Accelerating Irregular Applications via Efficient Synchronization and Data Access Techniques

Christina Giannoula

PhD Thesis Oral

Committee

Georgios Goumas (NTUA)
Nectarios Koziris (NTUA)
Onur Mutlu (ETH Zürich)
Dionisios Pnevmatikatos (NTUA)
Stefanos Kaxiras (Uppsala University)
Dimitris Gizopoulos (University of Athens)
Vasileios Papaefstathiou (University of Crete)

Irregular Applications

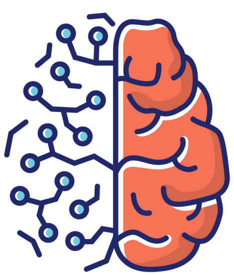
Graph Analytics

Databases

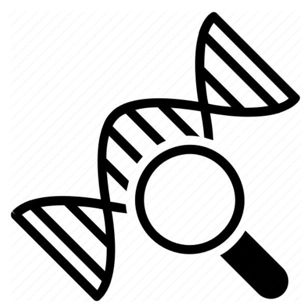
Medical Imaging

How can we accelerate the irregular applications?

Neural Networks



Bioinformatics

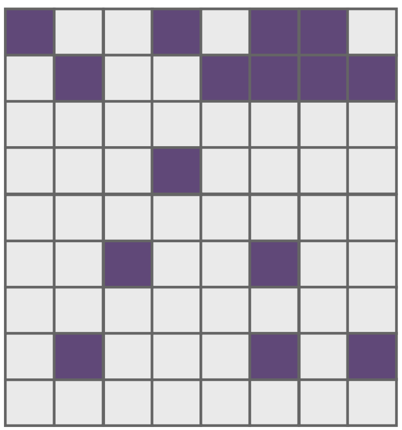


Economic Modeling

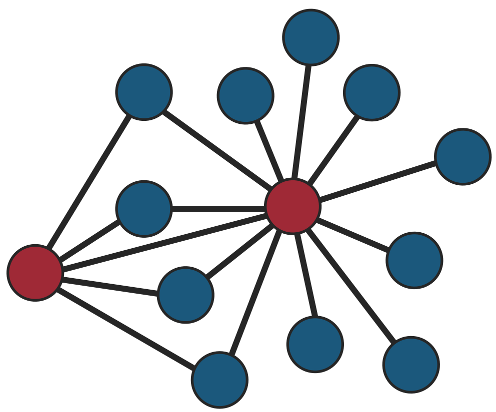
Characteristic 1: Inherent Imbalance

- The objects involved are not of equal size

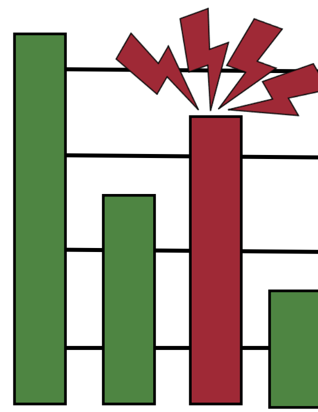
Sparse Matrix



Power-Law Graph



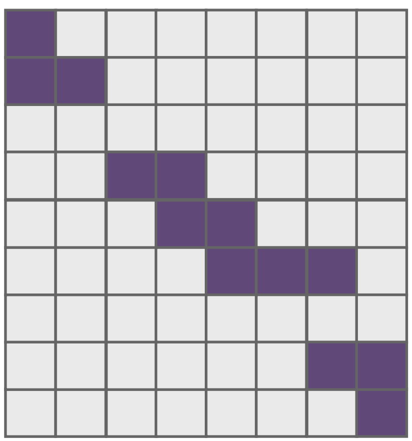
Zipfian Query Distribution



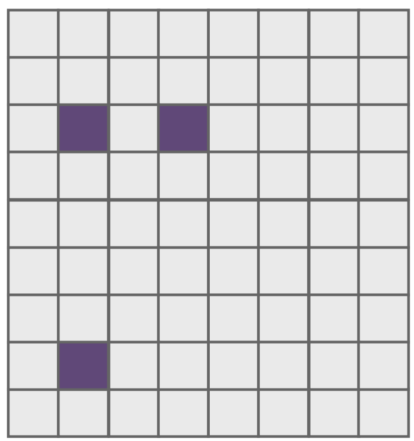
Characteristic 2: Random Memory Accesses

- Not sequential
- Not strided
- Input-driven

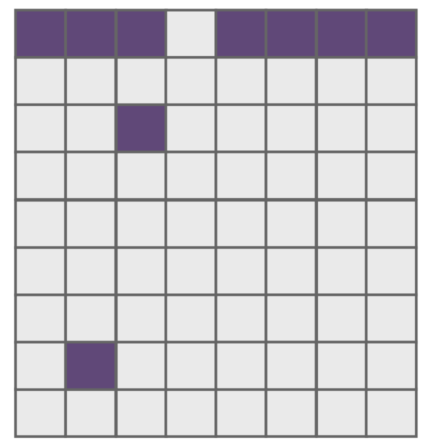
Diagonal Matrix



Highly Sparse Matrix

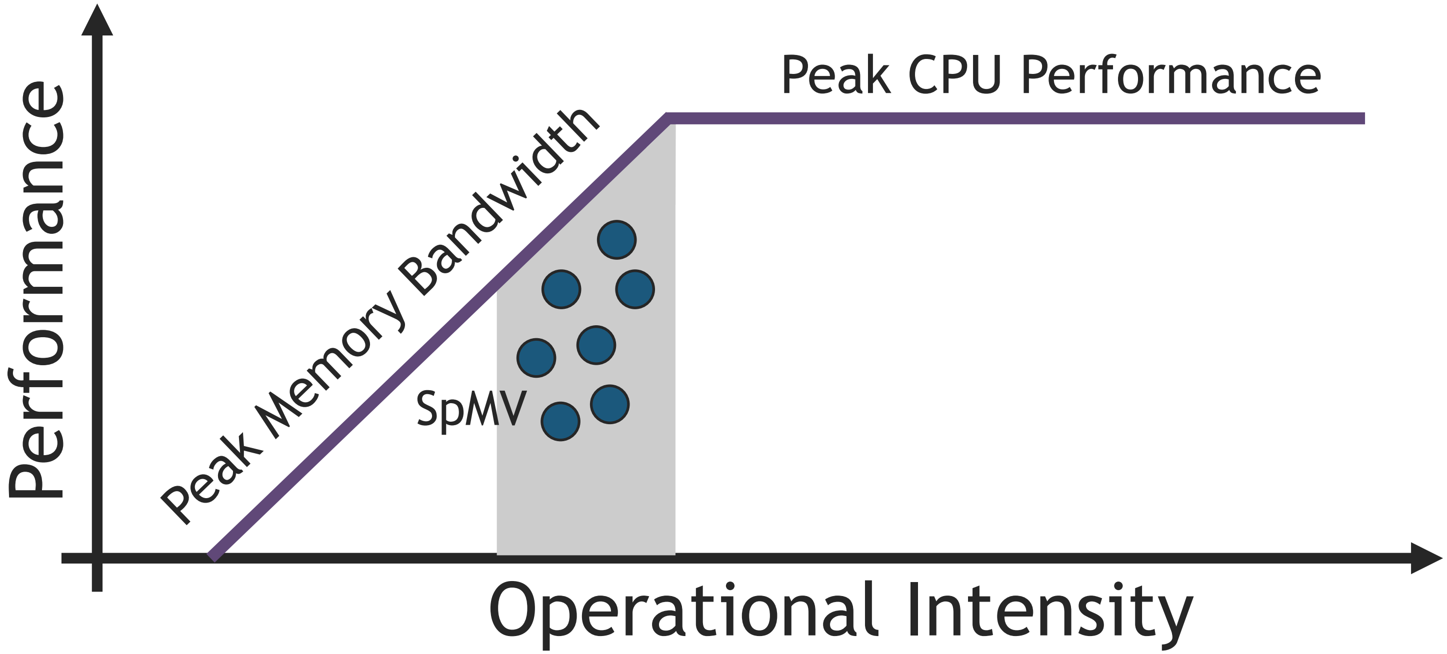


Highly Skewed Matrix



Characteristic 3: Low Operational Intensity

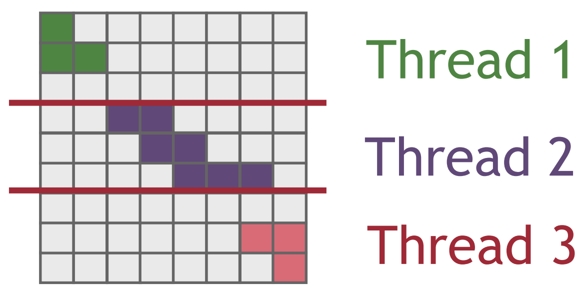
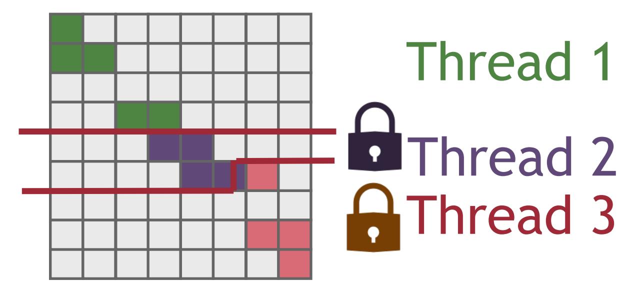
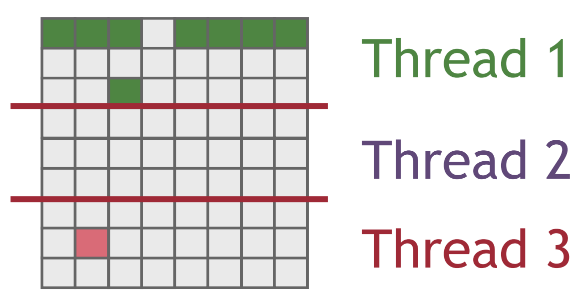
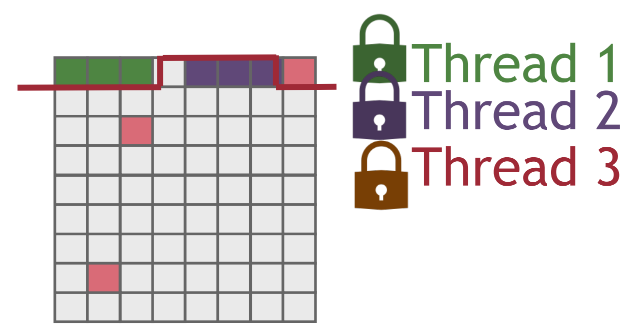
- **High bottleneck** by the memory subsystem



Challenge 1: Excessive Synchronization

- Inherent Imbalance
- Random Memory Accesses

SpMV

	Coarse-Grained Approach	Fine-Grained Approach
Diagonal Matrix		
Highly Skewed Matrix		

Challenge 1: Excessive Synchronization

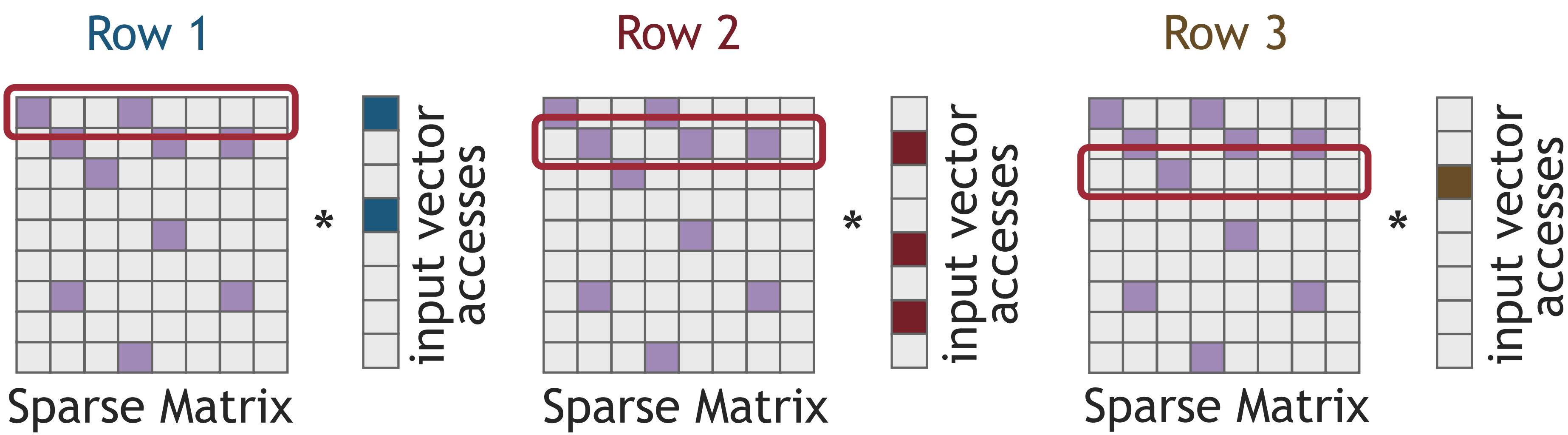
- Inherent Imbalance
- Random Memory Accesses

A large amount of processors' cycles is spent on synchronization

Challenge 2: High Memory Intensity

- Random Memory Accesses
- Low Operational Intensity

The SpMV Execution



Challenge 2: High Memory Intensity

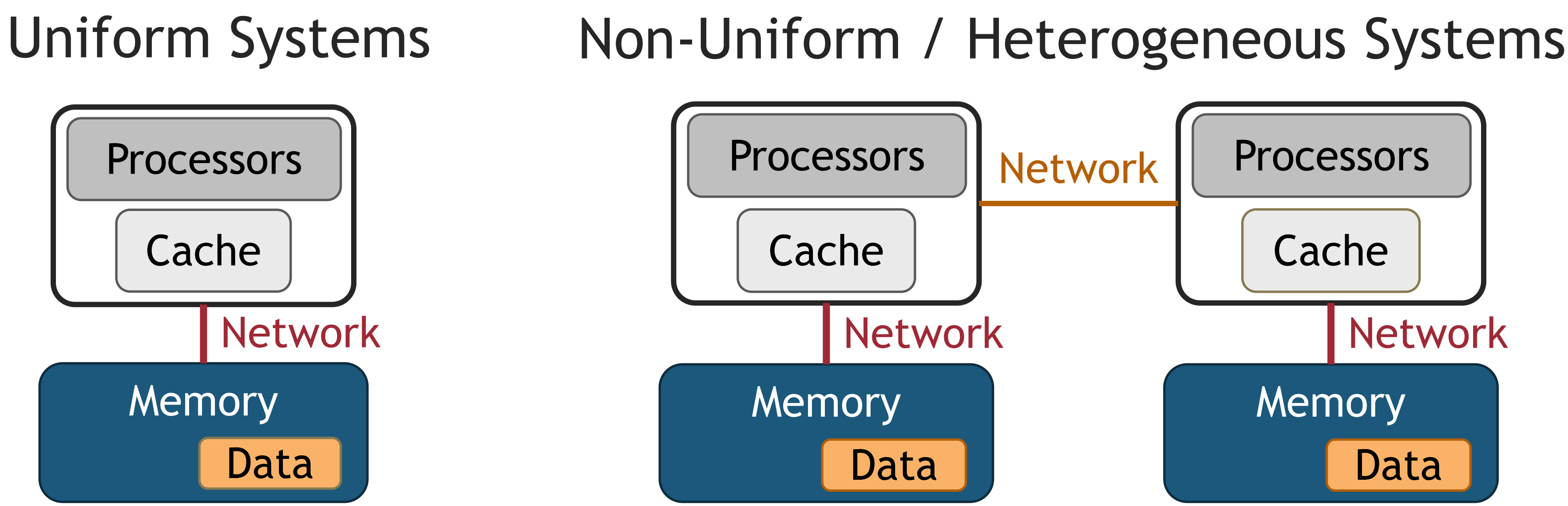
- Random Memory Accesses
- Low Operational Intensity

A large amount of processors' cycles is spent on data accesses

Challenge 2: High Memory Intensity

- Random Memory Accesses
- Low Operational Intensity

The Era of Heterogeneity



Our Approach

Synchronization of Threads

+

Management of Data

The two major priorities

in the execution

of irregular applications

Our Approach

Efficient Synchronization

- High load balance
- Low-cost inter-thread communication
- High levels of parallelism

+

Efficient Data Management

- Low-cost data accesses
- High memory bandwidth

The two major priorities in the execution of irregular applications

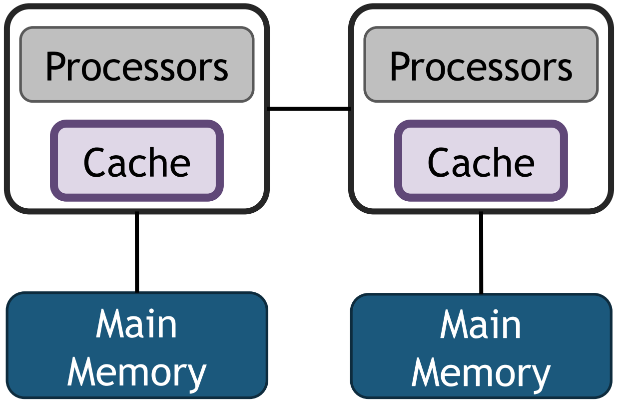
Thesis Statement

Low-overhead synchronization approaches
in cooperation with
well-crafted data access techniques
can significantly improve
performance and energy efficiency
of emerging irregular applications.

Thesis Goal

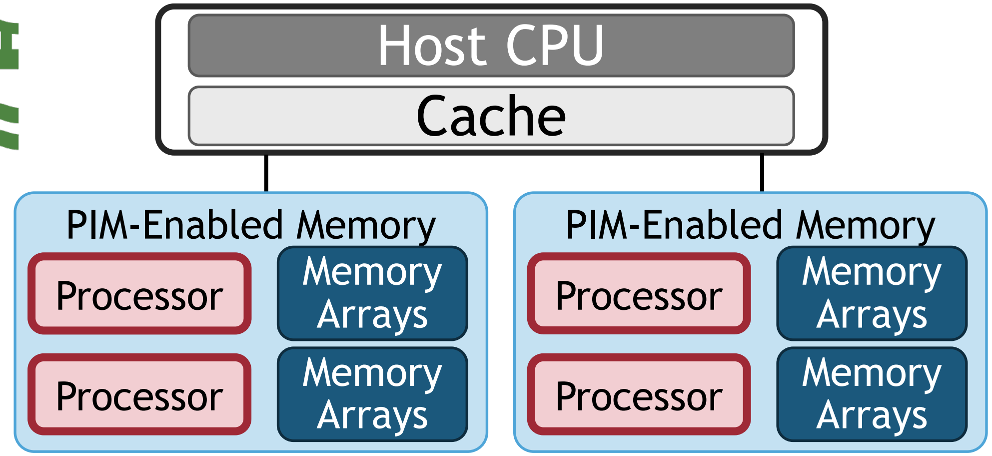
CPU System

(processor-centric)



Processing-In-Memory (PIM) System

(memory-centic)

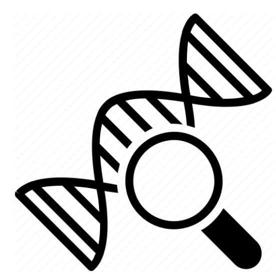


Irregular Applications: important yet difficult

Graph Analytics

Databases

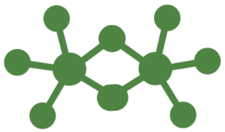
Bioinformatics



Core Contributions

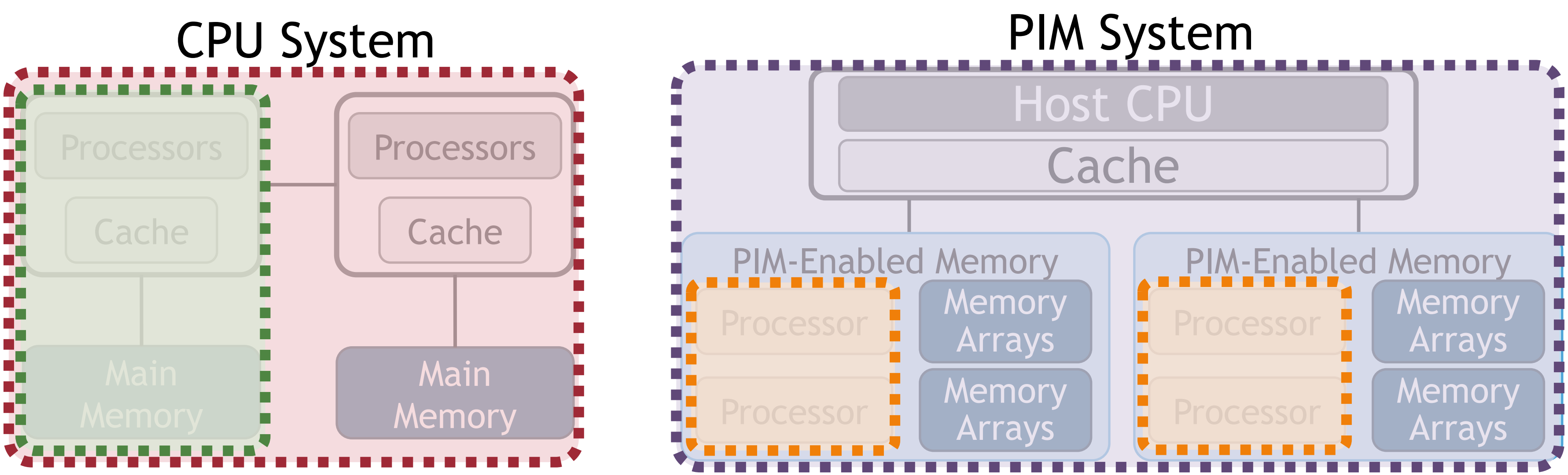
1 Graph Processing

ColorTM (ISC'18, SRC PACT'18)
High-Performance Graph Coloring for CPU Systems



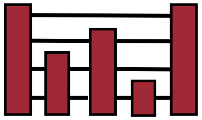
3 Irregular Workloads

SynCron (HPCA'21)
A Lightweight Synchronization Mechanism for PIM Systems



2 Pointer-Chasing

SmartPQ (CF'19)
An Adaptive Priority Queue for NUMA CPU Systems



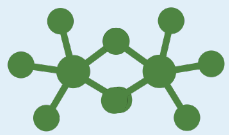
4 Sparse Linear Algebra

SparseP (Sigmetrics'22)
A Library of Efficient Sparse Matrix Vector Multiplication Kernels for Real PIM Systems

Core Contributions

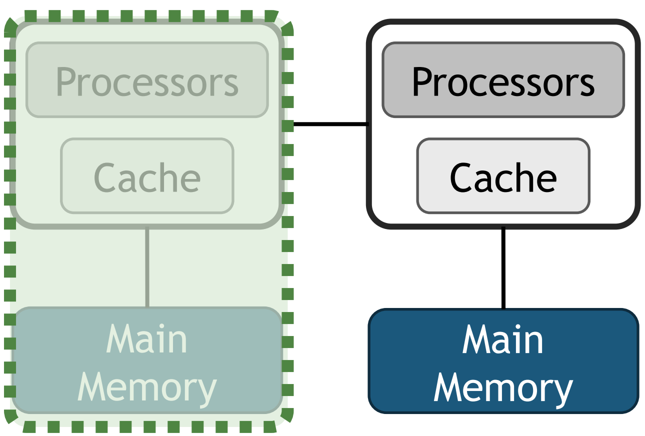
1 Graph Processing

ColorTM (ISC'18, SRC PACT'18)

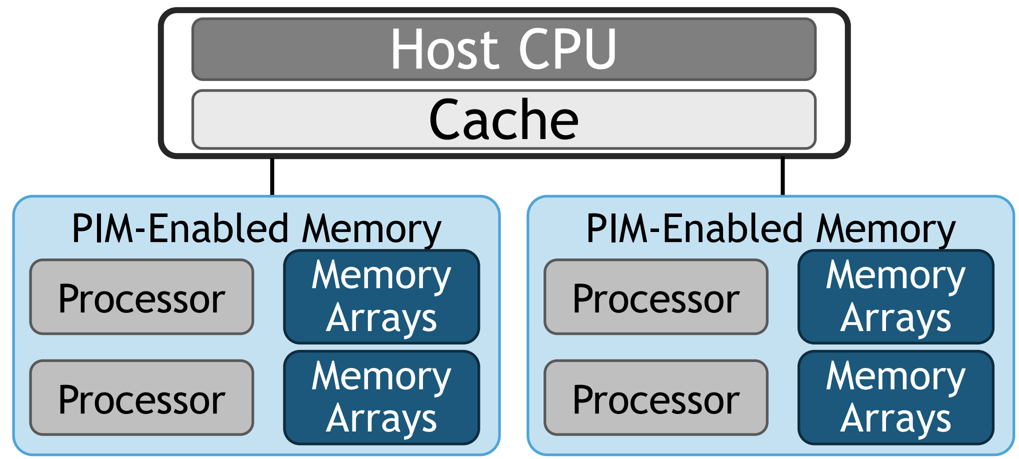


High-Performance Graph Coloring for CPU Systems

CPU System



PIM System

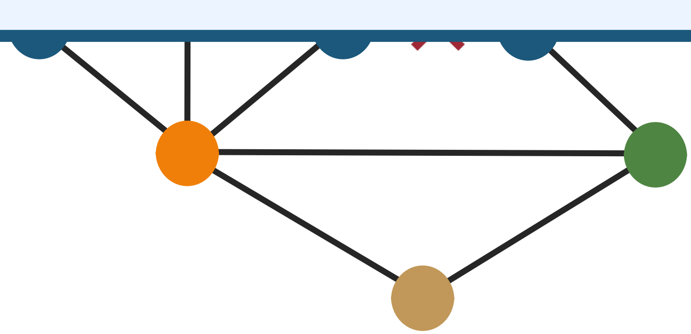


Trade-off between using synchronization with lower data access costs

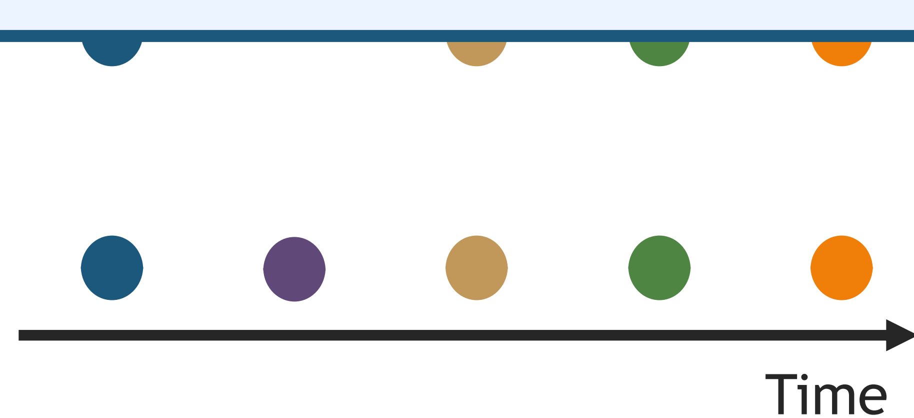
Graph Coloring

Applications: PageRank, Community Detection, Resource Allocation …

The Problem

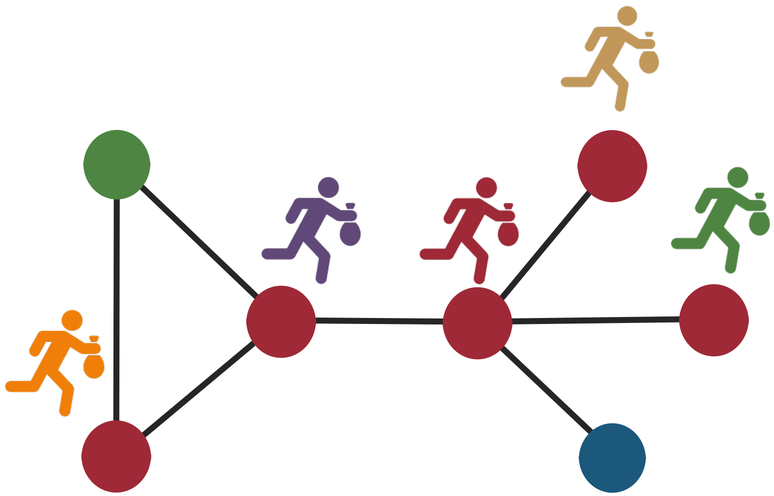


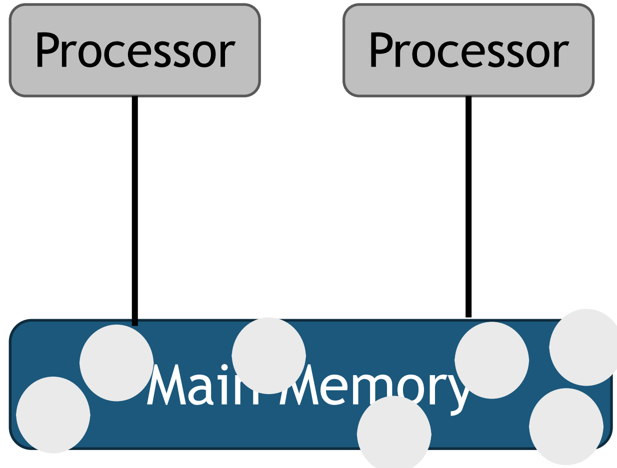
Chromatic Scheduling



How can we accelerate the graph coloring kernel?

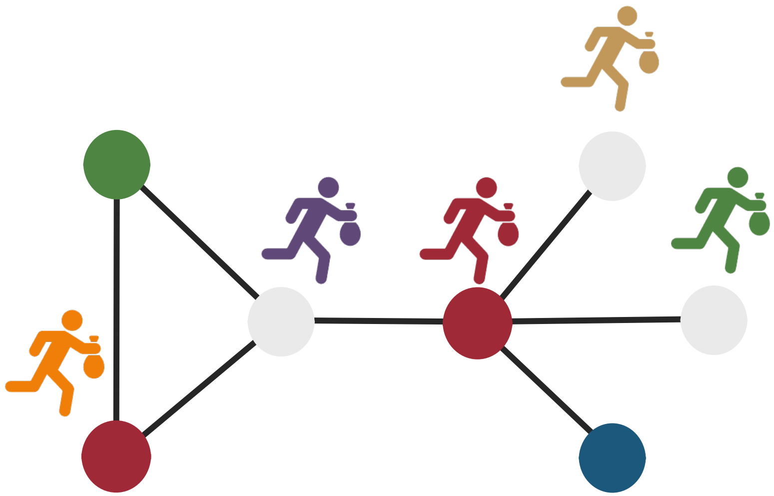
Prior Parallel Algorithms

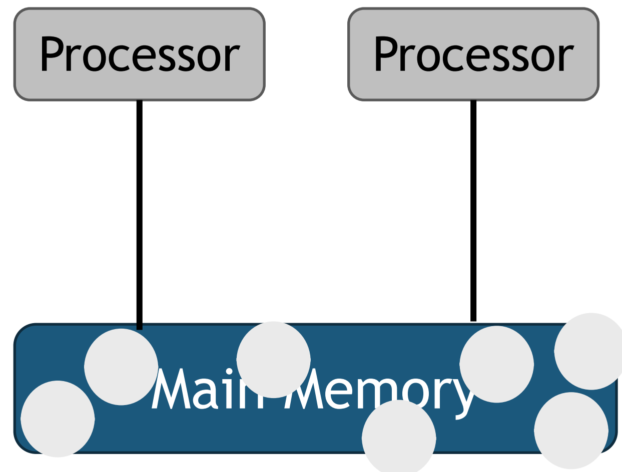




1 Parallel Graph Coloring – No Synchronization

Prior Parallel Algorithms



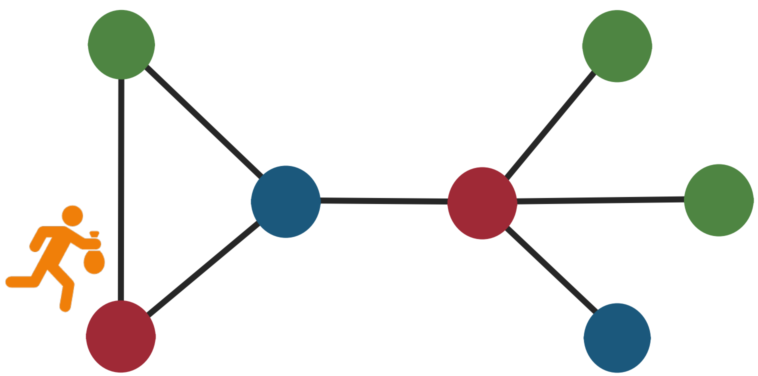


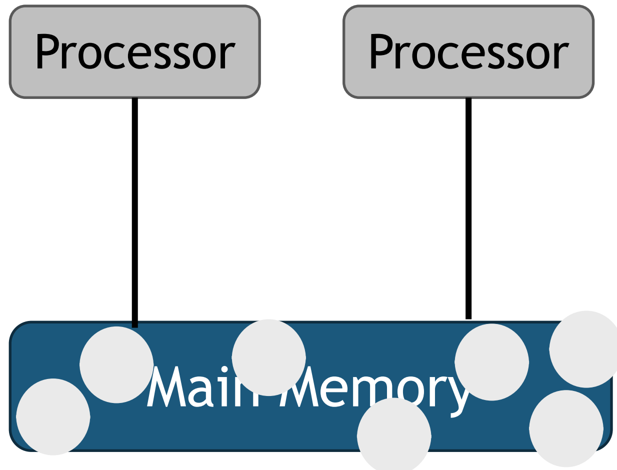
1 Parallel Graph Coloring – No Synchronization

2 Detect Coloring Conflicts

Prior Parallel Algorithms

Sequential Solving (SeqSolve [Gebr.+'00])





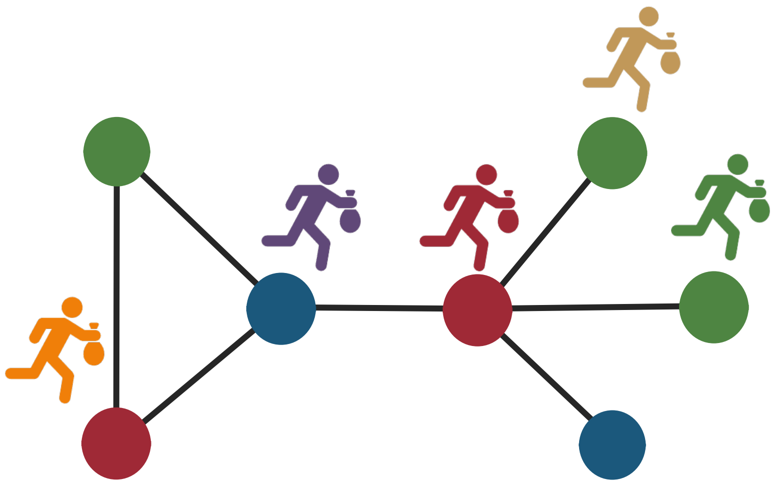
1 Parallel Graph Coloring – No Synchronization

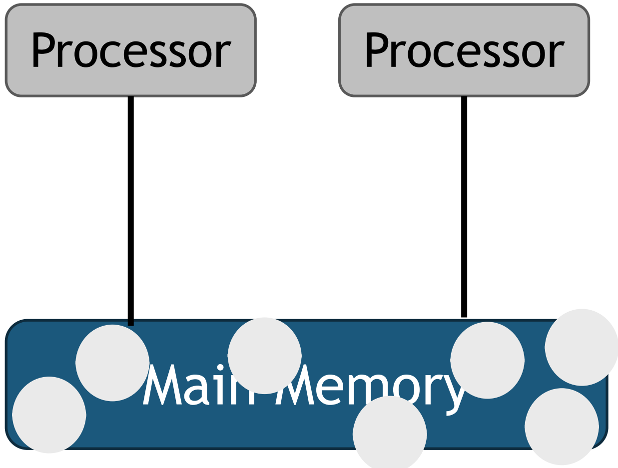
2 Detect Coloring Conflicts

3 Resolve Coloring Conflicts Sequentially

Prior Parallel Algorithms

Iterative Solving (IterSlv [Boman.+'05], IterSlvR [Rokos.+'15])





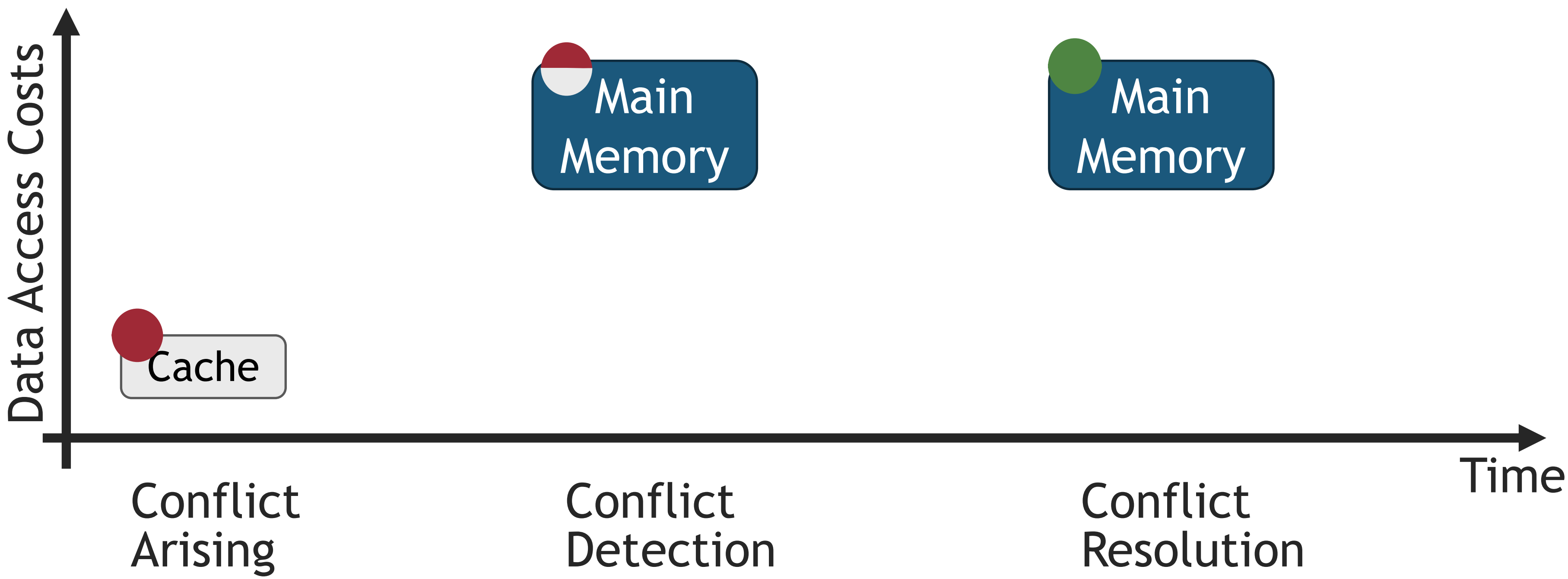
1 Parallel Graph Coloring – No Synchronization

2 Detect Coloring Conflicts

3 Repeat Steps 1 + 2 Multithreaded

Prior Parallel Algorithms

Lazy Iterative Coloring (e.g., SeqSlv, IterSlv, IterSlvR)



Prior Parallel Algorithms

Lazy Iterative Coloring (e.g., SeqSlv, IterSlv, IterSlvR)

- At least 2 iterations on the whole graph
- Lazy coloring conflict detection + resolution

	SeqSlv	IterSlv	IterSlvR
Parallelism	+ +	+ + +	+ + +
Synchronization	+ + +	+ + +	+ + +
Data Accesses	−	− − −	− − −

ColorTM [ISC'18, SRC PACT'18]

Eager Iterative Coloring

- Eager coloring conflict detection + resolution
- Speculative computation + synchronization

	SeqSlv	IterSlv	IterSlvR	ColorTM
Parallelism	+ +	+ + +	+ + +	+ +
Synchronization	+ + +	+ + +	+ + +	+
Data Accesses	−	− − −	− − −	+ + +

ColorTM: Key Idea 1

Eager Conflict Detection + Resolution

✓ Iterate on <u>each</u> <u>vertex</u> until a valid coloring is found

✓ Low Data Access Costs

✓ Low Latency

ColorTM: Key Idea 2

Speculative Synchronization + Computation

✓ Employ hardware transactional memory

✓ Perform most computations speculatively – outside the critical section

✓ Low Synchronization Costs
✓ High Amount of Parallelism

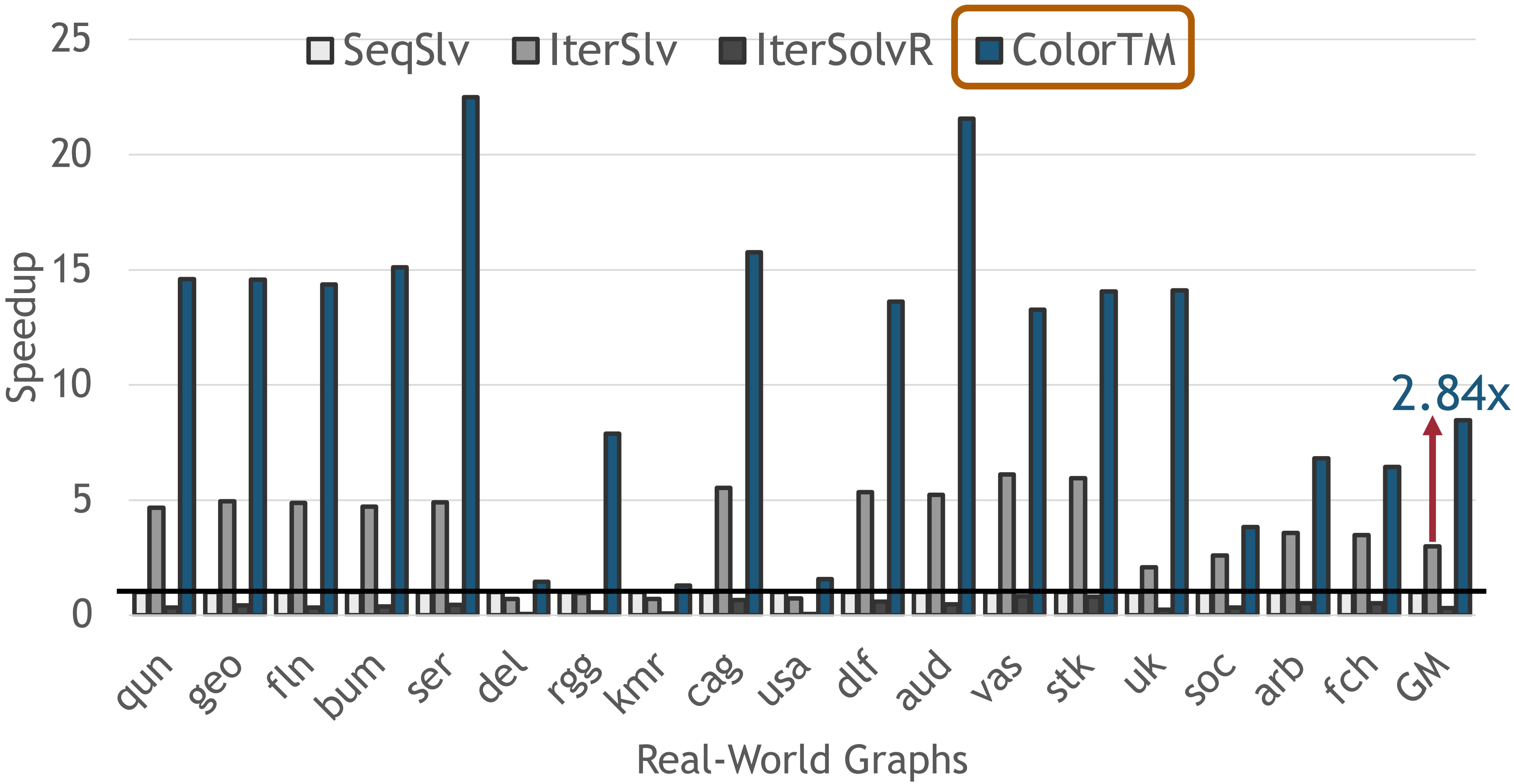
<u>each</u> <u>vertex</u>

2 begin_HTM();

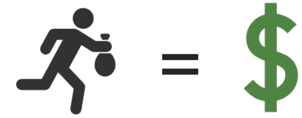
3 validate_color();

4 end_HTM();

Performance Analysis



Balanced ColorTM



Imbalanced Chromatic Scheduling

Balanced Chromatic Scheduling

1.91x faster than prior works using 56 threads

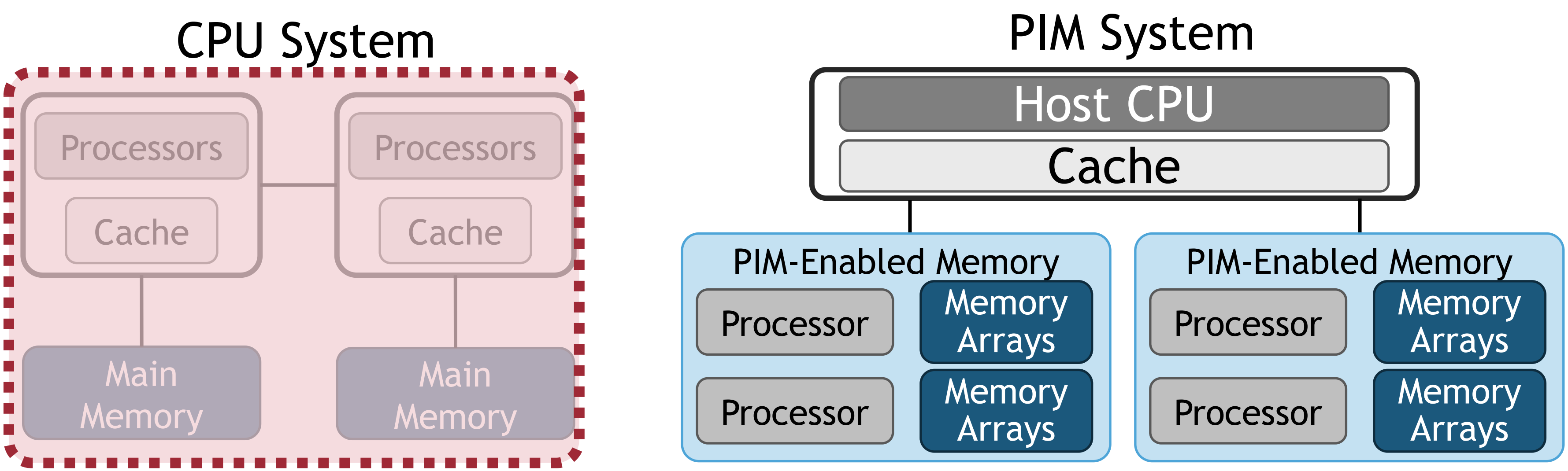
Community Detection: 1.12x faster over the imbalanced variant using 56 threads

Low Resource Utilization

High Resource Utilization

Core Contributions

High contention → low data access costs
Low contention → lightweight synchronization

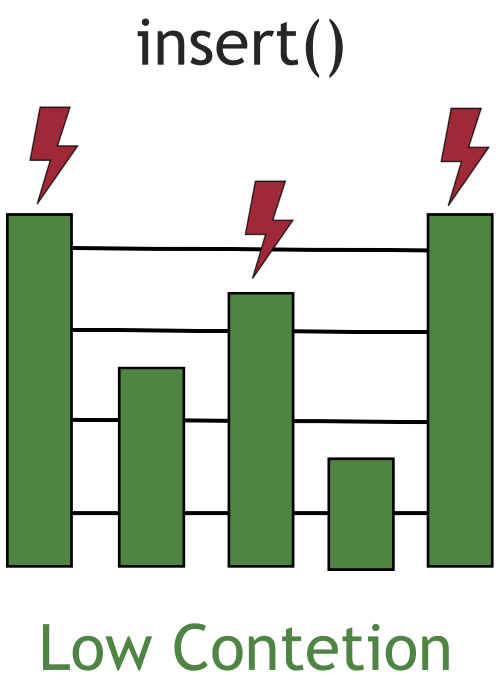


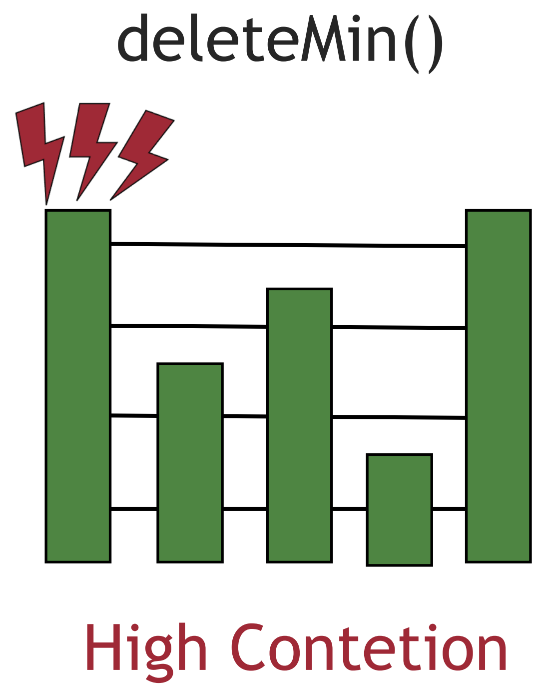
2 Pointer-Chasing

SmartPQ (CF’19)
An Adaptive Priority Queue for NUMA CPU Systems

Motivation

- Priority Queues (PQs) are widely used in graph processing kernels, discrete event simulations …
- Key Observations:
  1. PQs exhibit medium contention

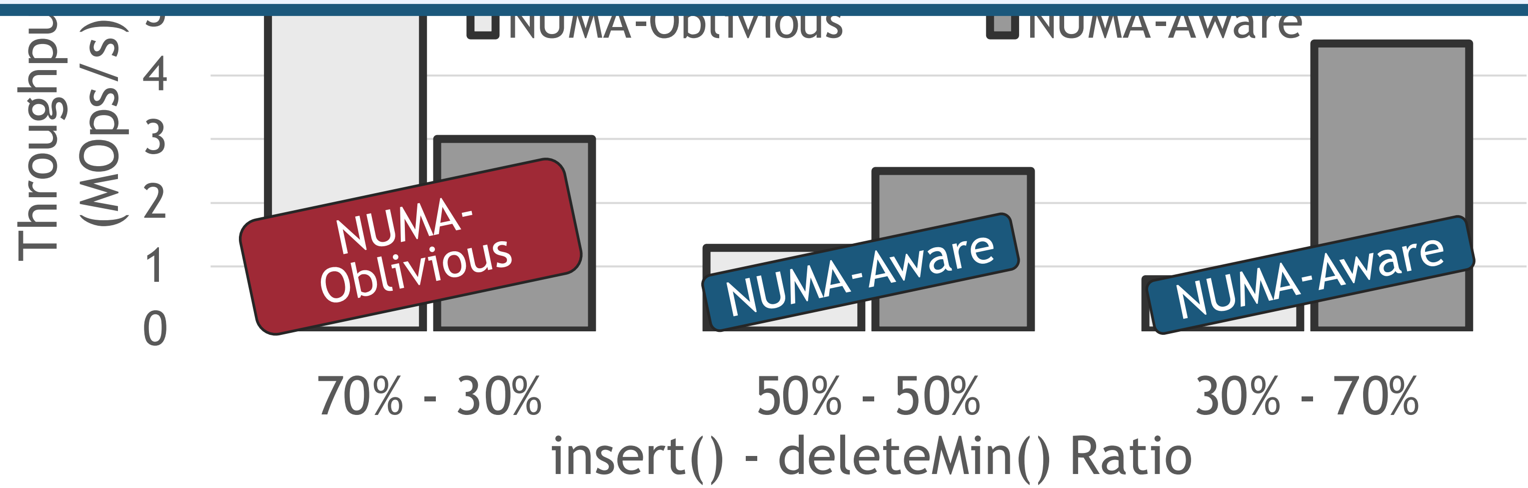




Motivation

- Priority Queues (PQs) are widely used in graph processing kernels, discrete event simulations …
- Key Observations:
  1. PQs exhibit medium contention

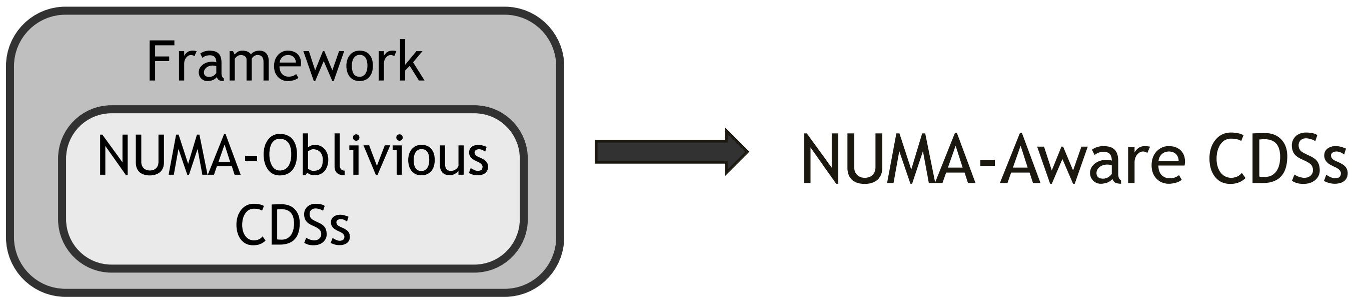
Can we design an ‘intelligent’ PQ to always perform best?



Key Contributions

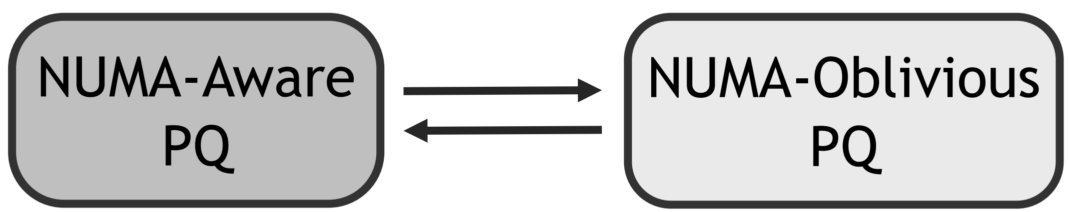
1. A black-box approach to provide high-performance NUMA-aware Concurrent Data Structures (CDSs)

Nuddle:



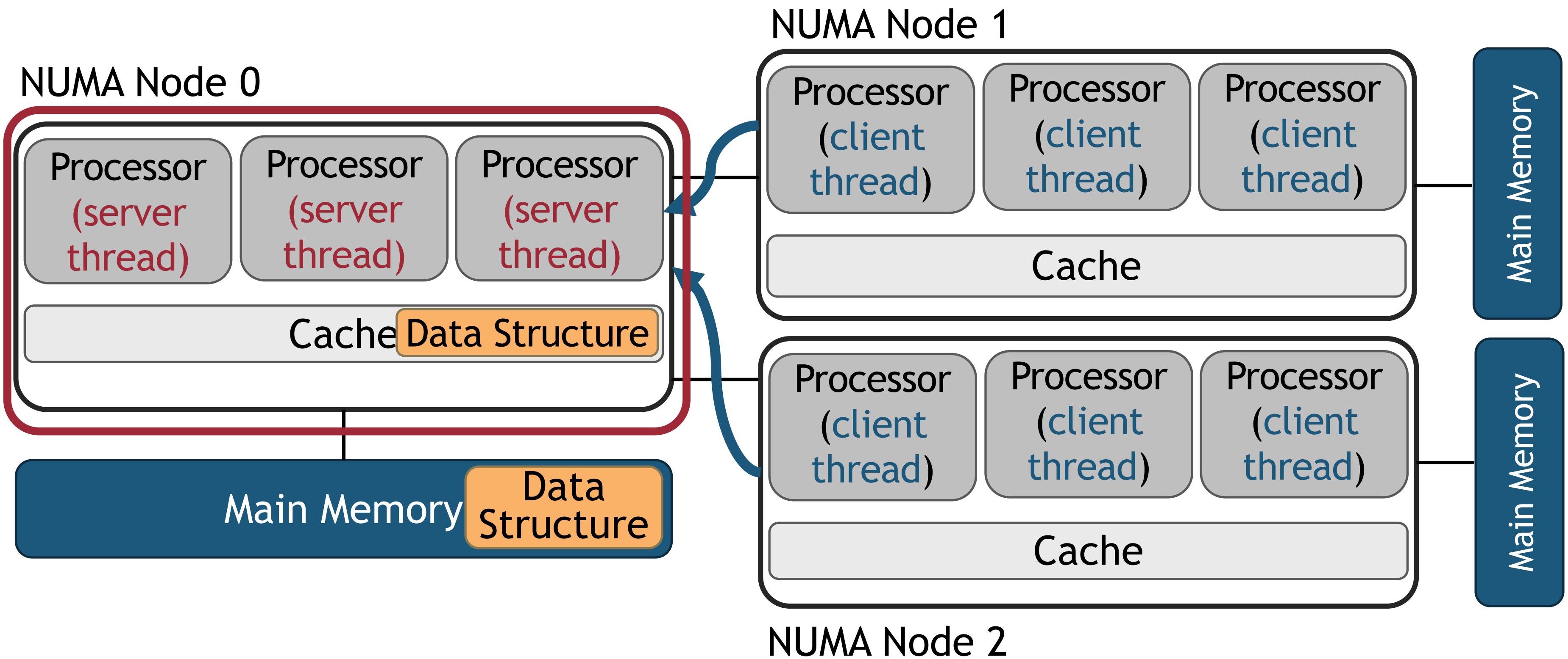
2. An adaptive PQ to perform best under various contention workloads

SmartPQ:



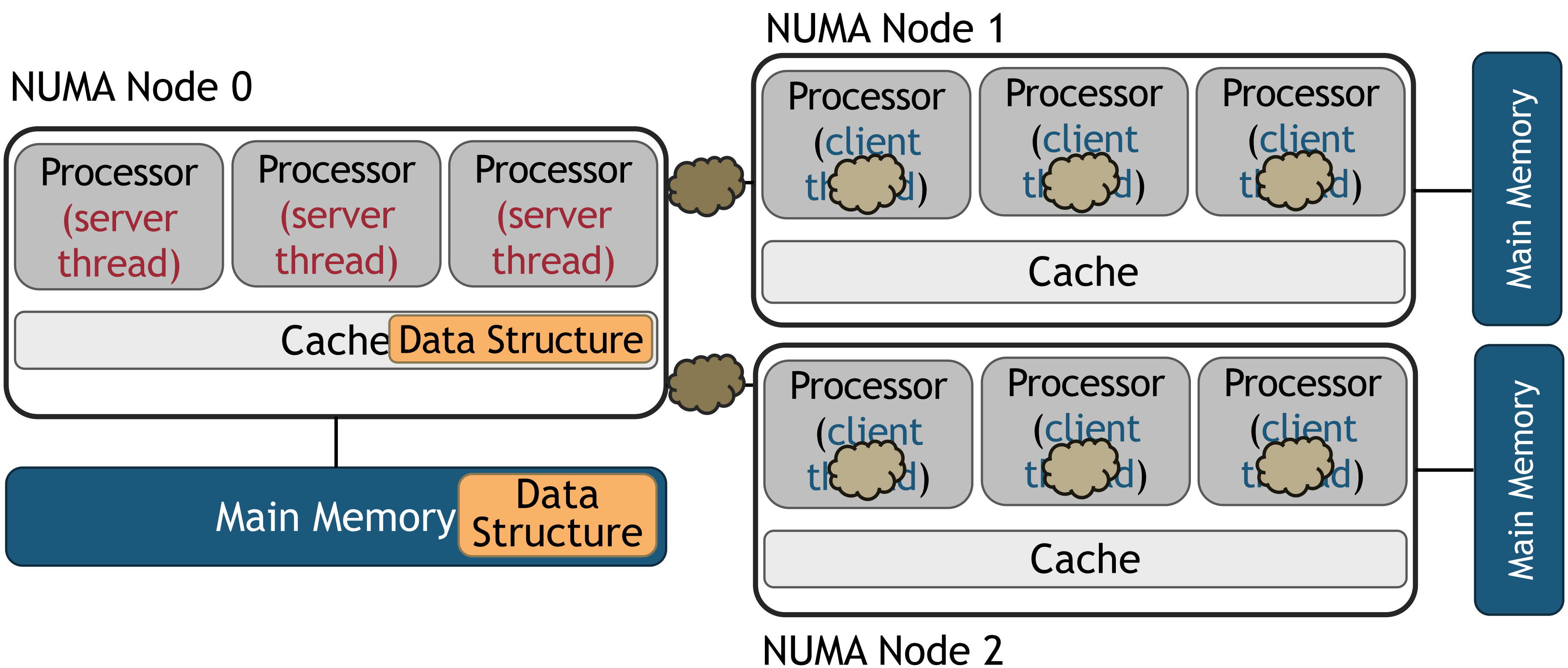
1. NUMA Node Delegation (Nuddle)

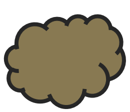
A generic framework to design NUMA-aware CDSs

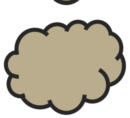


1. NUMA Node Delegation (Nuddle)

A generic framework to design NUMA-aware CDSs

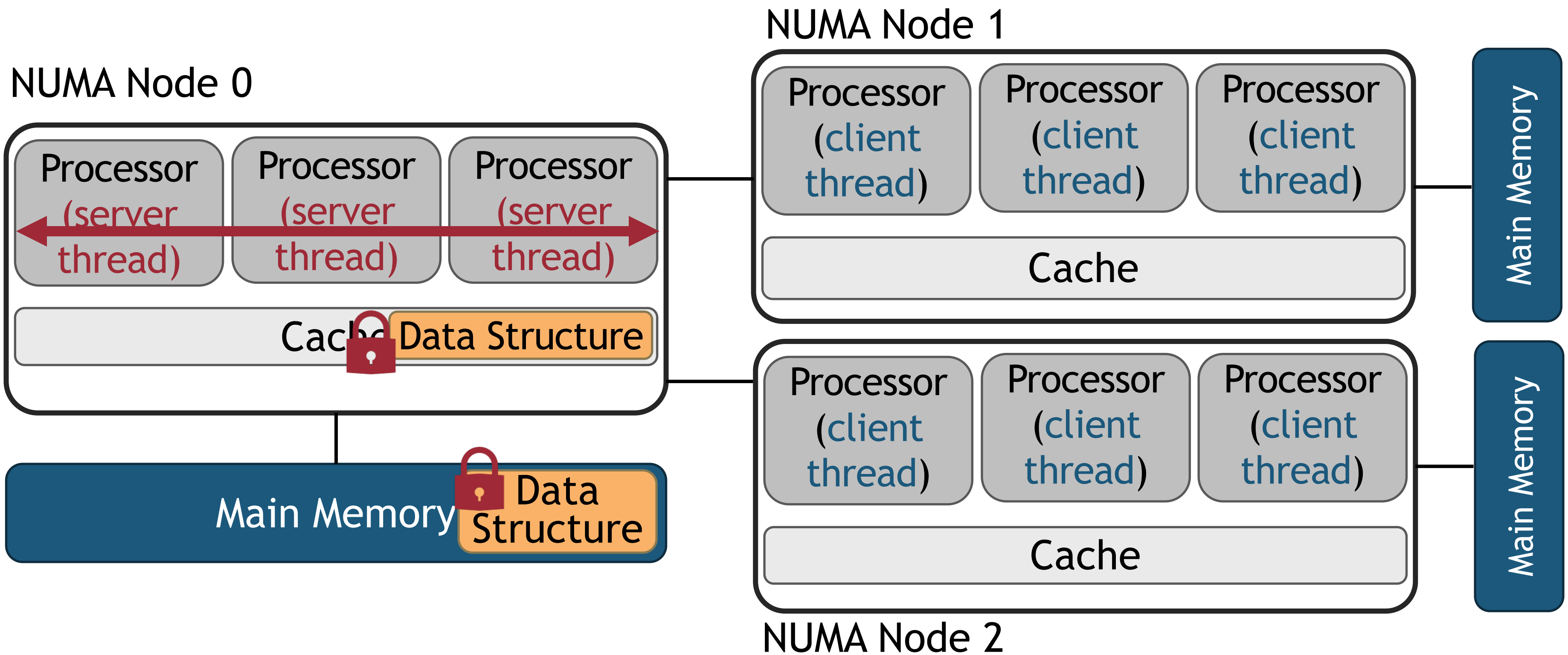



response msg (up 15 clients) [Roghanchi+ SOSP'17]: 1 cache line


request msg (1 client) [Roghanchi+ SOSP'17]: 1 cache line

1. NUMA Node Delegation (Nuddle)

A generic framework to design NUMA-aware CDSs

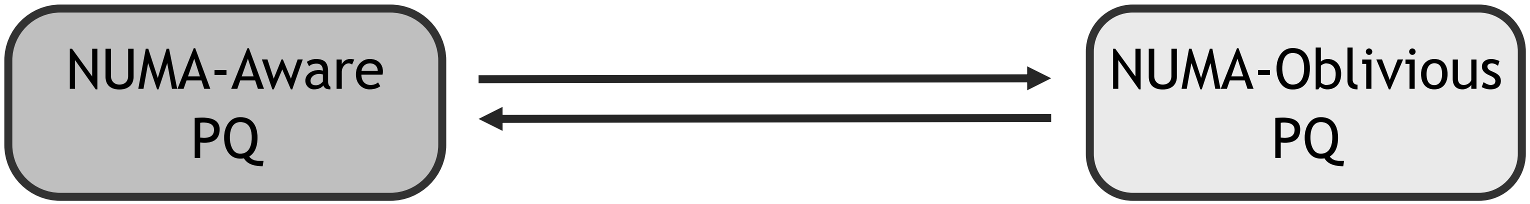


+ Any NUMA-oblivious → NUMA-aware CDS
+ Minimizes the memory traffic
+ Low-overhead communication protocol

− Needs synchronization
− Parallelization is limited to the number of server threads

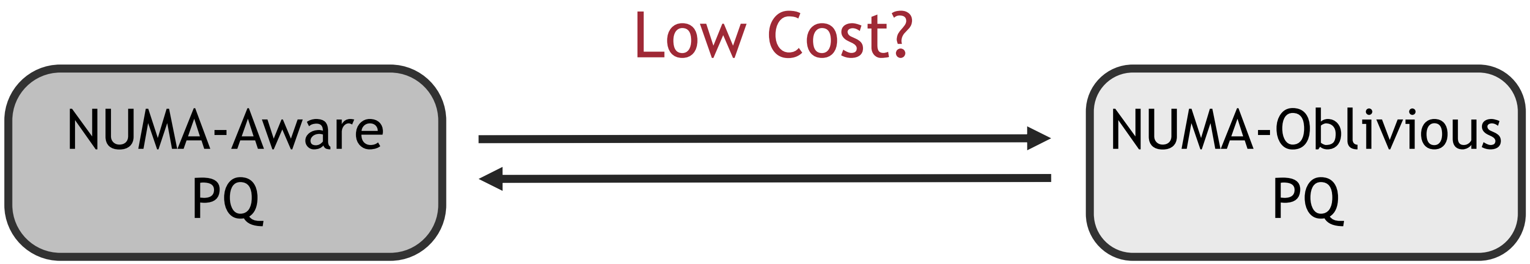
2. SmartPQ

An adaptive PQ that switches
between two algorithmic modes whenever it is needed



2. SmartPQ

An adaptive PQ that switches between two algorithmic modes whenever it is needed

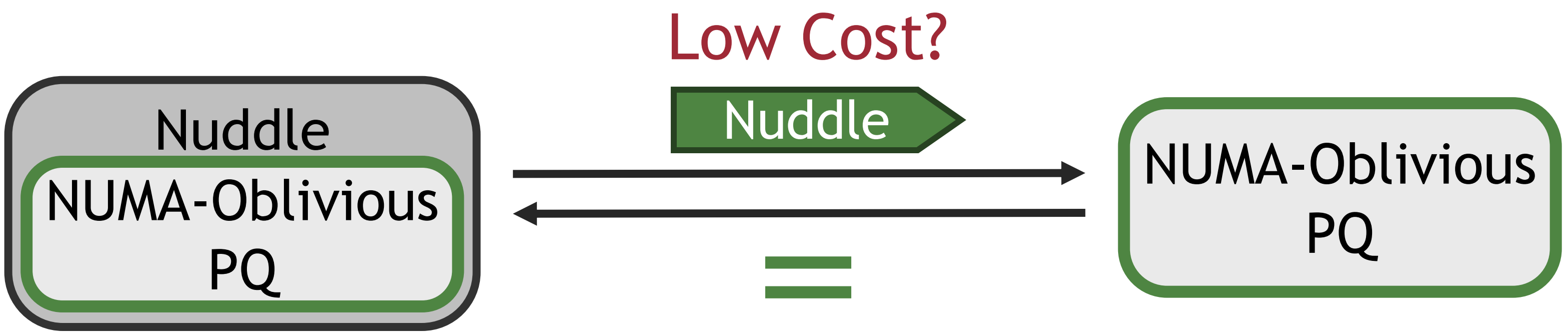


Key Challenges:

1. How to switch between the two modes with low synchronization overheads?

2. SmartPQ

An adaptive PQ that switches
between two algorithmic modes whenever it is needed

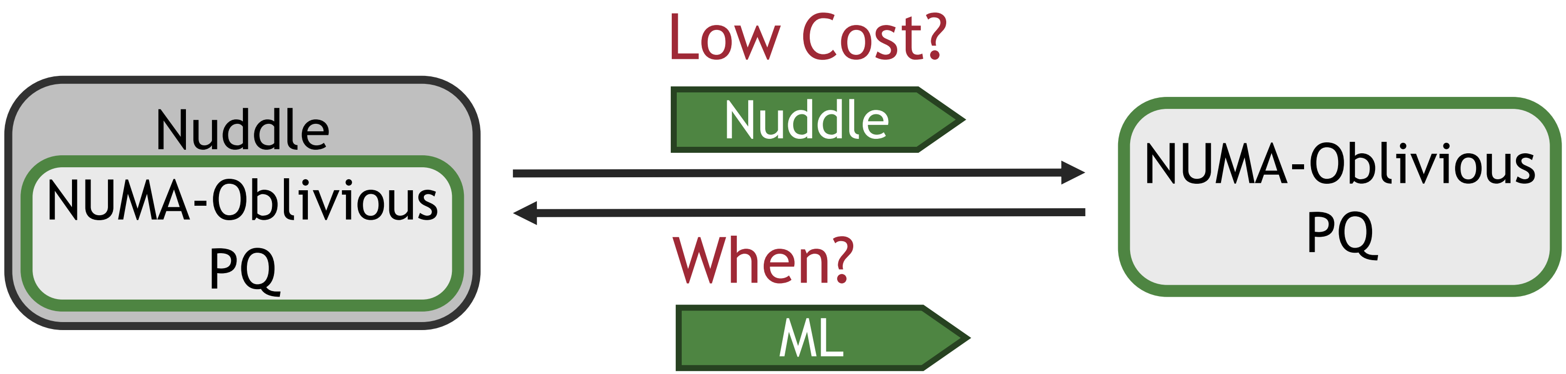


Key Challenges:

1. How to switch between the two modes with low synchronization overheads?

2. SmartPQ

An adaptive PQ that switches
between two algorithmic modes whenever it is needed



Key Challenges:

1. How to switch between the two modes with low synchronization overheads?
2. When to switch from the one to the other mode?

Decission Tree Classifier

NUMA-Aware, NUMA-Oblivious, Neutral

2. SmartPQ

An adaptive PQ that switches
between two algorithmic modes whenever it is needed
Low Cost?

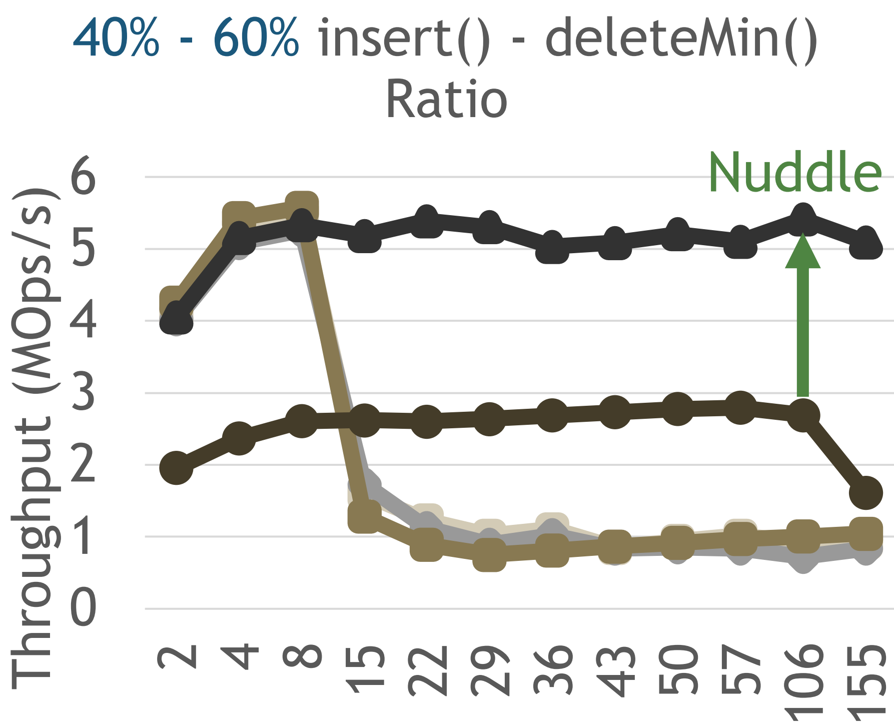
2-4 ms traversal time with 180 nodes
and a very low tree depth of 8

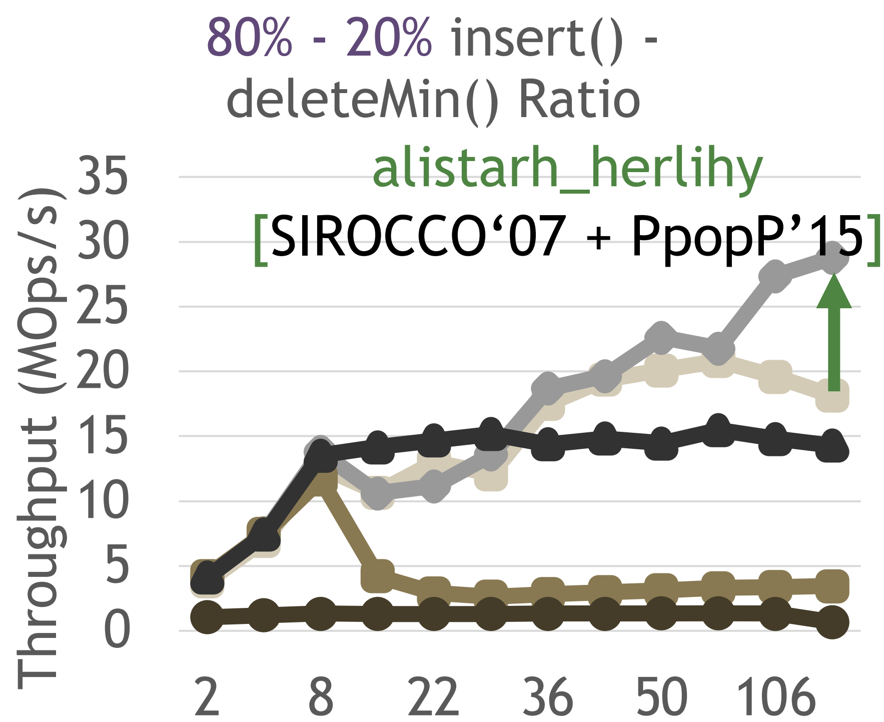
87.9% accuracy in a test set
of 10K different contention workloads

Decission Tree Classifier

→ NUMA-Aware, NUMA-Oblivious, Neutral

Throughput Evaluation

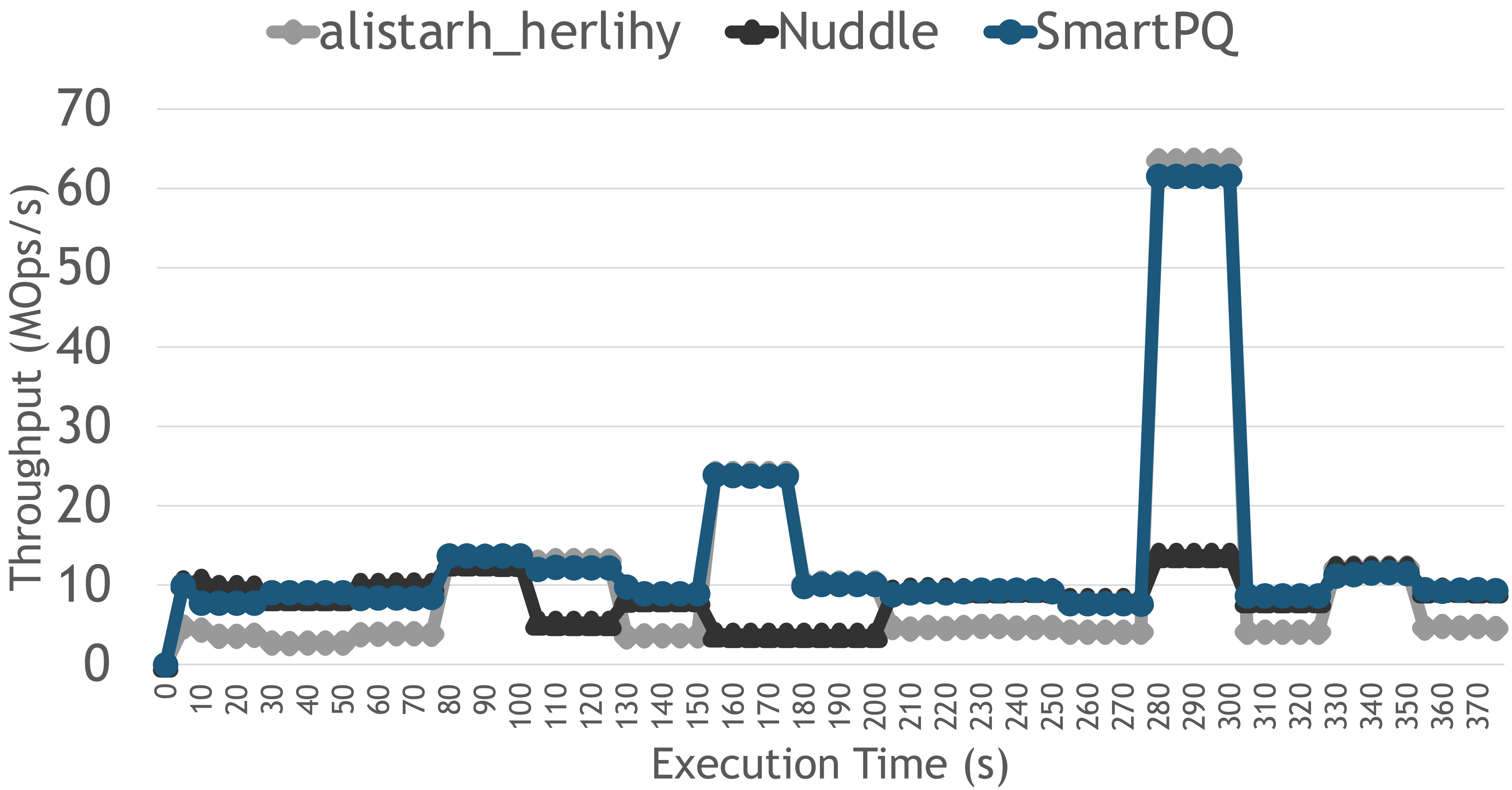




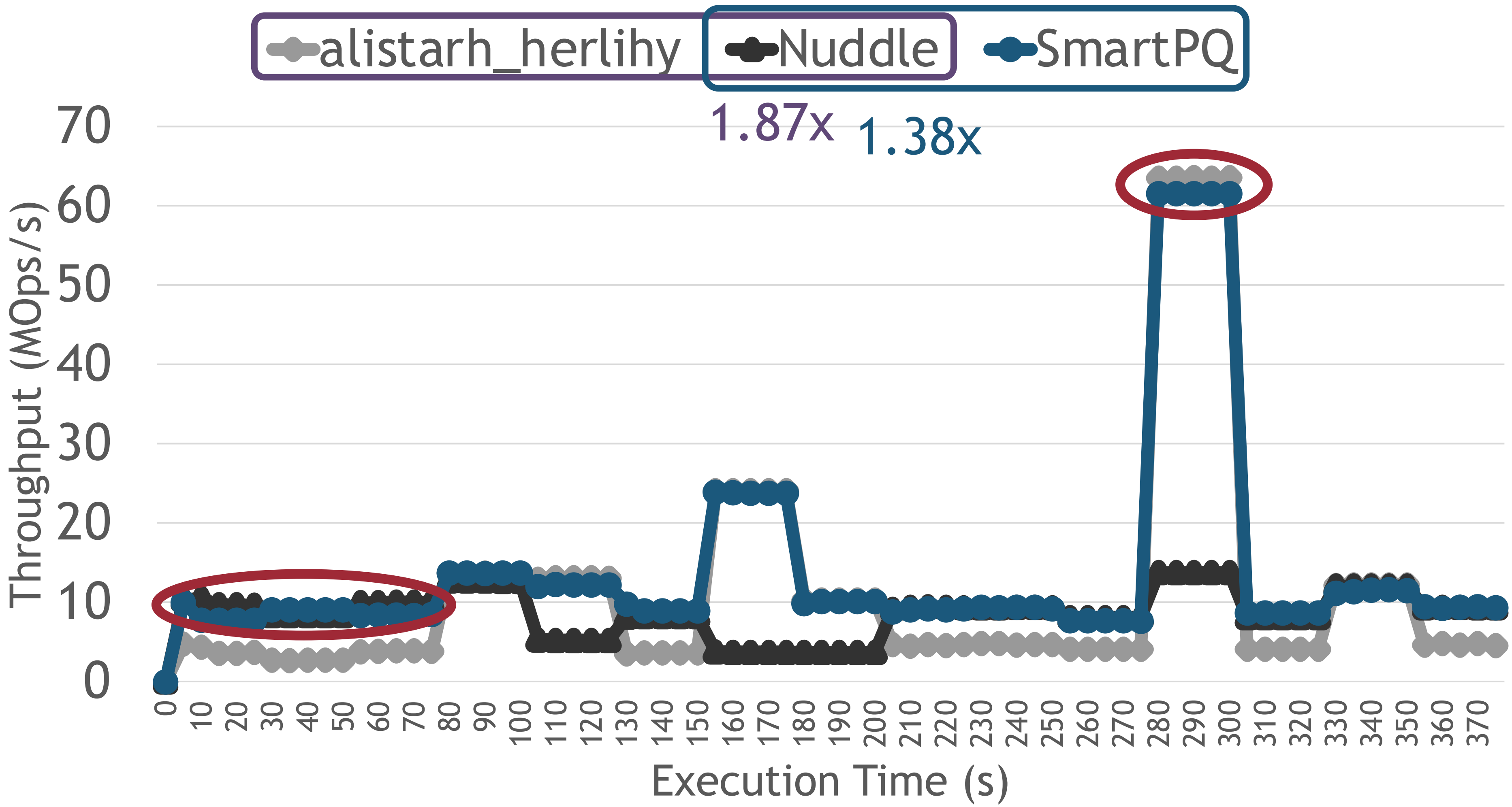
alistarh_fraser alistarh_herlihy lotan_shavit ffwd Nuddle

NUMA-Oblivious NUMA-Aware

Throughput with Varying Contention



Throughput with Varying Contention

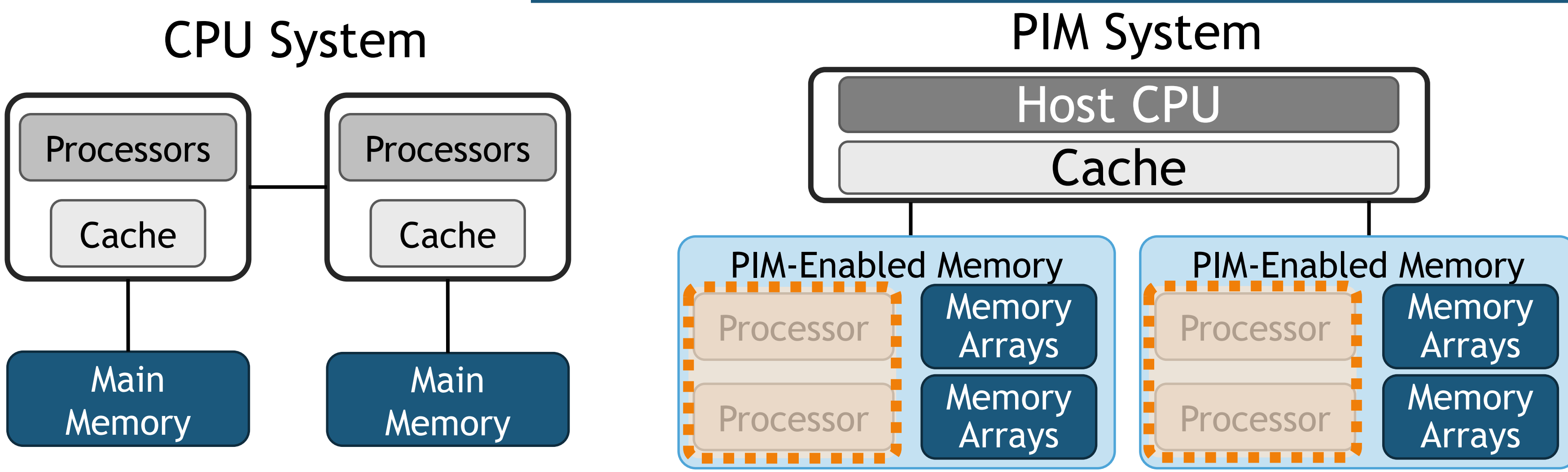


Core Contributions

3 Irregular Workloads

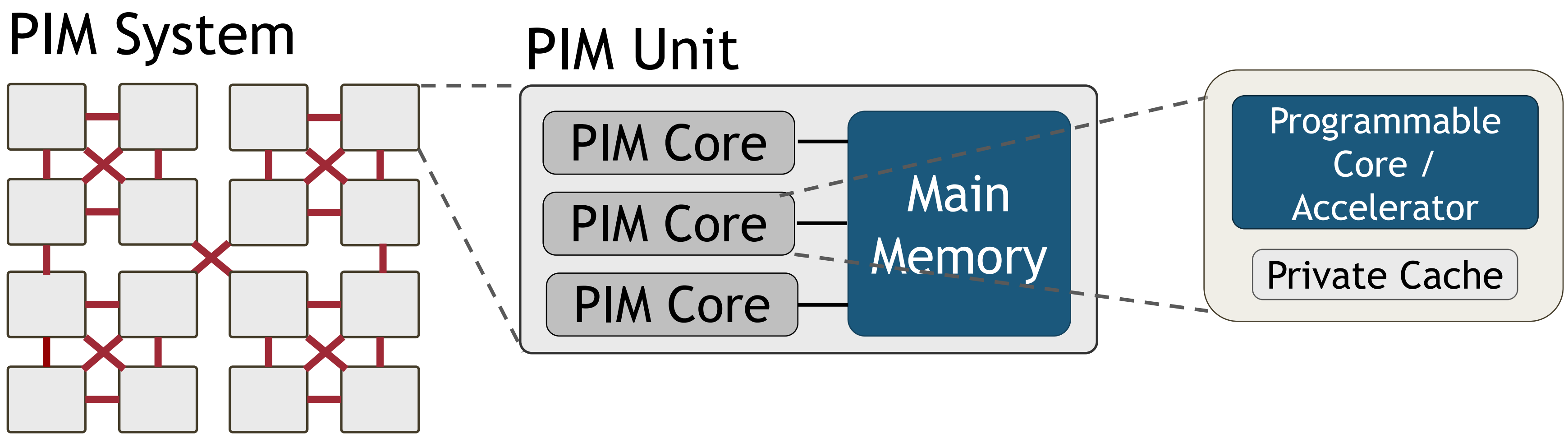
SynCron (HPCA'21)

A Lightweight Synchronization Mechanism for PIM Systems



Enabling very low synchronization costs

Processing-In-Memory (PIM) Architecture



Synchronization challenges in PIM systems:

(1) Lack of hardware cache coherence support

(2) Lack of a shared level of cache memory

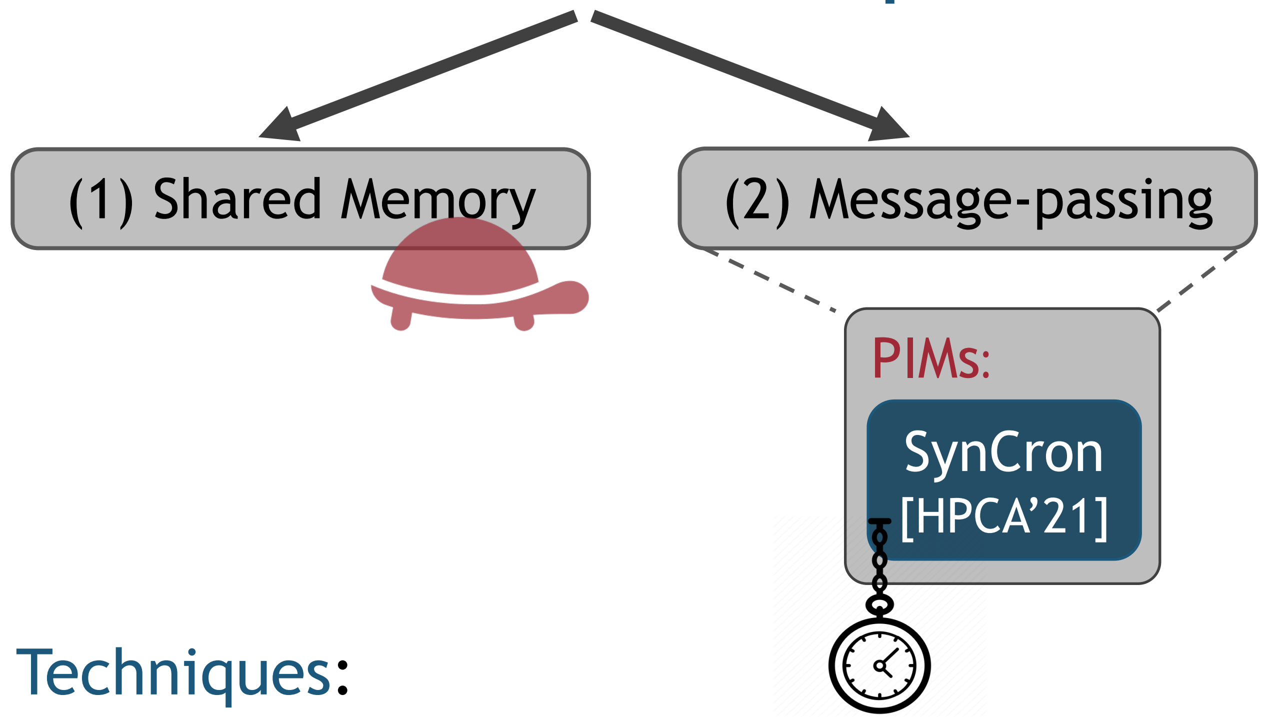
(3) Expensive communication across PIM units

PIM Synchronization Solution Space

(1) Shared Memory

(2) Message-passing

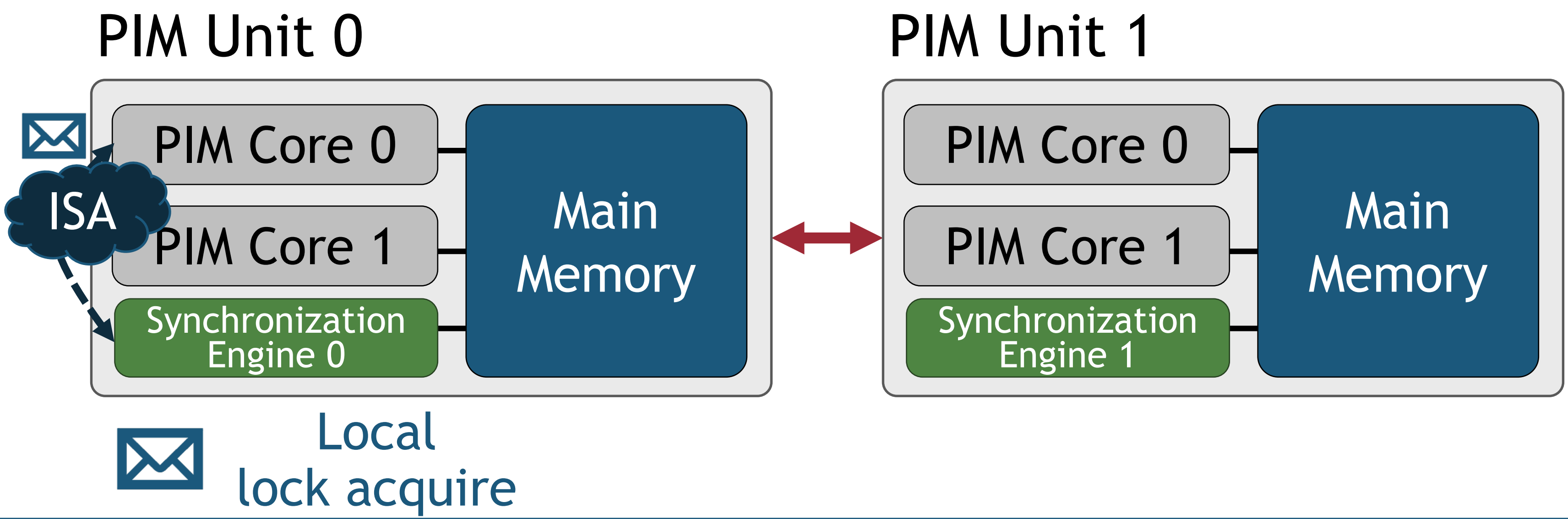
PIM Synchronization Solution Space



SynCron’s Key Techniques:

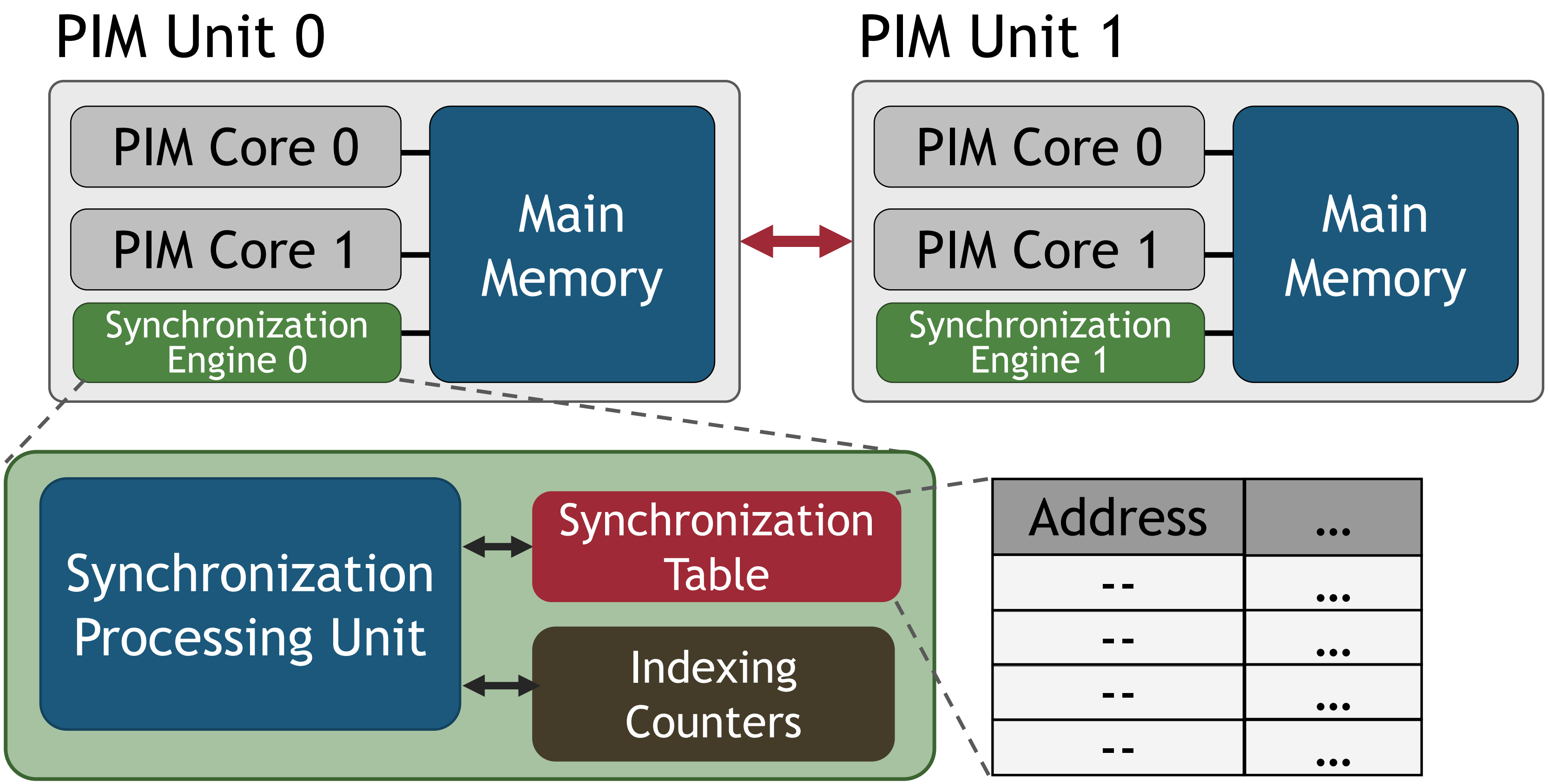
1. Hardware support for synchronization acceleration
2. Direct buffering of synchronization variables
3. Hierarchical message-passing communication
4. Integrated hardware-only overflow management

1. Hardware Synchronization Support



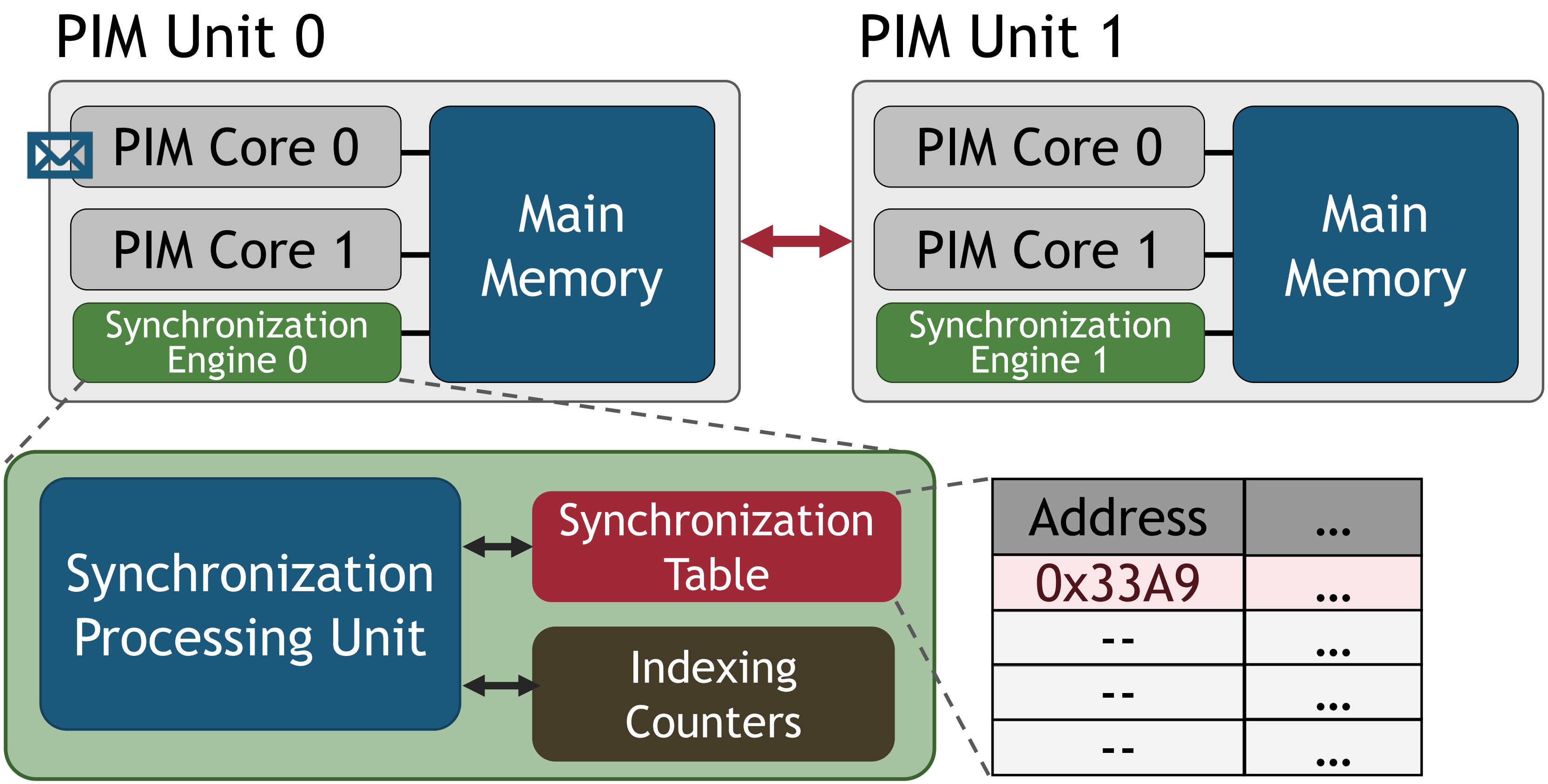
- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

2. Direct Buffering of Variables



Address	...
--	...
--	...
--	...
--	...

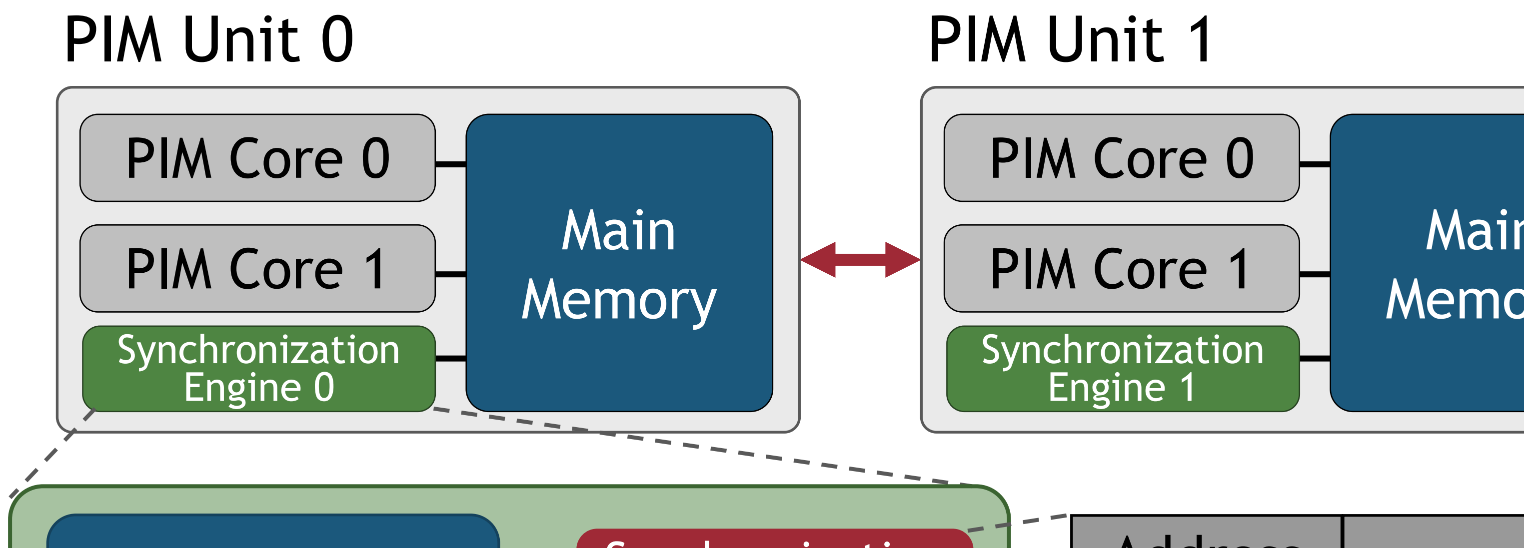
2. Direct Buffering of Variables



Address	...
0x33A9	...
--	...
--	...
--	...

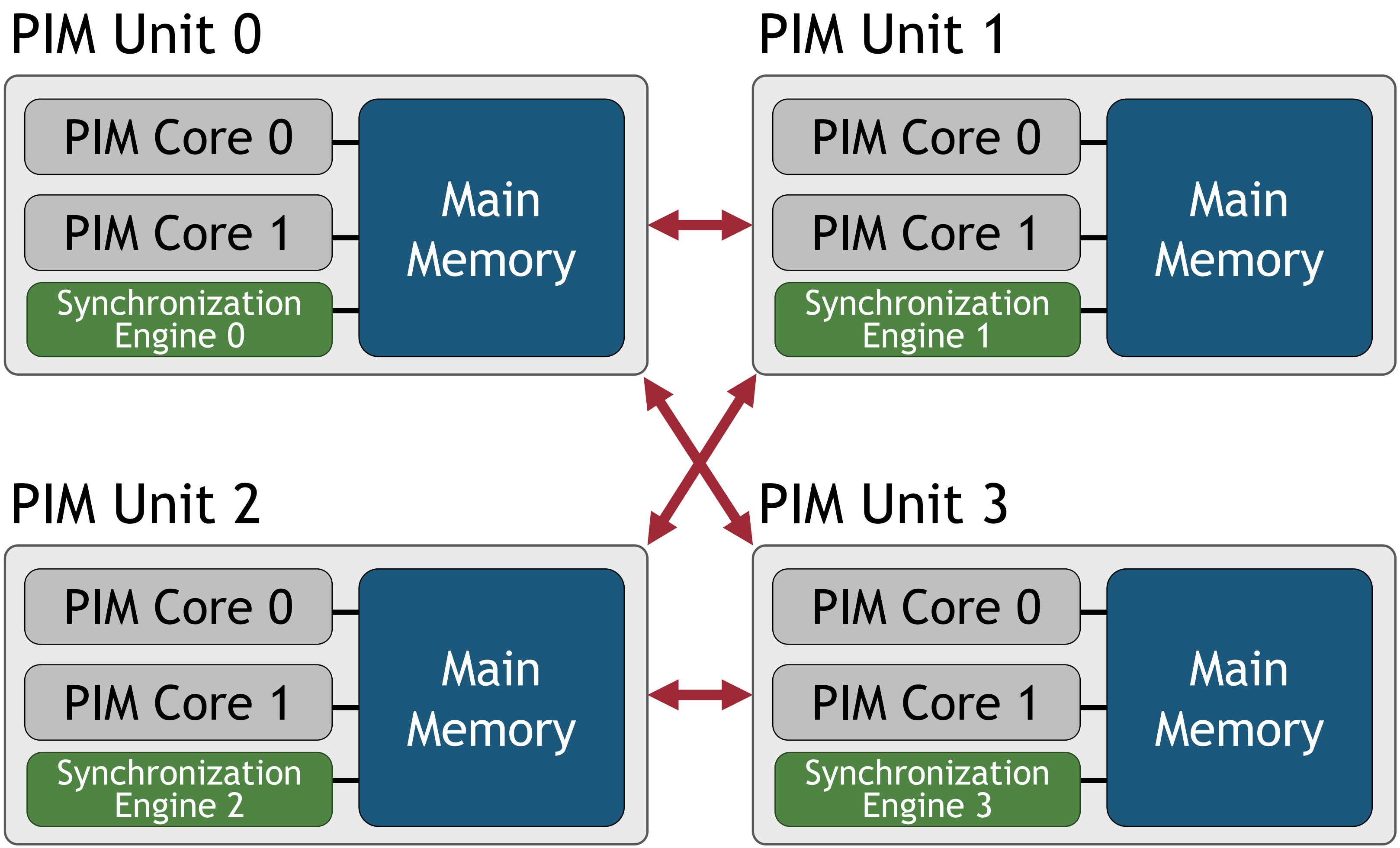
Local
lock acquire

2. Direct Buffering of Variables

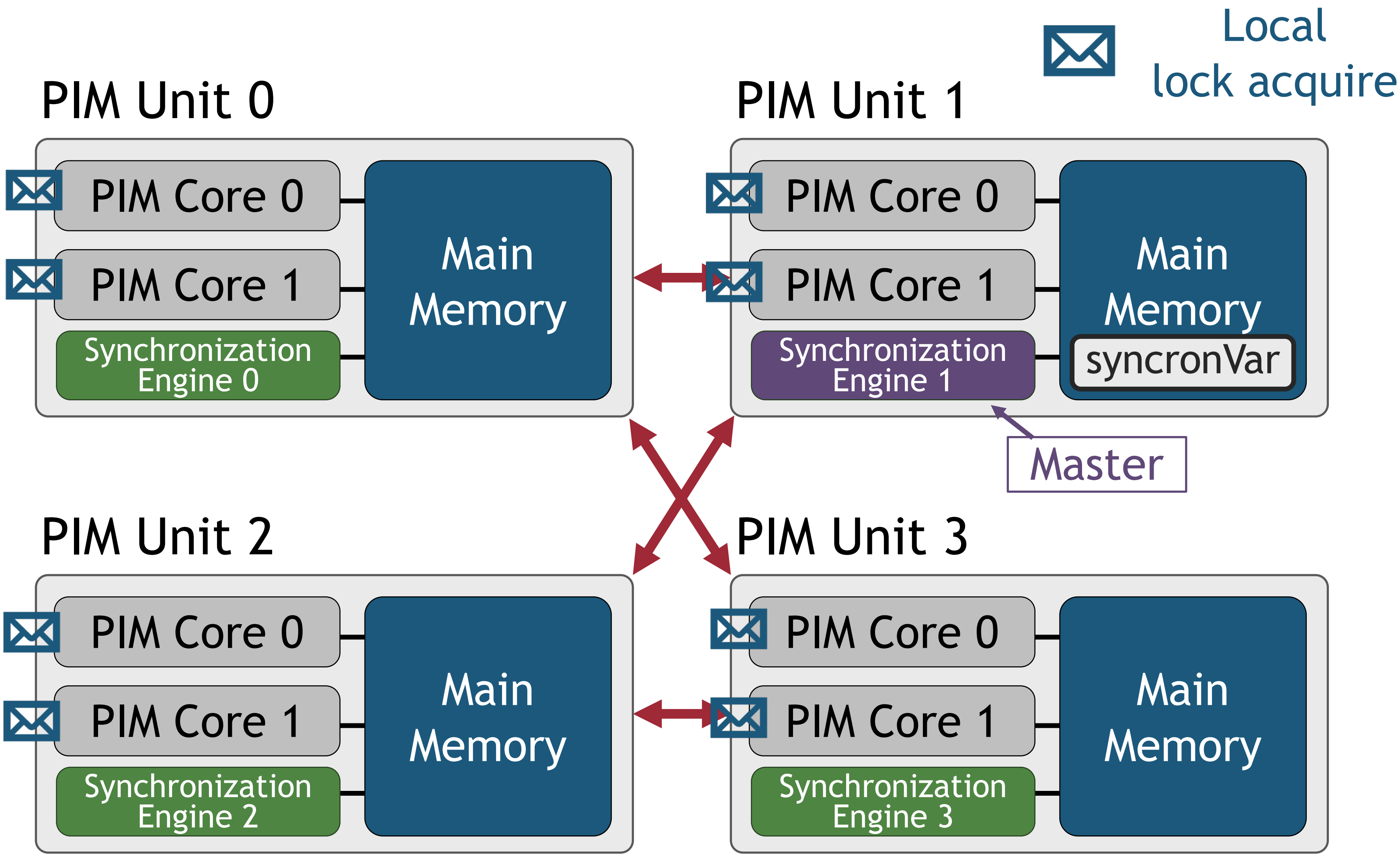


✓ No Costly Memory Accesses
✓ Low Latency

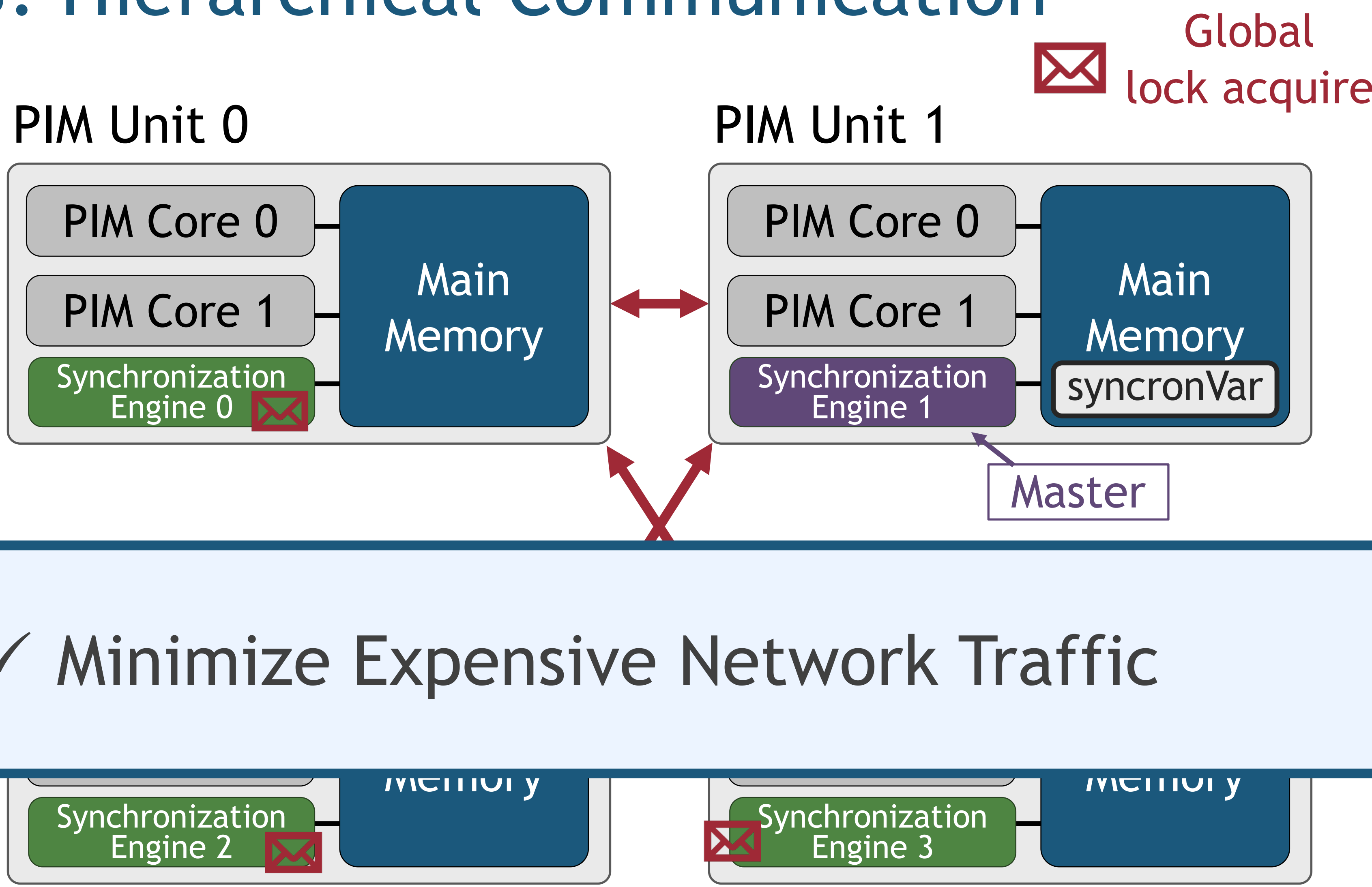
3. Hierarchical Communication



3. Hierarchical Communication

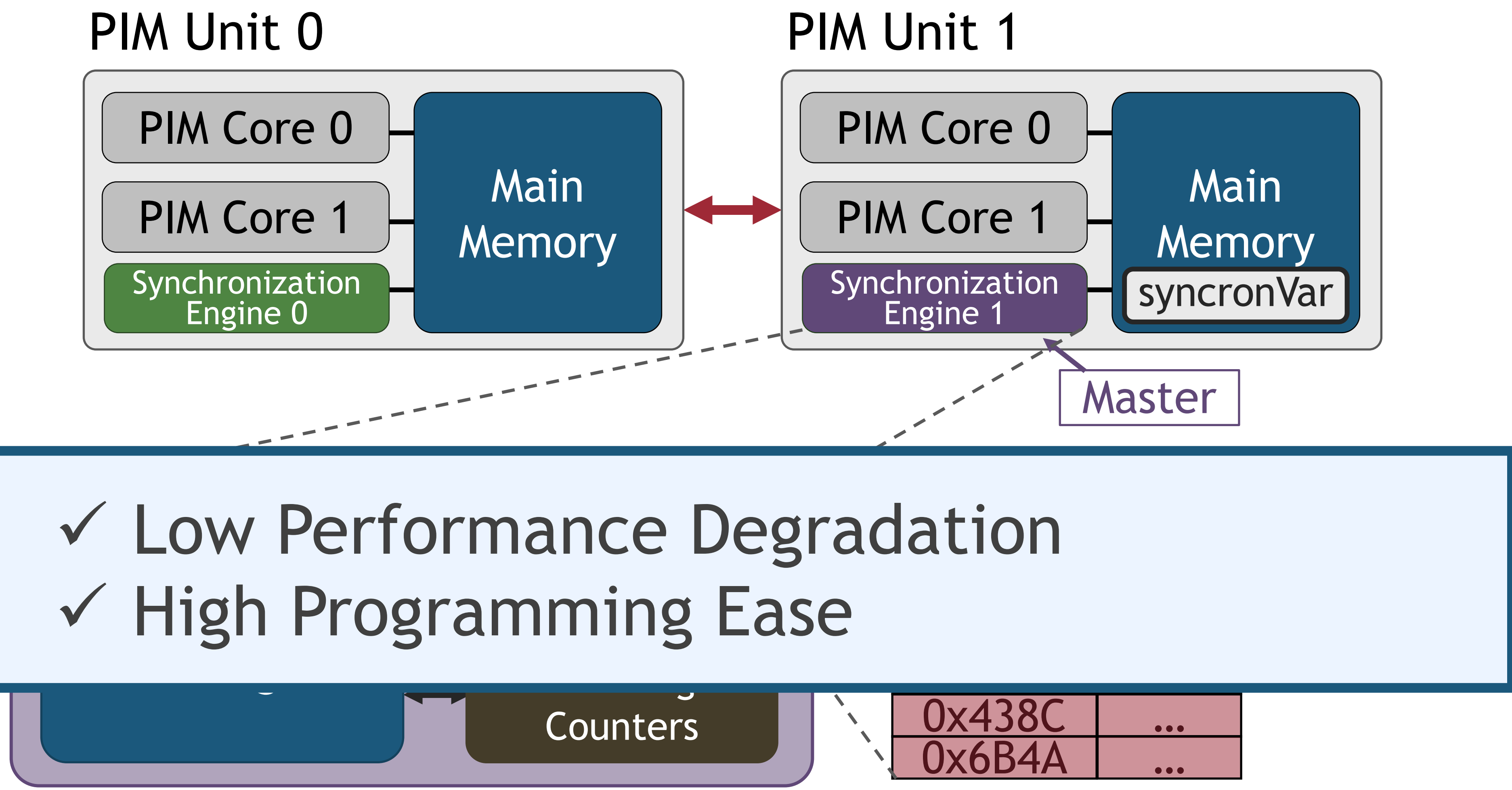


3. Hierarchical Communication



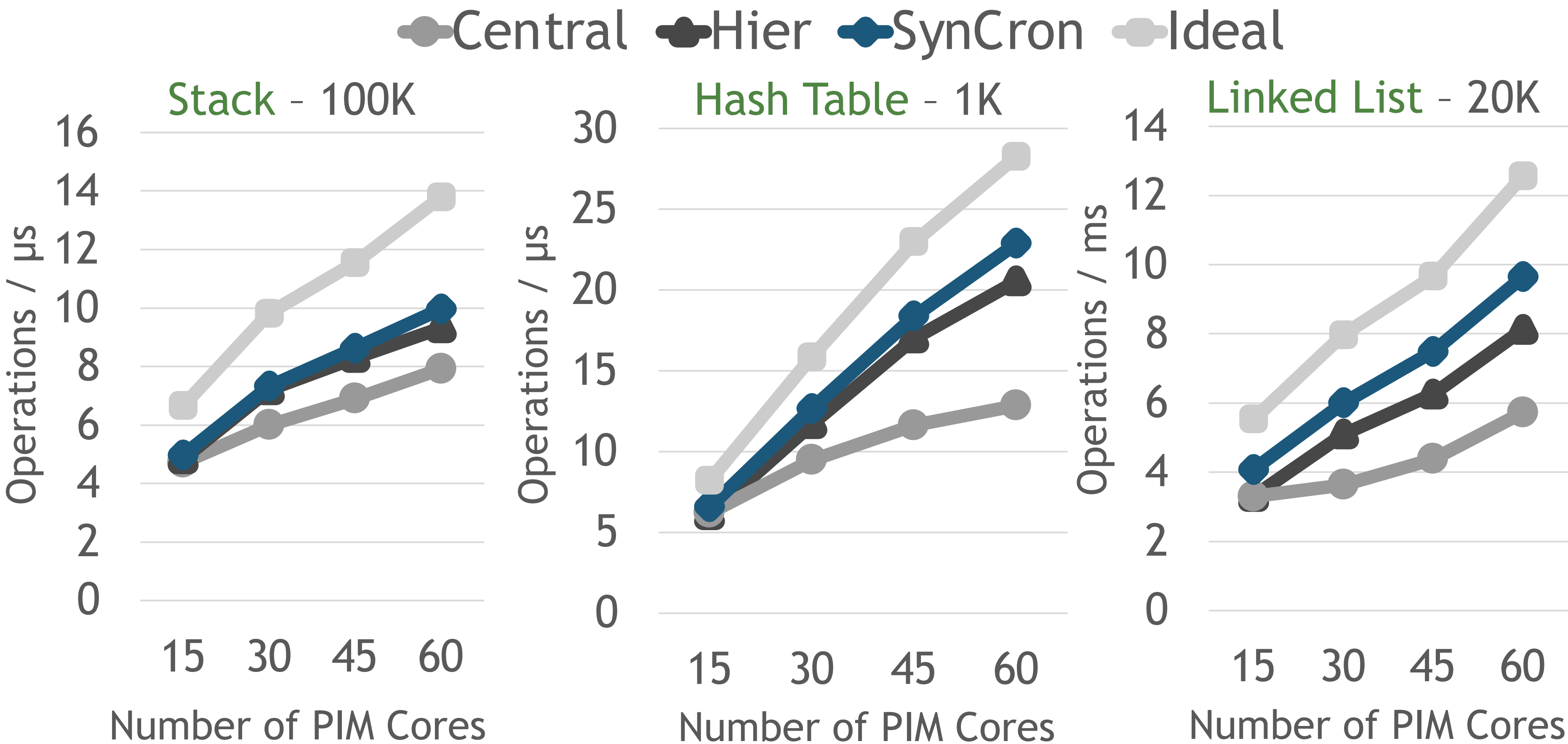
✓ Minimize Expensive Network Traffic

4. Integrated Overflow Management



✓ Low Performance Degradation
✓ High Programming Ease

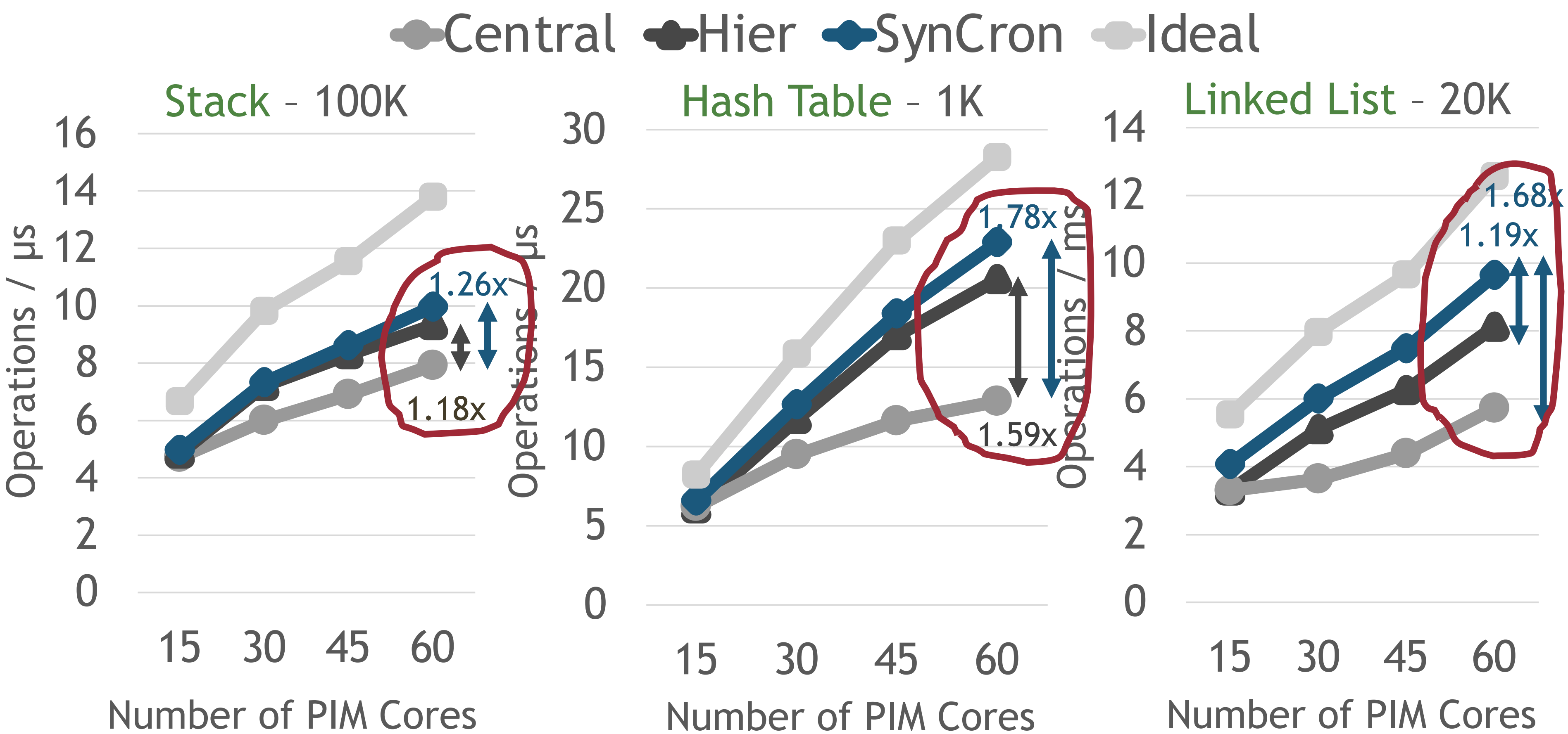
Throughput of Pointer Chasing

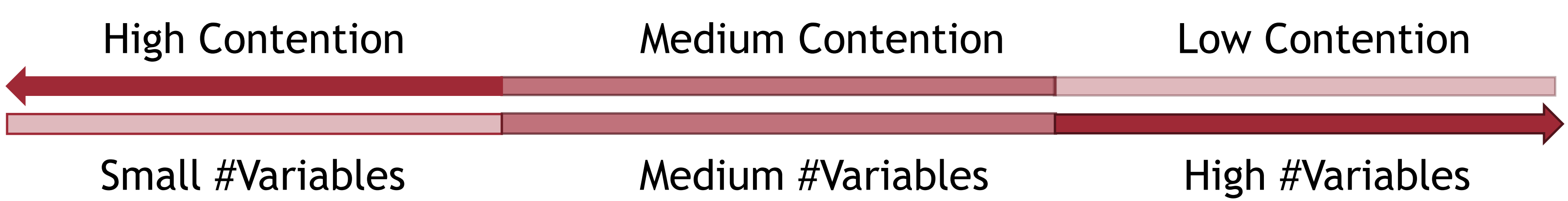


High Contention | Medium Contention | Low Contention

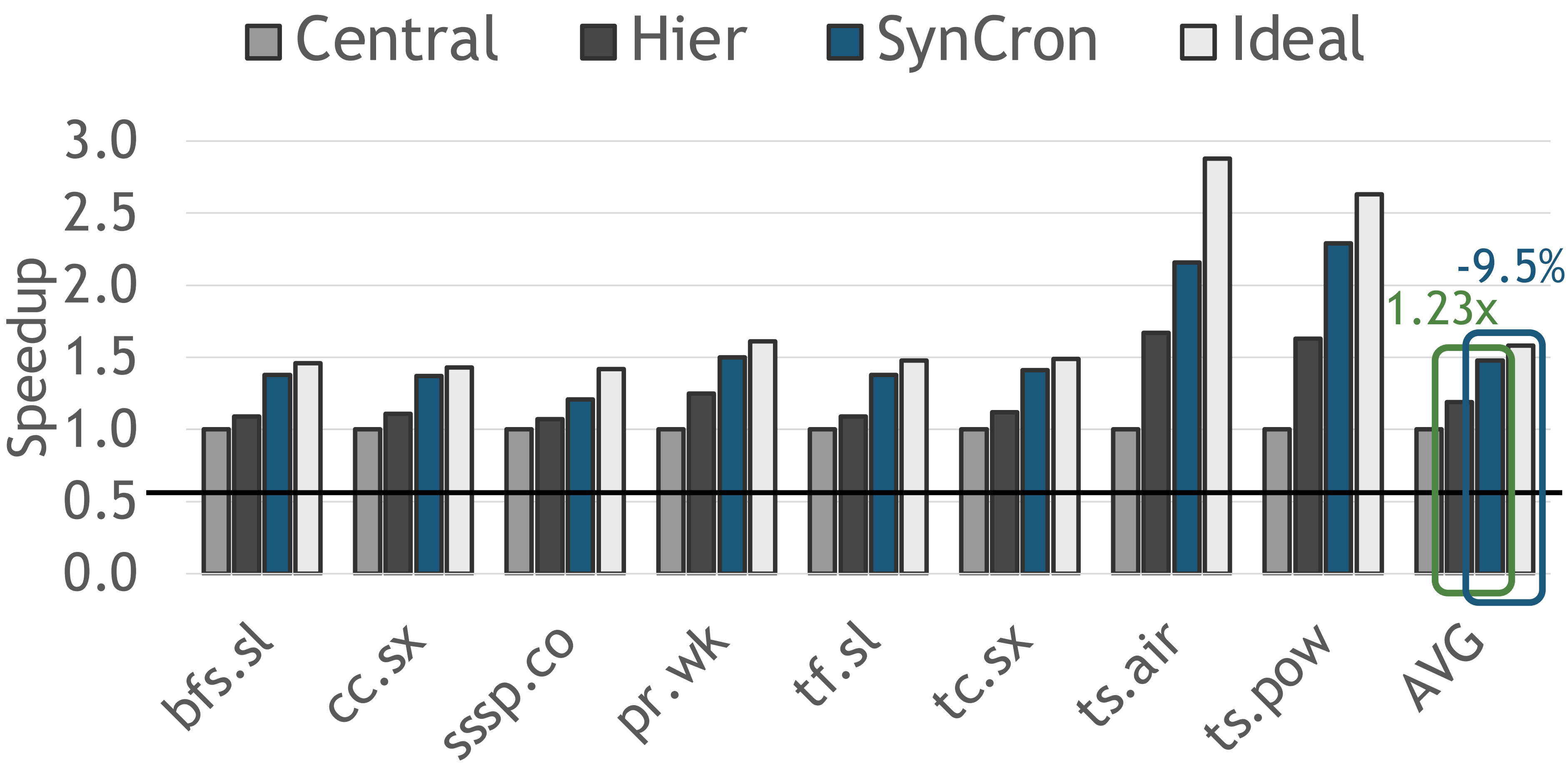
Small #Variables | Medium #Variables | High #Variables

Throughput of Pointer Chasing

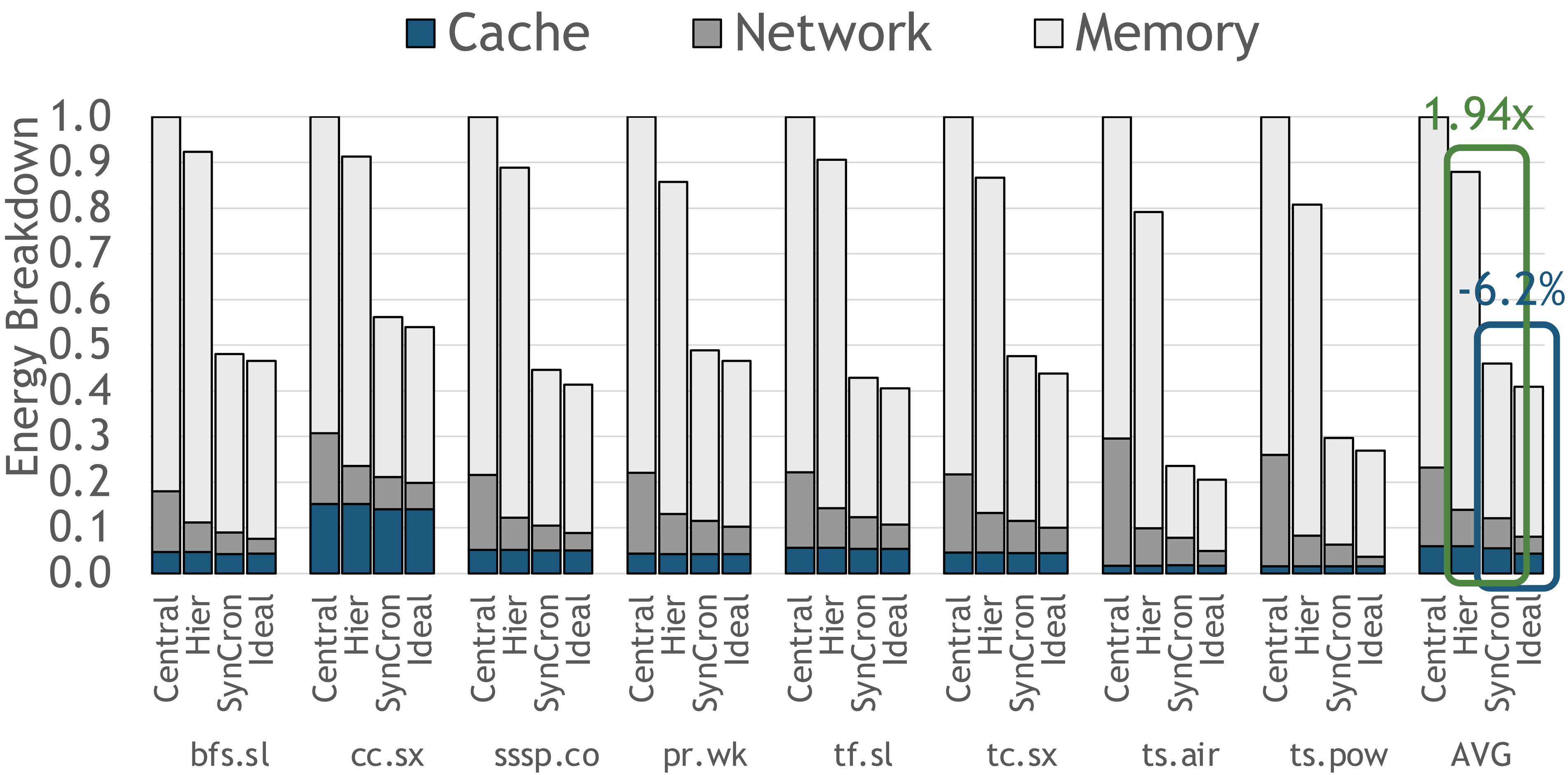




Performance in Data Analytics

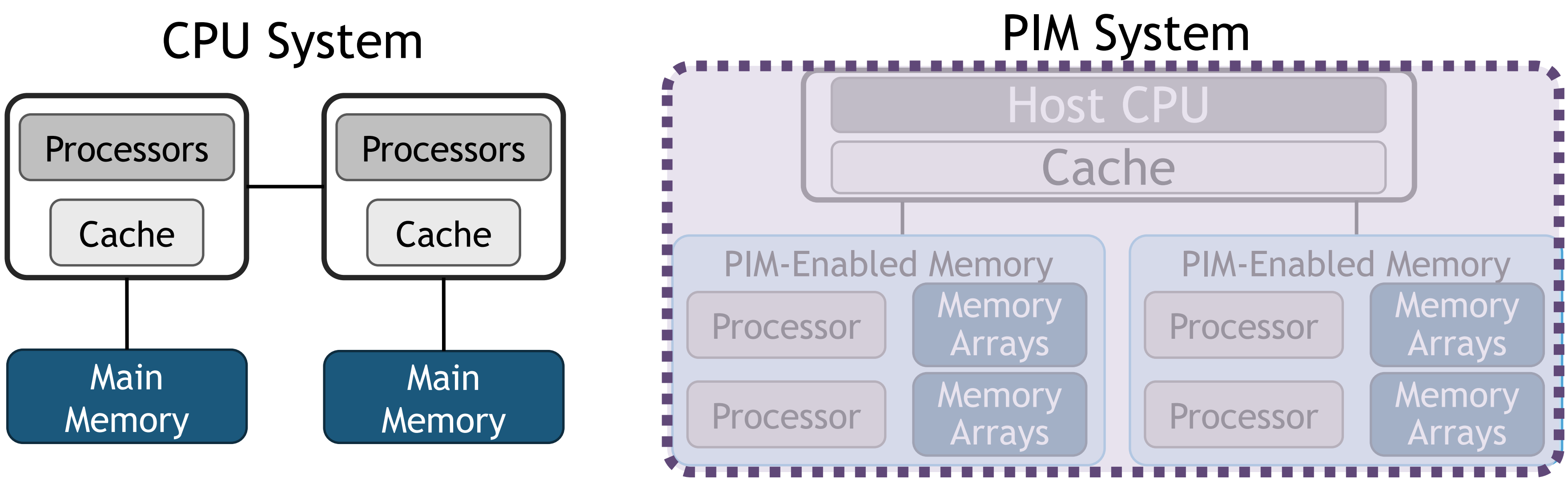


System Energy in Data Analytics



Core Contributions

Across multiple PIM cores → low data transfer costs
Across multiple threads → lightweight synchronization



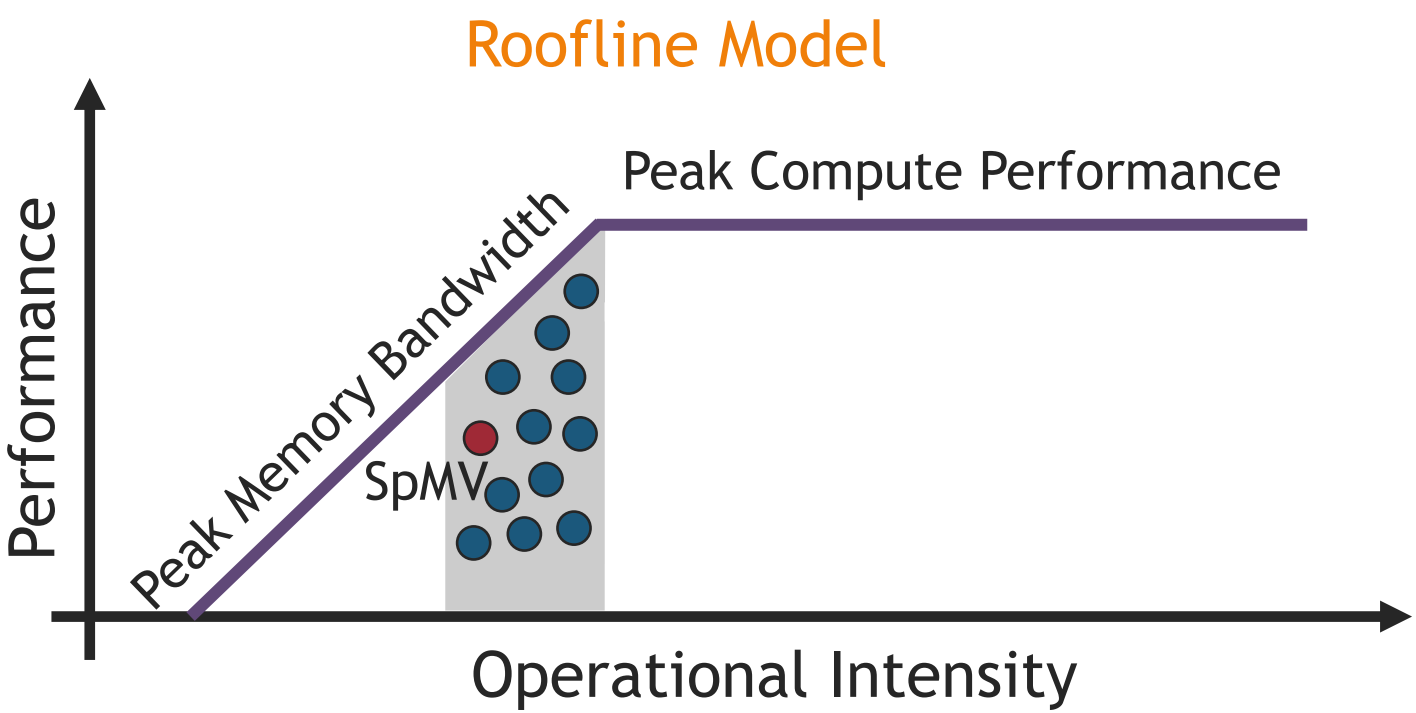
4 Sparse Linear Algebra

SparseP (Sigmetrics'22)

A Library of Efficient Sparse Matrix Vector Multiplication Kernels for Real PIM Systems

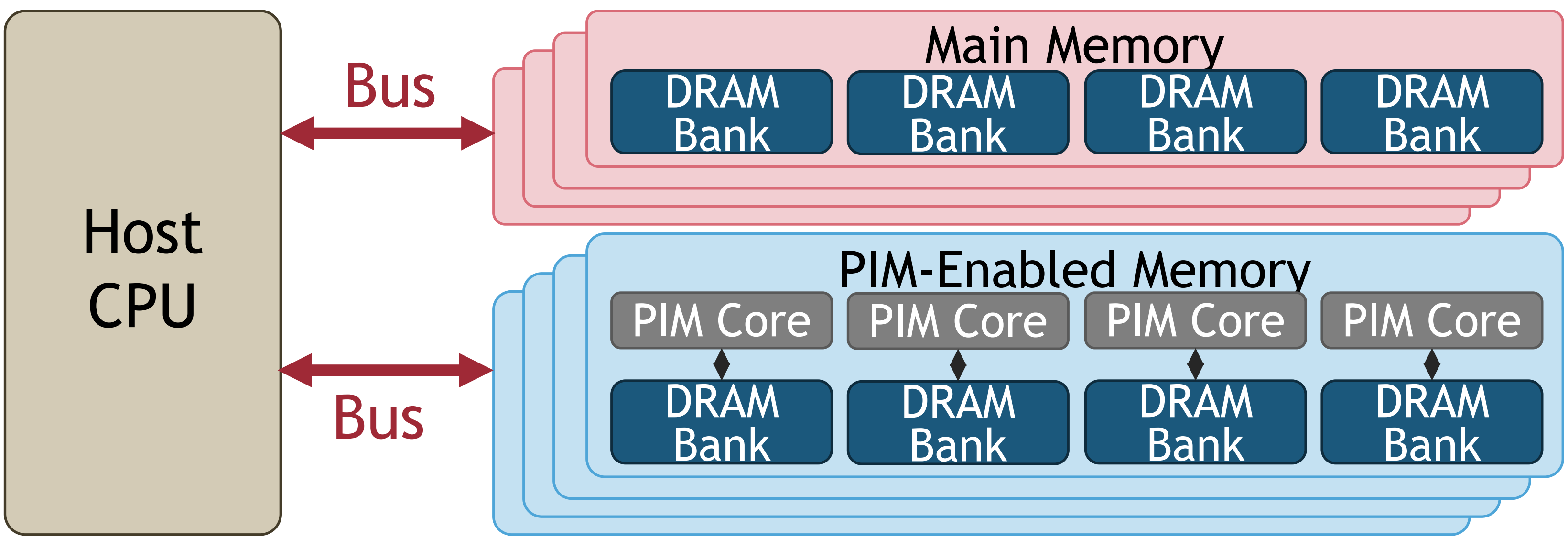
Motivation

- Sparse Matrix Vector Multiplication (SpMV):
  - Widely used in machine learning, graph analytics, scientific computing…
  - A highly bandwidth-bound kernel



Motivation

- Sparse Matrix Vector Multiplication (SpMV):
  - Widely used in machine learning, graph analytics, scientific computing…
  - A highly bandwidth-bound kernel
- Real Near-Bank Processing-In-Memory (PIM) Systems:
  - High levels of parallelism
  - Large aggregate memory bandwidth



Key Contributions

1. Design efficient SpMV kernels for current and future PIM systems
   - SparseP = 25 SpMV kernels

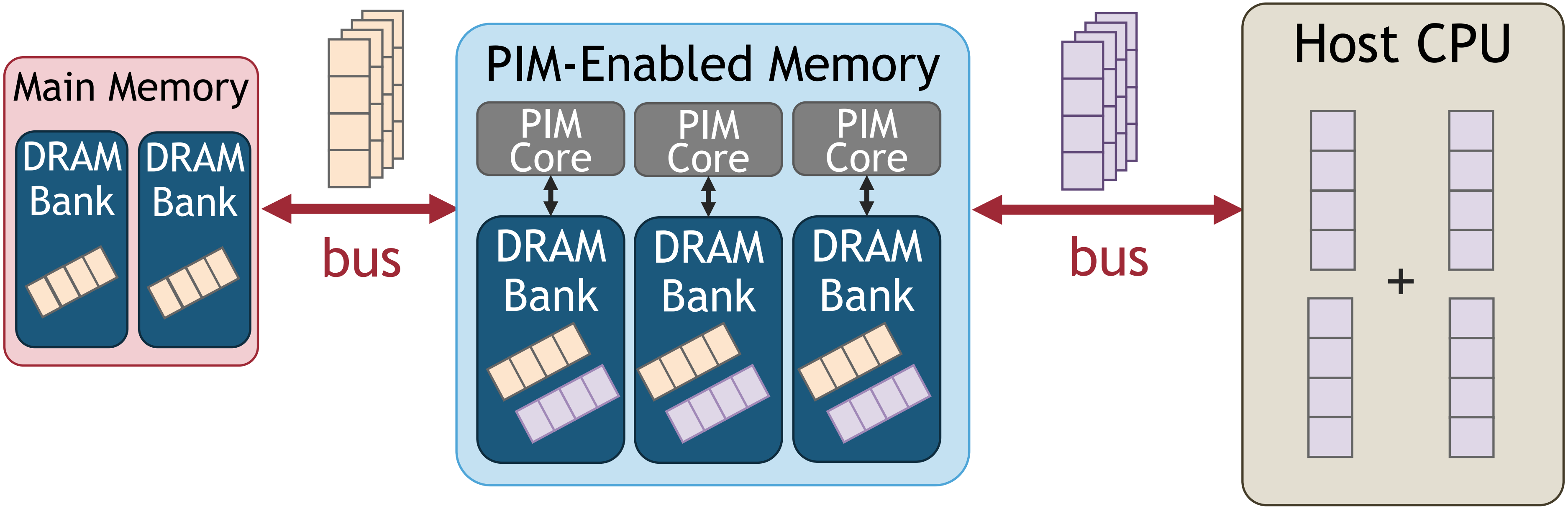
SparseP is Open-Source
SparseP: https://github.com/CMU-SAFARI/SparseP

2. Provide a comprehensive analysis of SpMV on the first commercially-available real PIM system
   - 26 sparse matrices
   - Comparisons to state-of-the-art CPU and GPU systems
   - Recommendations for software, system and hardware designers

Recommendations for Architects and Programmers
Full Paper: https://arxiv.org/pdf/2201.05072.pdf

SpMV Execution on a PIM System

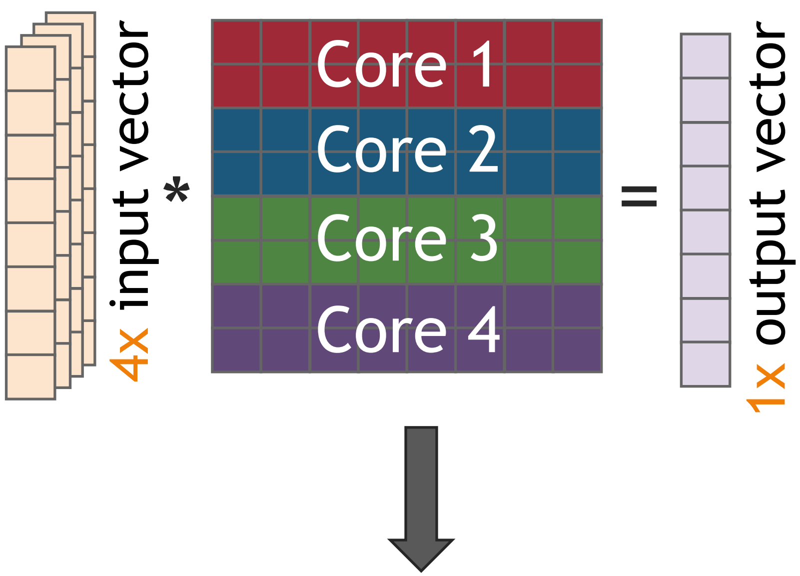
1. Load the input vector
2. Execute the kernel
3. Retrieve the partial results
4. Merge the partial results



Compute-Aware vs Data-Aware SpMV

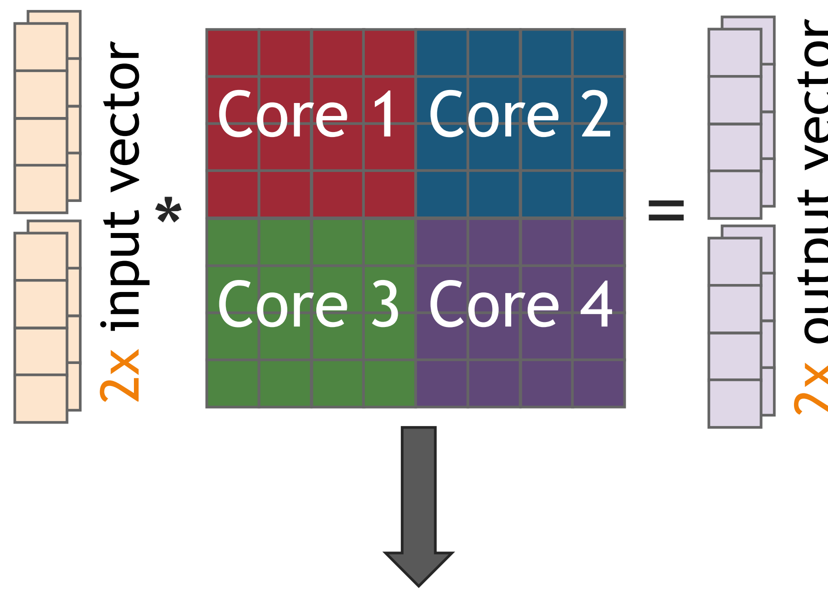
SparseP supports two types of data partitioning techniques:

1D Partitioning (compute-aware)



perform the complete SpMV computation only on PIM cores

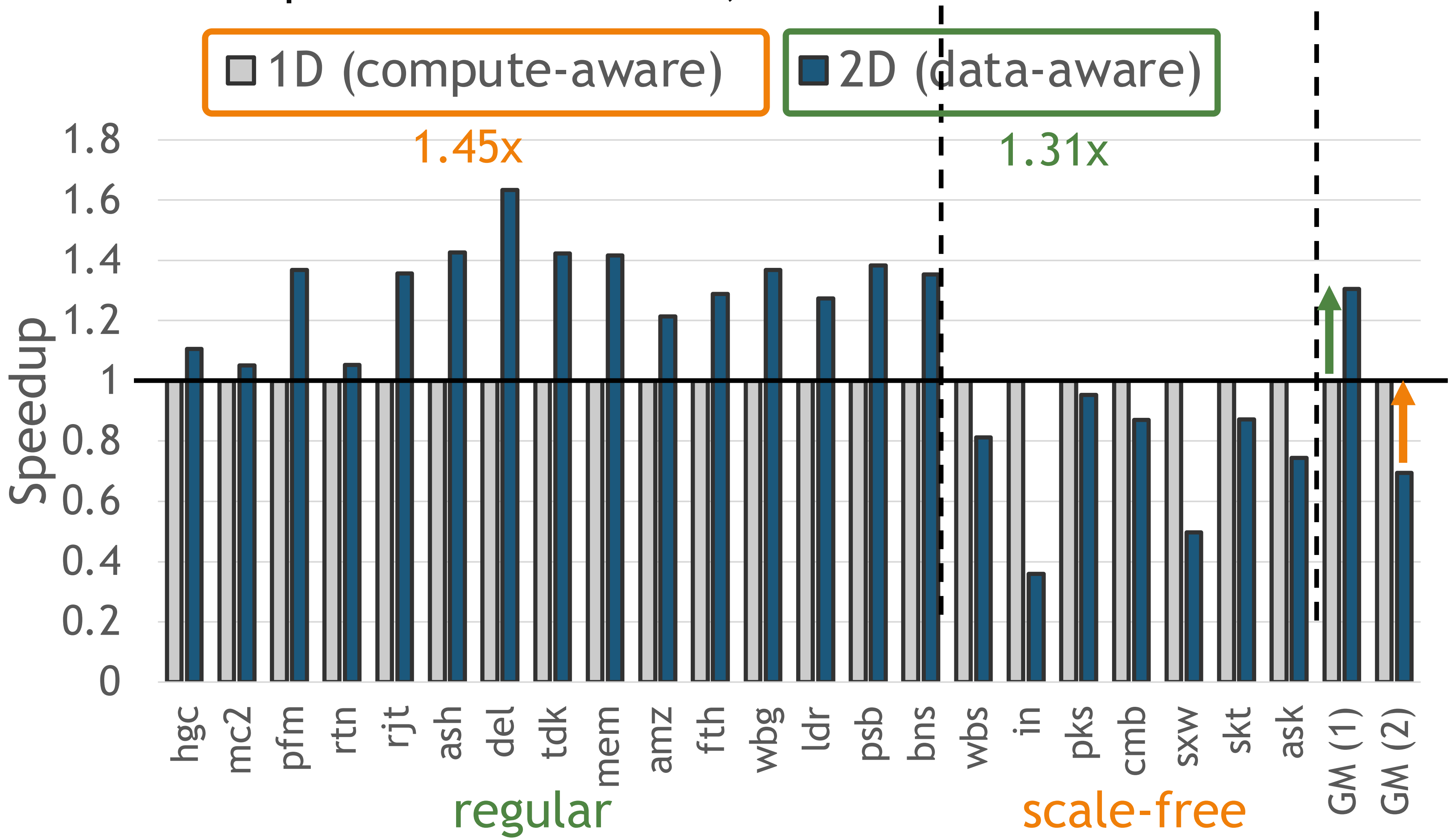
2D Partitioning (data-aware)



trade-off computation vs data transfer costs

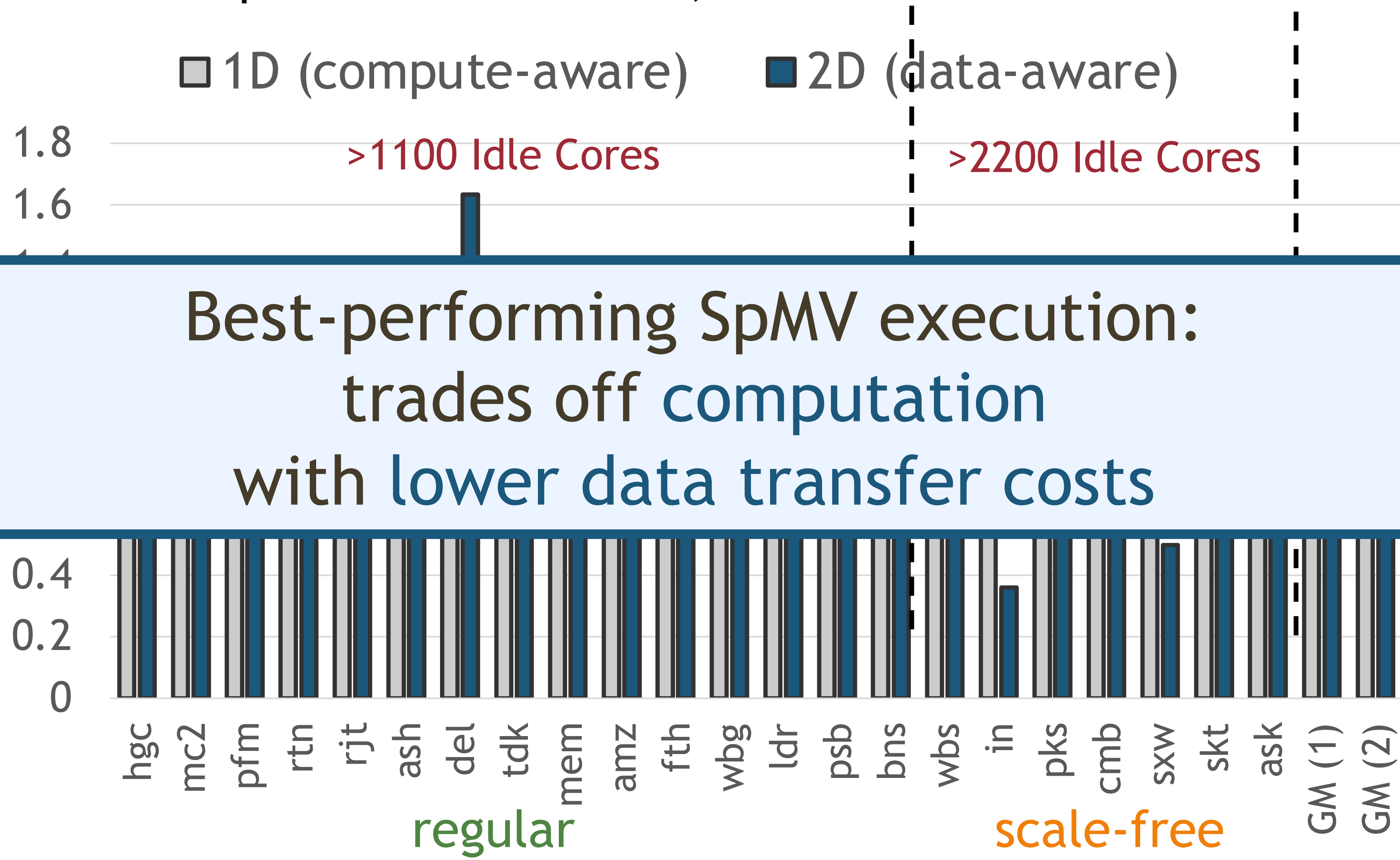
Compute-Aware vs Data-Aware SpMV

From 64 up to 2528 PIM Cores, 32-bit float



Compute-Aware vs Data-Aware SpMV

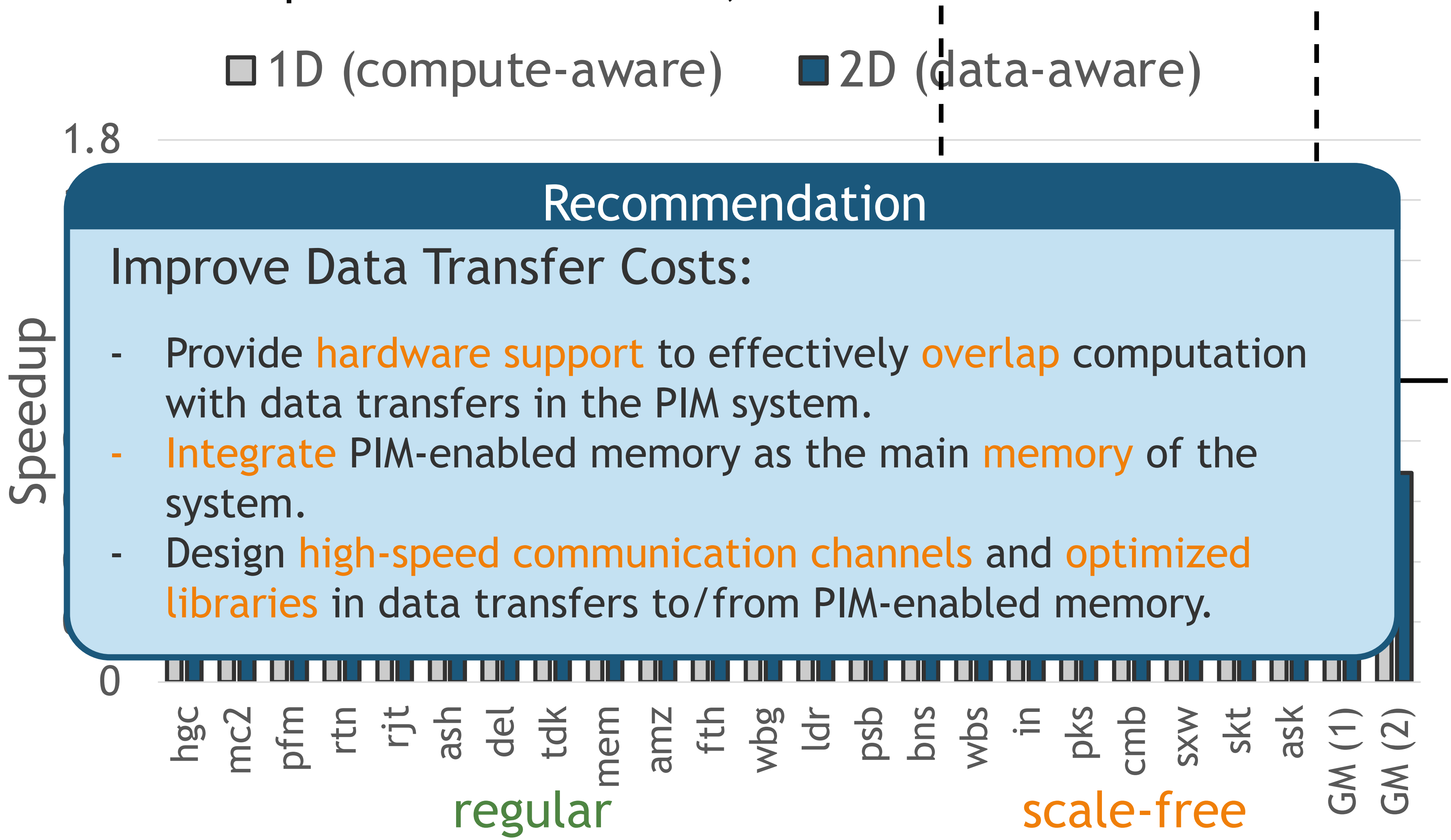
From 64 up to 2528 PIM Cores, 32-bit float



Best-performing SpMV execution:
trades off computation
with lower data transfer costs

Compute-Aware vs Data-Aware SpMV

From 64 up to 2528 PIM Cores, 32-bit float

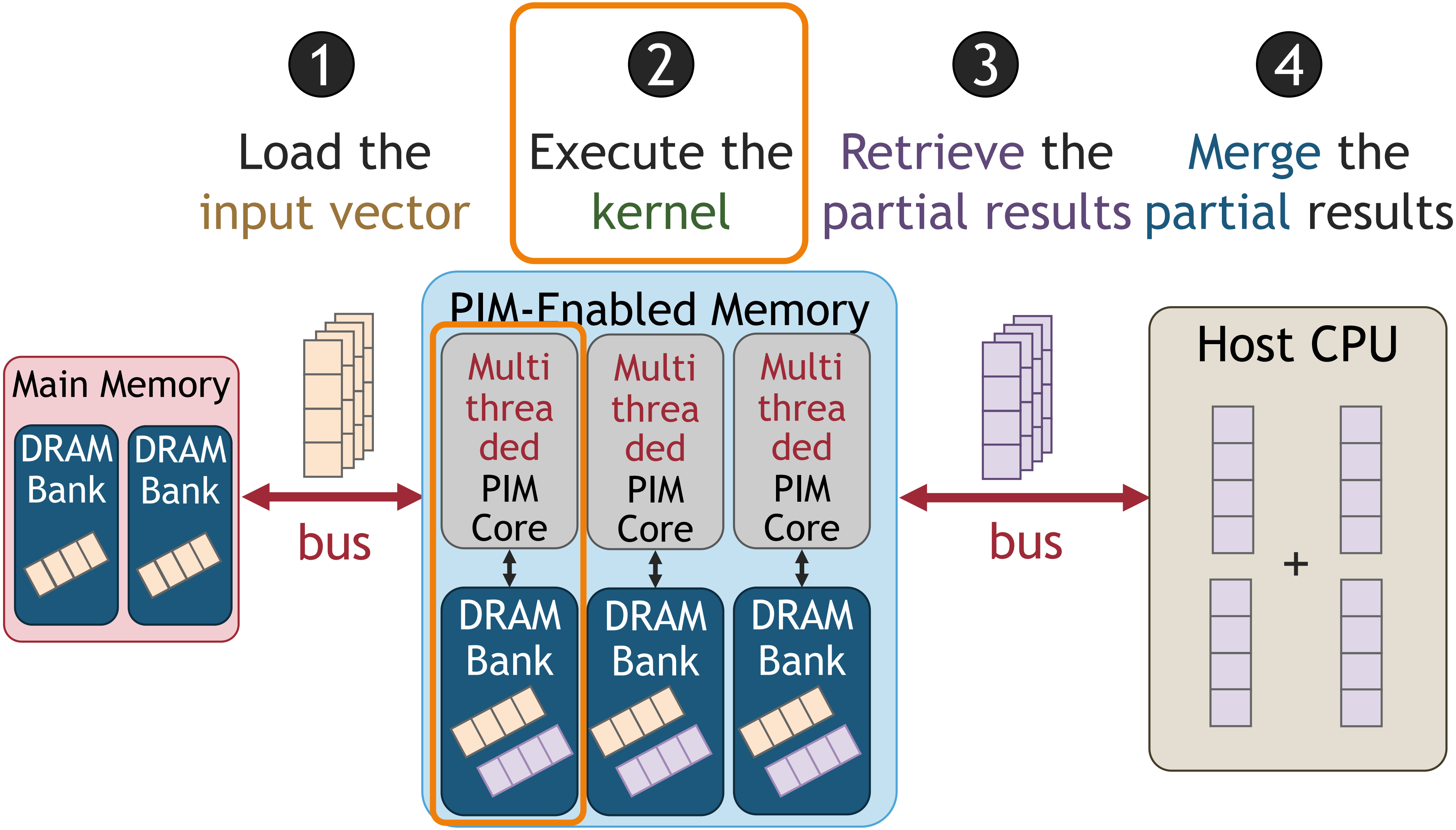


Recommendation

Improve Data Transfer Costs:

- Provide hardware support to effectively overlap computation with data transfers in the PIM system.
- Integrate PIM-enabled memory as the main memory of the system.
- Design high-speed communication channels and optimized libraries in data transfers to/from PIM-enabled memory.

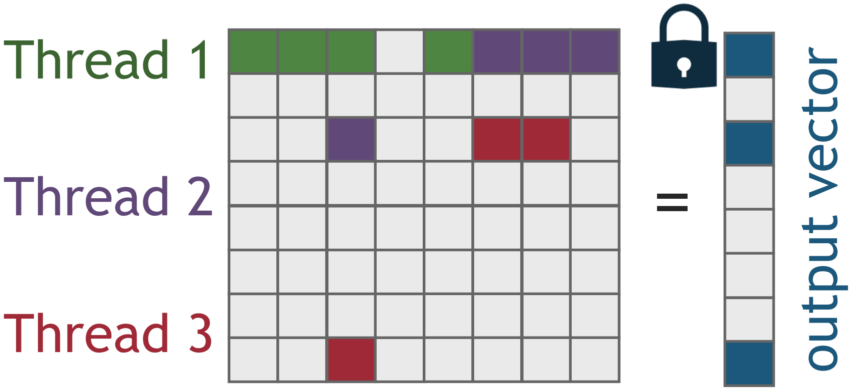
SpMV Execution on a PIM System



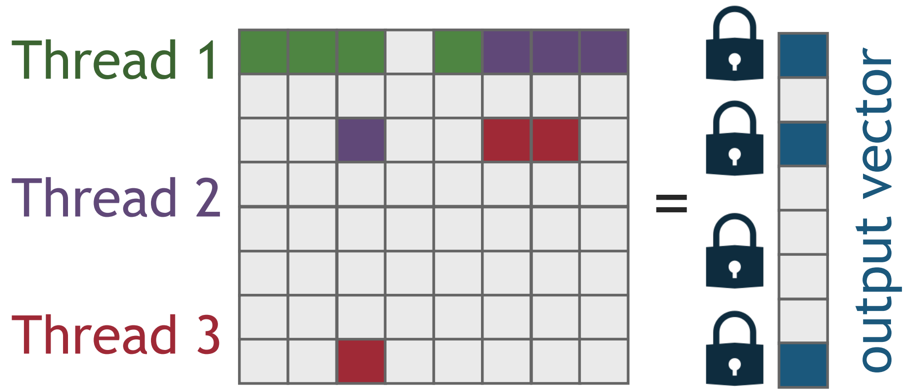
Synchronization Approaches

Multithreaded PIM Core:

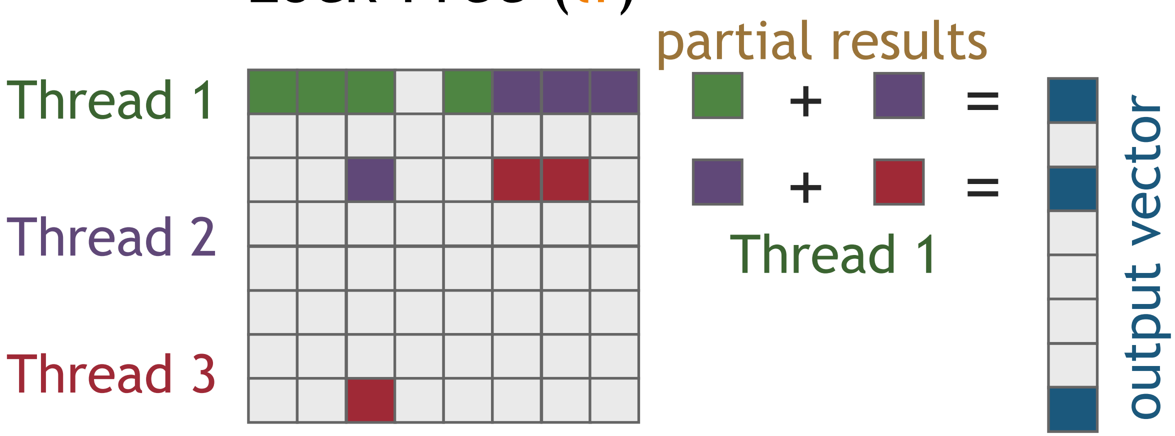
Coarse-Grained (lb-cg)



Fine-Grained (lb-fg)

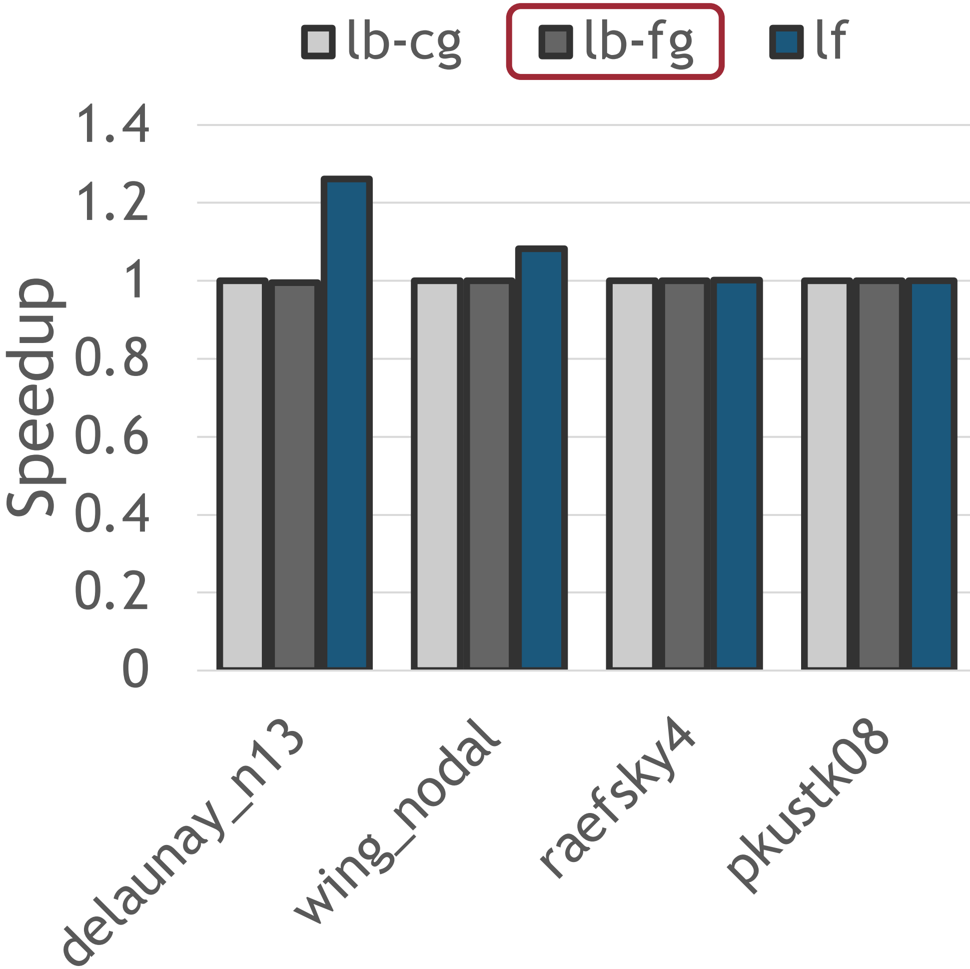


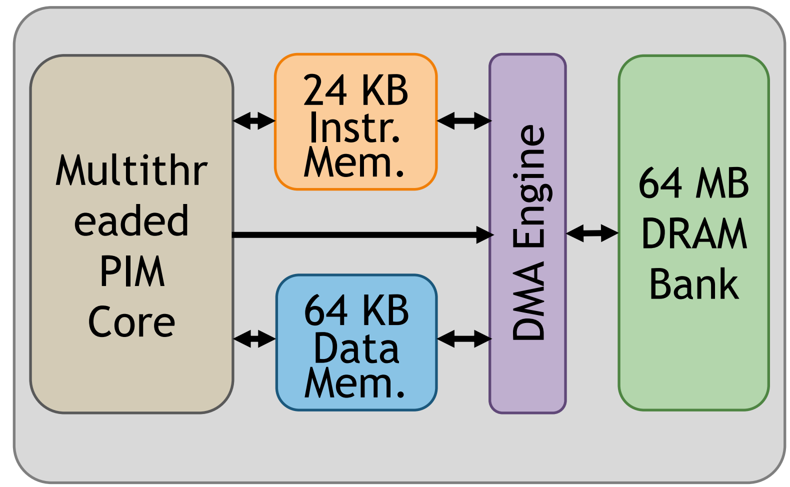
Lock-Free (lf)



Performance of Synchronization Schemes

16 threads, 32-bit integer

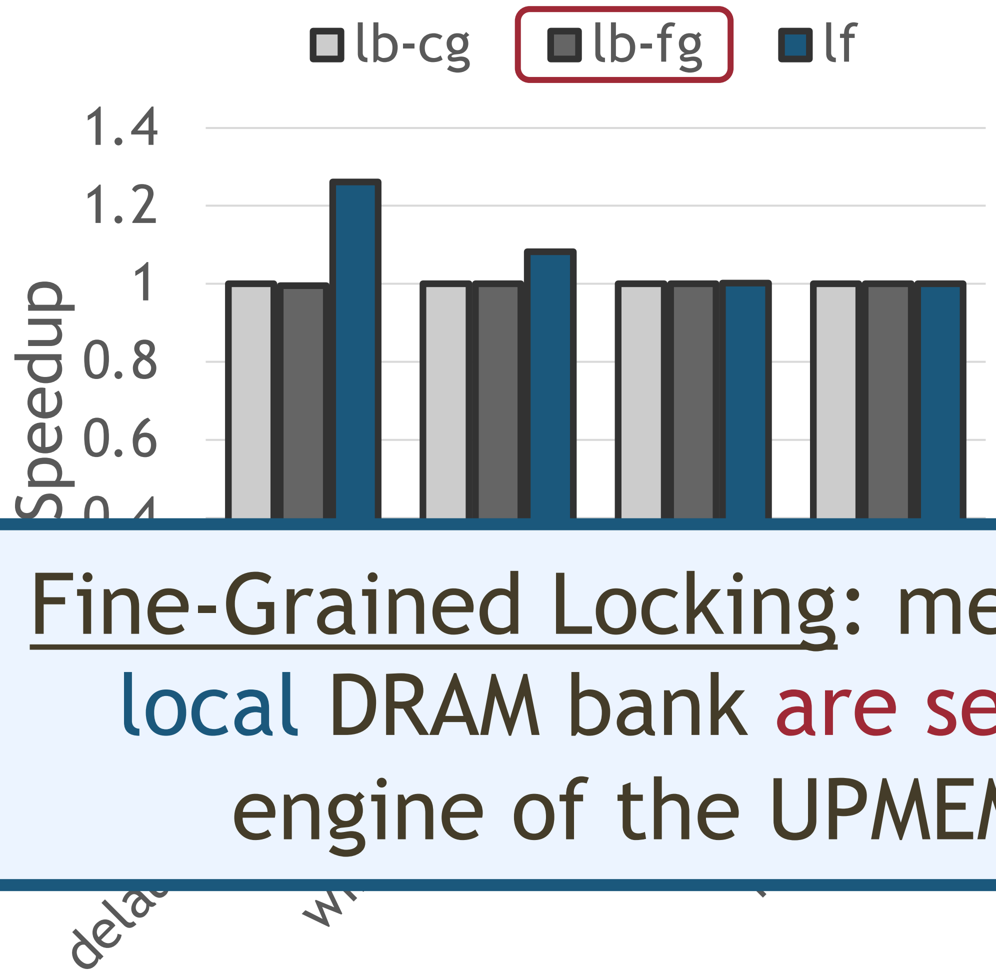


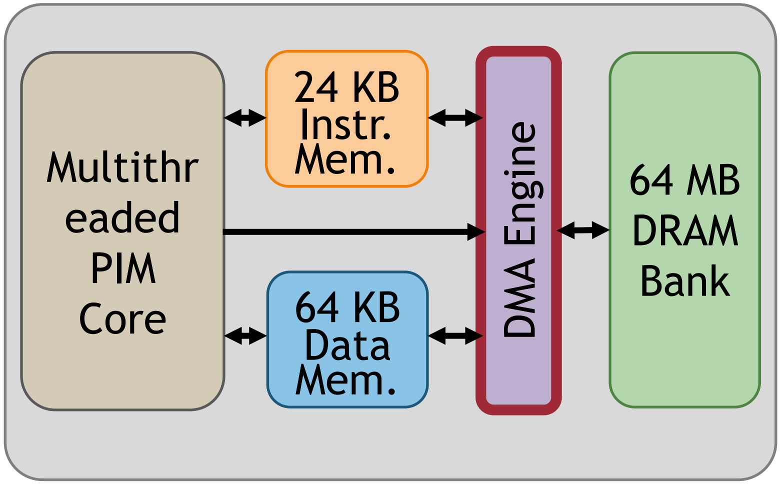


Fine-grained locking (lb-fg) does not improve performance over coarse-grained locking (lb-cg)

Performance of Synchronization Schemes

16 threads, 32-bit integer





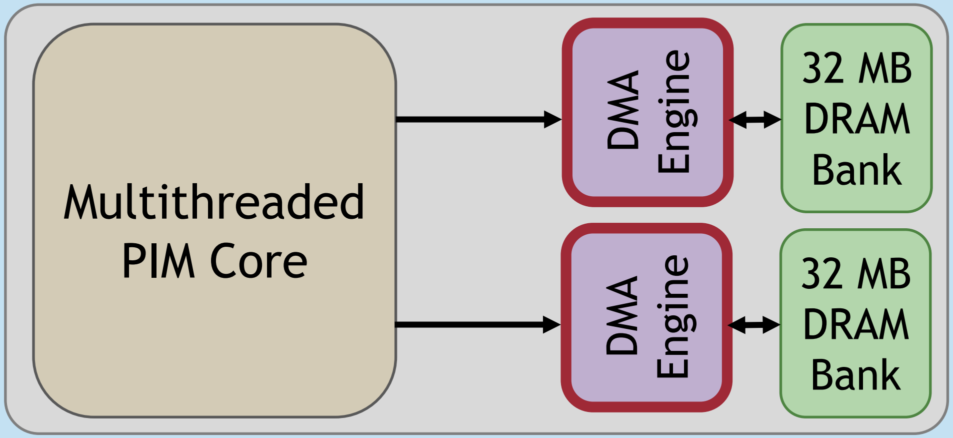
Fine-Grained Locking: memory accesses to the local DRAM bank are serialized in the DMA engine of the UPMEM PIM hardware.

Performance of Synchronization Schemes

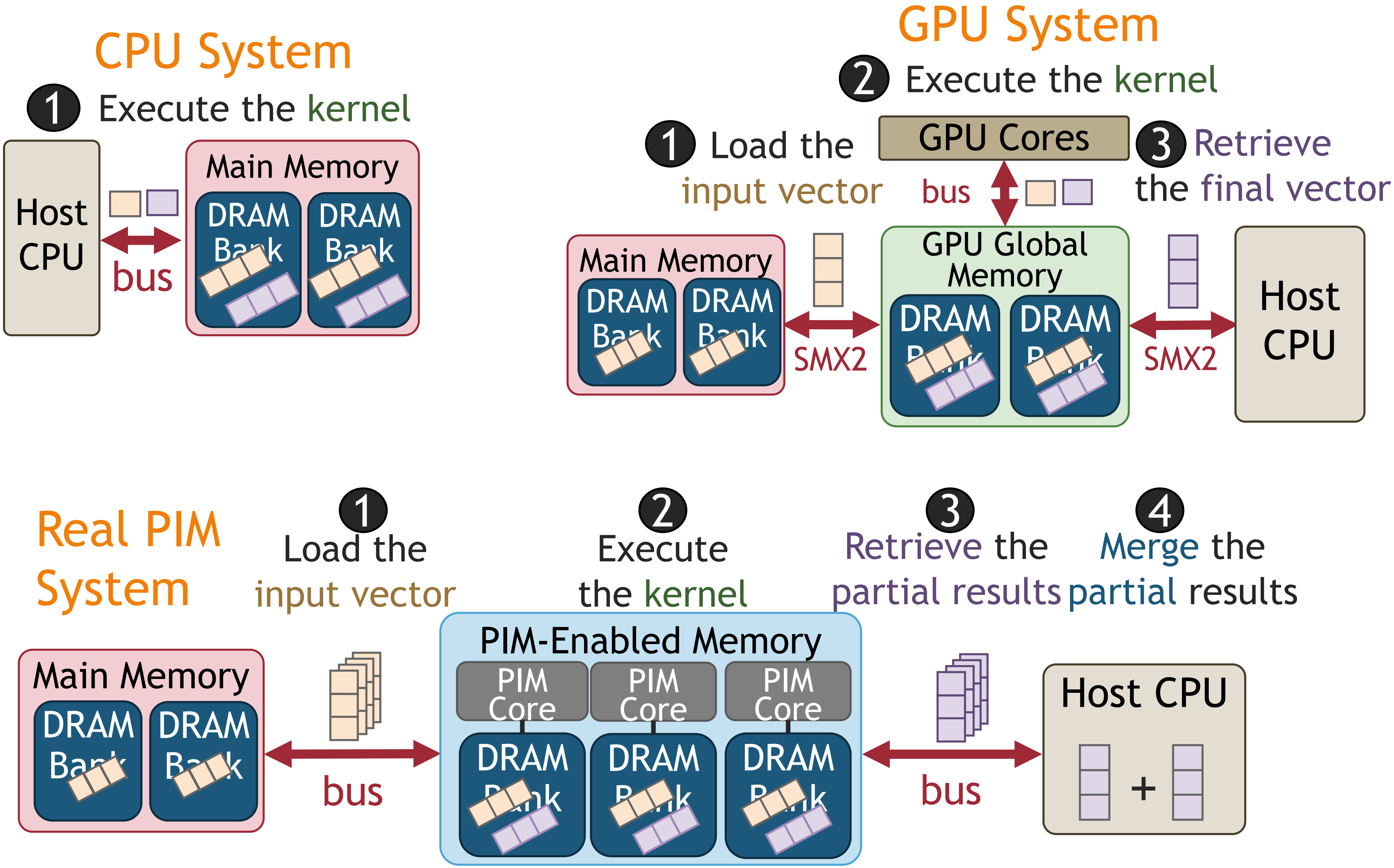
Recommendation

Improve Synchronization:

- Provide low-cost synchronization mechanisms for a multithreaded PIM core.
- Design hardware support to enable concurrent memory accesses to the local DRAM bank.
- Integrate multiple DRAM banks per PIM core to increase execution parallelism.



SpMV Execution on Various Systems



CPU/GPU Comparisons

- Kernel-Only (32-bit float):
  - CPU = 0.51% of Peak Perf.
  - GPU = 0.21% of Peak Perf.
  - PIM (1D) = 50.7% of Peak Perf.
- Kernel-Energy (32-bit float):
  - CPU = 0.247 J
  - GPU = 0.051 J
  - PIM (1D) = 0.179 J
- End-to-End (32-bit float):
  - CPU = 4.08 GFlop/s
  - GPU = 1.92 GFlop/s
  - PIM (1D) = 0.11 GFlop/s

System		Peak Performance	Bandwidth	TDP	
CPU	Intel Xeon Silver 4110	660 GFlops	23.1 GB/s	2x85 W	Processor-Centric
GPU	NVIDIA Tesla V100	14.13 TFlops	897 GB/s	300 W	
PIM	UPMEM 1st Gen.	4.66 GFlops	1.77 TB/s	379 W	Memory-Centric

CPU/GPU Comparisons

- Kernel-Only (32-bit float):
  - CPU = 0.51% of Peak Perf.
  - GPU = 0.21% of Peak Perf.
  - PIM (1D) = 50.7% of Peak Perf.
- Kernel-Energy (32-bit float):
  - CPU = 0.247 J
  - GPU = 0.051 J
  - PIM (1D) = 0.179 J
- End-to-End (32-bit float):
  - CPU = 4.08 GFlop/s
  - GPU = 1.92 GFlop/s
  - PIM (1D) = 0.11 GFlop/s

PIM: 1.38x higher energy efficiency over CPU

System		Peak Performance	Bandwidth	TDP	
CPU	Intel Xeon Silver 4110	660 GFlops	23.1 GB/s	2x85 W	Processor-Centric
GPU	NVIDIA Tesla V100	14.13 TFlops	897 GB/s	300 W	
PIM	UPMEM 1st Gen.	4.66 GFlops	1.77 TB/s	379 W	Memory-Centric

Core Contributions

1 Graph Processing

ColorTM (ISC'18, SRC PACT'18)

High-Performance Graph Coloring for CPU Systems

3 Irregular Workloads

SynCron (HPCA'21)

A Lightweight Synchronization Mechanism for PIM Systems

Thesis Statement:

Low-overhead synchronization approaches in cooperation with well-crafted data access techniques can significantly improve performance and energy efficiency of emerging irregular applications.

2 Pointer-Chasing

SmartPQ (CF'19)

An Adaptive Priority Queue for NUMA CPU Systems

4 Sparse Linear Algebra

SparseP (Sigmetrics'22)

A Library of Efficient Sparse Matrix Vector Multiplication Kernels for Real PIM Systems

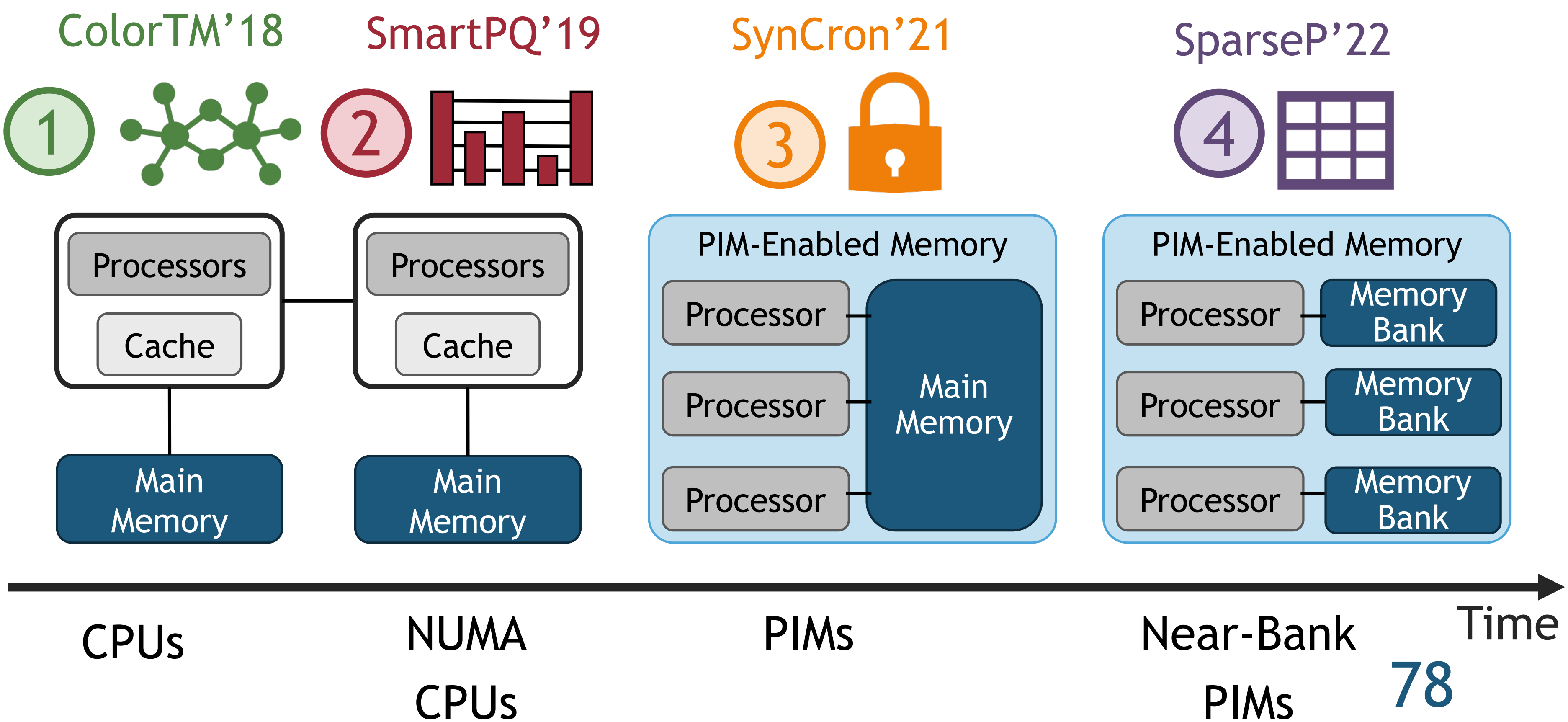
Thesis Summary

- Irregular applications exhibit:

Inherent imbalance, random accesses, low operational intensity

- Key optimization opportunities:

Lightweight synchronization + well-crafted data access policies



Future Research Directions

- Designing new adaptive approaches for irregular applications to capture dynamic workload demands and contention:
  - Adaptive algorithmic designs
  - Adaptive runtime systems
  - Adaptive hardware mechanisms

- Extending the techniques that we propose to accelerate irregular applications in new/unconventional systems:
  - Hybrid/heterogeneous memory systems
  - Disaggregated memory systems

- Leveraging the key insights and recommendations that we provide to improve multiple aspects of CPU and PIM hardware and software

Thesis Publications

1. "*Combining HTM with RCU to Speed up Graph Coloring on Multicore Platforms*"
   Christina Giannoula, Georgios Goumas, Nectarios Koziris
   [ISC 2018]
2. "*An Adaptive Priority Queue for NUMA Architectures*"
   Foteini Strati*, **Christina Giannoula***, Dimitrios Siakavaras, Georgios Goumas, Nectarios Koziris
   [CF 2019] (* joint first authors)
3. "*SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures*"
   Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, Onur Mutlu
   [HPCA 2021]
4. "*SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures*"
   Christina Giannoula, Ivan Fernandez, Juan Gómez-Luna, Nectarios Koziris, Georgios Goumas, Onur Mutlu
   [ISC 2018]
5. "*High-Performance and Balanced Parallel Graph Coloring on Multicore Platforms*"
   Christina Giannoula, Athanasios Peppas, Georgios Goumas, Nectarios Koziris
   [Journal of Supercomputing 2022]

Other Publications

1. "*SMASH: Co-designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations*"
   [Kanellopoulos+, **MICRO 2019**]
2. "*NATSA: A Near-Data Processing Accelerator for Time Series Analysis*"
   [Fernandez+, **ICCD 2020**]
3. "*Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware*"
   [Gómez-Luna+, **CUT 2021**]
4. "*Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System*"
   [Gómez-Luna+, **IEEE ACCESS 2022**]
5. "*An MRAM-based Accelerator for Time Series Analysis*"
   [Fernandez+, **Under Submission 2022**]
6. "*Architectural Support for Efficient Data Movement in Disaggregated Systems*"
   [Giannoula+, **Under Submission 2022**]
7. "*Architecting the Processor Core and Cache Hierarchy for Systems with Monolithically-Integrated Logic and Memory*"
   [Mansouri Ghiasi+, **Under Submission 2022**]

PhD Scholarships & Awards

- Foundation for Education and European Culture:
  - September 2021 – October 2022
- Hellenic Foundation for Research and Innovation + General Secretariat for Research and Technology:
  - October 2017 – March 2020
- NTUA Thomaidion Awards:
  - SynCron – HPCA 2021
  - NATSA - ICCD 2020
  - ColorTM - ISC 2018
- HiPEAC Research Award:
  - SynCron – HPCA 2021
- 2nd Place Winner at SRC Competition:
  - Balanced ColorTM – SRC PACT 2018

Acknowledgments

- Advisors:
  - Georgios Goumas, Nectarios Koziris, Onur Mutlu
- Committee Members:
  - Dionisios Pnevmatikatos, Stefanos Kaxiras, Dimitris Gizopoulos, Vasileios Papaefstathiou
- Co-Authors and Close Collaborators
- CSLab Group Members
- SAFARI Group Members
- Friends
- Family

National Technical University of Athens CSLab

SAFARI

Accelerating Irregular Applications via Efficient Synchronization and Data Access Techniques

Christina Giannoula

PhD Thesis Oral

Thank you!