PIMDAL
Mitigating the Memory Bottleneck in Data Analytics using a Real Processing-in-Memory System

Manos Frouzakis, Juan Gómez-Luna, Geraldo F. Oliveira, Mohammad Sadrosadati, Onur Mutlu

SAFARI

ETH zürich

Executive Summary

Problem:

Performance of database operators used for analytical query processing is limited by accesses to off-chip main memory

Motivation:

Processing-in-Memory systems move computation to where the data resides

Evaluate a real-world PIM system for analytical query processing

Challenge:

Current database implementations on conventional systems cannot straightforwardly be ported due to the limitations of current PIM systems

Goal:

Create implementation of TPC-H queries that achieves optimal performance on UPMEM PIM system, to evaluate potential of PIM for analytical query processing

Results:

PIM outperforms CPU in 4 out of 5 TPC-H queries, by 3.9× on average

Outperforms GPU on 2 of the queries that require little communication

Outline

Background and Goal

PIMDAL

Experiments and Methodology

Evaluation and Insights

Conclusion & Outlook

Outline

Background and Goal

PIMDAL

Experiments and Methodology

Evaluation and Insights

Conclusion & Outlook

Database Management Systems

- Development of DBMSs started in the late 1950s
  - Even before the invention of the microprocessor
- DBMSs evolved together with computer architectures
- Larger DRAM sizes enabled the development of in-memory databases, where data is kept in main memory
  - Performance and energy improvements compared to moving data from storage
- In-memory databases still suffer from data movement bottlenecks due to off-chip main memory accesses

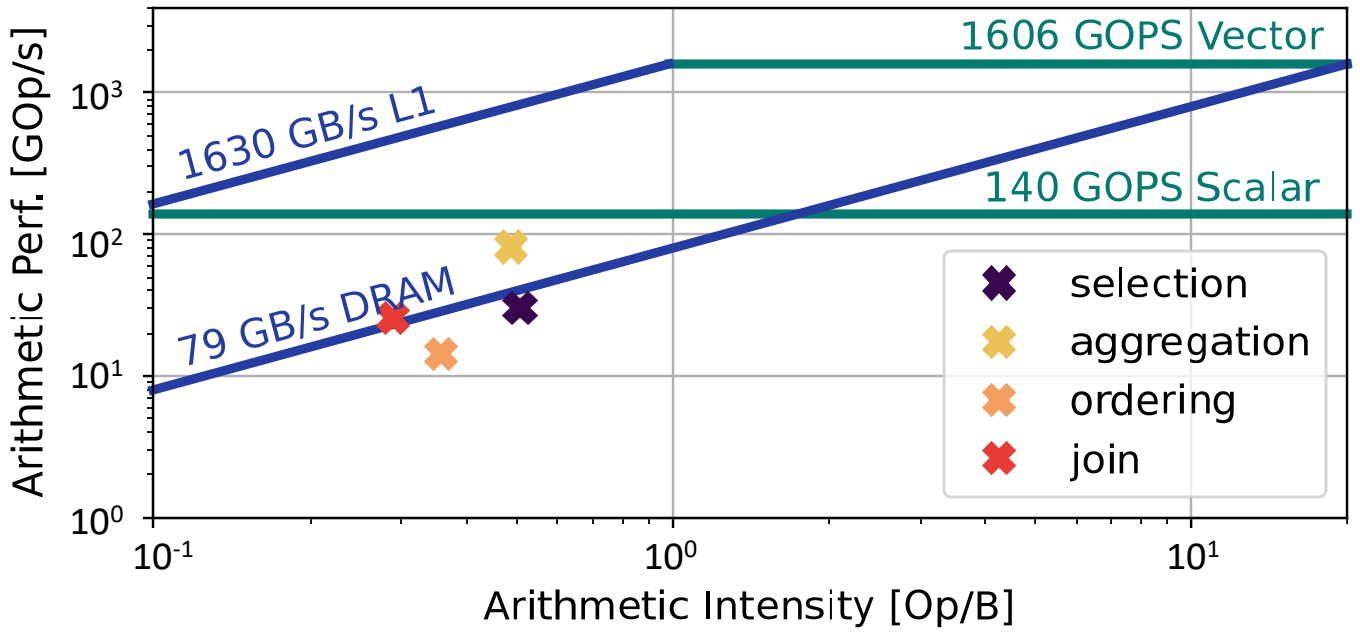
Database Operators

- Relational databases consist of tables where each data point is a row
  - Row is a tuple where each entry belongs to a set
- Database (DB) operators perform computation on rows and columns of one or multiple tables

DB Operator	Function
Selection	Filter tuples based on a predicate on columns
Aggregation	Grouping with aggregations groups columns with equal keys together and aggregates remaining columns
Ordering	Orders tuples based on some columns
Join	Concatenates rows of table to rows of another table based on key columns

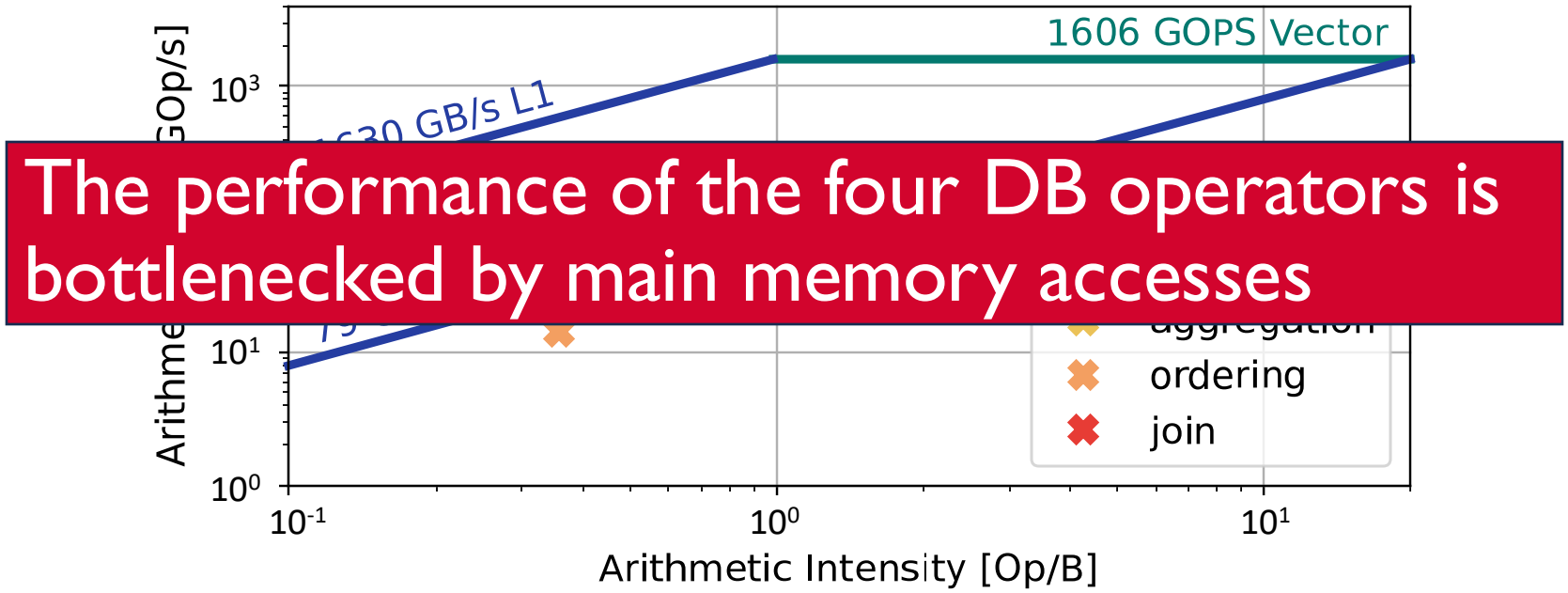
Memory-Bound DB Operators

- Roofline model of DB operators running on an Intel Xeon 6226R Gold CPU with 128 GB of DDR4 DRAM



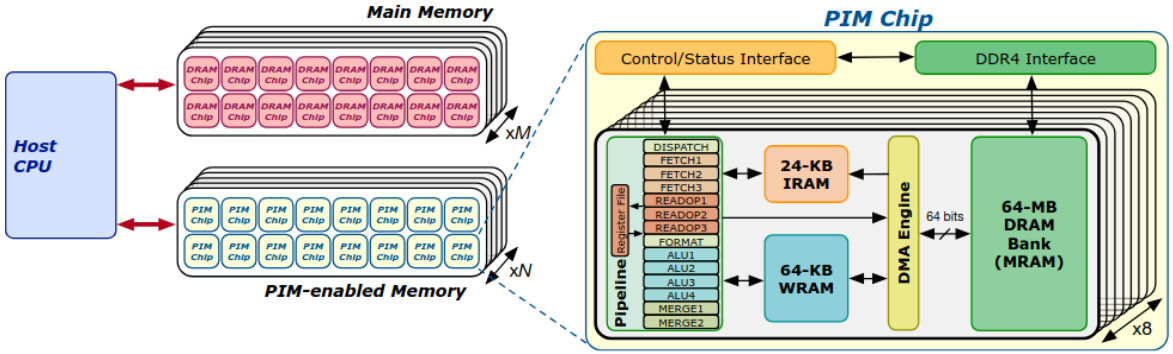
Memory-Bound DB Operators

- Roofline model of DB operators running on an Intel Xeon 6226R Gold CPU with 128 GB of DDR4 DRAM



The performance of the four DB operators is bottlenecked by main memory accesses

UPMEM System Characteristics



- Complement standard DDR4 DIMMs with PIM DIMMs equipped with PIM-cores (DPUs) for processing
- PIM-cores can access internal DRAM with high bandwidth
- PIM-cores are multi-threaded, in-order, RISC-like cores
- PIM-threads can independently execute their own code
- Use scratchpad memory (SPM) instead of caches, which is managed by programmer
- Inter PIM-core communication happens over host

Challenges and Limitations

Hardware unaware implementation of DB operators on UPMEM can lead to subpar performance due to three main reasons

Data movement in PIM code must be explicitly managed by programmer and can degrade performance if suboptimal

DB operators can require multiplication or division, which is costly on UPMEM due to the lack of hardware support

DB operators can require data redistribution and thus communication over the host

Goal

Create high performance implementation of TPC-H queries on UPMEM PIM system, to evaluate the potential of PIM for analytical query processing

- Implement DB operators on a real-world PIM system, while addressing the limitations previously mentioned
- Discover strengths and limitations of current memory-centric architectures for analytical query processing
- To this end we propose PIMDAL, a library for analytical query processing on the UPMEM system

Outline

Background and Goal

PIMDAL

Experiments and Methodology

Evaluation and Insights

Conclusion & Outlook

Design Principles

- From a previous UPMEM characterization[1] we deduce 2 design principles for PIMDAL
- Transfer latency from DRAM to scratchpad memory (SPM) has the form $\alpha + \beta \times size$

  ⟶ **Use as big, continuous memory transfers as possible**
- Full pipeline occupancy and throughput achieved at 11 threads

  ⟶ **Use at least 11 PIM-threads**
- Challenge: SPM is shared by all the threads, design principles can interfere with each other

[1] Gómez-Luna+, Benchmarking Memory-centric Computing Systems: Analysis of Real Processing-in-Memory Hardware, *IGSC* 2021

Structure

- PIMDAL consists of a PIM and a host component
- Host component feeds PIM system with data and provides communication capabilities
  - Load data from disk and transfer data between host and PIM system
  - Redistribute data between PIM-cores
- PIM component implements the four DB operators: *selection*, *aggregation*, *ordering* and *join*
  - Operates on data stored in column-store format in DRAM, where each column in a table is stored contiguously in memory

Two Algorithmic Building Blocks

- Two common algorithms for implementing aggregation and join are sorting and hashing:
- **Aggregation**
  - Group equal elements together by sorting them
  - Join elements by mapping them to the same hash table entry
- **Join**
  - Sort-merge: Order both tables in the same order, iterate efficiently, match the keys and join them
  - Hash: Store inner relation in hash table, probe with outer relation and join them
- Key Challenges: Parallelism and very irregular memory accesses

Outline

Background and Goal

PIMDAL

Experiments and Methodology

Evaluation and Insights

Conclusion & Outlook

Micro-Benchmarks

- Measure performance of standalone operators and compare to CPU/GPU
  - Kernel execution time
  - IPC of in-order pipeline
  - Weak and strong scaling analysis of end-to-end execution
- Main goal is to gain insights into performance of DB operators
- Comparison to CPU and GPU reference implementations
  - Comparison to CPU using custom implementation based on Apache Arrow
  - Comparison to GPU using cudf

TPC-H Queries

- TPC-H benchmark widely used for comparing different commercial DBMSs
- Suite of *business oriented ad-hoc queries*
- Use 5 queries from the TPC-H benchmark that do not rely on variable-length datatypes
  - Includes combinations of all DB operators
  - All require addition, 4 of them multiplication
- Reference implementations:
  - PyArrow on the CPU
  - Cudf on the GPU

System Configurations

	UPMEM PIM System	Intel Xeon Gold 6226R	Nvidia A6000 GPU
PEs	2048 DPUs	16 cores (32 threads)	108 SM
Memory	131 GB	128 GB	48 GB
Int32 Throughput	760 GOps	1606 GOps	38 TOps
Bandwidth	1.2 TB/s	79 GB/s	768 GB/s

Outline

Background and Goal

PIMDAL

PIMDAL: Data Movement Optimization

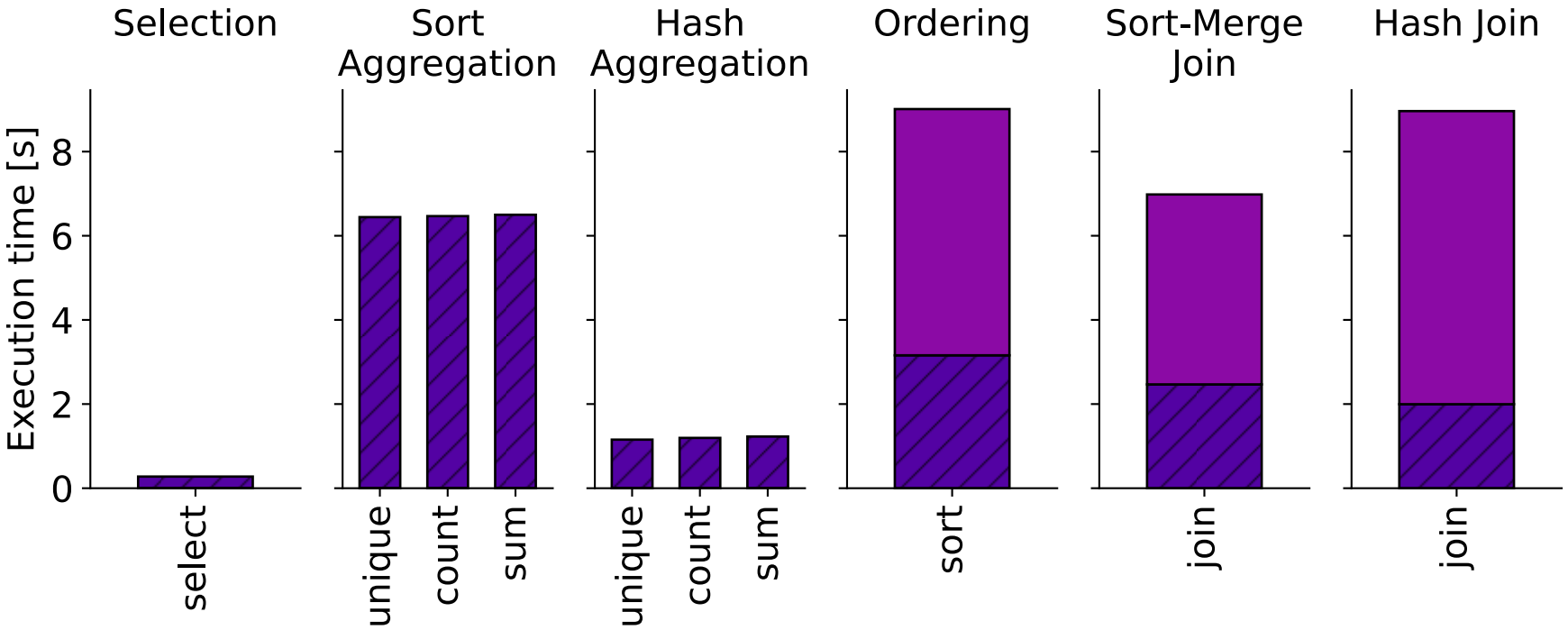
Experiments and Methodology

Evaluation and Insights

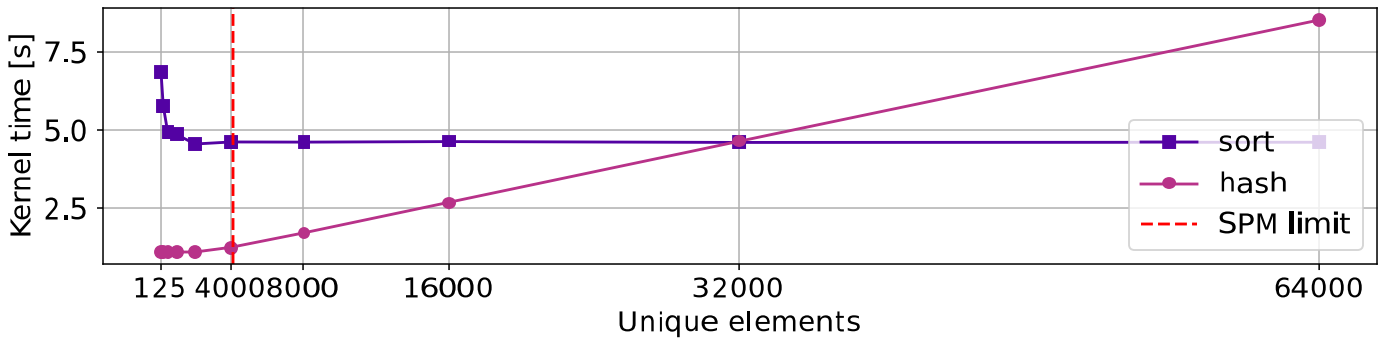
Conclusion

Towards Data-Centric Architectures

Kernel Execution Time Overview

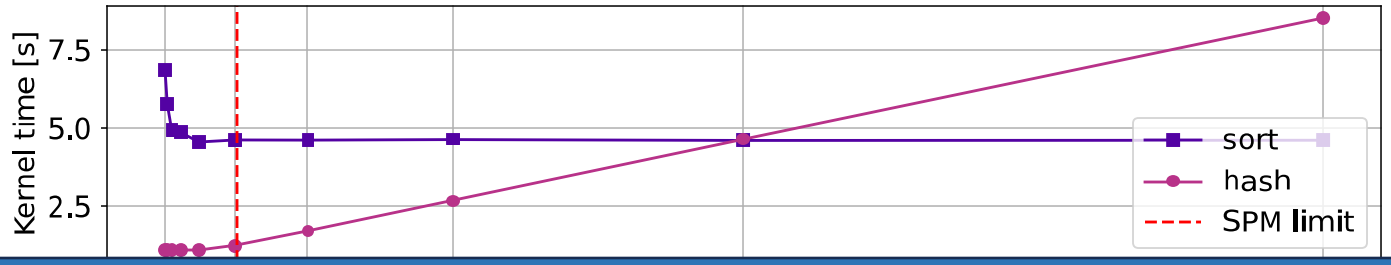


Aggregation



- Comparing hash to sort based aggregation for different number of unique elements/groups
- Sorting is inefficient for small number of output elements but performance remains constant for a growing number
  - Few elements cannot be partitioned efficiently
- Hash aggregation faster for smaller number of output elements
- Execution time increases for hash aggregation because steps are repeated if output does not fit in SPM

Aggregation



Key Takeaway 1: SPM or cache utilization still plays a role in the performance of PIM architectures, as the example of hash and sort based aggregation shows.

performance remains constant for a growing number

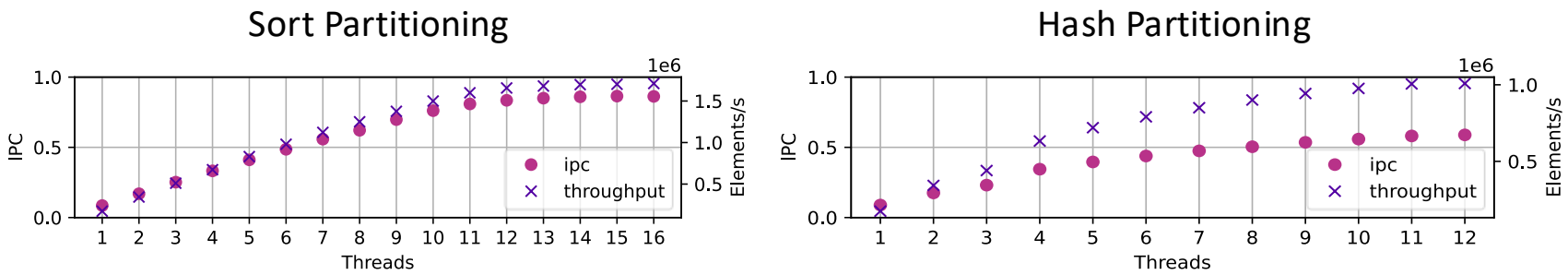
- Few elements cannot be partitioned efficiently

- Hash aggregation faster for smaller number of output elements
- Execution time increases for hash aggregation because steps are repeated if output does not fit in SPM

Join

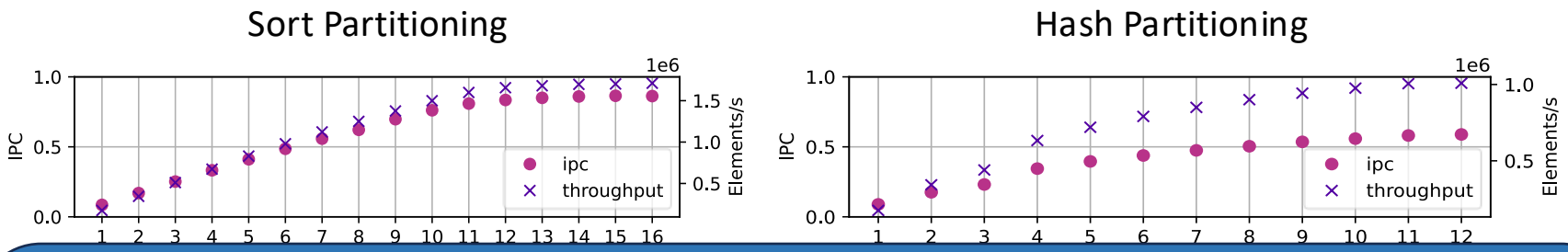
- Hash and sort-merge join are two commonly used join algorithms
- On conventional systems hash join achieves significantly better performance in the general use-case
- A surprising result is that on PIM systems sort-merge join outperforms hash join
- What we currently know about algorithm performance does not necessarily translate to PIM systems
- For more insights we can look into sort compared to hash partitioning performance

Sort vs Hash Partitioning



- Sort outperforms hash partitioning
- Sort partitioning has lower arithmetic intensity and simpler memory accesses
- Hash partitioning has higher arithmetic intensity and more complex memory accesses
- This is exactly where conventional and PIM systems differ
- We try to mitigate this using the radix-cluster algorithm
  - This leads to other issues such as synchronization

Sort vs Hash Partitioning

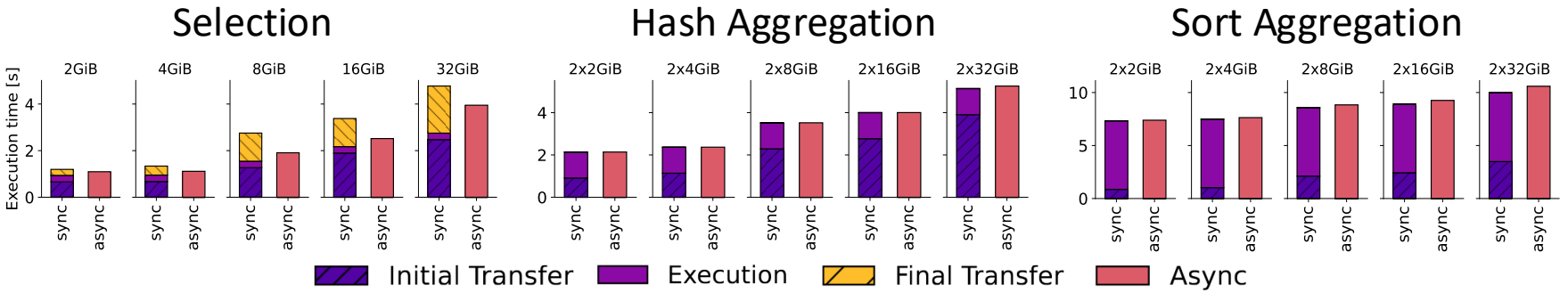


Key Takeaway 2: The most efficient algorithms for PIM can differ from the state-of-the-art on other architectures. One such example is sort-merge compared to hash join

complex memory accesses

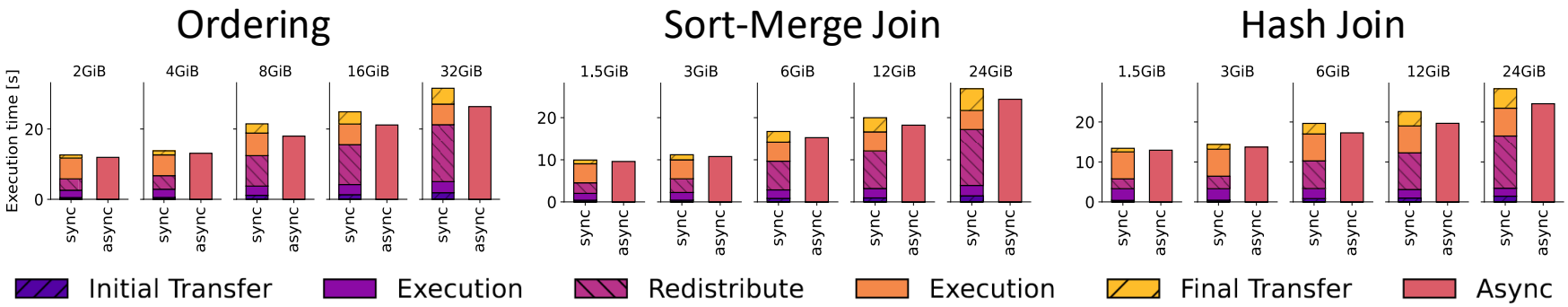
- This is exactly where conventional and PIM systems differ
- We try to mitigate this using the radix-cluster algorithm
  - This leads to other issues such as synchronization

Weak Scaling Analysis I



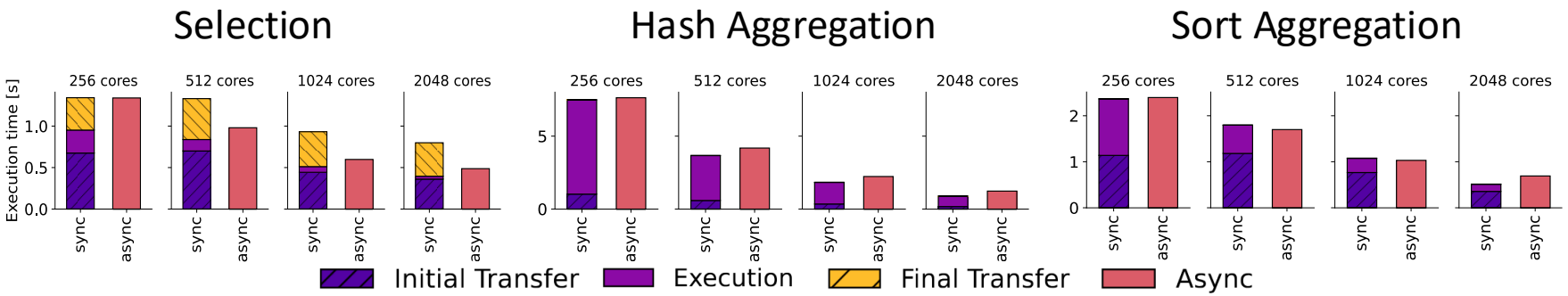
- The main bottleneck for hash aggregations is the initial, and for selection also the final data transfers
- Sort aggregation is less limited by transfers due to longer execution time
- Asynchronous execution only improves performance when there are significant transfers from and to the host
  - Transfers can be performed successively, improving bandwidth utilization

Weak Scaling Analysis II



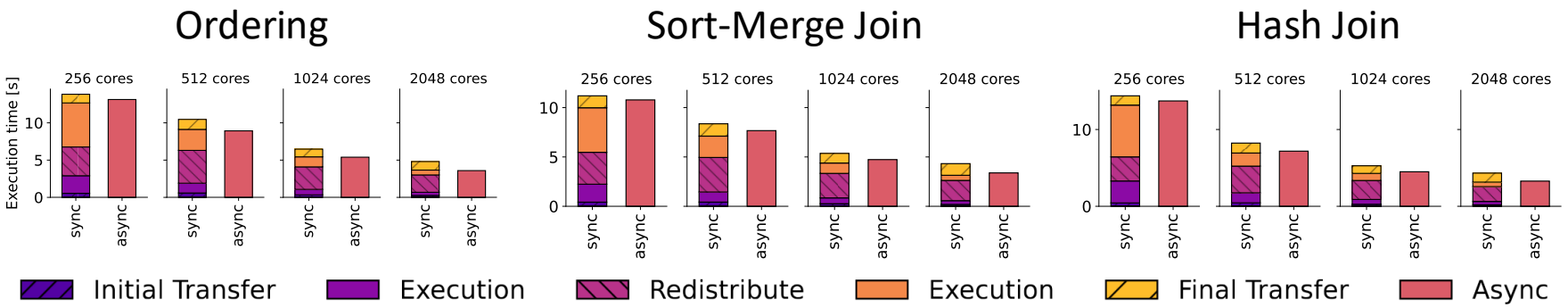
- Performance is similar for all three operators
- The biggest bottleneck in operator performance is data redistribution between PIM-cores
- For all DB operators the weak scaling with the number of PIM-cores or ranks is far from optimal
  - PIM execution time remains constant
  - Data transfer time increases due to contention

Strong Scaling Analysis I



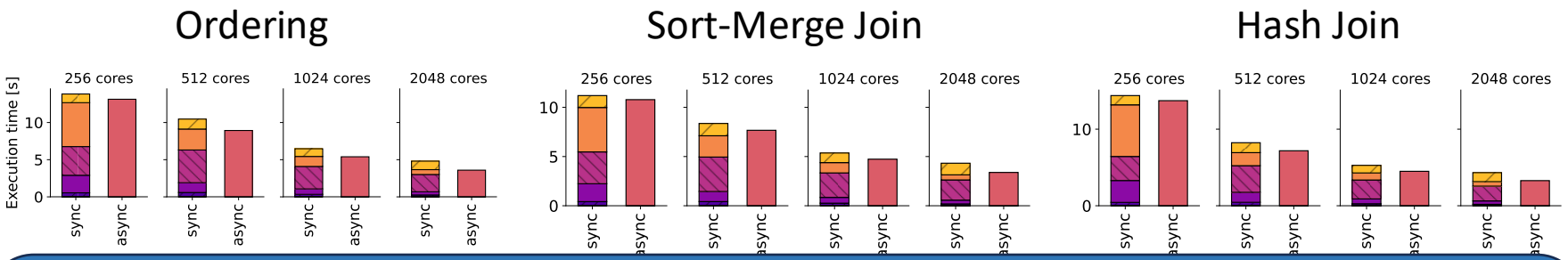
- Transfer and execution time improves with more PIM-cores, but not optimally
- Bandwidth does not fully scale with PIM-cores, since memory channels are shared by PIM-ranks

Strong Scaling Analysis II



- The behavior is similar for the more complex operators
- Overall, using as many PIM-cores as possible is always beneficial for the performance of DB operators on the UPMEM system
- The main bottleneck of complex DB operators remains data redistribution between PIM-cores

Strong Scaling Analysis II

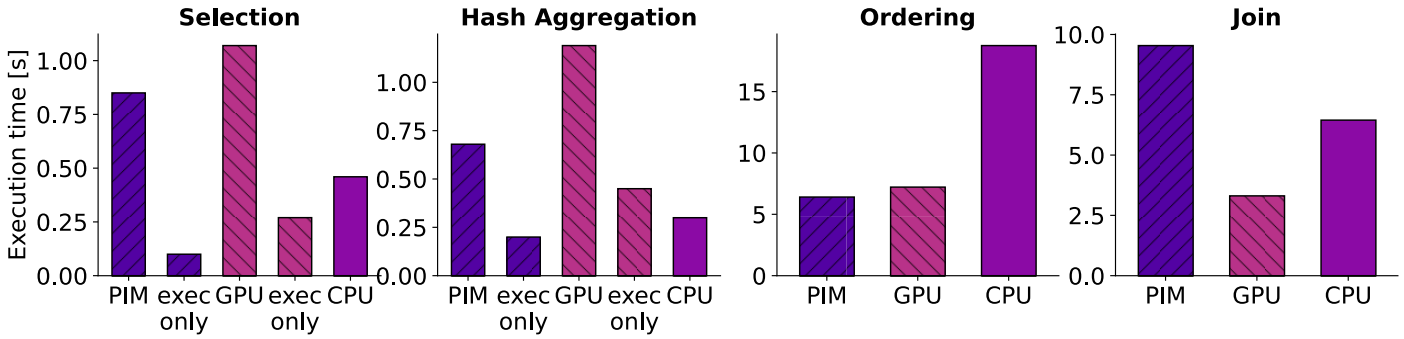


Key Takeaway 3: Current PIM systems are still limited by the weaknesses of processor-centric architectures. This is because processors have to be used for data transfers.

for the performance of DB operators on the UPMEM system

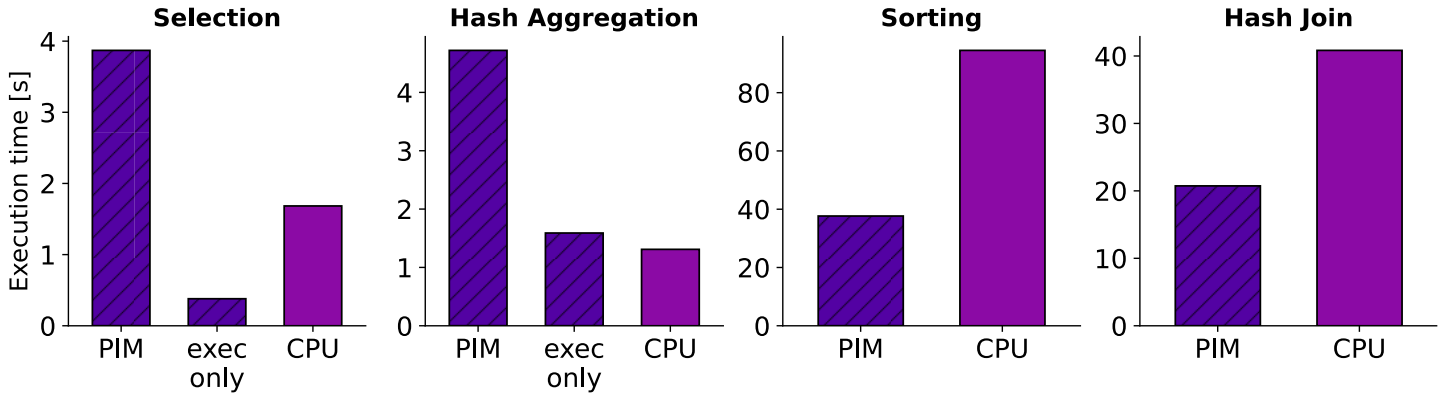
- The main bottleneck of complex DB operators remains data redistribution between PIM-cores

Comparison to CPU and GPU (8 GB)



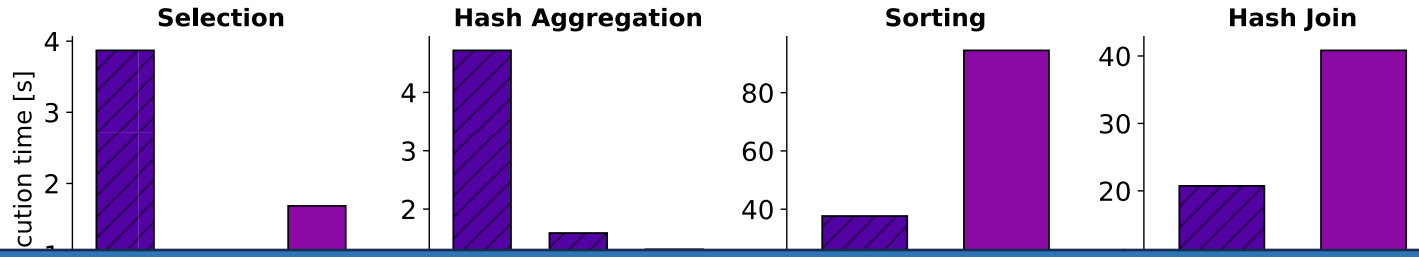
- Simpler operators selection and aggregation show no speedup on PIM and GPU when including transfer times
- Ordering with more memory accesses outperforms the CPU on both PIM and GPU
- Join is slower on PIM than on GPU due to data redistribution
- On CPU it is slower due to small data size as we show next

Comparison to CPU (32 GB)



- Comparison of selection remains the same
- For aggregation speedup gets worse, likely because the cache provides a higher performance uplift on the CPU
- The two more memory-bound operators perform the best on PIM compared to the CPU

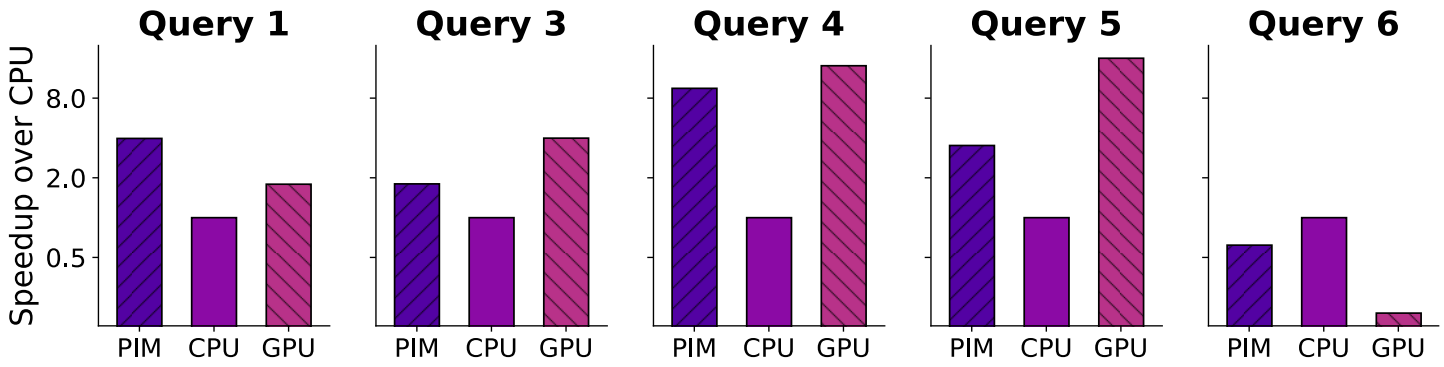
Comparison to CPU (32 GB)



Key Takeaway 4: Queries using ordering and join are better suited for acceleration using PIM systems compared to selection and aggregation.

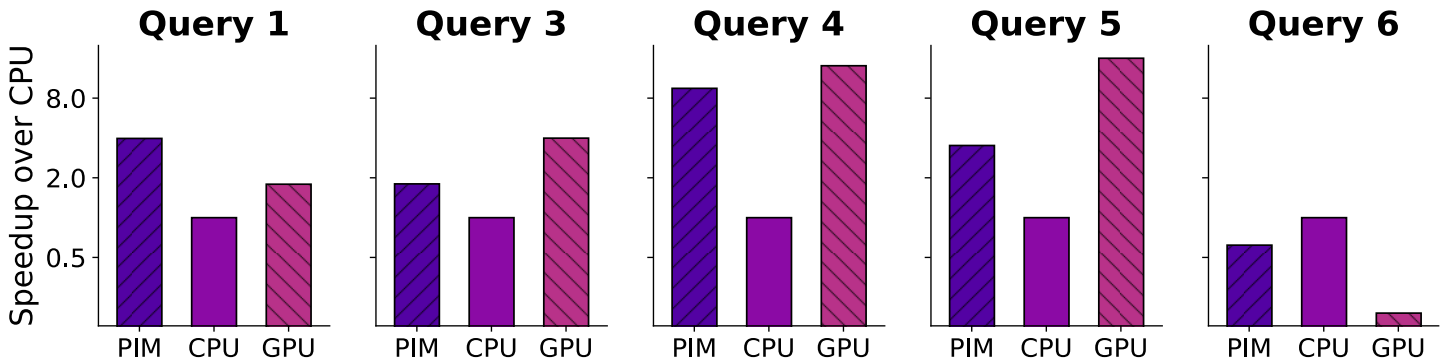
- For aggregation speedup gets worse, likely because the cache provides a higher performance uplift on the CPU
- The two more memory-bound operators perform the best on PIM compared to the CPU

TPC-H Benchmark CPU



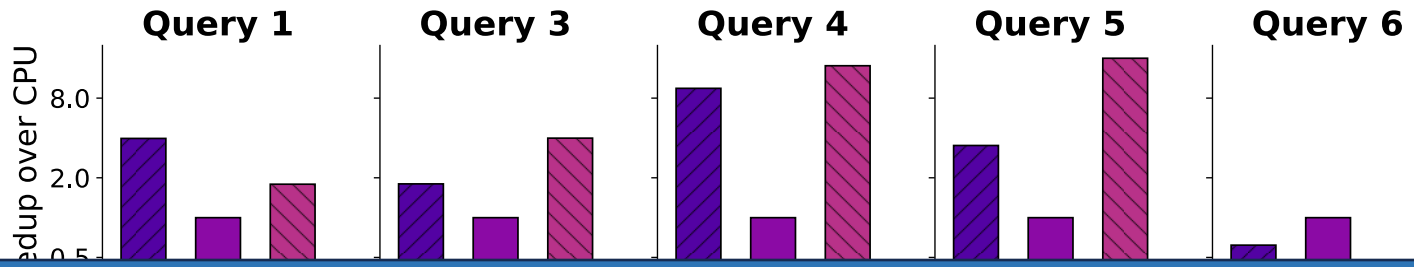
- Outperform CPU on all queries but 6, by 3.9× on average
- Speedup in queries 3 to 5 is mainly due to joins
- In query 1 due to high number of columns aggregated
- Outperform CPU even with complex arithmetic operations
  - Can represent decimals as 64-bit integers, multiplication is simple due to magnitudes of numbers used
  - Query 4 without multiplication performs best on PIM

TPC-H Benchmark GPU



- Outperform GPU in queries 1 and 6 that do not rely on join by 2.2× and 3.3× respectively
- Join requires data redistribution over the host which is slow
  - Single GPU has unified memory
- Overall TPC-H performance is limited by main memory
- Inter PIM-core communication is currently an issue, but scaling PIM memory should be easier than GPU memory

TPC-H Benchmark GPU



Key Takeaway 5: Data analytics is well suited for more data-centric architectures as the results from PIM and also the GPU show.

- Join requires data redistribution over the host which is slow
  - Single GPU has unified memory
- Overall TPC-H performance is limited by main memory
- Inter PIM-core communication is currently an issue, but scaling PIM memory should be easier than GPU memory

Outline

Background and Goal

PIMDAL

Experiments and Methodology

Evaluation and Insights

Conclusion & Outlook

Conclusion & Outlook

- Memory-centric architectures can significantly accelerate data analytics as PIMDAL shows on the UPMEM system
- We find strengths and weaknesses of PIM for implemented DB operators using HW performance metrics
- Communication between PIM-cores is a key limitation of current PIM systems
  - Might require a fundamentally new memory architecture
- Memory-centric systems can play a key role in data analytics in the future, especially if they can support even more complex operations as found in machine learning for example
- Main memory is only one part of the equation, what happens if data needs to be loaded from storage?

Questions?

"PIMDAL: Mitigating the Memory Bottleneck in Data Analytics using a Real Processing-in-Memory System" with more insights can be found on arXiv:

https://arxiv.org/abs/2504.01948