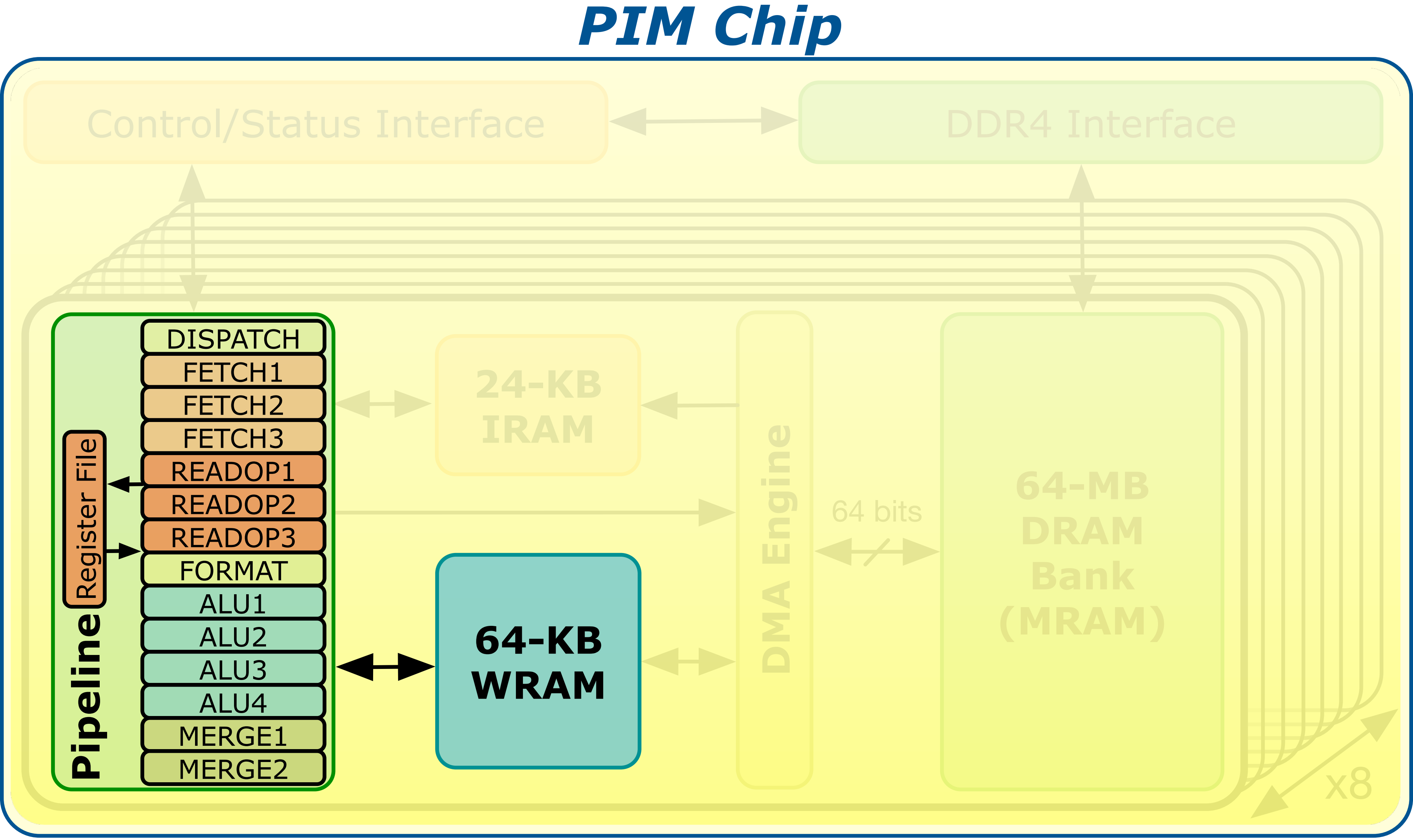


KEY OBSERVATION 2

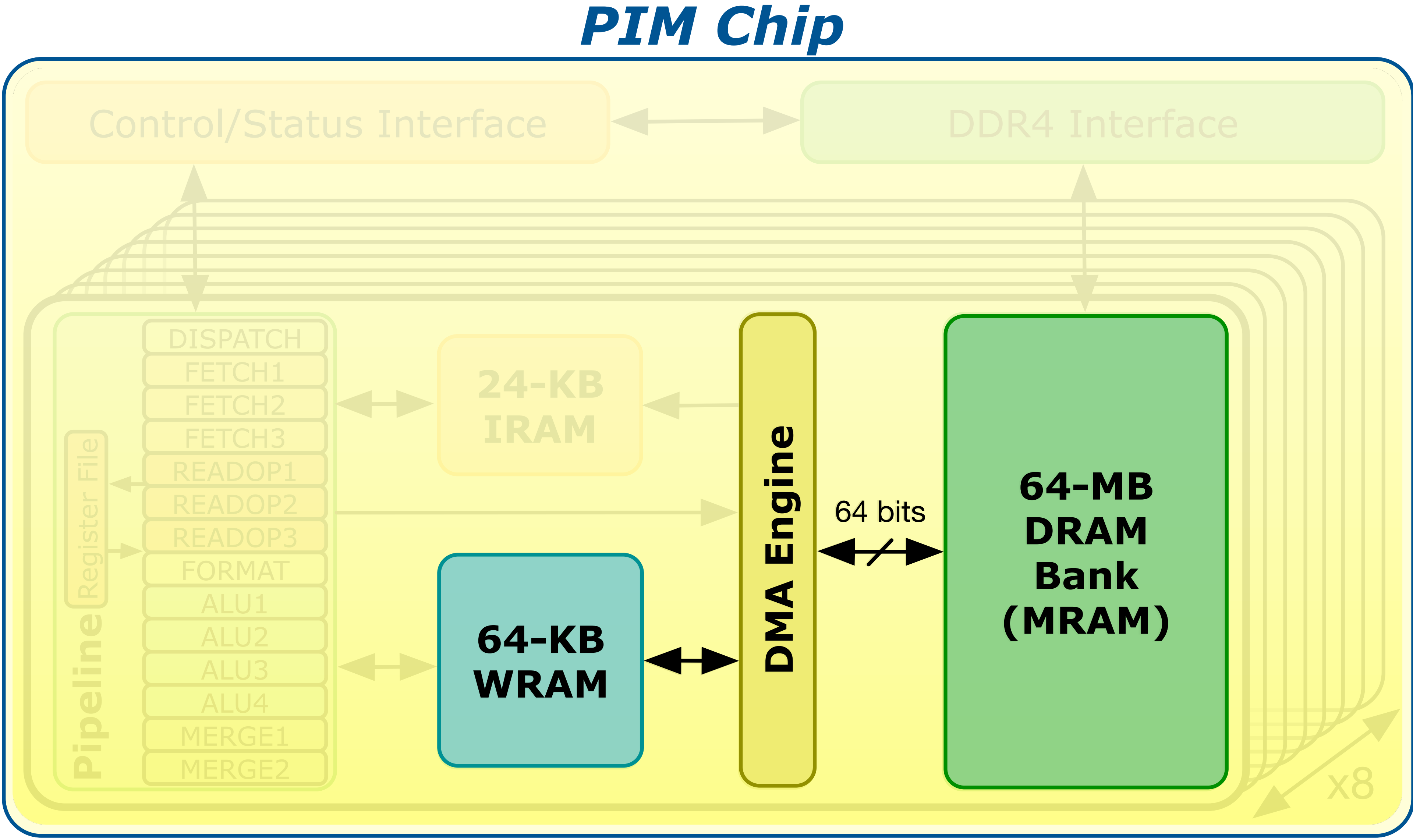
- DPUs provide **native hardware support for 32- and 64-bit integer addition and subtraction**, leading to high throughput for these operations.

- DPUs do ***not* natively support 32- and 64-bit multiplication and division, and floating point operations**. These operations are **emulated by the UPMEM runtime library**, leading to much lower throughput.

DPU: WRAM Bandwidth



DPU: MRAM Latency and Bandwidth



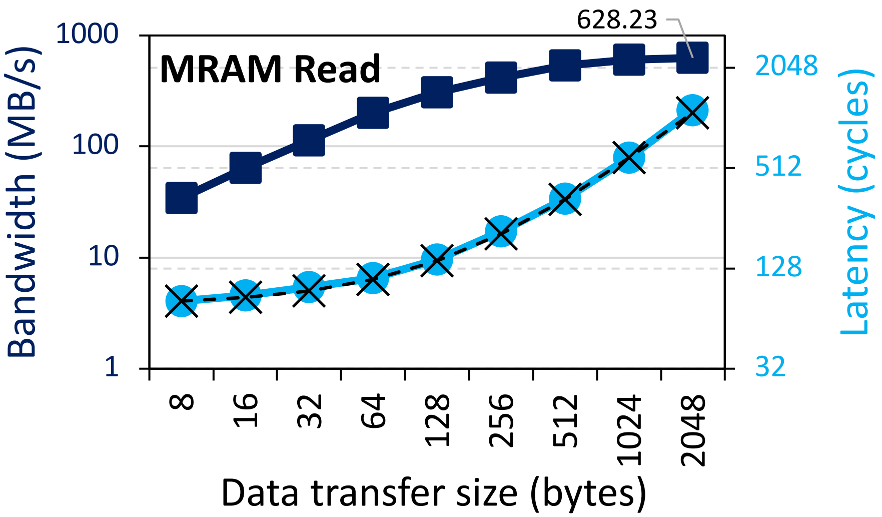
MRAM Bandwidth

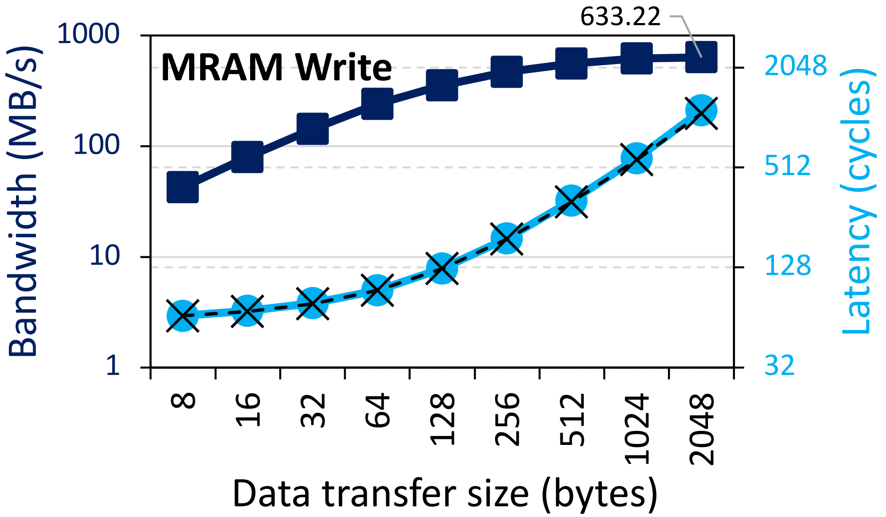
- Goal
  - Measure MRAM bandwidth for different access patterns

- Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    - `mram_read();   // MRAM-WRAM DMA transfer`
    - `mram_write();  // WRAM-MRAM DMA transfer`
  - STREAM benchmark
    - COPY, COPY-DMA
    - ADD, SCALE, TRIAD
  - Strided access pattern
    - Coarse-grain strided access
    - Fine-grain strided access
  - Random access pattern (GUPS)

- We do include accesses to MRAM

MRAM Read and Write Latency (I)





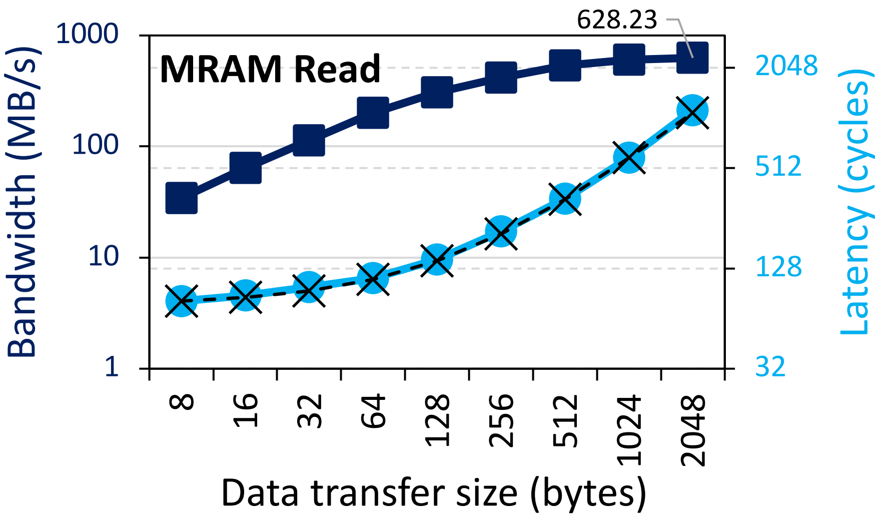
$$MRAM\ Bandwidth\ \left(in\ \frac{B}{S}\right) = \frac{size \times frequency_{DPU}}{MRAM\ Latency}$$

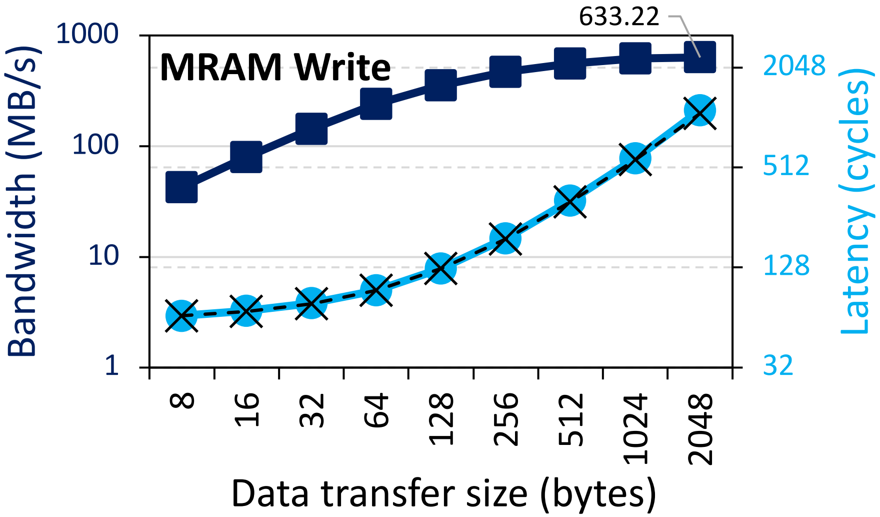
We can model the MRAM latency with a linear expression

$$MRAM\ Latency\ (in\ cycles) = \alpha + \beta \times size$$

In our measurements, β equals 0.5 cycles/byte.
Theoretical maximum MRAM bandwidth = 700 MB/s at 350 MHz

MRAM Read and Write Latency (II)





KEY OBSERVATION 4

- The DPU's **Main memory (MRAM) bank access latency increases linearly** with the transfer size.
- The maximum theoretical MRAM **bandwidth is 2 bytes per cycle**.

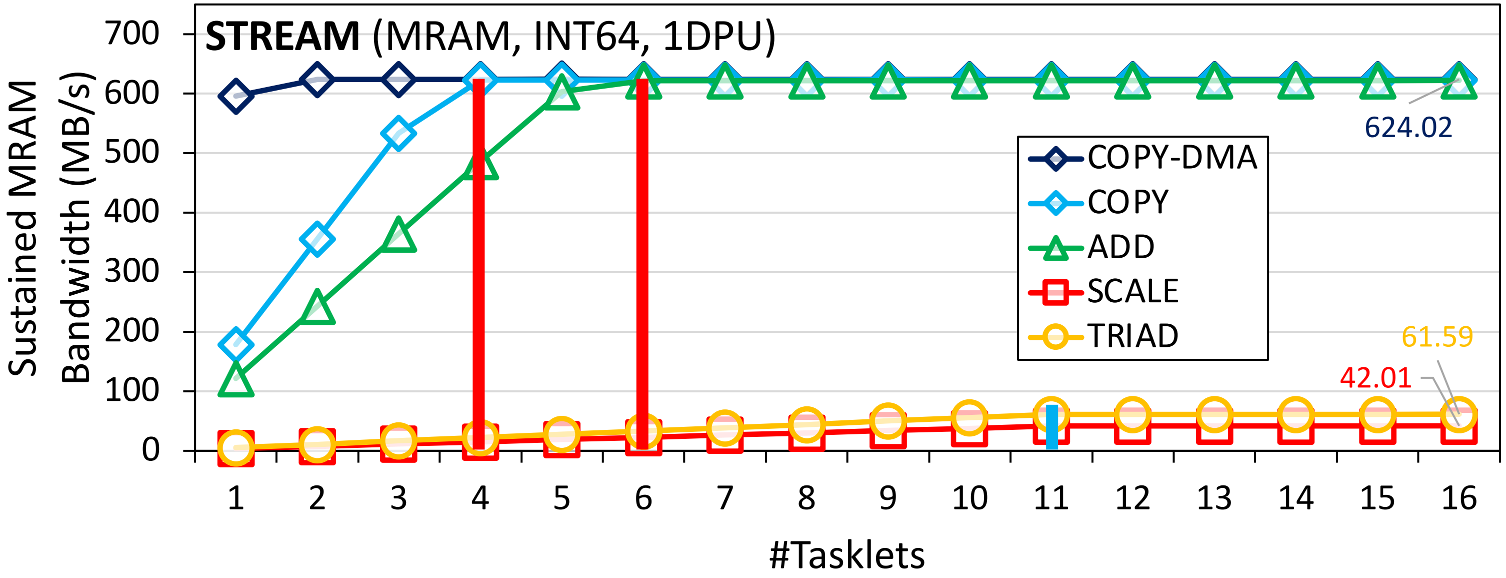
MRAM Bandwidth

- Goal
  - Measure MRAM bandwidth for different access patterns

- Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    - `mram_read();   // MRAM-WRAM DMA transfer`
    - `mram_write(); // WRAM-MRAM DMA transfer`
  - STREAM benchmark
    - COPY, COPY-DMA
    - ADD, SCALE, TRIAD
  - Strided access pattern
    - Coarse-grain strided access
    - Fine-grain strided access
  - Random access pattern (GUPS)

- We do include accesses to MRAM

STREAM Benchmark: Bandwidth Saturation



KEY OBSERVATION 5

- **When the access latency to an MRAM bank** for a streaming benchmark (COPY-DMA, COPY, ADD) **is larger than the pipeline latency** (i.e., execution latency of arithmetic operations and WRAM accesses), the performance of the DPU saturates at a number of tasklets smaller than 11. **This is a memory-bound workload.**
- **When the pipeline latency** for a streaming benchmark (SCALE, TRIAD) **is larger than the MRAM access latency**, the performance of a DPU saturates at 11 tasklets. **This is a compute-bound workload.**

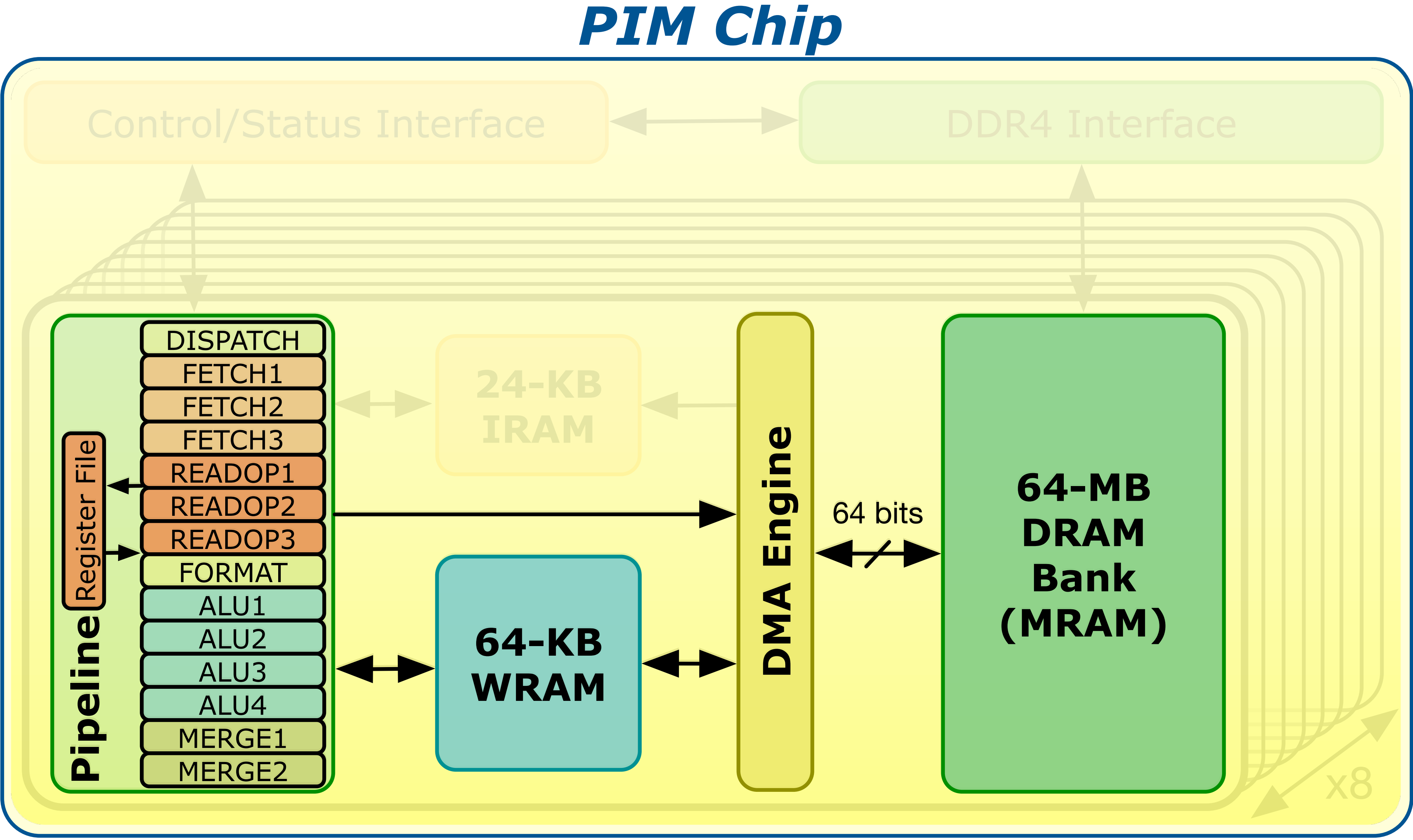
MRAM Bandwidth

- Goal
  - Measure MRAM bandwidth for different access patterns

- Microbenchmarks
  - Latency of a single DMA transfer for different transfer sizes
    - `mram_read();   // MRAM-WRAM DMA transfer`
    - `mram_write();  // WRAM-MRAM DMA transfer`
  - STREAM benchmark
    - COPY, COPY-DMA
    - ADD, SCALE, TRIAD
  - Strided access pattern
    - Coarse-grain strided access
    - Fine-grain strided access
  - Random access pattern (GUPS)

- We do include accesses to MRAM

DPU: Arithmetic Throughput vs. Operational Intensity



Arithmetic Throughput vs. Operational Intensity (I)

- Goal
  - Characterize memory-bound regions and compute-bound regions for different datatypes and operations

- Microbenchmark
  - We load one chunk of an MRAM array into WRAM
  - Perform a variable number of operations on the data
  - Write back to MRAM

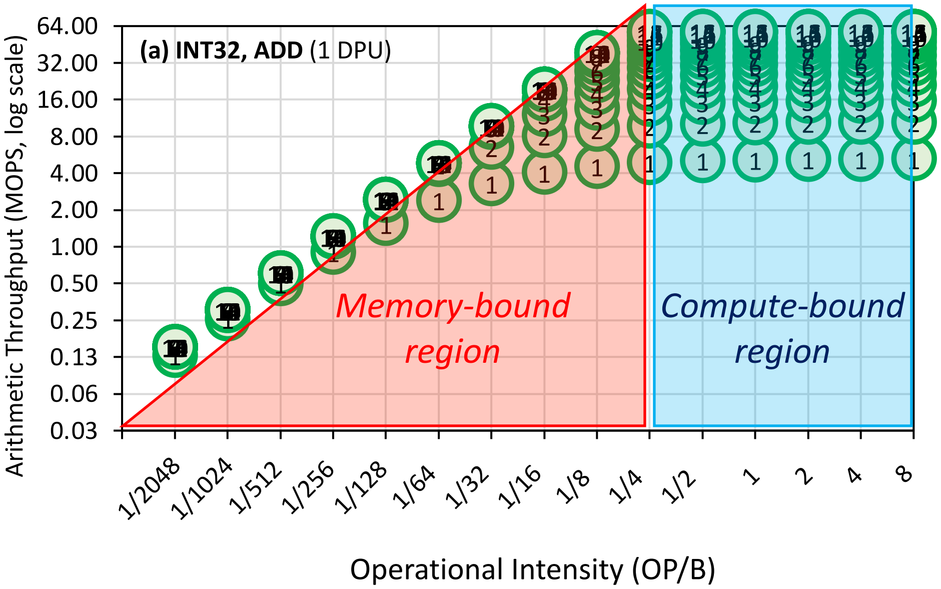
- The experiment is inspired by the Roofline model*

- We define operational intensity (OI) as the number of arithmetic operations performed per byte accessed from MRAM (OP/B)

- The pipeline latency changes with the operational intensity, but the MRAM access latency is fixed

*S. Williams et al., “Roofline: An Insightful Visual Performance Model for Multi-core Architectures,” CACM, 2009

Arithmetic Throughput vs. Operational Intensity (II)



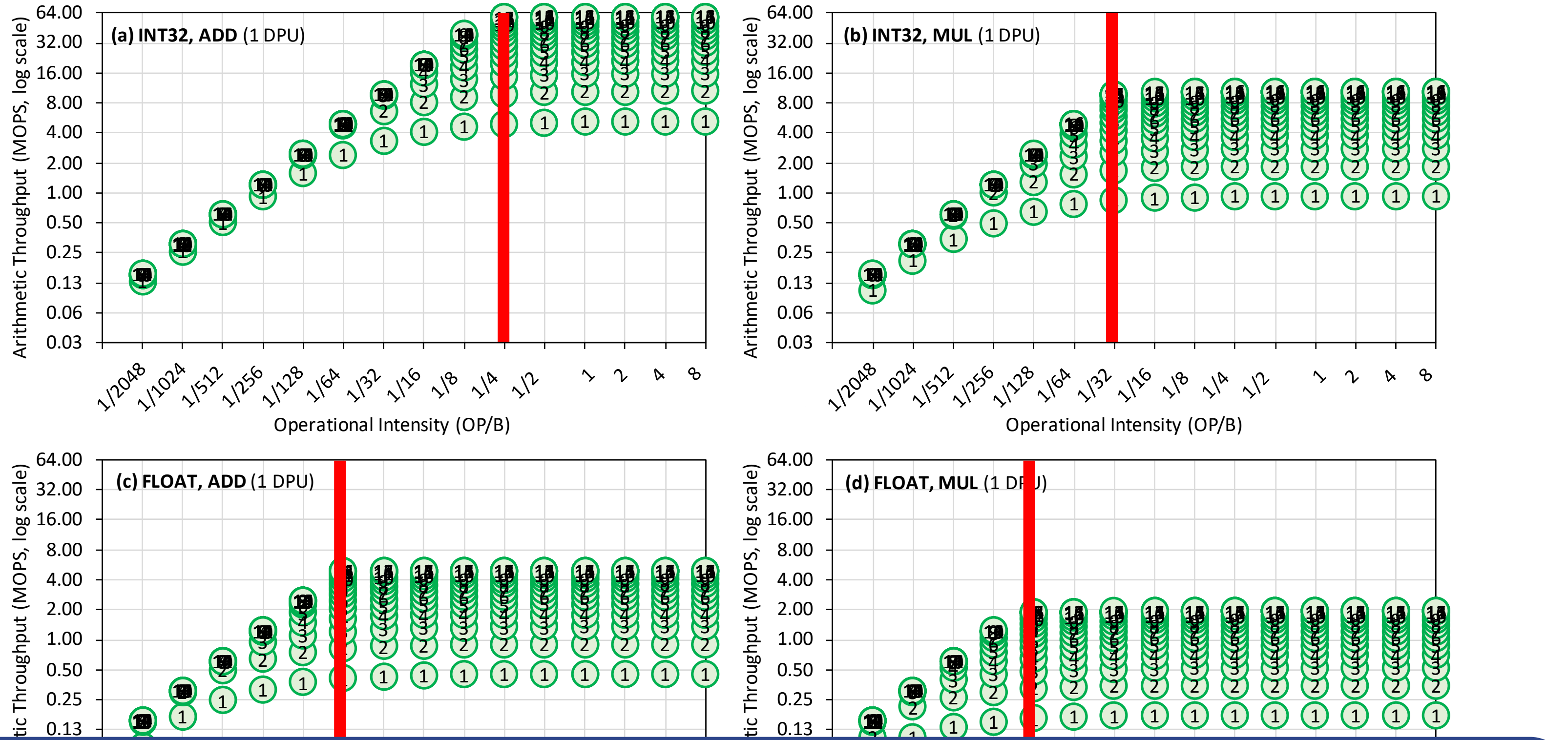
In the memory-bound region, the arithmetic throughput increases with the operational intensity

In the compute-bound region, the arithmetic throughput is flat at its maximum

The *throughput saturation point* is the operational intensity where the transition between the memory-bound region and the compute-bound region happens

The throughput saturation point is as low as ¼ OP/B, i.e., 1 integer addition per every 32-bit element fetched

Arithmetic Throughput vs. Operational Intensity (III)



KEY OBSERVATION 6

The arithmetic throughput of a DRAM Processing Unit (DPU) saturates at low or very low operational intensity (e.g., 1 integer addition per 32-bit element). Thus, **the DPU is fundamentally a compute-bound processor.**
We expect **most real-world workloads be compute-bound in the UPMEM PIM architecture.**

Outline

- Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview
- UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth
- PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity
- Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU
- Key Takeaways

PrIM Benchmarks

- Goal
  - A common set of workloads that can be used to
    - evaluate the UPMEM PIM architecture,
    - compare software improvements and compilers,
    - compare future PIM architectures and hardware

- Two key selection criteria:
  - Selected workloads from different application domains
  - Memory-bound workloads on processor-centric architectures

- 14 different workloads, 16 different benchmarks*

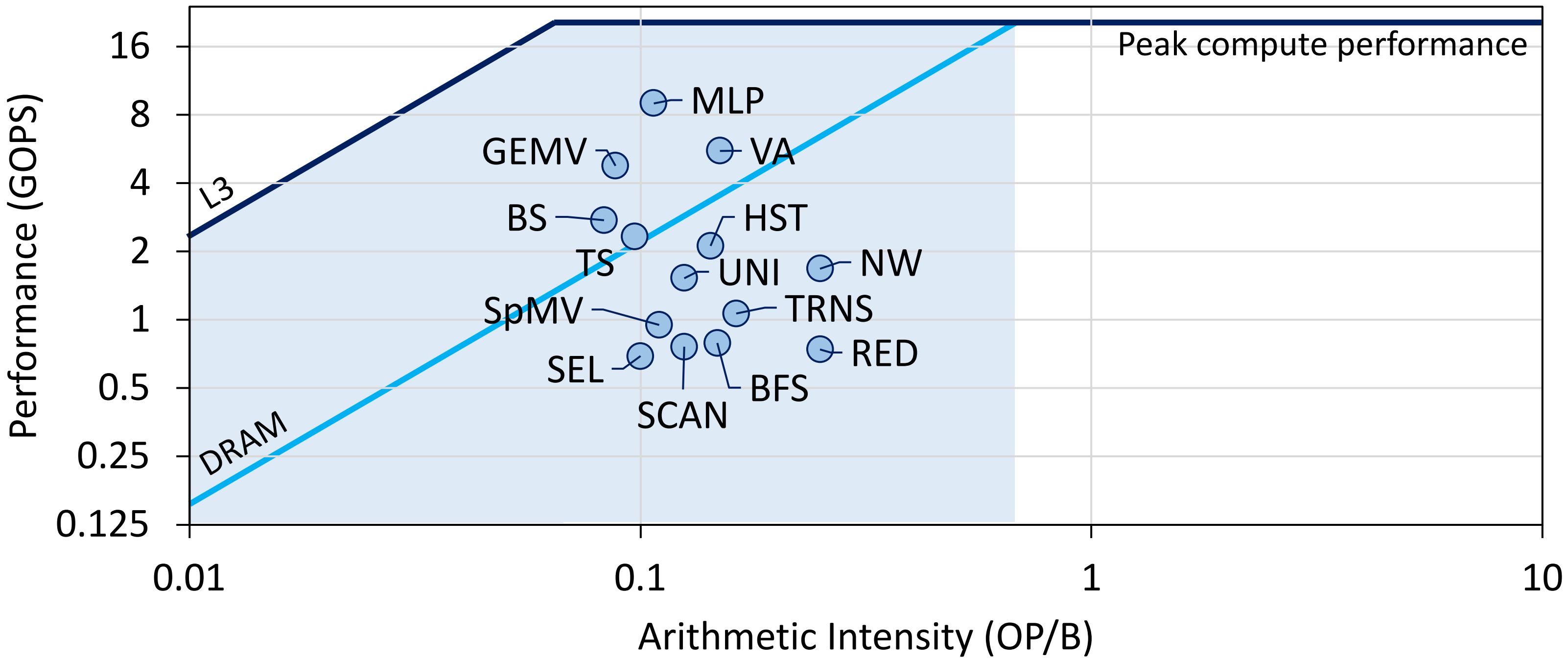
*There are two versions for two of the workloads (HST, SCAN).

PrIM Benchmarks: Application Domains

Domain	Benchmark	Short name
Dense linear algebra	Vector Addition	VA
	Matrix-Vector Multiply	GEMV
Sparse linear algebra	Sparse Matrix-Vector Multiply	SpMV
Databases	Select	SEL
	Unique	UNI
Data analytics	Binary Search	BS
	Time Series Analysis	TS
Graph processing	Breadth-First Search	BFS
Neural networks	Multilayer Perceptron	MLP
Bioinformatics	Needleman-Wunsch	NW
Image processing	Image histogram (short)	HST-S
	Image histogram (large)	HST-L
Parallel primitives	Reduction	RED
	Prefix sum (scan-scan-add)	SCAN-SSA
	Prefix sum (reduce-scan-scan)	SCAN-RSS
	Matrix transposition	TRNS

Roofline Model

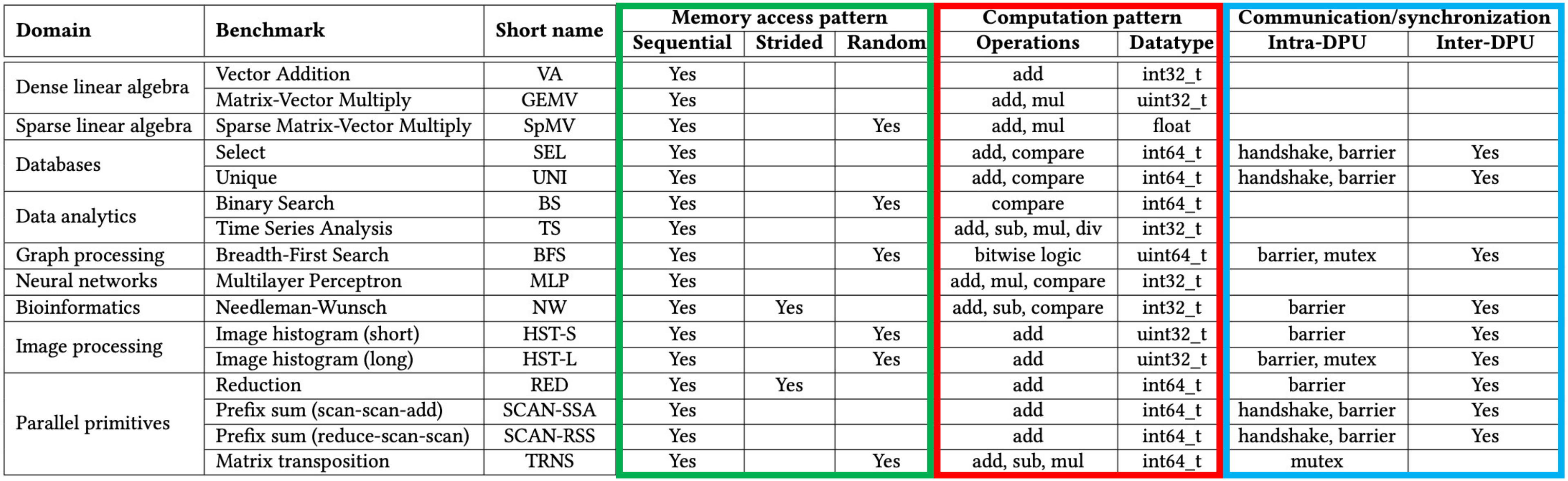
- Intel Advisor on an Intel Xeon E3-1225 v6 CPU



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

PrIM Benchmarks: Diversity

- PrIM benchmarks are diverse:
  - Memory access patterns
  - Operations and datatypes
  - Communication/synchronization



Domain	Benchmark	Short name	Memory access pattern			Computation pattern		Communication/synchronization	
			Sequential	Strided	Random	Operations	Datatype	Intra-DPU	Inter-DPU
Dense linear algebra	Vector Addition	VA	Yes			add	int32_t		
	Matrix-Vector Multiply	GEMV	Yes			add, mul	uint32_t		
Sparse linear algebra	Sparse Matrix-Vector Multiply	SpMV	Yes		Yes	add, mul	float		
Databases	Select	SEL	Yes			add, compare	int64_t	handshake, barrier	Yes
	Unique	UNI	Yes			add, compare	int64_t	handshake, barrier	Yes
Data analytics	Binary Search	BS	Yes		Yes	compare	int64_t		
	Time Series Analysis	TS	Yes			add, sub, mul, div	int32_t		
Graph processing	Breadth-First Search	BFS	Yes		Yes	bitwise logic	uint64_t	barrier, mutex	Yes
Neural networks	Multilayer Perceptron	MLP	Yes			add, mul, compare	int32_t		
Bioinformatics	Needleman-Wunsch	NW	Yes	Yes		add, sub, compare	int32_t	barrier	Yes
Image processing	Image histogram (short)	HST-S	Yes		Yes	add	uint32_t	barrier	Yes
	Image histogram (long)	HST-L	Yes		Yes	add	uint32_t	barrier, mutex	Yes
Parallel primitives	Reduction	RED	Yes	Yes		add	int64_t	barrier	Yes
	Prefix sum (scan-scan-add)	SCAN-SSA	Yes			add	int64_t	handshake, barrier	Yes
	Prefix sum (reduce-scan-scan)	SCAN-RSS	Yes			add	int64_t	handshake, barrier	Yes
	Matrix transposition	TRNS	Yes		Yes	add, sub, mul	int64_t	mutex	

Outline

- Introduction
  - Accelerator Model
  - UPMEM-based PIM System Overview
- UPMEM PIM Programming
  - Vector Addition
  - CPU-DPU Data Transfers
  - Inter-DPU Communication
  - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
  - Arithmetic Throughput
  - WRAM and MRAM Bandwidth
- PrIM Benchmarks
  - Roofline Model
  - Benchmark Diversity
- Evaluation
  - Strong and Weak Scaling
  - Comparison to CPU and GPU
- Key Takeaways

Evaluation Methodology

- We evaluate the 16 PrIM benchmarks on two UPMEM-based systems:
  - 2,556-DPU system
  - 640-DPU system
- Strong and weak scaling experiments on the 2,556-DPU system
  - 1 DPU with different numbers of tasklets
  - 1 rank (strong and weak)
  - Up to 32 ranks

Strong scaling refers to how the execution time of a program solving a particular problem varies with the number of processors for a fixed problem size

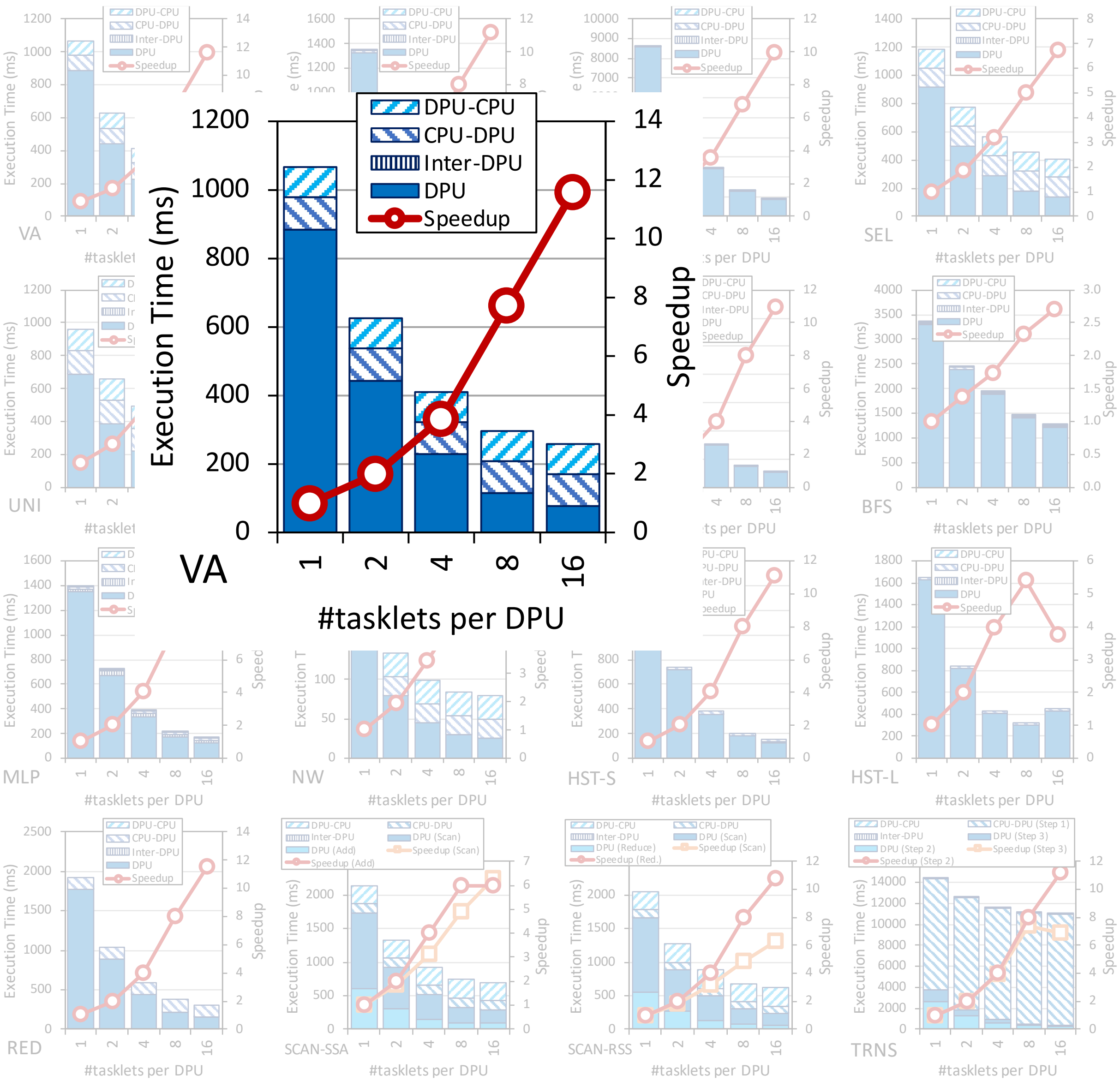
Weak scaling refers to how the execution time of a program solving a particular problem varies with the number of processors for a fixed problem size per processor

Evaluation Methodology

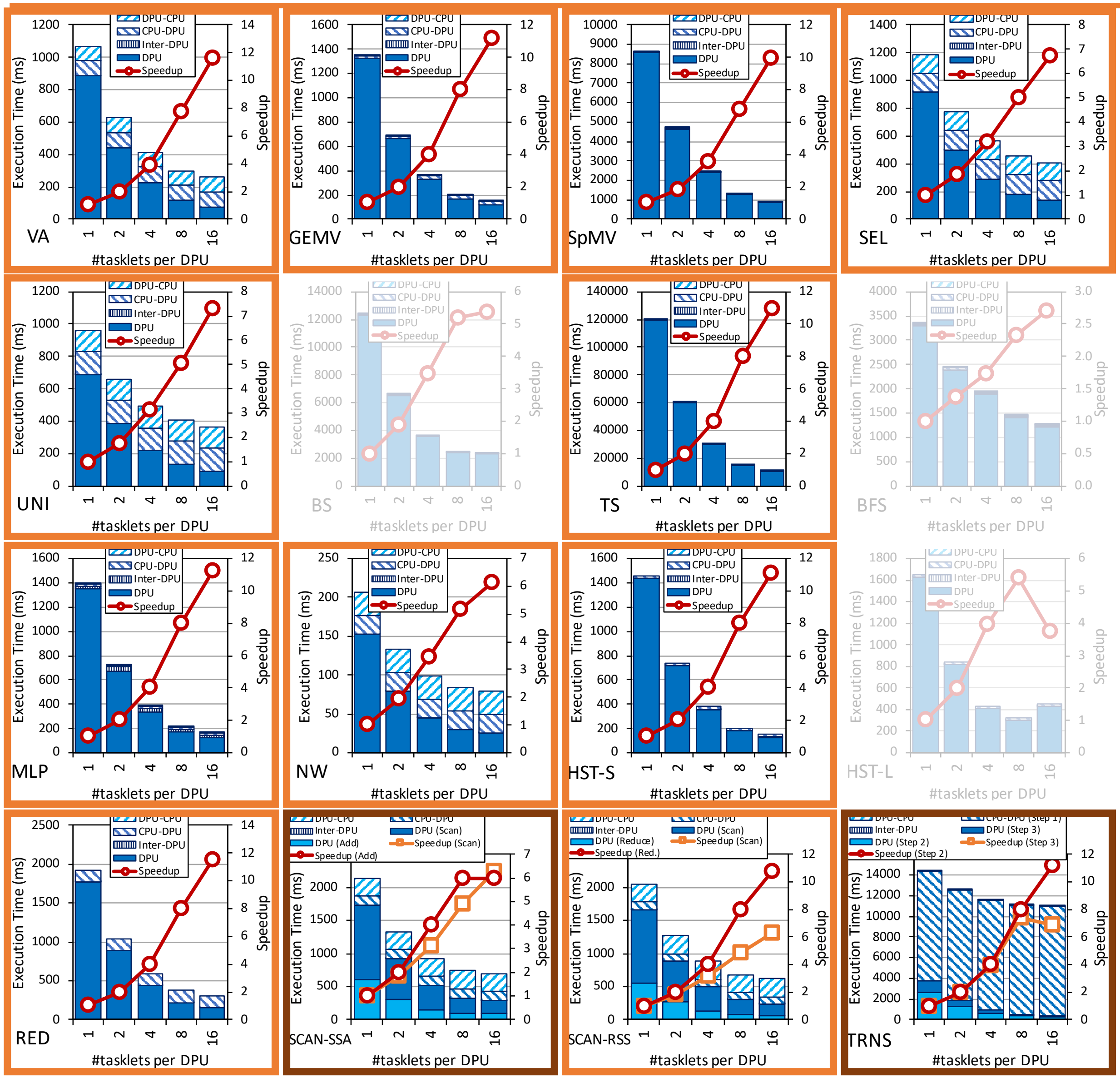
- We evaluate the 16 PrIM benchmarks on two UPMEM-based systems:
  - 2,556-DPU system
  - 640-DPU system
- Strong and weak scaling experiments on the 2,556-DPU system
  - 1 DPU with different numbers of tasklets
  - 1 rank (strong and weak)
  - Up to 32 ranks
- Comparison of both UPMEM-based PIM systems to state-of-the-art CPU and GPU
  - Intel Xeon E3-1240 CPU
  - NVIDIA Titan V GPU

Strong Scaling: 1 DPU (I)

- Strong scaling experiments on 1 DPU
  - We set the number of tasklets to 1, 2, 4, 8, and 16
  - We show the breakdown of execution time:
    - DPU: Execution time on the DPU
    - Inter-DPU: Time for inter-DPU communication via the host CPU
    - CPU-DPU: Time for CPU to DPU transfer of input data
    - DPU-CPU: Time for DPU to CPU transfer of final results
  - Speedup over 1 tasklet



Strong Scaling: 1 DPU (II)



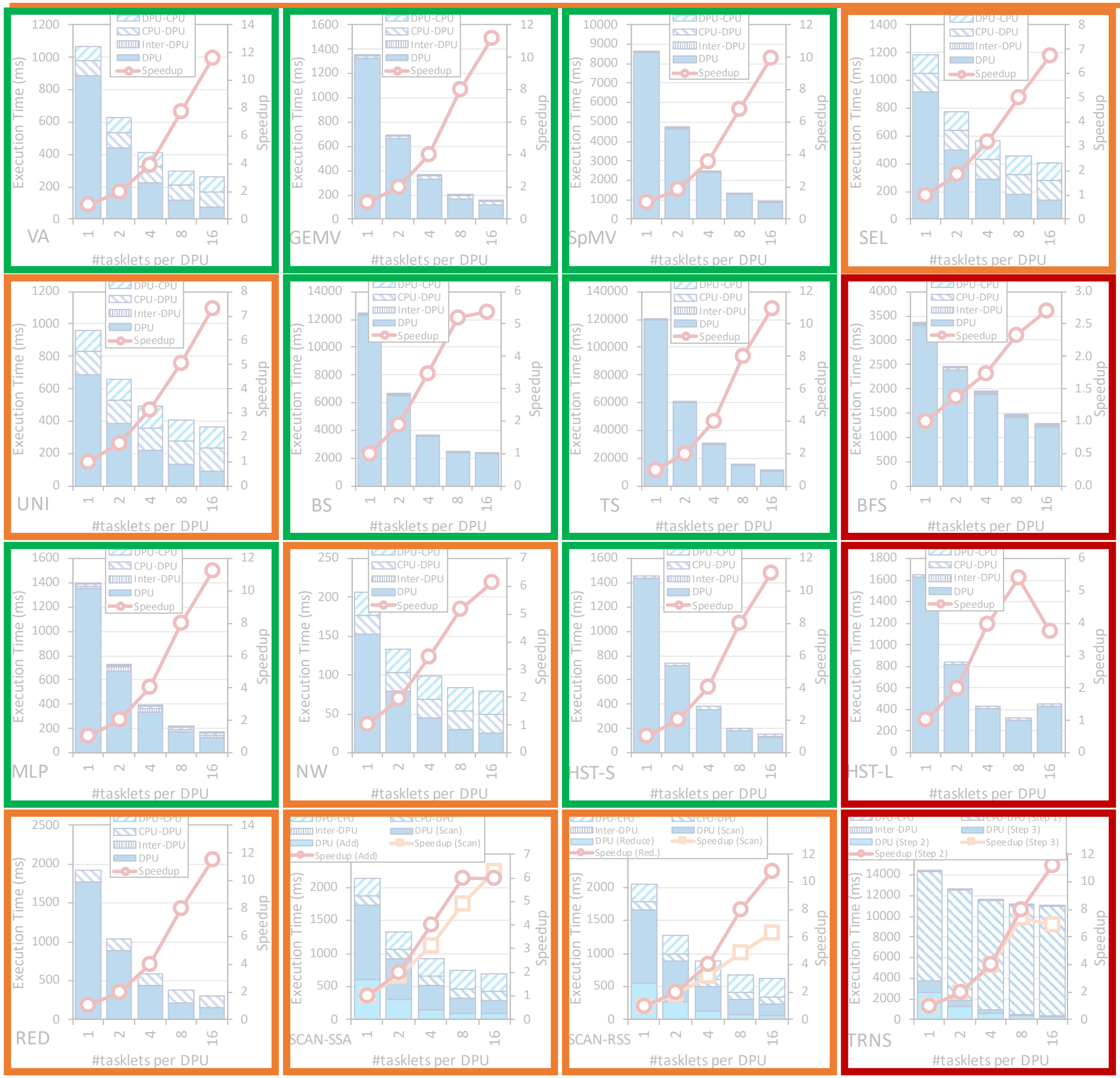
VA, GEMV, SpMV, SEL, UNI, TS, MLP, NW, HST-S, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), and TRNS (Step 2 kernel), the best performing number of tasklets is 16

Speedups 1.5-2.0x as we double the number of tasklets from 1 to 8. Speedups 1.2-1.5x from 8 to 16, since the pipeline throughput saturates at 11 tasklets

KEY OBSERVATION 10

A number of tasklets greater than 11 is a good choice for most real-world workloads we tested (16 kernels out of 19 kernels from 16 benchmarks), as it fully utilizes the DPU's pipeline.

Strong Scaling: 1 DPU (III)

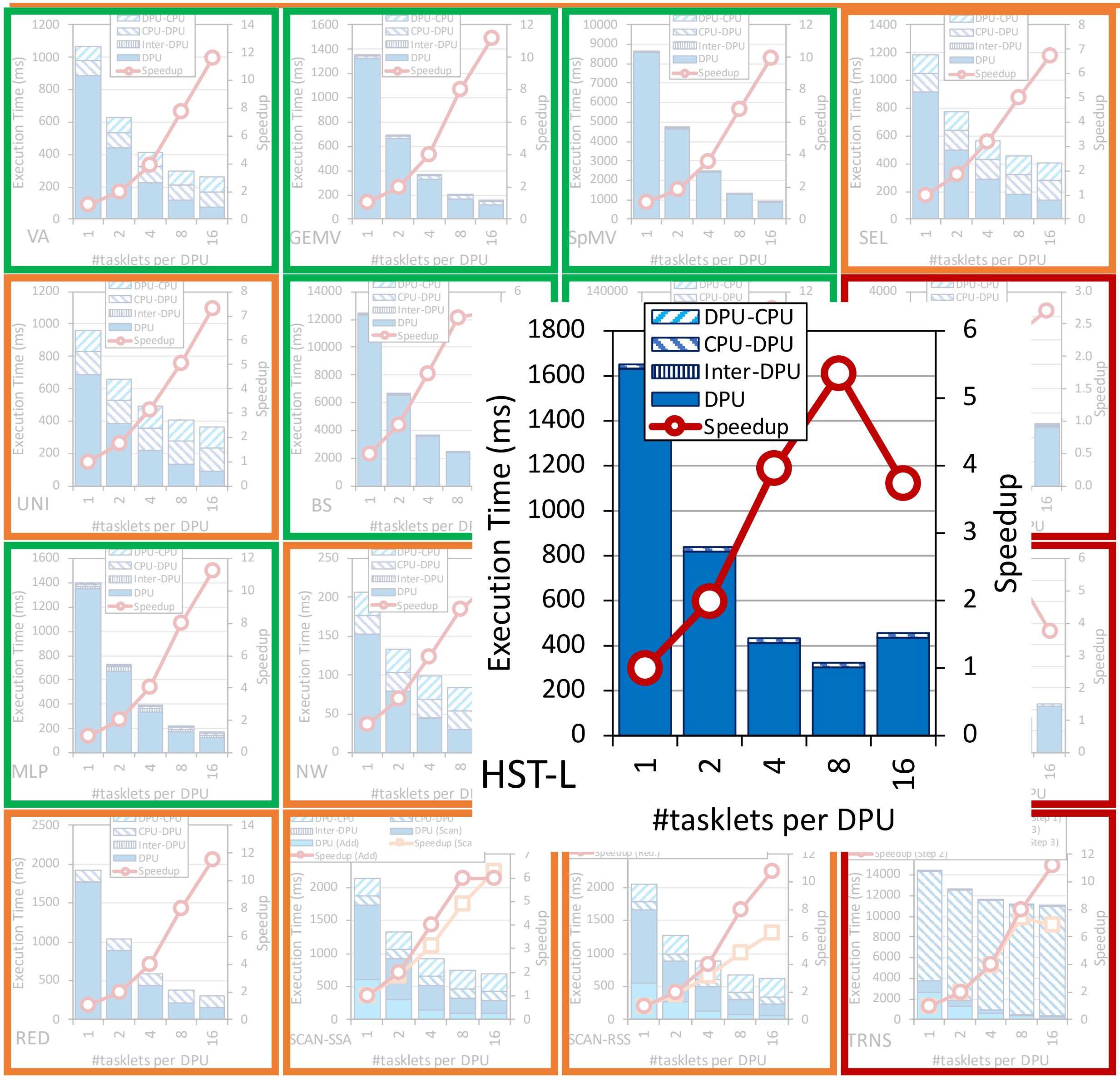


VA, GEMV, SpMV, BS, TS, MLP, HST-S do not use intra-DPU synchronization primitives

In SEL, UNI, NW, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), synchronization is lightweight

BFS, HST-L, TRNS (Step 3) use mutexes, which cause contention when accessing shared data structures

Strong Scaling: 1 DPU (IV)



VA, GEMV, SpMV, BS, TS, MLP, HST-S do not use synchronization primitives

In SEL, UNI, NW, RED, SCAN-SSA (Scan kernel), SCAN-RSS (both kernels), synchronization is lightweight

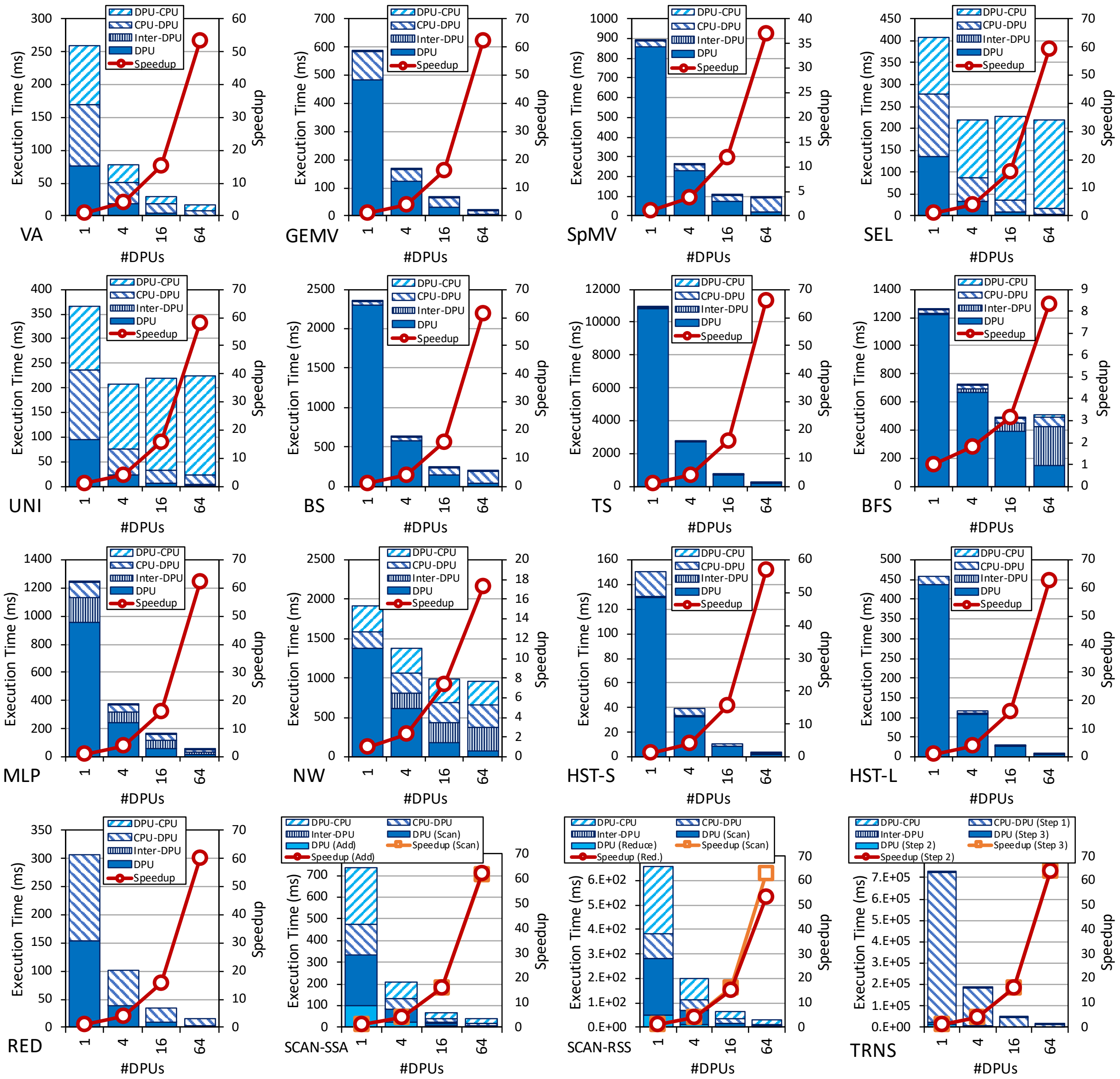
BFS, HST-L, TRNS (Step 3) use mutexes, which cause contention when accessing shared data structures

KEY OBSERVATION 11

Intensive use of **intra-DPU synchronization across tasklets (e.g., mutexes, barriers, handshakes) may limit scalability**, sometimes causing the best performing number of tasklets to be lower than 11.

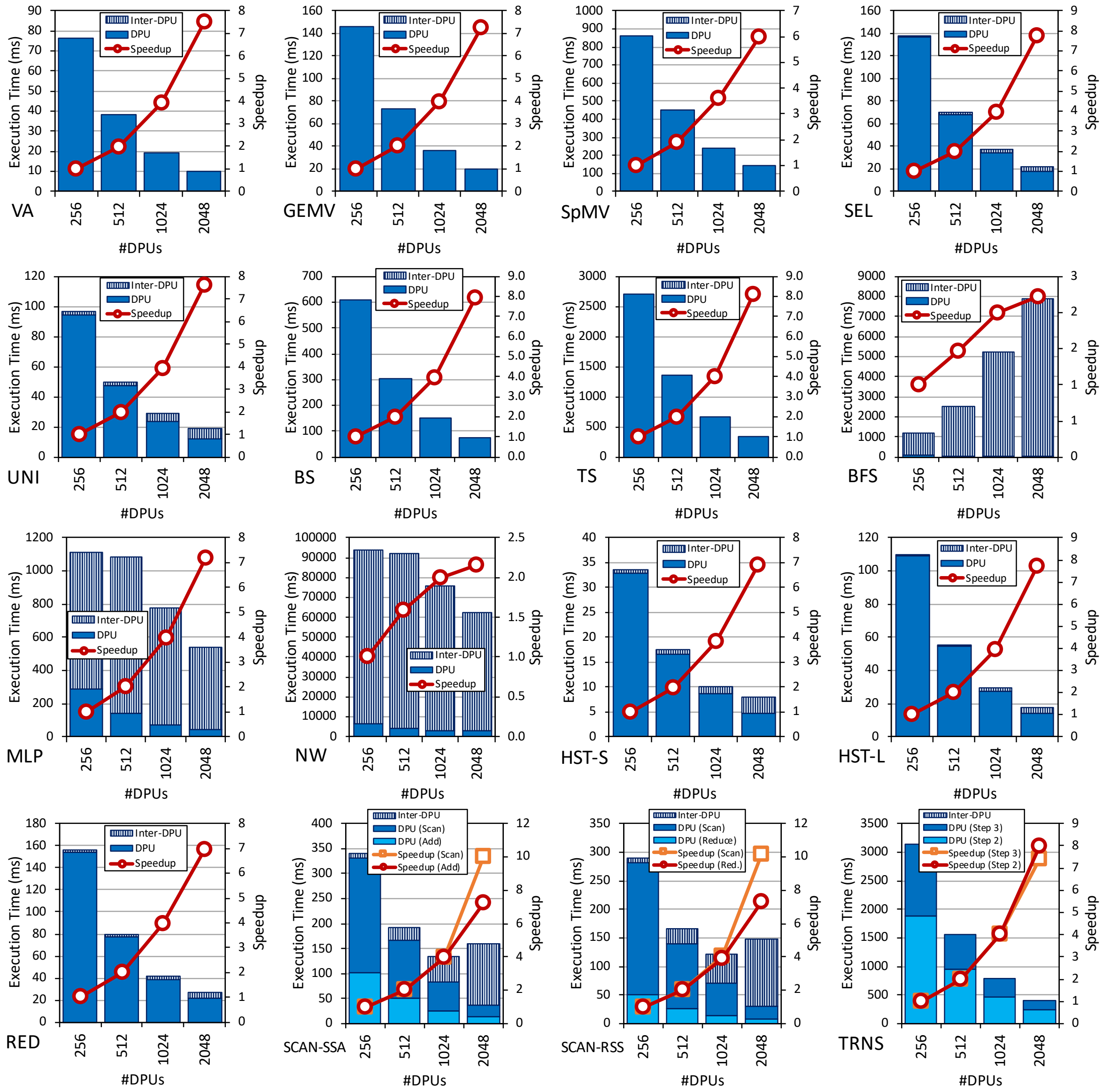
Strong Scaling: 1 Rank

- Strong scaling experiments on 1 rank
  - We set the number of tasklets to the best performing one
  - The number of DPUs is 1, 4, 16, 64
  - We show the breakdown of execution time:
    - DPU: Execution time on the DPU
    - Inter-DPU: Time for inter-DPU communication via the host CPU
    - CPU-DPU: Time for CPU to DPU transfer of input data
    - DPU-CPU: Time for DPU to CPU transfer of final results
  - Speedup over 1 DPU

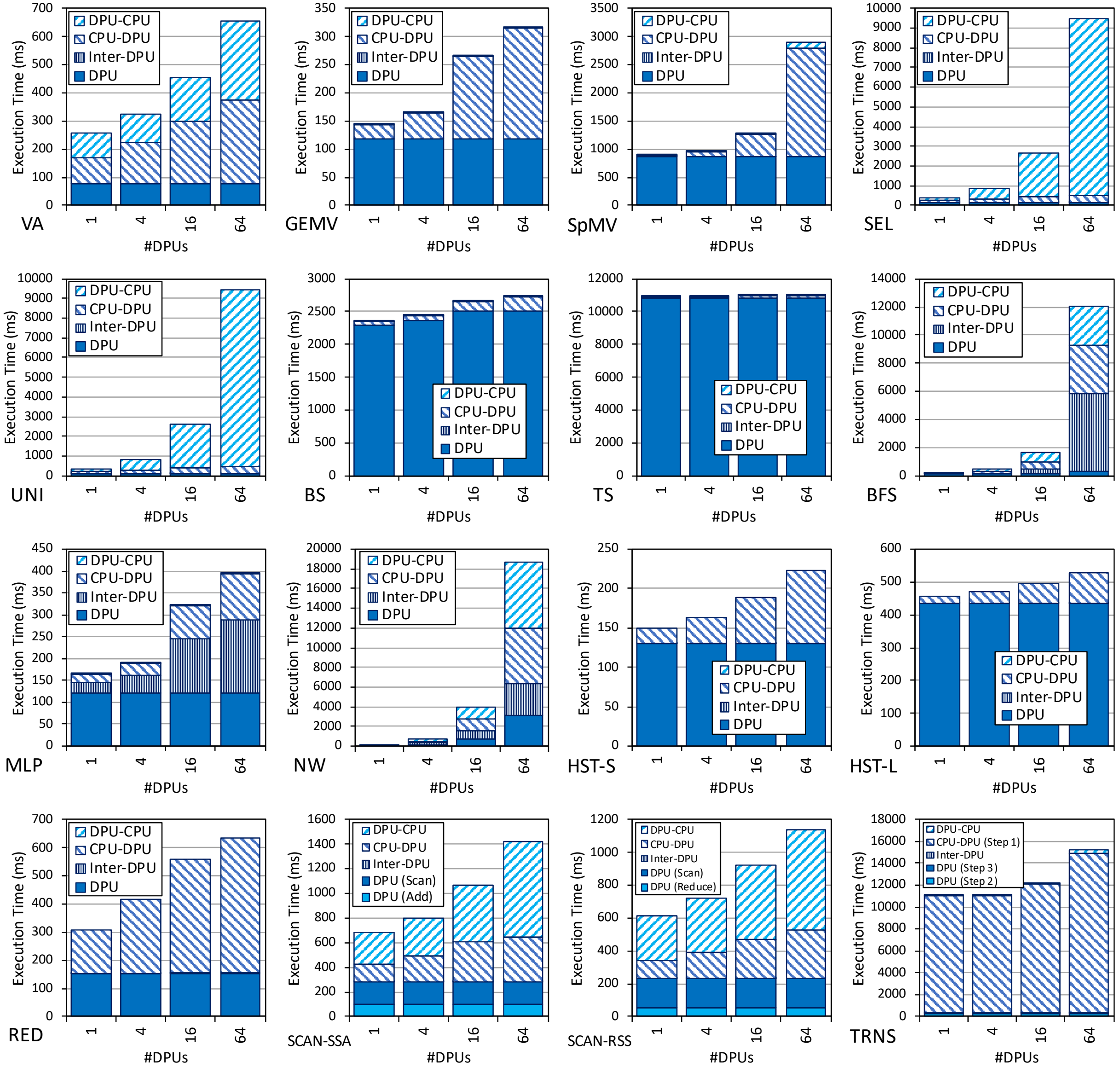


Strong Scaling: 32 Ranks

- Strong scaling experiments on 32 rank
  - We set the number of tasklets to the best performing one
  - The number of DPUs is 256, 512, 1024, 2048
  - We show the breakdown of execution time:
    - DPU: Execution time on the DPU
    - Inter-DPU: Time for inter-DPU communication via the host CPU
    - We do not show CPU-DPU/DPU-CPU transfer times
  - Speedup over 256 DPUs



Weak Scaling: 1 Rank



KEY OBSERVATION 17

Equally-sized problems assigned to different DPUs and little/no inter-DPU synchronization lead to linear weak scaling of the execution time spent on the DPUs (i.e., constant execution time when we increase the number of DPUs and the dataset size accordingly).

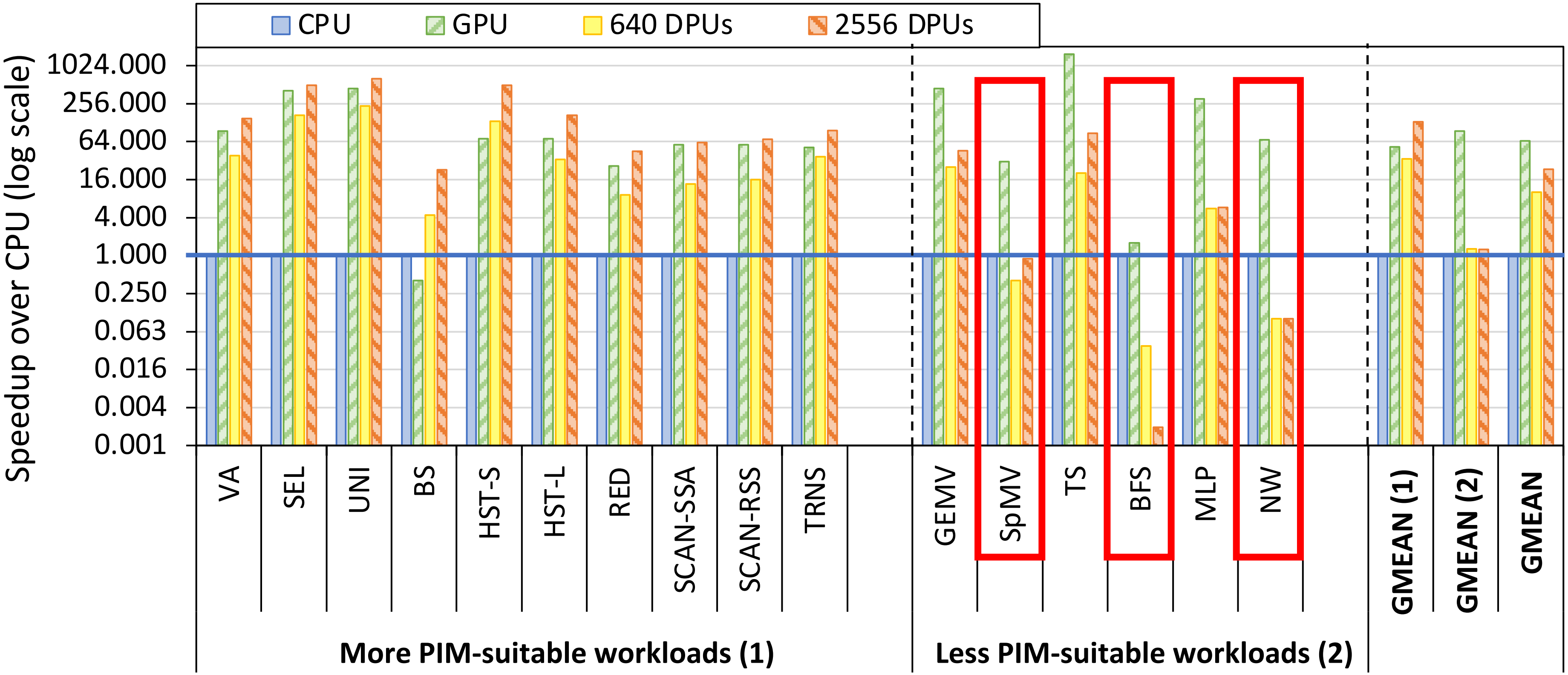
KEY OBSERVATION 18

Sustained bandwidth of parallel CPU-DPU/DPU-CPU transfers inside a rank of DPUs increases sublinearly with the number of DPUs.

CPU/GPU: Evaluation Methodology

- Comparison of both UPMEM-based PIM systems to state-of-the-art CPU and GPU
  - Intel Xeon E3-1240 CPU
  - NVIDIA Titan V GPU
- We use state-of-the-art CPU and GPU counterparts of PrIM benchmarks
  - https://github.com/CMU-SAFARI/prim-benchmarks
- We use the largest dataset that we can fit in the GPU memory
- We show overall execution time, including DPU kernel time and inter DPU communication

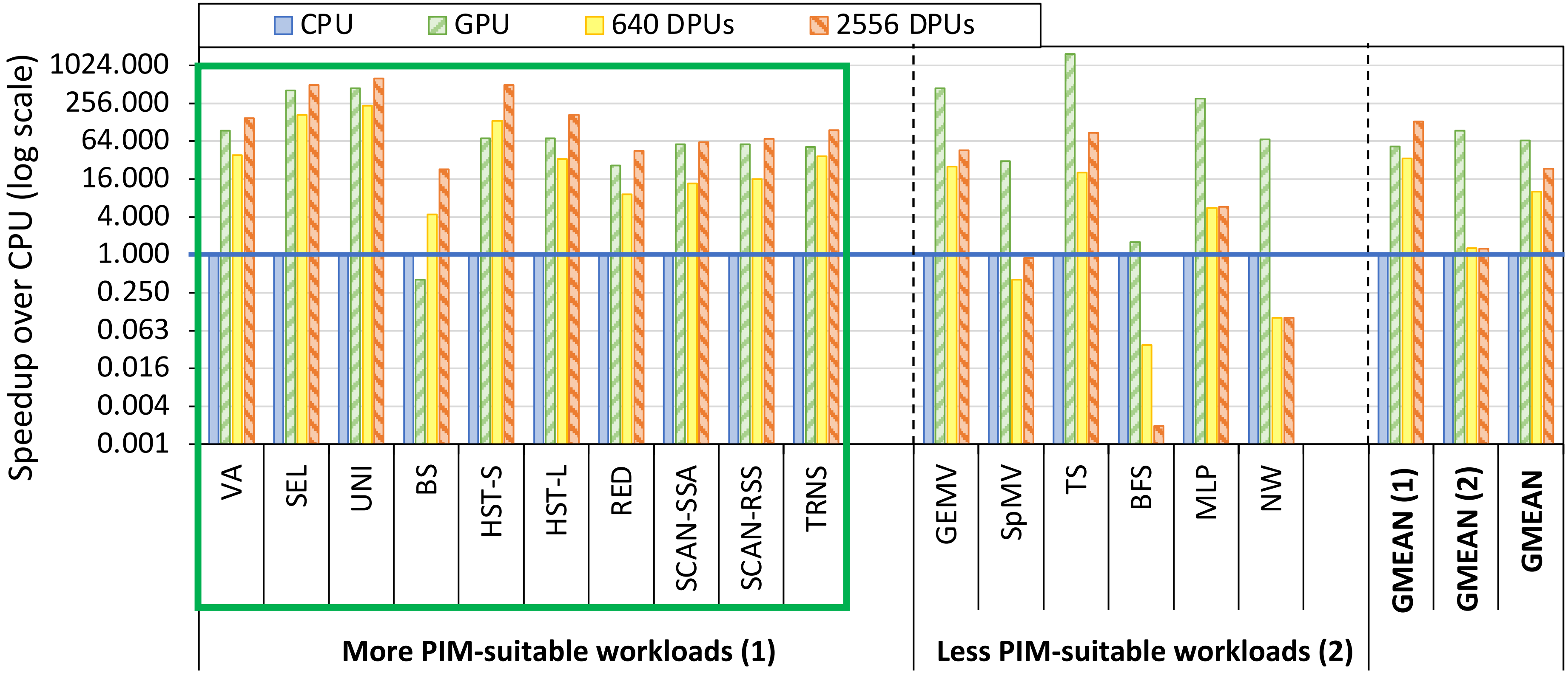
CPU/GPU: Performance Comparison (I)



The 2,556-DPU and the 640-DPU systems outperform the CPU for all benchmarks except SpMV, BFS, and NW

The 2,556-DPU and the 640-DPU are, respectively, 93.0x and 27.9x faster than the CPU for 13 of the PrIM benchmarks

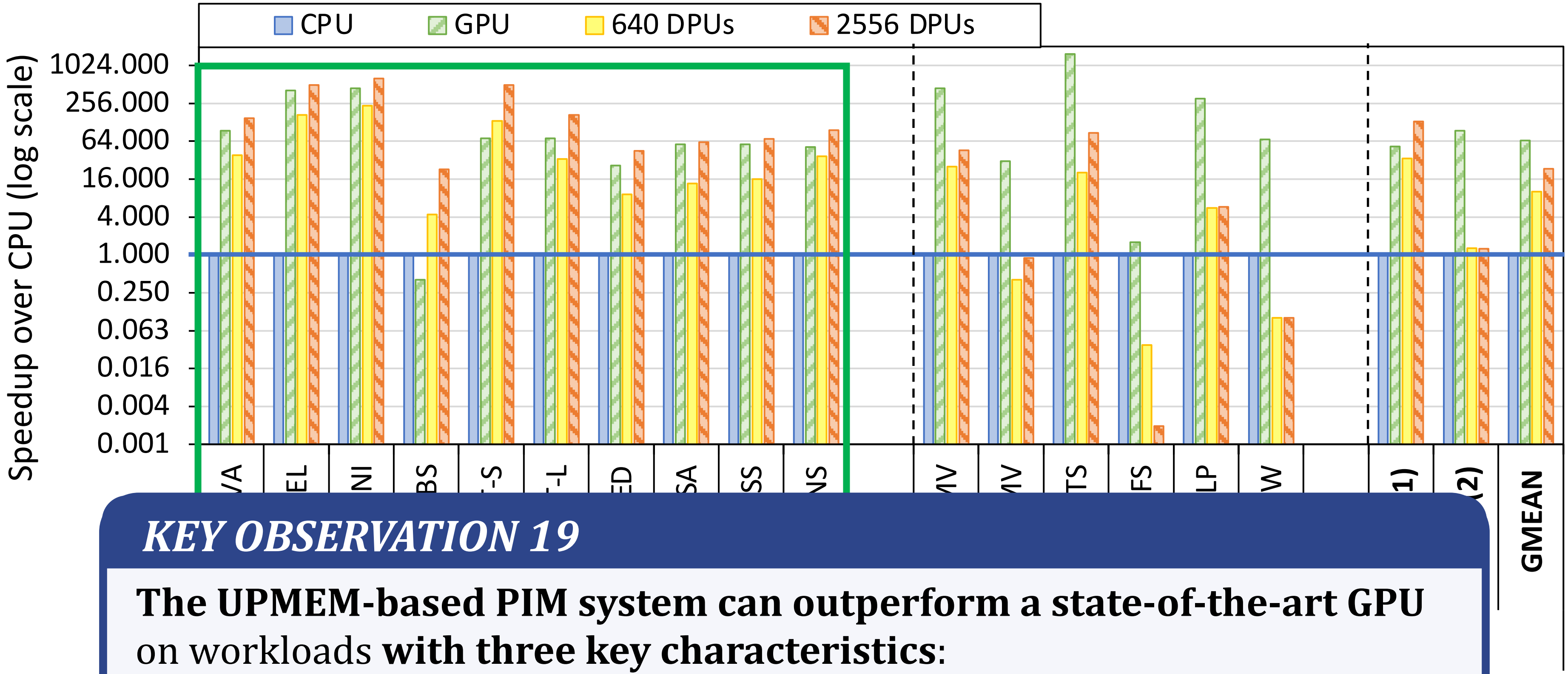
CPU/GPU: Performance Comparison (II)



The 2,556-DPU outperforms the GPU
for 10 PrIM benchmarks with an average of 2.54x

The performance of the 640-DPU is within 65%
the performance of the GPU for the same 10 PrIM benchmarks

CPU/GPU: Performance Comparison (III)



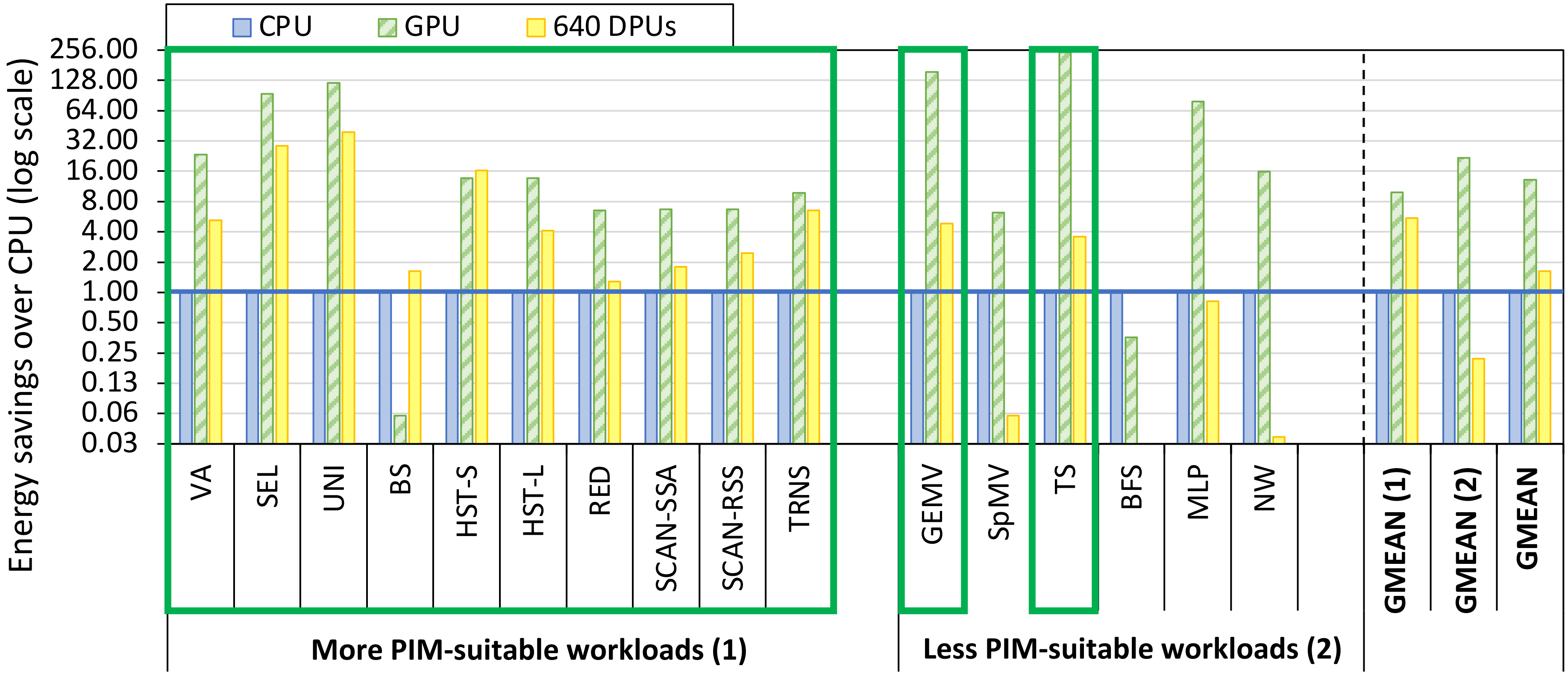
KEY OBSERVATION 19

The UPMEM-based PIM system can outperform a state-of-the-art GPU on workloads **with three key characteristics**:

1. Streaming memory accesses
2. No or little inter-DPU synchronization
3. No or little use of integer multiplication, integer division, or floating point operations

These three key characteristics make a **workload potentially suitable to the UPMEM PIM architecture**.

CPU/GPU: Energy Comparison



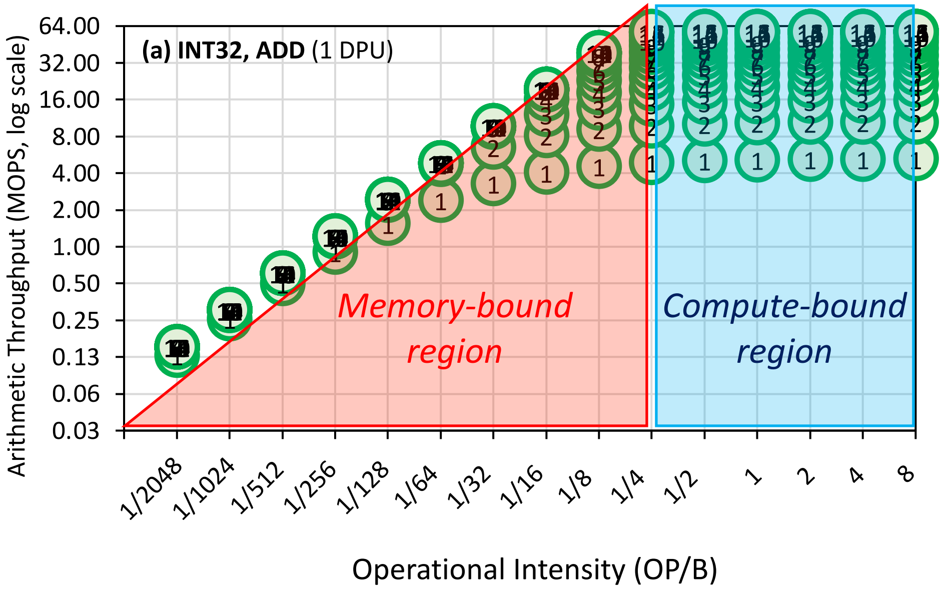
The 640-DPU system consumes on average 1.64x less energy than the CPU for all 16 PrIM benchmarks

For 12 benchmarks, the 640-DPU system provides energy savings of 5.23x over the CPU

Outline

- Introduction
    - Accelerator Model
    - UPMEM-based PIM System Overview
- UPMEM PIM Programming
    - Vector Addition
    - CPU-DPU Data Transfers
    - Inter-DPU Communication
    - CPU-DPU/DPU-CPU Transfer Bandwidth
- DRAM Processing Unit
    - Arithmetic Throughput
    - WRAM and MRAM Bandwidth
- PrIM Benchmarks
    - Roofline Model
    - Benchmark Diversity
- Evaluation
    - Strong and Weak Scaling
    - Comparison to CPU and GPU
- Key Takeaways

Key Takeaway 1

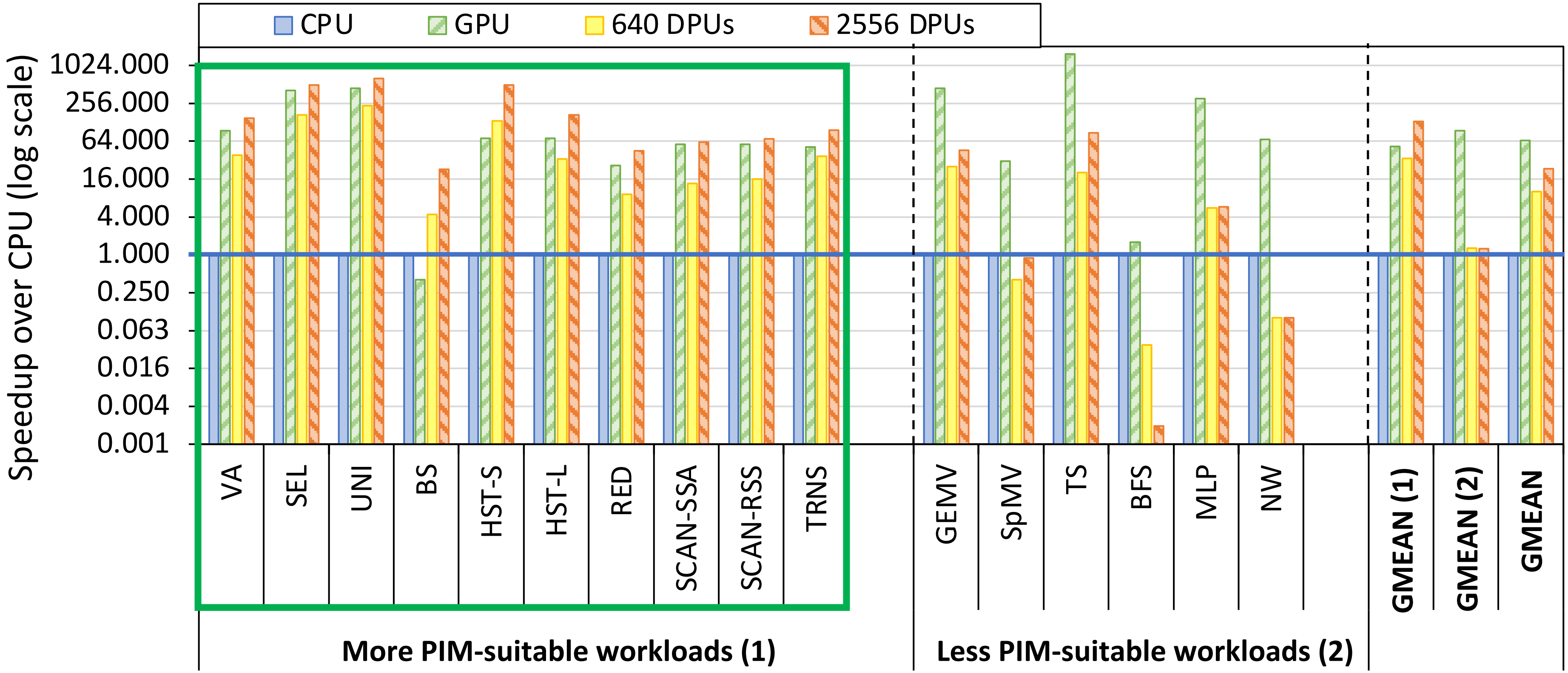


The throughput saturation point is as low as ¼ OP/B, i.e., 1 integer addition per every 32-bit element fetched

KEY TAKEAWAY 1

The UPMEM PIM architecture is fundamentally compute bound. As a result, **the most suitable workloads are memory-bound.**

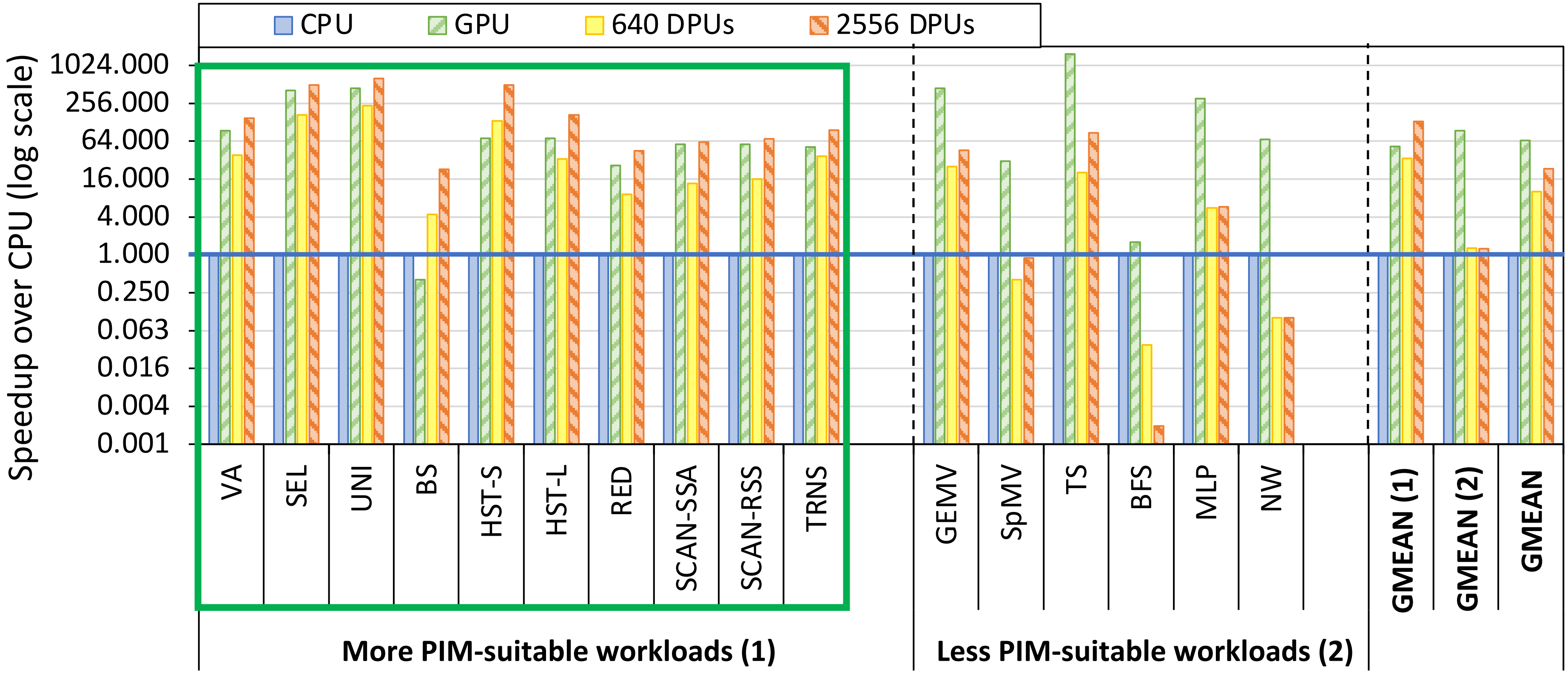
Key Takeaway 2



KEY TAKEAWAY 2

The most well-suited workloads for the UPMEM PIM architecture use no arithmetic operations or use only simple operations (e.g., bitwise operations and integer addition/subtraction).

Key Takeaway 3



KEY TAKEAWAY 3

The most well-suited workloads for the UPMEM PIM architecture require little or no communication across DPUs (inter-DPU communication).

Key Takeaway 4

KEY TAKEAWAY 4

- UPMEM-based PIM systems **outperform state-of-the-art CPUs in terms of performance** (by 23.2× on 2,556 DPUs for 16 PrIM benchmarks) **and energy efficiency on most of PrIM benchmarks.**

- UPMEM-based PIM systems **outperform state-of-the-art GPUs on a majority of PrIM benchmarks** (by 2.54× on 2,556 DPUs for 10 PrIM benchmarks), and the outlook is even more positive for future PIM systems.

- UPMEM-based PIM systems are **more energy-efficient than state-of-the-art CPUs and GPUs on workloads that they provide performance improvements** over the CPUs and the GPUs.

Executive Summary

- Data movement between memory/storage units and compute units is a major contributor to execution time and energy consumption
- Processing-in-Memory (PIM) is a paradigm that can tackle the *data movement bottleneck*
  - Though explored for +50 years, technology challenges prevented the successful materialization
- UPMEM has designed and fabricated the first publicly-available real-world PIM architecture
  - DDR4 chips embedding in-order multithreaded DRAM Processing Units (DPUs)
- Our work:
  - Introduction to UPMEM programming model and PIM architecture
  - Microbenchmark-based characterization of the DPU
  - Benchmarking and workload suitability study
- Main contributions:
  - Comprehensive characterization and analysis of the first commercially-available PIM architecture
  - **PrIM** (Processing-In-Memory) benchmarks:
    - 16 workloads that are memory-bound in conventional processor-centric systems
    - Strong and weak scaling characteristics
  - Comparison to state-of-the-art CPU and GPU
- Takeaways:
  - Workload characteristics for PIM suitability
  - Programming recommendations
  - Suggestions and hints for hardware and architecture designers of future PIM systems
  - PrIM: (a) programming samples, (b) evaluation and comparison of current and future PIM systems

Long arXiv Version

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]

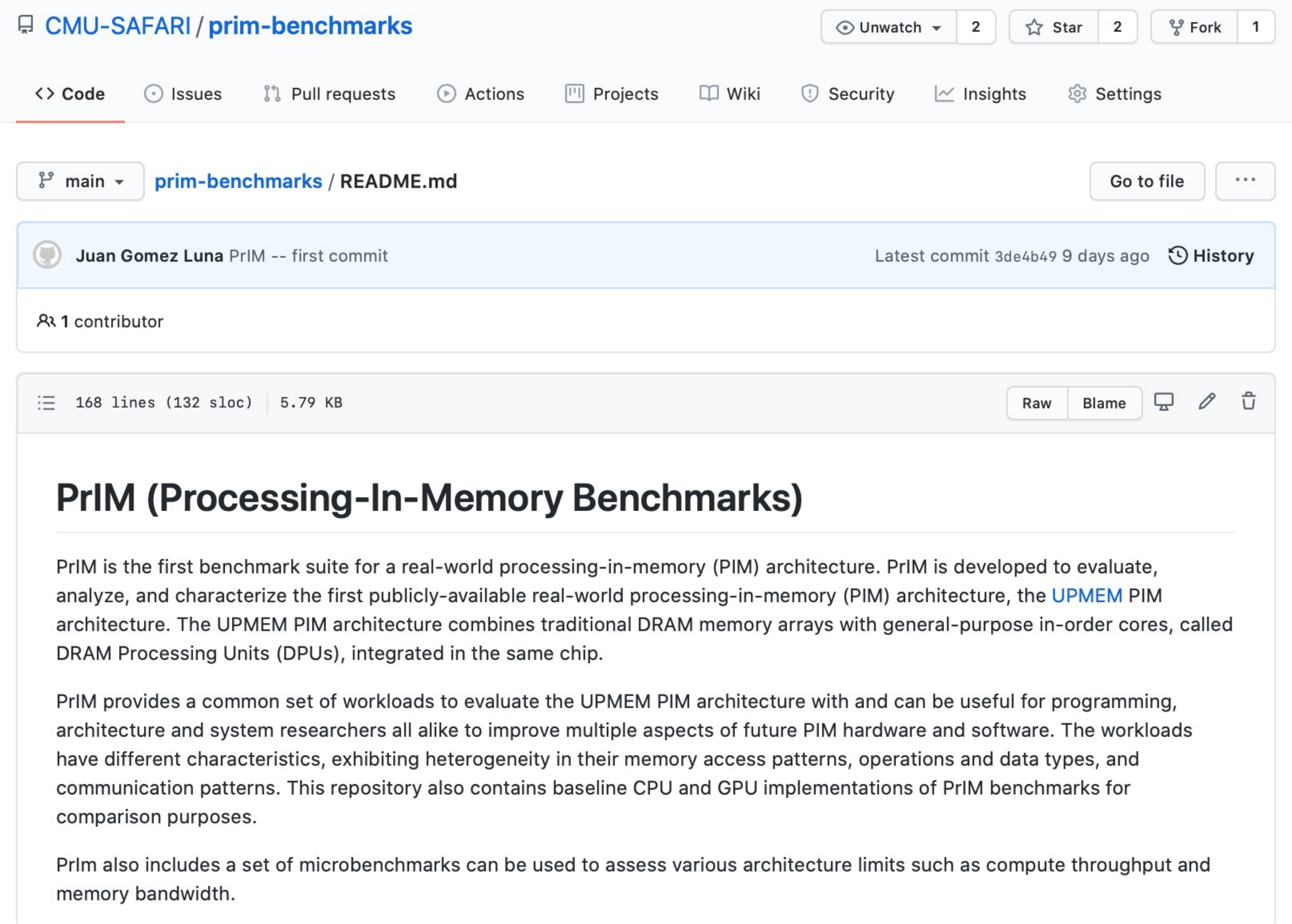
[1]ETH Zürich [2]American University of Beirut [3]University of Malaga [4]National Technical University of Athens

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

https://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

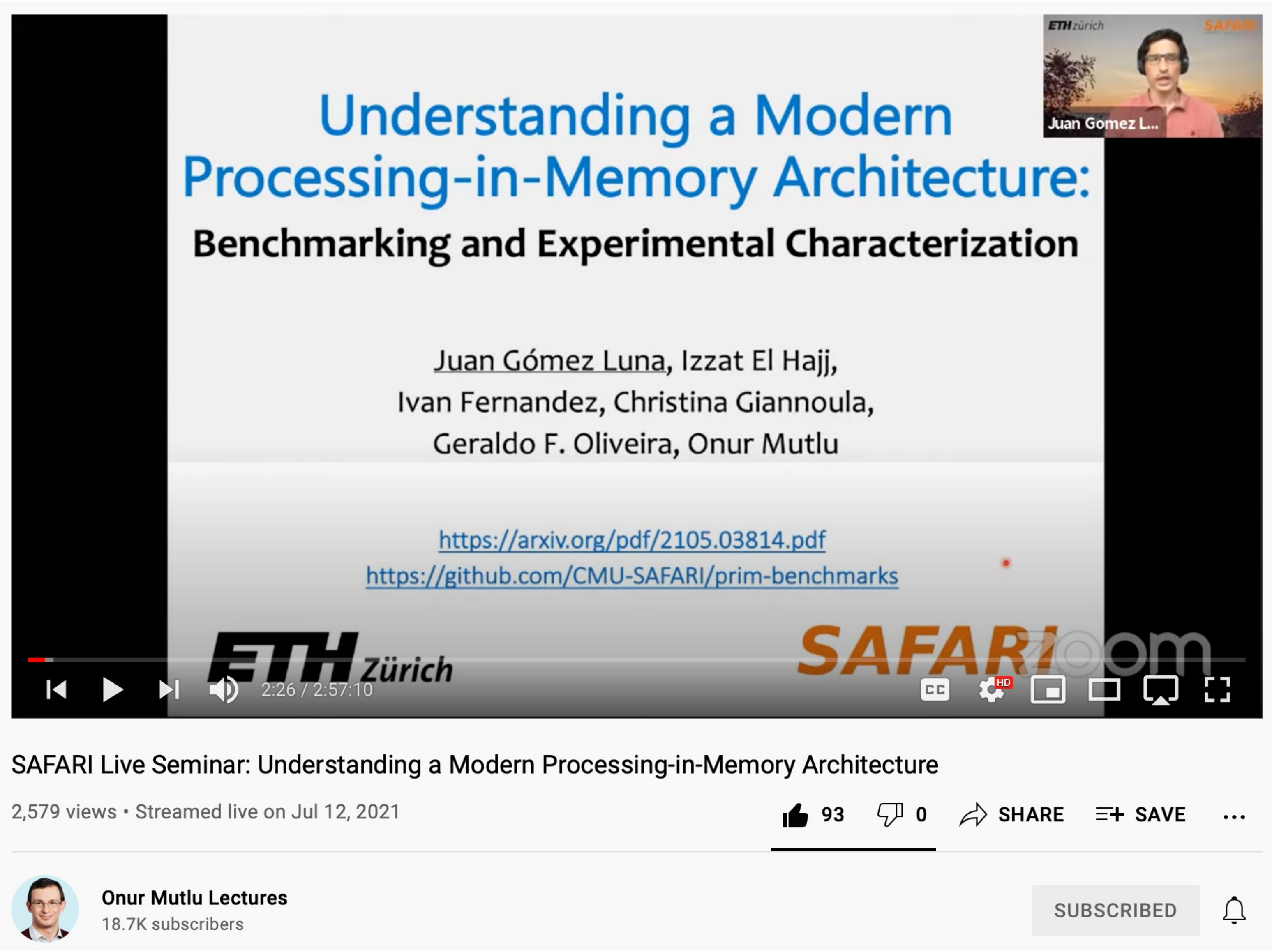
PrIM (Processing-In-Memory Benchmarks)

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.

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

Understanding a Modern PIM Architecture



Understanding a Modern Processing-in-Memory Architecture:

Benchmarking and Experimental Characterization

Juan Gómez Luna, Izzat El Hajj,
Ivan Fernandez, Christina Giannoula,
Geraldo F. Oliveira, Onur Mutlu

https://arxiv.org/pdf/2105.03814.pdf
https://github.com/CMU-SAFARI/prim-benchmarks

SAFARI Live Seminar: Understanding a Modern Processing-in-Memory Architecture

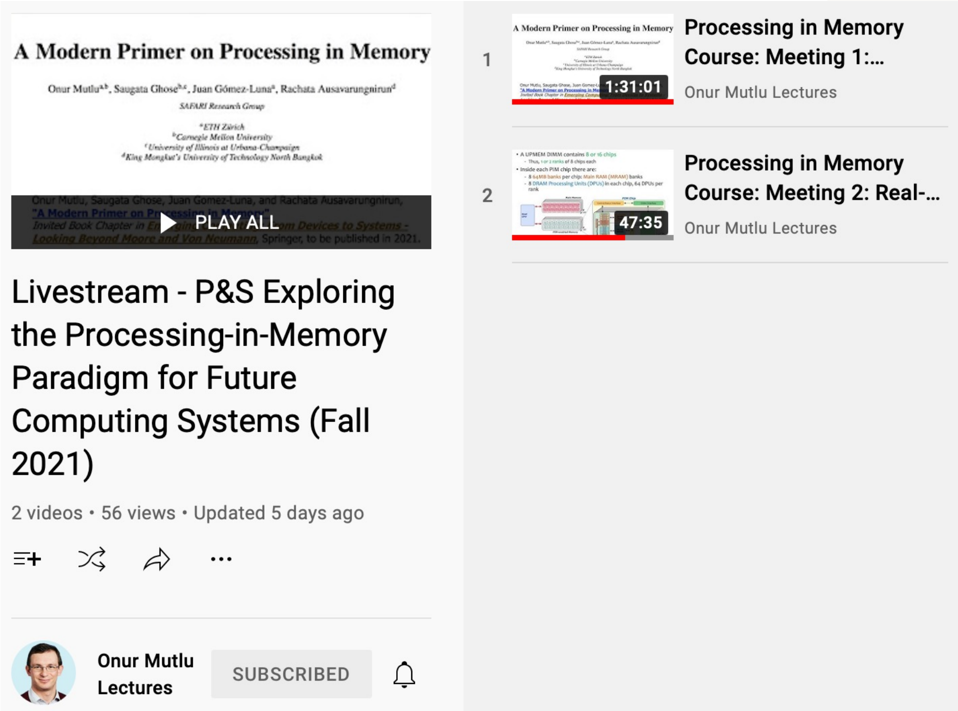
2,579 views • Streamed live on Jul 12, 2021

Onur Mutlu Lectures
18.7K subscribers

https://www.youtube.com/watch?v=D8Hjy2iU9l4&list=PL5Q2soXY2Zi_tOTAYm--dYByNPL7JhwR9

Processing-in-Memory Course (Fall 2021)

- Short weekly lectures
- Hands-on projects



A Modern Primer on Processing in Memory

Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, Rachata Ausavarungnirun

SAFARI Research Group

ETH Zürich
Carnegie Mellon University
University of Illinois at Urbana-Champaign
King Mongkut's University of Technology North Bangkok

▶ PLAY ALL

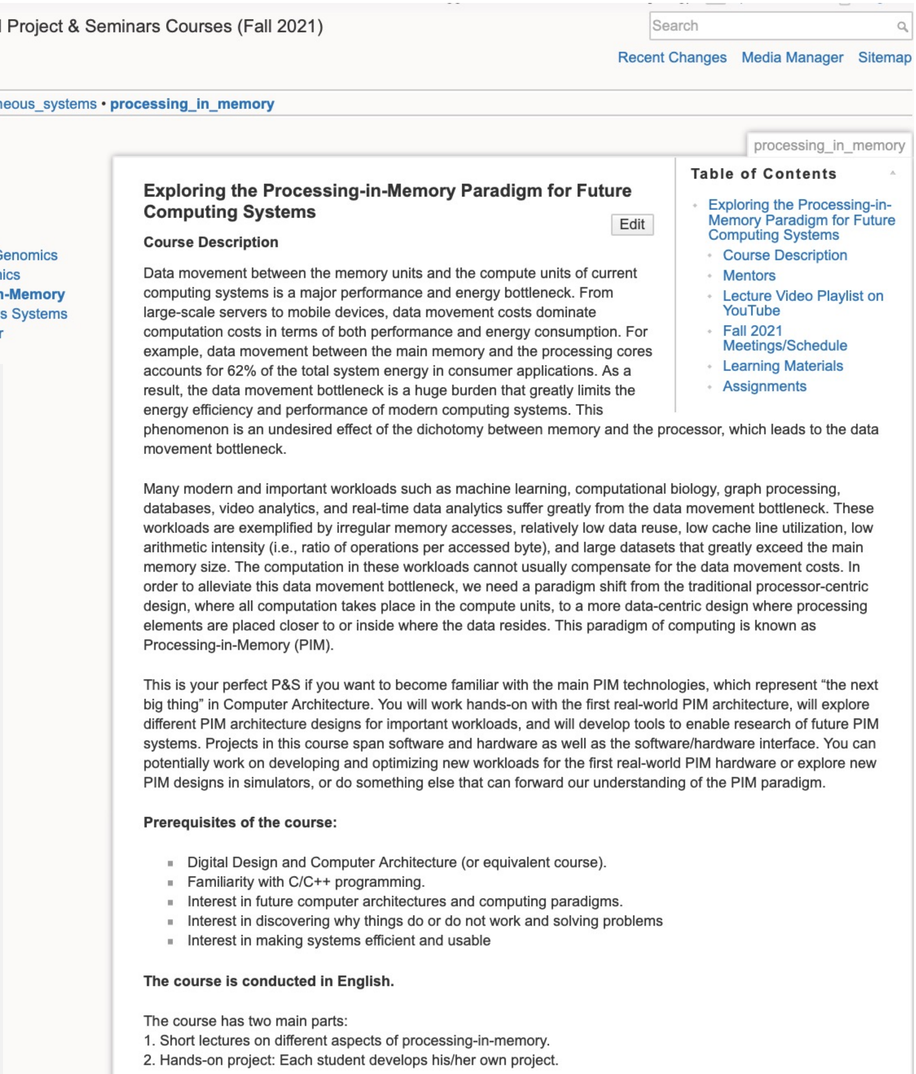
Livestream - P&S Exploring the Processing-in-Memory Paradigm for Future Computing Systems (Fall 2021)

2 videos • 56 views • Updated 5 days ago

Onur Mutlu Lectures SUBSCRIBED

1 Processing in Memory Course: Meeting 1:... 1:31:01
Onur Mutlu Lectures

2 Processing in Memory Course: Meeting 2: Real-... 47:35
Onur Mutlu Lectures



SAFARI Project & Seminars Courses (Fall 2021)

Search

Recent Changes Media Manager Sitemap

Trace: • heterogeneous_systems • processing_in_memory

Home

Projects

- SoftMC
- Ramulator
- Accelerating Genomics
- Mobile Genomics
- **Processing-in-Memory**
- Heterogeneous Systems
- SSD Simulator

processing_in_memory

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

Edit

Table of Contents

- Exploring the Processing-in-Memory Paradigm for Future Computing Systems
  - Course Description
  - Mentors
  - Lecture Video Playlist on YouTube
  - Fall 2021 Meetings/Schedule
  - Learning Materials
  - Assignments

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.

Prerequisites of the course:

- Digital Design and Computer Architecture (or equivalent course).
- Familiarity with C/C++ programming.
- Interest in future computer architectures and computing paradigms.
- Interest in discovering why things do or do not work and solving problems
- Interest in making systems efficient and usable

The course is conducted in English.

The course has two main parts:
1. Short lectures on different aspects of processing-in-memory.
2. Hands-on project: Each student develops his/her own project.

https://youtube.com/playlist?list=PL5Q2soXY2Zi-841fUYYUK9EsXKhQKRPyX

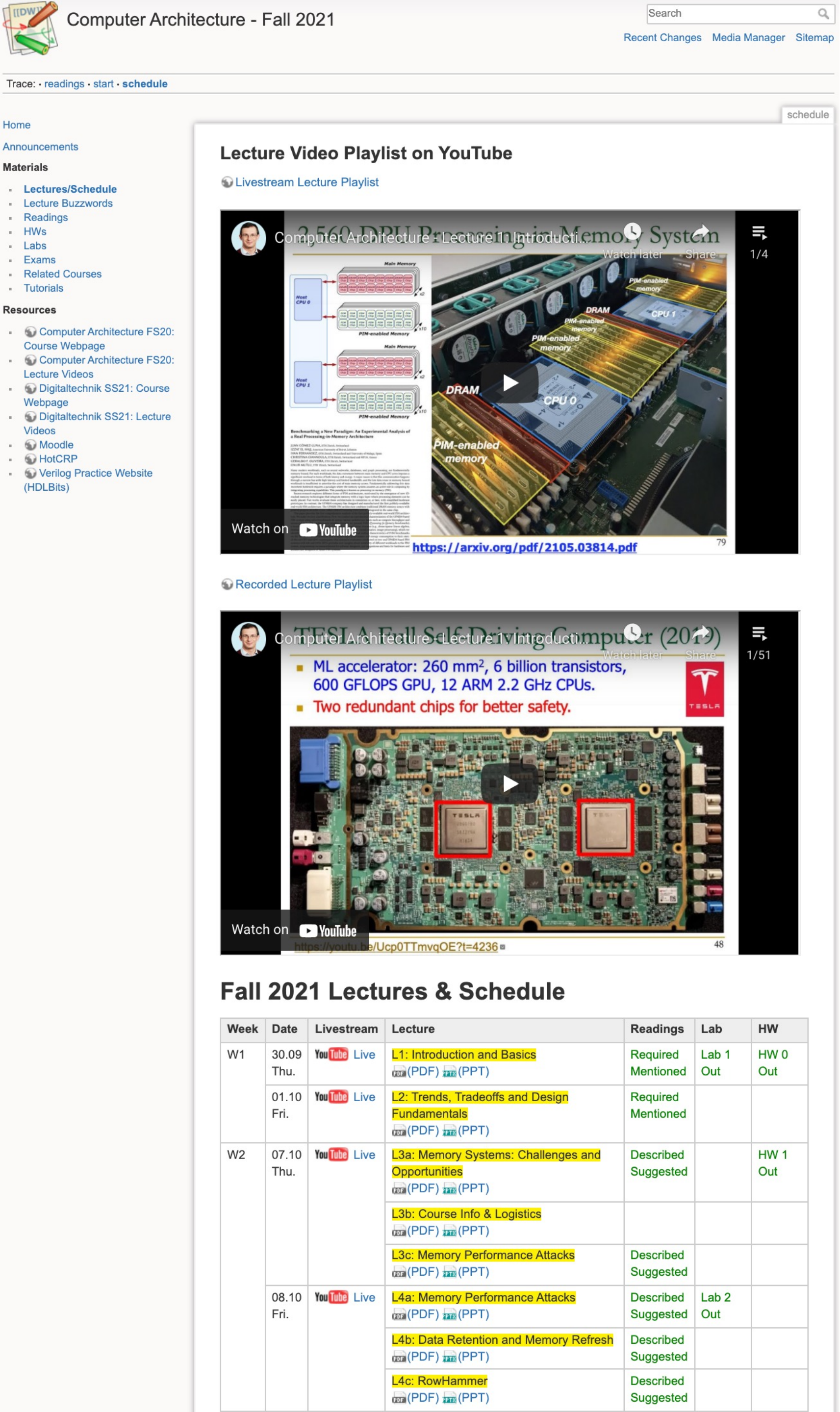
https://safari.ethz.ch/projects_and_seminars/fall2021/doku.php?id=processing_in_memory

Comp Arch (Fall 2021)

- https://safari.ethz.ch/architecture/fall2021/doku.php?id=schedule

- **Youtube Livestream:**
  - https://www.youtube.com/watch?v=4yfkM_5EFgo&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF

- Master's level course
  - Taken by Bachelor's/Masters/PhD students
  - Cutting-edge research topics + fundamentals in Computer Architecture
  - 5 Simulator-based Lab Assignments
  - Potential research exploration
  - Many research readings



Computer Architecture - Fall 2021

Search

Recent Changes Media Manager Sitemap

Trace: • readings • start • **schedule**

schedule

Home

Announcements

Materials

- **Lectures/Schedule**
- Lecture Buzzwords
- Readings
- HWs
- Labs
- Exams
- Related Courses
- Tutorials

Resources

- Computer Architecture FS20: Course Webpage
- Computer Architecture FS20: Lecture Videos
- Digitaltechnik SS21: Course Webpage
- Digitaltechnik SS21: Lecture Videos
- Moodle
- HotCRP
- Verilog Practice Website (HDLBits)

Lecture Video Playlist on YouTube

Livestream Lecture Playlist

Computer Architecture - Lecture 1: Introducti...
Watch later
Share
1/4
2,560-DPU Processing-in-Memory System
Main Memory
Host CPU 0
Host CPU 1
PIM-enabled Memory
PIM-enabled memory
DRAM
CPU 0
CPU 1
Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture
Watch on YouTube
https://arxiv.org/pdf/2105.03814.pdf
79

Recorded Lecture Playlist

Computer Architecture - Lecture 1: Introducti...
Watch later
Share
1/51
TESLA Full Self-Driving Computer (2019)
ML accelerator: 260 mm², 6 billion transistors, 600 GFLOPS GPU, 12 ARM 2.2 GHz CPUs.
Two redundant chips for better safety.
TESLA
Watch on YouTube
https://youtu.be/Ucp0TTmvqOE?t=4236
48

Fall 2021 Lectures & Schedule

Week	Date	Livestream	Lecture	Readings	Lab	HW
W1	30.09 Thu.	YouTube Live	L1: Introduction and Basics (PDF) (PPT)	Required Mentioned	Lab 1 Out	HW 0 Out
	01.10 Fri.	YouTube Live	L2: Trends, Tradeoffs and Design Fundamentals (PDF) (PPT)	Required Mentioned		
W2	07.10 Thu.	YouTube Live	L3a: Memory Systems: Challenges and Opportunities (PDF) (PPT)	Described Suggested		HW 1 Out
			L3b: Course Info & Logistics (PDF) (PPT)			
			L3c: Memory Performance Attacks (PDF) (PPT)	Described Suggested		
	08.10 Fri.	YouTube Live	L4a: Memory Performance Attacks (PDF) (PPT)	Described Suggested	Lab 2 Out	
			L4b: Data Retention and Memory Refresh (PDF) (PPT)	Described Suggested		
			L4c: RowHammer (PDF) (PPT)	Described Suggested		

Workshop on Computing with Unconventional Technologies (CUT 2021)

Benchmarking Memory-Centric Computing Systems:

Analysis of Real Processing-in-Memory Hardware

Juan Gómez Luna, Izzat El Hajj,
Ivan Fernandez, Christina Giannoula,
Geraldo F. Oliveira, Onur Mutlu

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

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

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

ETH Zürich

SAFARI