CS 839: Design the Next-Generation Database

Lecture 14: Process in Memory

Xiangyao Yu

3/5/2020

Announcements

Upcoming deadlines:

- Proposal due: **Mar. 10**

Fill this Google sheet for course project information

- https://docs.google.com/spreadsheets/d/1W7ObfjLqjDChm49GqrLg49x6r4B28-f-PBpQPHX01Mk/edit?usp=sharing

Discussion Highlights

Prof. Stronebraker's comment

- Agree with the comment; future is unpredictable
- Not entirely true
- Recent several papers: looking for problems using new hardware as a solution

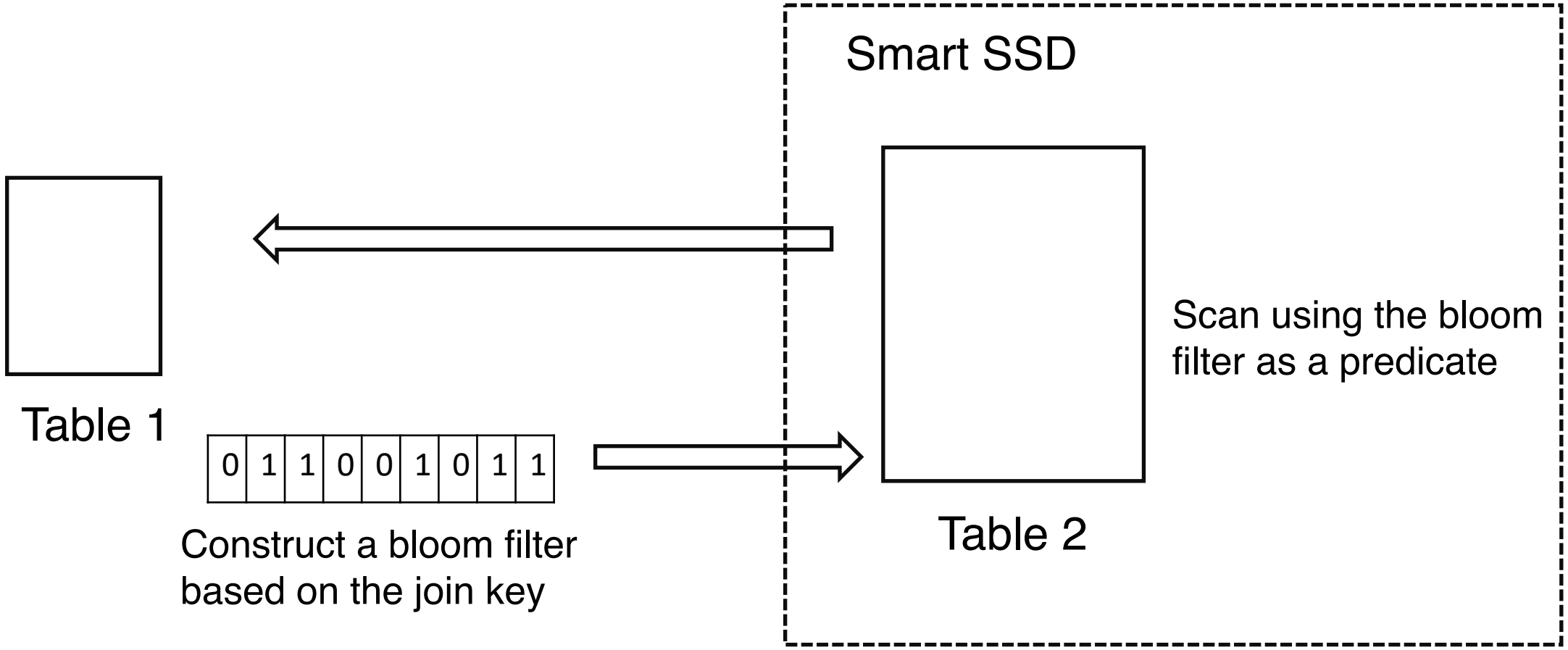
Fast IO/Network affect smart memory/storage?

- Closes internal/external bandwidth gap => less gain from smart SSD
- Cost and energy

Supporting complex operators

- Join: Small table fits in Smart SSD memory; computation simple enough
- Breakdown the complex operators
- Not wise to push join entirely
- Push some simple group-by
- Data partitioning in Smart SSD

Bloom Join


Smart SSD
Table 1
0 1 1 0 0 1 0 1 1
Construct a bloom filter based on the join key
Table 2
Scan using the bloom filter as a predicate

Today's Paper

NEAR-DATA PROCESSING: INSIGHTS FROM A MICRO-46 WORKSHOP

Rajeev Balasubramonian
University of Utah

Jichuan Chang
Google

Troy Manning
Micron

Jaime H. Moreno
IBM Thomas J. Watson Research Center

Richard Murphy
Micron

Ravi Nair
IBM Thomas J. Watson Research Center

Steven Swanson
University of California, San Diego

AFTER A DECADE-LONG DORMANCY, INTEREST IN NEAR-DATA PROCESSING (NDP) HAS SPIKED. A WORKSHOP ON NDP WAS ORGANIZED AT MICRO-46 AND WAS WELL ATTENDED. GIVEN THE INTEREST, THE ORGANIZERS AND KEYNOTE SPEAKERS HAVE ATTEMPTED TO CAPTURE THE KEY INSIGHTS FROM THE WORKSHOP FOR WIDER DISSEMINATION. THIS ARTICLE DESCRIBES WHY NDP IS COMPELLING TODAY AND IDENTIFIES UPCOMING CHALLENGES IN REALIZING ITS POTENTIAL.

Processing in large-scale systems is shifting from the traditional computing-centric model successfully used for many decades into one that is more data centric. This transition is driven by the evolving nature of computing, which is no longer dominated by the execution of arithmetic and logic calculations but instead by the handling of large data volume and the cost of moving data to the locations where computations are performed. The computing-centric model where data lives on disk, or even tape, and moves as needed to a central computing engine across a deep storage hierarchy is sufficient when computational aspects dominate data movement aspects. In contrast, in the data-centric model, data lives in different storage levels within the hierarchy, with processing engines surrounding the data and operating on such data without moving it across the system.

The trend toward big data is leading to changes in the computing paradigm, and in particular to the notion of moving computation to data, in what we call the *near-data processing* (NDP) approach. Data movement impacts performance, power efficiency, and reliability, three fundamental attributes of a system. NDP seeks to minimize data movement by computing at the most appropriate location in the hierarchy, considering the location of the data and the information that needs to be extracted from that data. Thus, in NDP, computation can be performed right at the data's home, either in caches, main memory, or persistent storage. This is in contrast to the movement of data toward a CPU independent of where it resides, as is done traditionally. Examples of NDP already exist in systems that execute computations close to the disk, filtering or preprocessing the data streaming from the disks so that a minimal number of items are transferred for processing at other parts of the system. Conceptually, the same principle can be applied at other levels of a system's memory and storage hierarchy by placing computing resources close to where data is located, and restructuring applications to exploit the resulting distributed computing infrastructure.

At the MICRO-46 conference, a workshop was held to bring together experts from academia and industry, who presented recent advances in the development of large systems employing NDP principles (www.cs.utah.

Published by the IEEE Computer Society

IEEE MICRO 2014

Database Processing-in-Memory: An Experimental Study

Tiago R. Kepe
UFPR & IFPR, Brazil
trkepe@inf.ufpr.br

Eduardo C. de Almeida
UFPR, Brazil
eduardo@inf.ufpr.br

Marco A. Z. Alves
UFPR, Brazil
mazalves@inf.ufpr.br

ABSTRACT

The rapid growth of "big-data" intensified the problem of data movement when processing data analytics: Large amounts of data need to move through the memory up to the CPU before any computation takes place. To tackle this costly problem, Processing-in-Memory (PIM) inverts the traditional data processing by pushing computation to memory with an impact on performance and energy efficiency. In this paper, we present an experimental study on processing database SIMD operators in PIM compared to current x86 processor (i.e., using AVX512 instructions). We discuss the execution time gap between those architectures. However, this is the first experimental study, in the database community, to discuss the trade-offs of execution time and energy consumption between PIM and x86 in the main query execution systems: materialized, vectorized, and pipelined. We also discuss the results of a hybrid query scheduling when interleaving the execution of the SIMD operators between PIM and x86 processing hardware. In our results, the hybrid query plan reduced the execution time by 45%. It also drastically reduced energy consumption by more than 2× compared to hardware-specific query plans.

PVLDB Reference Format:
Tiago R. Kepe, Marco A. Z. Alves, and Eduardo C. de Almeida. Database Processing-in-Memory: An Experimental Study. *PVLDB*, 13(3): 334-347, 2019.
DOI: https://doi.org/10.14778/3368289.3368298

1. INTRODUCTION

Applications based on data analysis need to move large amounts of data between memory and processing units to look for patterns. Computers have relied on this traditional computing-centric execution since the introduction of the Von Neumann model. In this model, however, data movement severely affects performance and energy consumption. Recent studies show that data movement accounts for almost 63% of the total energy consumption and imposes high latencies [6, 36].

Traditional query execution systems have been operating only on computing-centric models [11]. The materialization query execution system generates lots of intermediate data that move along the memory hierarchy to process the operators of a query plan. The vectorized query execution system tries to exploit the caching mechanism and the CPU processing with a high interpretation overhead. The pipelined query execution system uses the Just-In-Time (JIT) compilation to fuse operators of the same pipeline into a monolithic code fragment. Although the authors of [28] call JIT as data-centric compilation, the query execution is still computing-centric by moving data to the CPU with many adaptations to make better use of the memory caches. In this paper, we study the data-centric execution model to tackle the data movement problem in query execution systems with logical units integrated closer to the data (inside memory devices), which is called Processing-in-Memory (PIM) [25, 34, 45].

Database engineers have been evaluating PIM approaches with processing components installed in magnetic disks [1, 12, 26], RAM [38], and more recently in flash disks [10, 13, 46]. However, commercial products have not been adopting those approaches for three main reasons: 1) Limitations of the hardware technology; 2) The continuous growth in CPU performance complied to the Moore's Law and Dennard scaling; 3) The lack of a general programming interface that leads to low abstraction level when handling hardware errors.

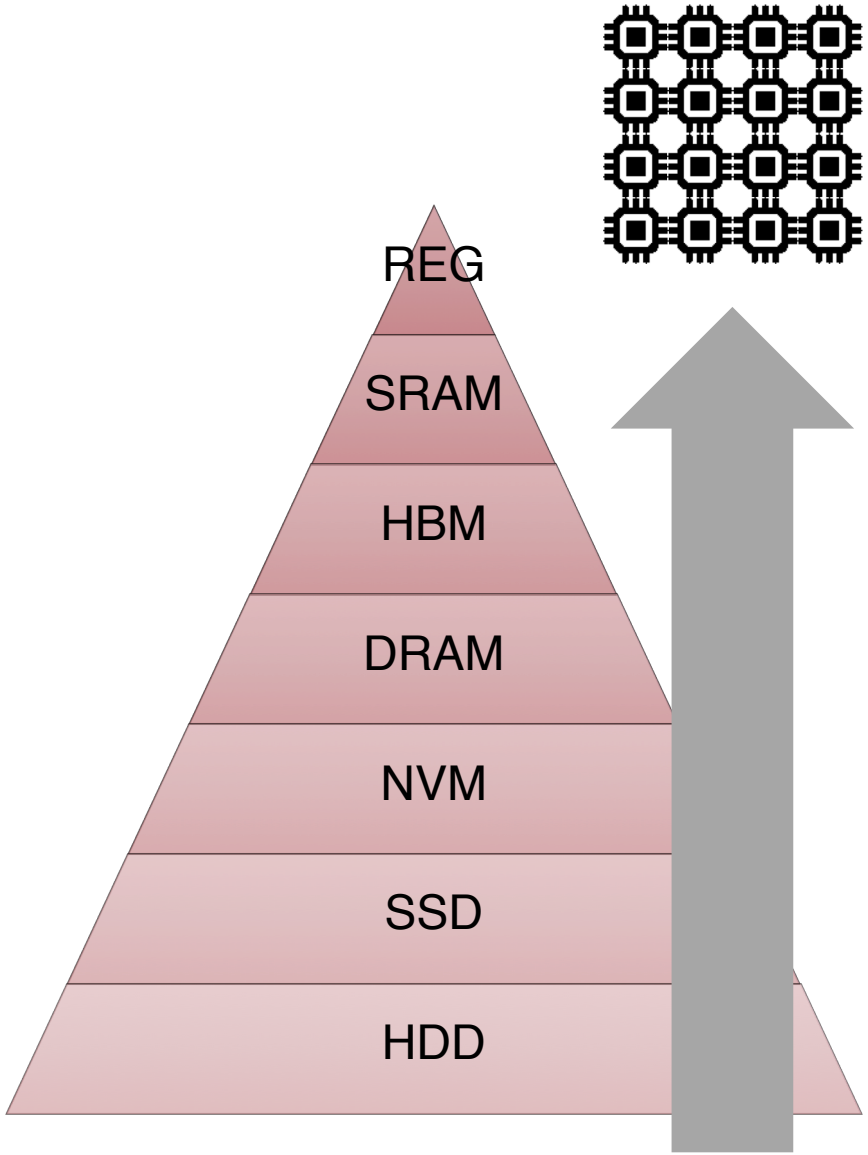
Now, modern PIM hardware integrate traditional DRAM dies and logic cells in the same chip area with the Through-Silicon Via (TSV), forming a 3D-stacked memory with a high degree of parallelism. Therefore, modern PIM can leverage current memory protocols to handle hardware errors. Current GPUs already embed the emerging 3D-stacked memories, such as the Hybrid Memory Cube (HMC) [23] and the High Bandwidth Memory (HBM) [30]. However, there has not been any in-depth study of query processing on PIM with Single Instruction Multiple Data (SIMD) support.

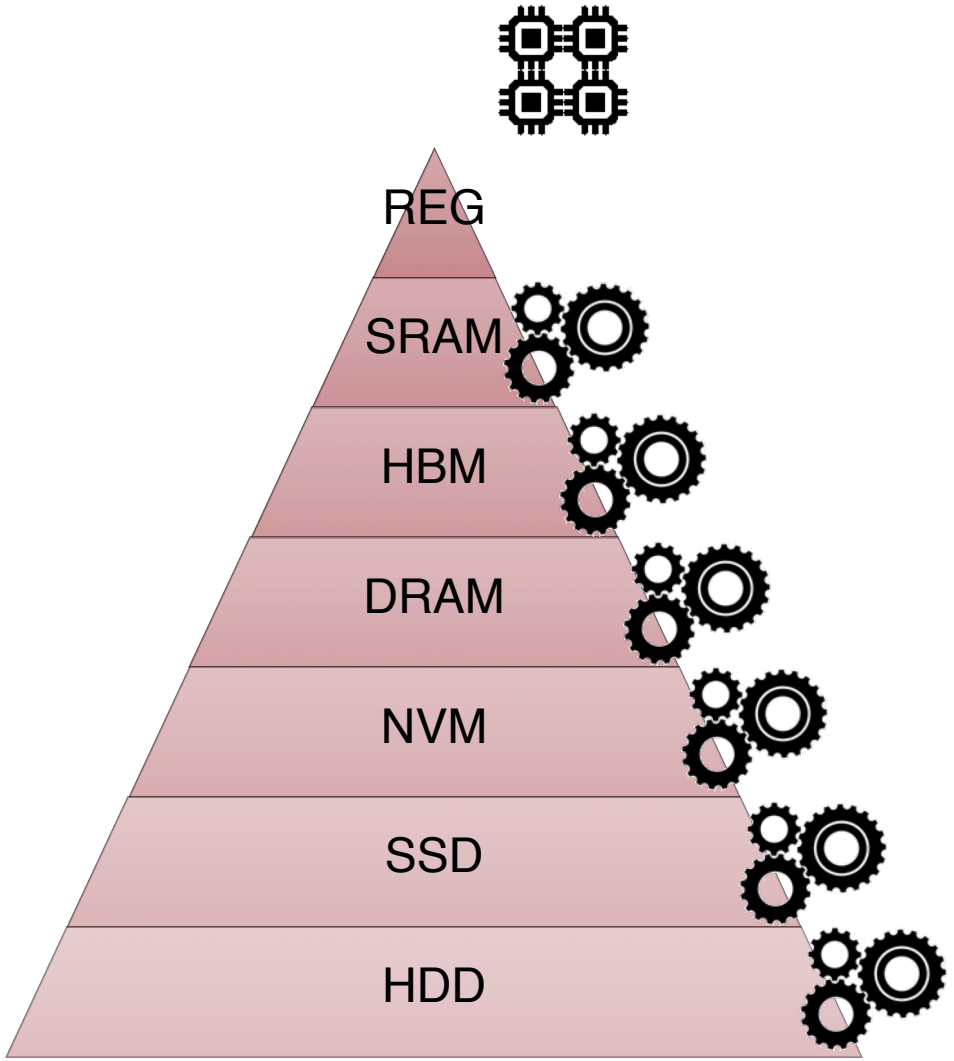
In this experimental paper, we detail the implementation of five major query operators into a simulator of PIM hardware: selection, projection, aggregation, sorting, and join (hash join, sort-merge join, and nested loop join). In particular, we present a new SIMD sorting algorithm that requires fewer memory instructions compared to the state of the art [21]. For each operator, we gauge the latency and energy spend to process TPC-H and Zipf distribution datasets. We evaluate the high levels of parallelism and data access when using AVX512 extensions from x86 processors, compared to modern PIM architectures with SIMD support on registers of 256-bytes wide. To the best of our knowledge,

Proceedings of the VLDB Endowment, Vol. 13, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3368289.3368298

VLDB 2019

Compute Centric vs. Data Centric





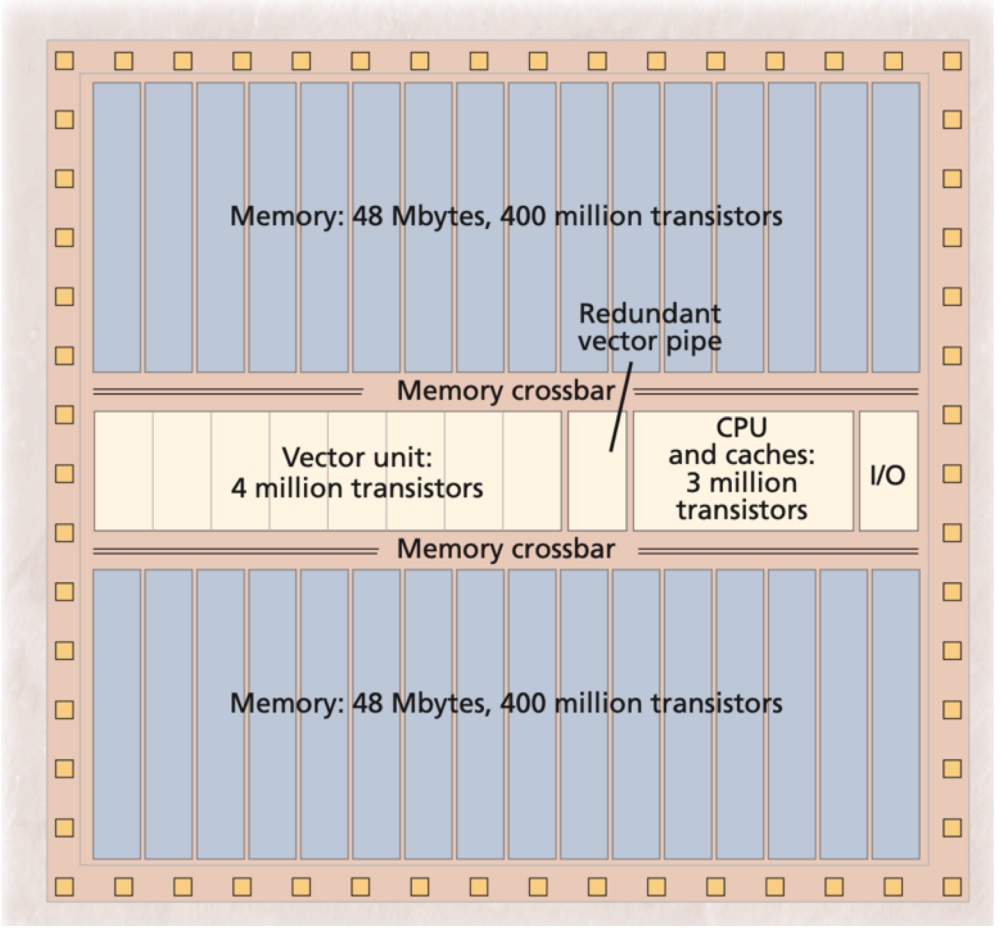
Process-in-Memory (PIM) in Late 1990's

[1] P.Kogge,"A Short History of PIM at Notre Dame," July 1999

[2] C.E. Kozyrakis et al., "Scalable Processors in the Billion Transistor Era: IRAM," Computer, 1997

[3] T.L. Sterling and H.P. Zima, "Gilgamesh: A Multithreaded Processor-in-Memory Architecture for Petaflops Computing", Supercomputing, 2002

[4] J. Draper et al., "The Architecture of the DIVA Processing-in-Memory Chip" Supercomputing, 2002



Reasons of PIM Failure in 2000s

Incompatibility of DRAM and CPU processes

- DRAM is designed with a costly logic process
- Logic designed with a process optimized for DRAM

PIM requires a new programming model

Top 10 reasons for a revitalized NDP 2.0

1. Necessity. Increasing overheads of computing-centric architectures

- Moving computation close to data reduces data movement and cache hierarchy overhead;
- Rebalance of computing-to-memory ratios;
- Specializing computation for the data transformation

2. Technology. 3D and 2.5D die-stacking technologies are mature

- Eliminating previous disadvantages of merged logic and memory fabrication
- The close proximity of computation => high bandwidth with low energy

Top 10 reasons for a revitalized NDP 2.0

3. Software. Distributed software frameworks (e.g., MapReduce)

- Smooth learning curve of programming NDP hardware
- Handle data layout, naming, scheduling, and fault tolerance

4. Interface. Impossible with DDR but memory interface will change

- Mobile DRAM is replacing desktop/server DRAM
- New interfaces such as HMC already includes preliminary NDP support

5. Hierarchy. New nonvolatile memories (NVMs) that combine memory-like performance with storage-like capacity enable a flattened memory/storage hierarchy and **self-contained NDP computing elements**. In essence, this flattened hierarchy eliminates the bottleneck of getting data on and off the NDP memory

Top 10 reasons for a revitalized NDP 2.0

6. Balance. Communication between NDP may be the new bottleneck

- New system-on- a-chip (SoC) and die-stacking technologies
- New opportunities for NDP-customized interconnect designs

7. Heterogeneity. NDP involves heterogeneity for specialization

8. Capacity. NVM in NDP has large device capacities and lower cost

- Early NDP designs were limited by small device capacities that forced too much fine-grained parallelism and inter device data movement

Top 10 reasons for a revitalized NDP 2.0

9. Anchor workloads. Big-data appliances

- For example, IBM's Netezza and Oracle's Exadata

10. Ecosystem. Prototypes, tools, and

- Software programming models: OpenMP4.0, OpenCL, and MapReduce
- Hardware prototypes: Adapteva, Micron, Vinray, and Samsung

Challenges of NDP

- Packaging and thermal constraints
- Communication interfaces
- Synchronization mechanisms
- Optimizing processing cores
- Programming model
- Security

Today's Paper

NEAR-DATA PROCESSING: INSIGHTS FROM A MICRO-46 WORKSHOP

Rajeev Balasubramonian
University of Utah

Jichuan Chang
Google

Troy Manning
Micron

Jaime H. Moreno
IBM Thomas J. Watson Research Center

Richard Murphy
Micron

Ravi Nair
IBM Thomas J. Watson Research Center

Steven Swanson
University of California, San Diego

AFTER A DECADE-LONG DORMANCY, INTEREST IN NEAR-DATA PROCESSING (NDP) HAS SPIKED. A WORKSHOP ON NDP WAS ORGANIZED AT MICRO-46 AND WAS WELL ATTENDED. GIVEN THE INTEREST, THE ORGANIZERS AND KEYNOTE SPEAKERS HAVE ATTEMPTED TO CAPTURE THE KEY INSIGHTS FROM THE WORKSHOP FOR WIDER DISSEMINATION. THIS ARTICLE DESCRIBES WHY NDP IS COMPELLING TODAY AND IDENTIFIES UPCOMING CHALLENGES IN REALIZING ITS POTENTIAL.

Processing in large-scale systems is shifting from the traditional computing-centric model successfully used for many decades into one that is more data centric. This transition is driven by the evolving nature of computing, which is no longer dominated by the execution of arithmetic and logic calculations but instead by the handling of large data volume and the cost of moving data to the locations where computations are performed. The computing-centric model where data lives on disk, or even tape, and moves as needed to a central computing engine across a deep storage hierarchy is sufficient when computational aspects dominate data movement aspects. In contrast, in the data-centric model, data lives in different storage levels within the hierarchy, with processing engines surrounding the data and operating on such data without moving it across the system.

The trend toward big data is leading to changes in the computing paradigm, and in particular to the notion of moving computation to data, in what we call the *near-data processing* (NDP) approach. Data movement impacts performance, power efficiency, and reliability, three fundamental attributes of a system. NDP seeks to minimize data movement by computing at the most appropriate location in the hierarchy, considering the location of the data and the information that needs to be extracted from that data. Thus, in NDP, computation can be performed right at the data's home, either in caches, main memory, or persistent storage. This is in contrast to the movement of data toward a CPU independent of where it resides, as is done traditionally. Examples of NDP already exist in systems that execute computations close to the disk, filtering or preprocessing the data streaming from the disks so that a minimal number of items are transferred for processing at other parts of the system. Conceptually, the same principle can be applied at other levels of a system's memory and storage hierarchy by placing computing resources close to where data is located, and restructuring applications to exploit the resulting distributed computing infrastructure.

At the MICRO-46 conference, a workshop was held to bring together experts from academia and industry, who presented recent advances in the development of large systems employing NDP principles (www.cs.utah.

Published by the IEEE Computer Society

IEEE MICRO 2014

Database Processing-in-Memory: An Experimental Study

Tiago R. Kepe
UFPR & IFPR, Brazil
trkepe@inf.ufpr.br

Eduardo C. de Almeida
UFPR, Brazil
eduardo@inf.ufpr.br

Marco A. Z. Alves
UFPR, Brazil
mazalves@inf.ufpr.br

ABSTRACT

The rapid growth of "big-data" intensified the problem of data movement when processing data analytics: Large amounts of data need to move through the memory up to the CPU before any computation takes place. To tackle this costly problem, Processing-in-Memory (PIM) inverts the traditional data processing by pushing computation to memory with an impact on performance and energy efficiency. In this paper, we present an experimental study on processing database SIMD operators in PIM compared to current x86 processor (i.e., using AVX512 instructions). We discuss the execution time gap between those architectures. However, this is the first experimental study, in the database community, to discuss the trade-offs of execution time and energy consumption between PIM and x86 in the main query execution systems: materialized, vectorized, and pipelined. We also discuss the results of a hybrid query scheduling when interleaving the execution of the SIMD operators between PIM and x86 processing hardware. In our results, the hybrid query plan reduced the execution time by 45%. It also drastically reduced energy consumption by more than 2× compared to hardware-specific query plans.

PVLDB Reference Format:
Tiago R. Kepe, Marco A. Z. Alves, and Eduardo C. de Almeida. Database Processing-in-Memory: An Experimental Study. *PVLDB*, 13(3): 334-347, 2019.
DOI: https://doi.org/10.14778/3368289.3368298

1. INTRODUCTION

Applications based on data analysis need to move large amounts of data between memory and processing units to look for patterns. Computers have relied on this traditional computing-centric execution since the introduction of the Von Neumann model. In this model, however, data movement severely affects performance and energy consumption. Recent studies show that data movement accounts for almost 63% of the total energy consumption and imposes high latencies [6, 36].

Proceedings of the VLDB Endowment, Vol. 13, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3368289.3368298

Traditional query execution systems have been operating only on computing-centric models [11]. The materialization query execution system generates lots of intermediate data that move along the memory hierarchy to process the operators of a query plan. The vectorized query execution system tries to exploit the caching mechanism and the CPU processing with a high interpretation overhead. The pipelined query execution system uses the Just-In-Time (JIT) compilation to fuse operators of the same pipeline into a monolithic code fragment. Although the authors of [28] call JIT as data-centric compilation, the query execution is still computing-centric by moving data to the CPU with many adaptations to make better use of the memory caches. In this paper, we study the data-centric execution model to tackle the data movement problem in query execution systems with logical units integrated closer to the data (inside memory devices), which is called Processing-in-Memory (PIM) [25, 34, 45].

Database engineers have been evaluating PIM approaches with processing components installed in magnetic disks [1, 12, 26], RAM [38], and more recently in flash disks [10, 13, 46]. However, commercial products have not been adopting those approaches for three main reasons: 1) Limitations of the hardware technology; 2) The continuous growth in CPU performance complied to the Moore's Law and Dennard scaling; 3) The lack of a general programming interface that leads to low abstraction level when handling hardware errors.

Now, modern PIM hardware integrate traditional DRAM dies and logic cells in the same chip area with the Through-Silicon Via (TSV), forming a 3D-stacked memory with a high degree of parallelism. Therefore, modern PIM can leverage current memory protocols to handle hardware errors. Current GPUs already embed the emerging 3D-stacked memories, such as the Hybrid Memory Cube (HMC) [23] and the High Bandwidth Memory (HBM) [30]. However, there has not been any in-depth study of query processing on PIM with Single Instruction Multiple Data (SIMD) support.

In this experimental paper, we detail the implementation of five major query operators into a simulator of PIM hardware: selection, projection, aggregation, sorting, and join (hash join, sort-merge join, and nested loop join). In particular, we present a new SIMD sorting algorithm that requires fewer memory instructions compared to the state of the art [21]. For each operator, we gauge the latency and energy spend to process TPC-H and Zipf distribution datasets. We evaluate the high levels of parallelism and data access when using AVX512 extensions from x86 processors, compared to modern PIM architectures with SIMD support on registers of 256-bytes wide. To the best of our knowledge,

VLDB 2019

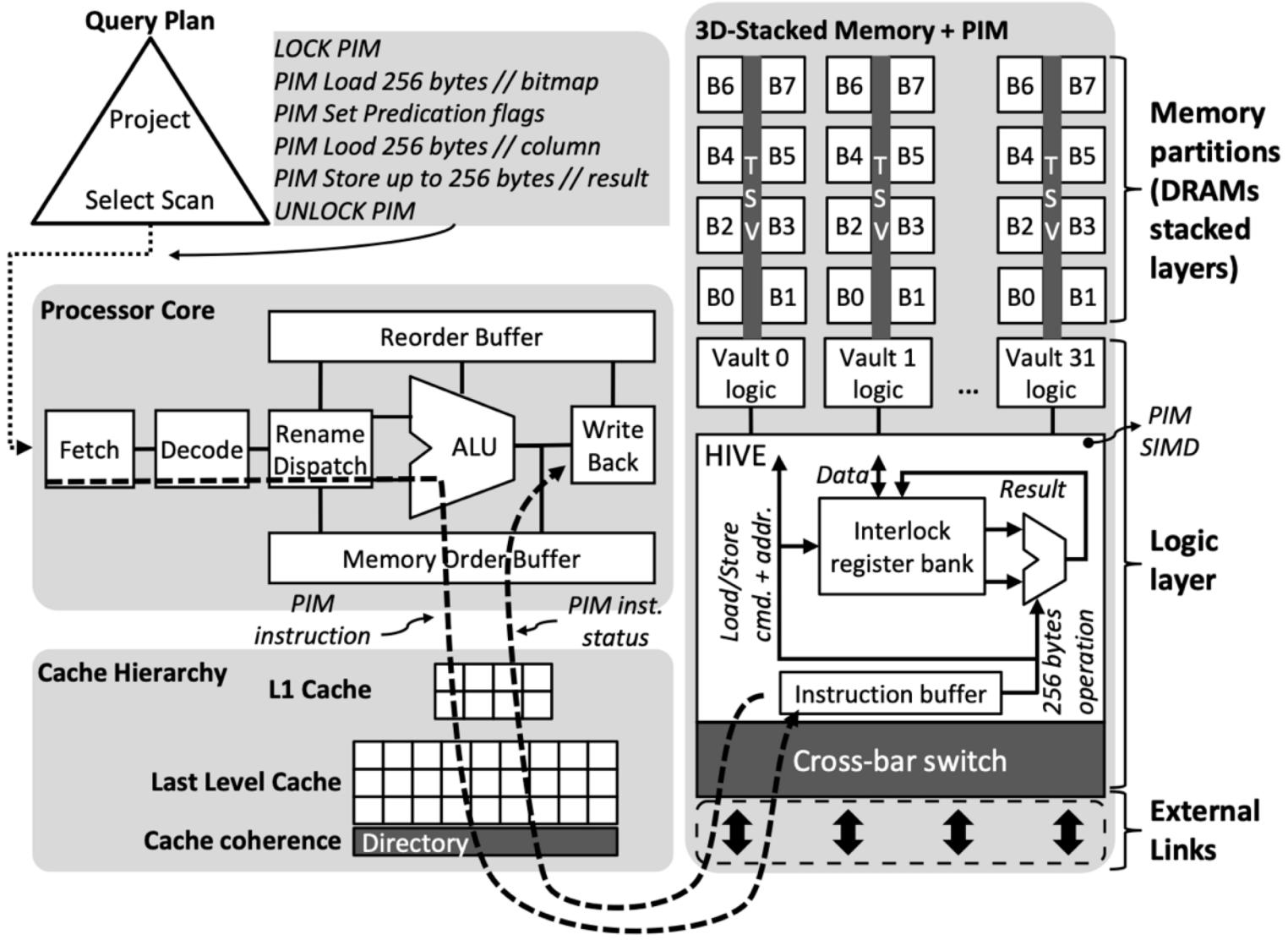
Previous NDP for Databases

Previous NDP-DB: Active disk, Intelligent disk, smart SSD

No commercial adoption of previous work

- Limitations of hardware technology
  => HBM and HMC
- Continuous growth in CPU performance
  => Moore's law is slowing down
- Lack of general programming interface
  => SIMD

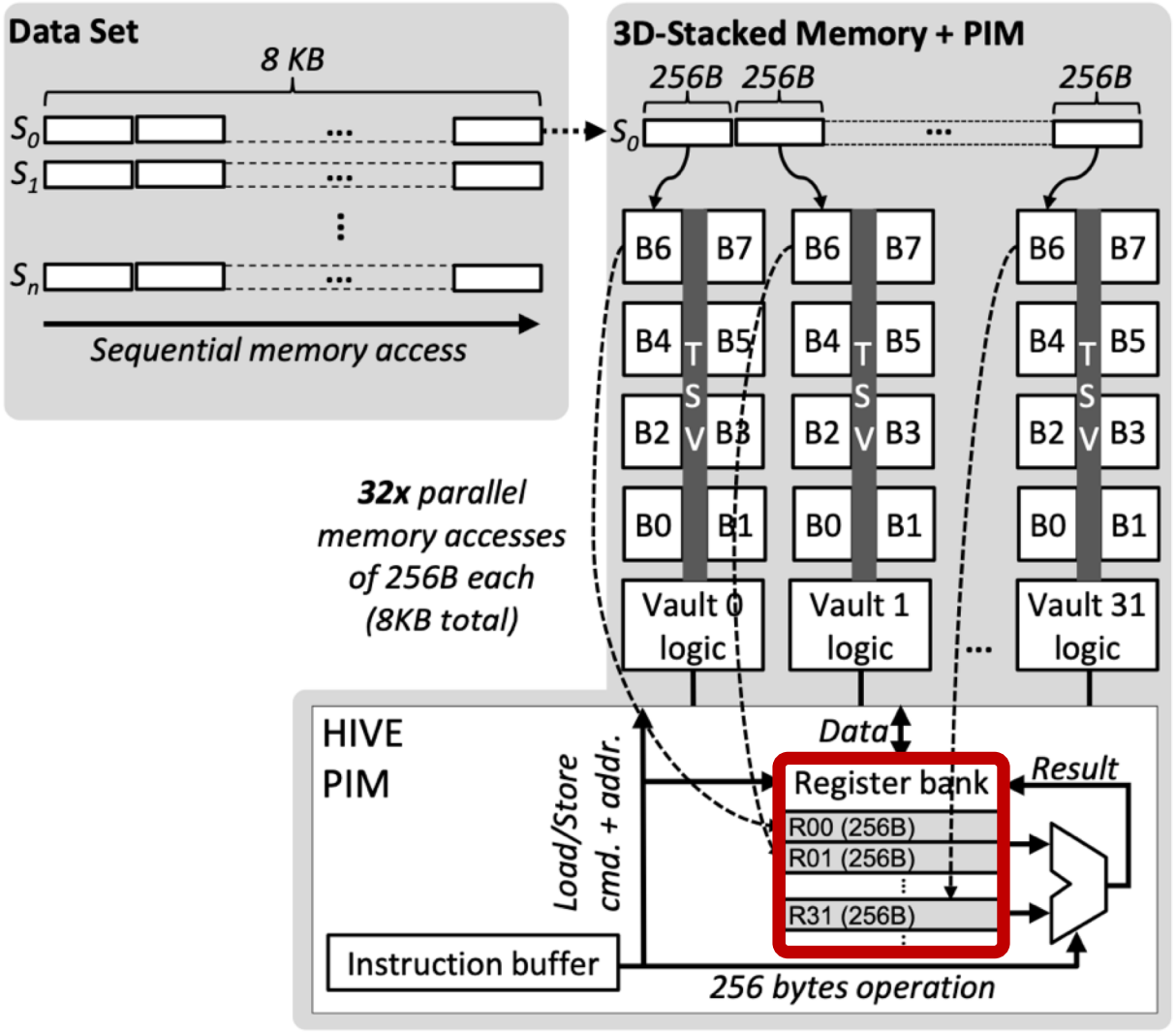
PIM-256B Architecture



- 32 vaults
- 8 DRAM banks per vault
- 256B per DRAM bank row accesses
- 512 parallel requests
- Bandwidth: 320 GB/s

- Coherence between PIM and cache?

PIM-256B Architecture



C Language

```
algebra.select (data[], filter)

for (int i = 0; i < size; i++)
    bitmap[i] = (data[i] < filter)
```

PIM-Selection Assembly Like

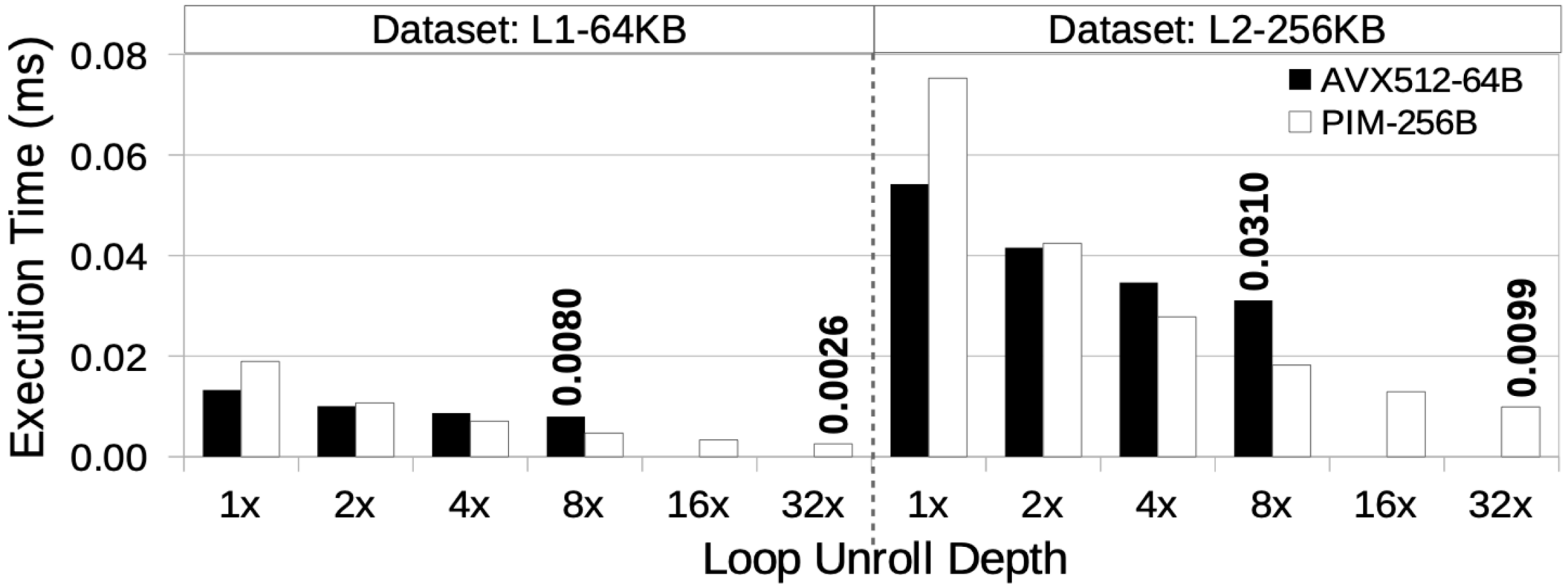
```
LOOP:                              algebra.select
  ...
  PIM_LD          V0 <-  data[i]
  PIM+SIMD_CMP    V0 <-  V0, VF
  PIM_ST          V0 ->  bitmap[i]
  ...
J LOOP
```

Loop Unrolling

```
int x;
for (x = 0; x < 100; x++)
{
        delete(x);
}
```

```
int x;
for (x = 0; x < 100; x += 5 )
{
        delete(x);
        delete(x + 1);
        delete(x + 2);
        delete(x + 3);
        delete(x + 4);
}
```

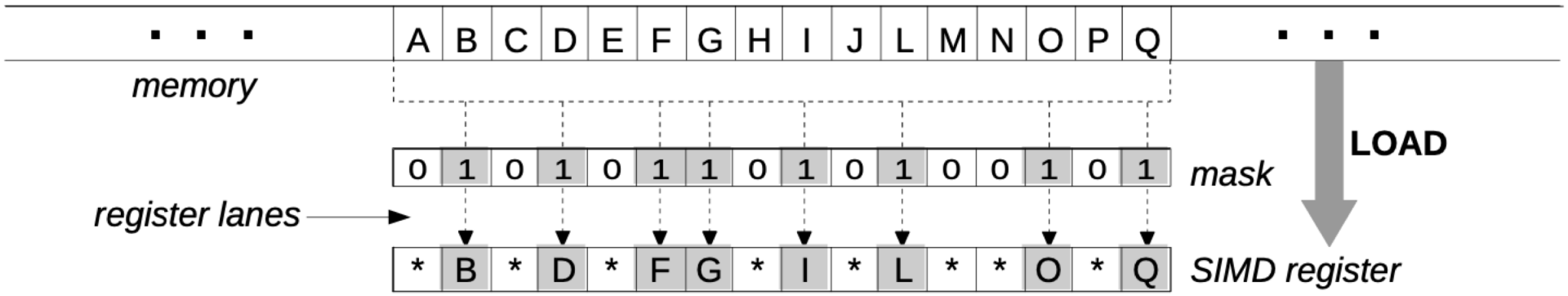
Benefits of PIM Processing (Selection)



"In this paper, we are using only a **single thread** to execute the operators on both systems …"

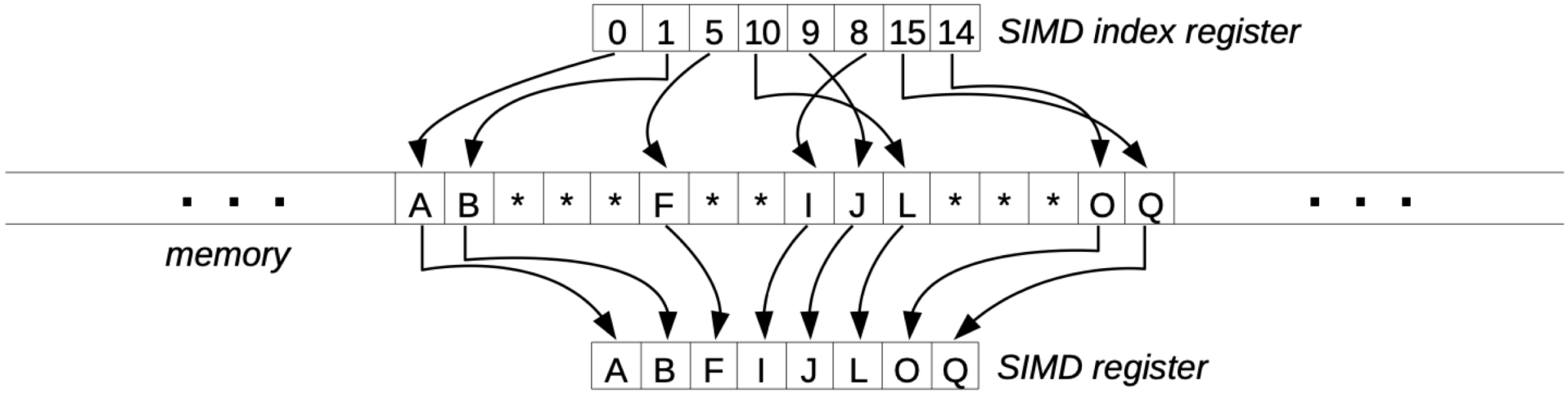
Selection

Bitmask



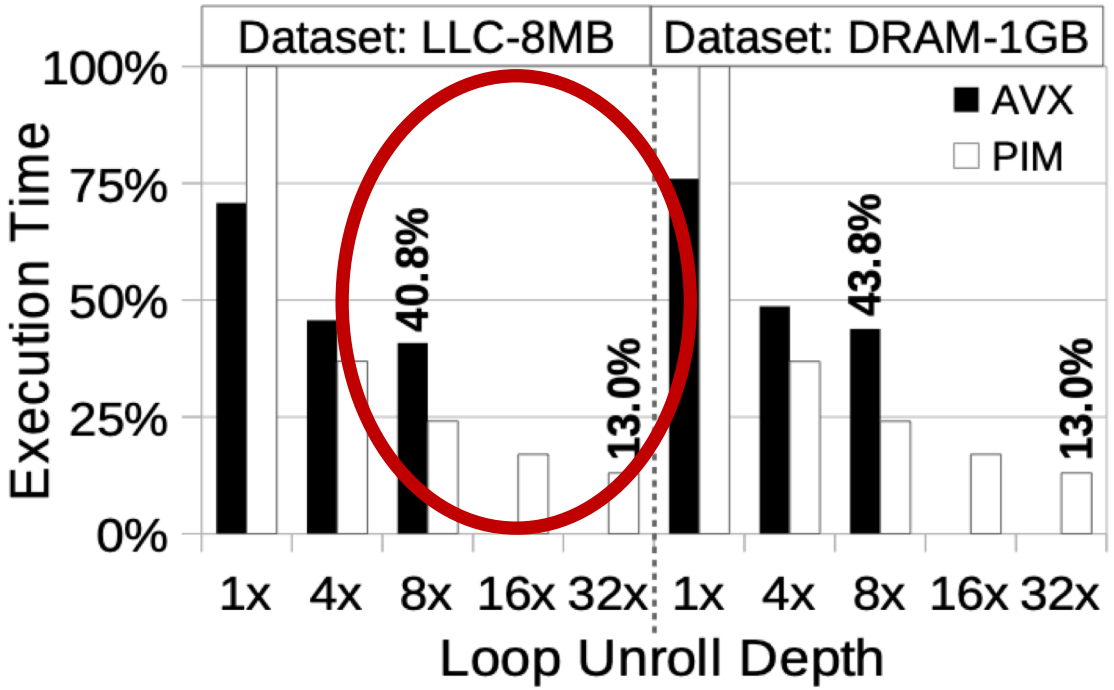
(a) Selective Load (Selection).

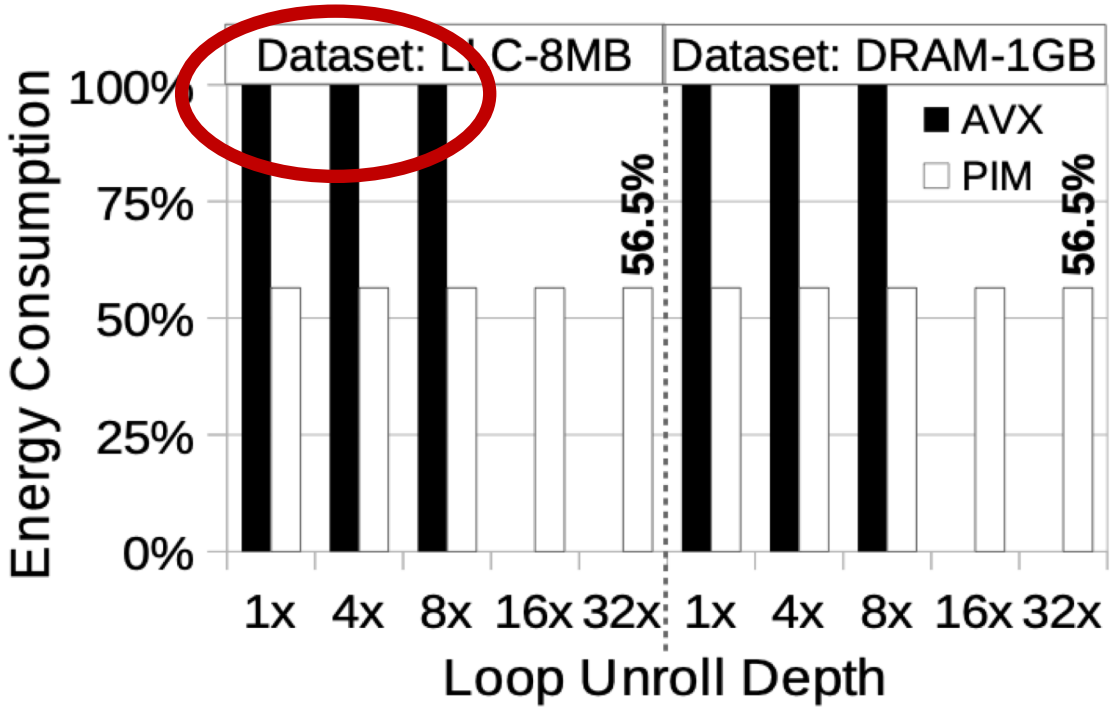
Index



(a) *Gather* Instruction.

Selection Evaluation

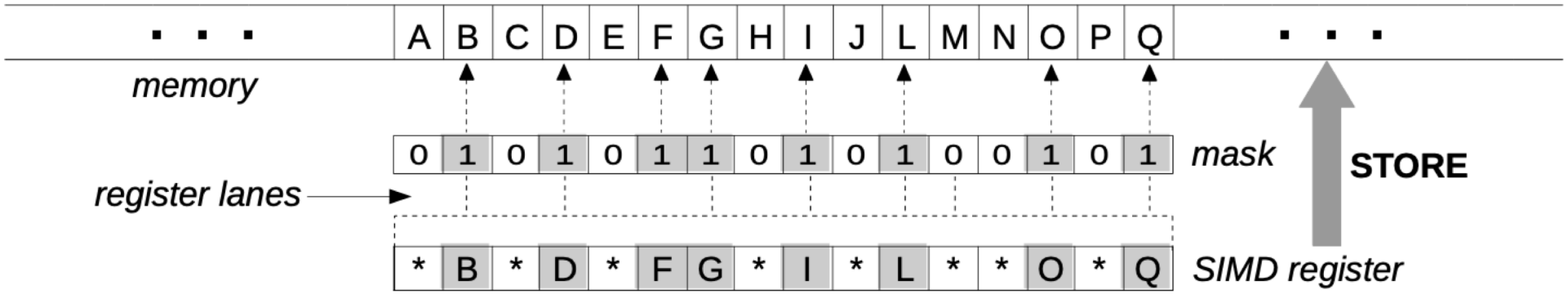




- PIM is 3x faster than AVX512
- PIM uses 45% less energy than AVX512

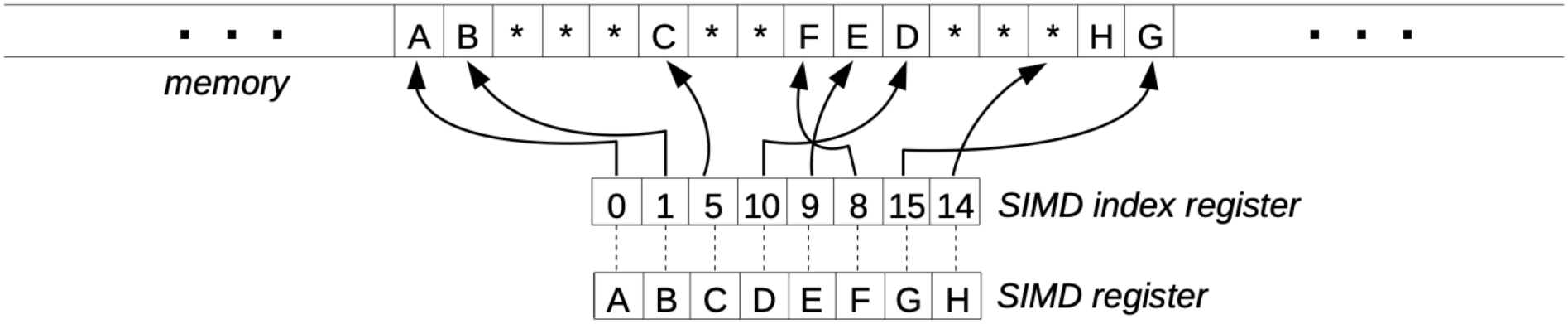
Projection

Bitmask



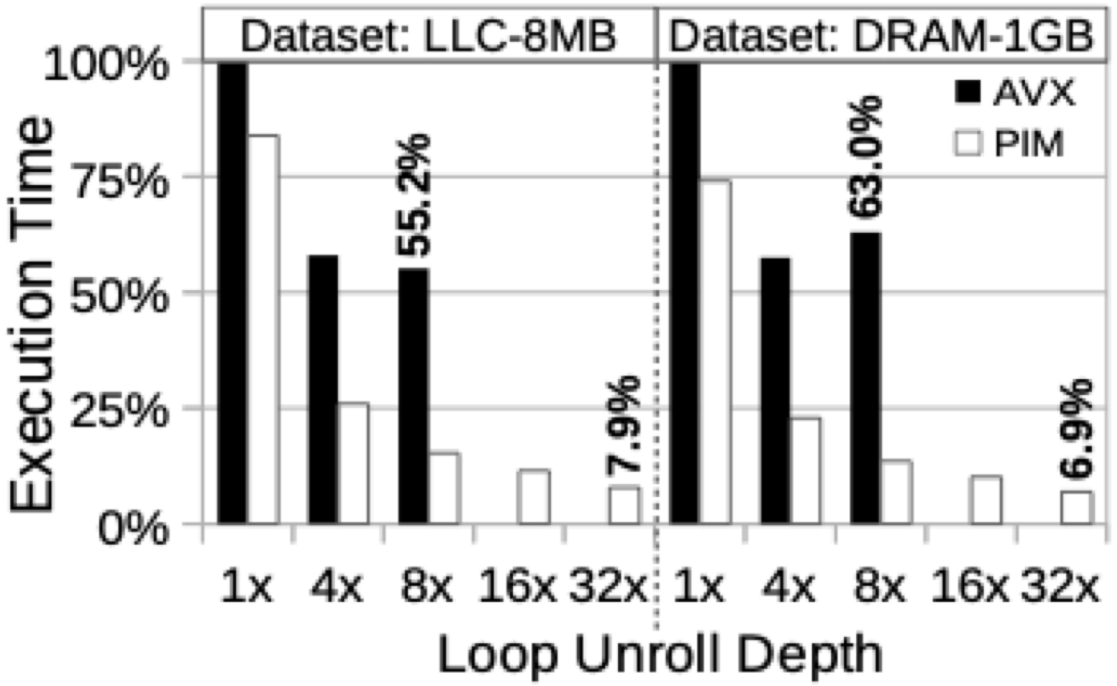
(b) Selective Store (Projection).

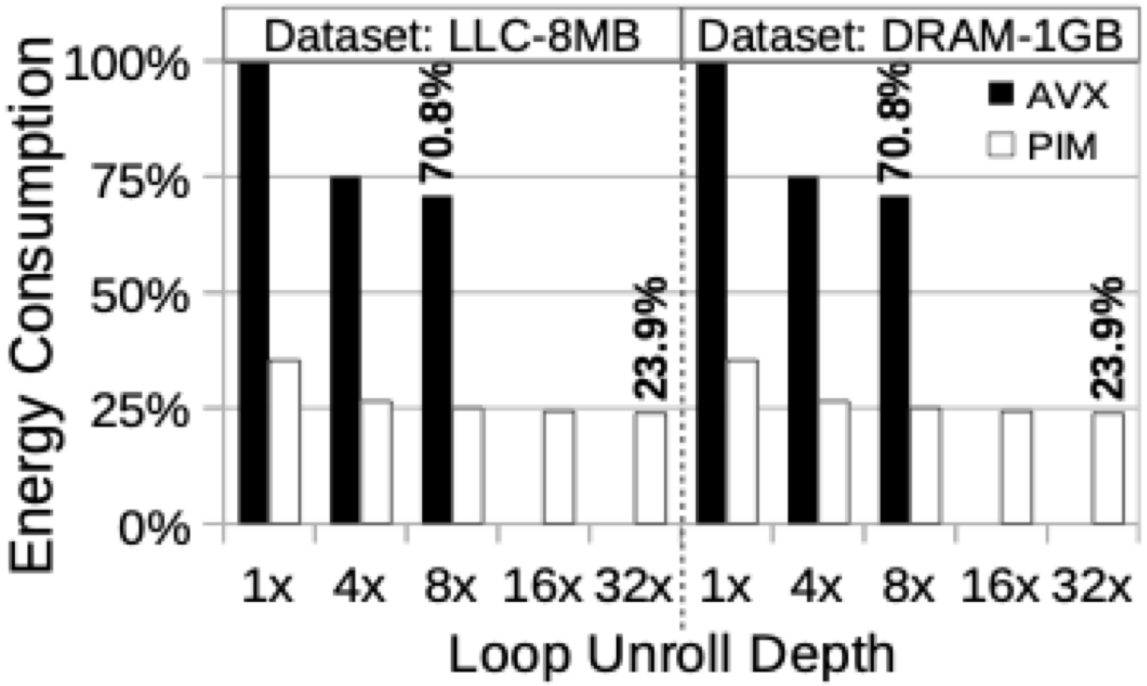
Index



(b) *Scatter* Instruction.

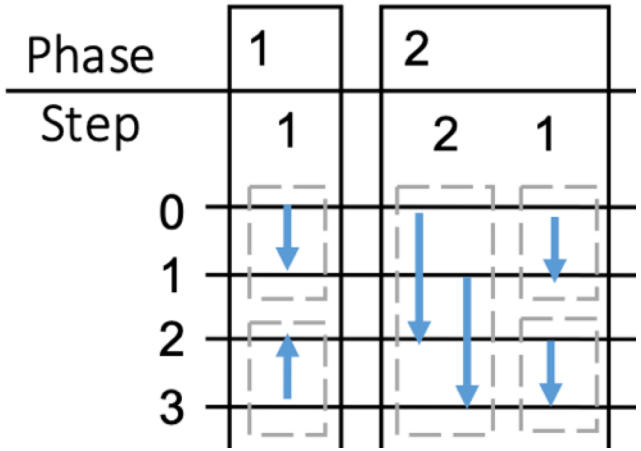
Projection Evaluation



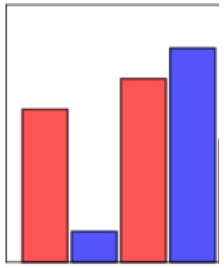


- PIM can be 10x faster than AVX512
- PIM reduces energy consumption by 3x

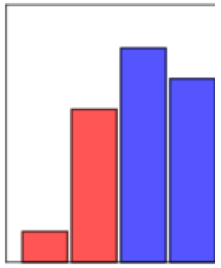
Bitonic Merge Sort



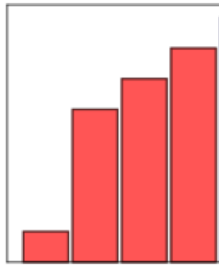
- Merge ascending array with descending array



Unsorted Input

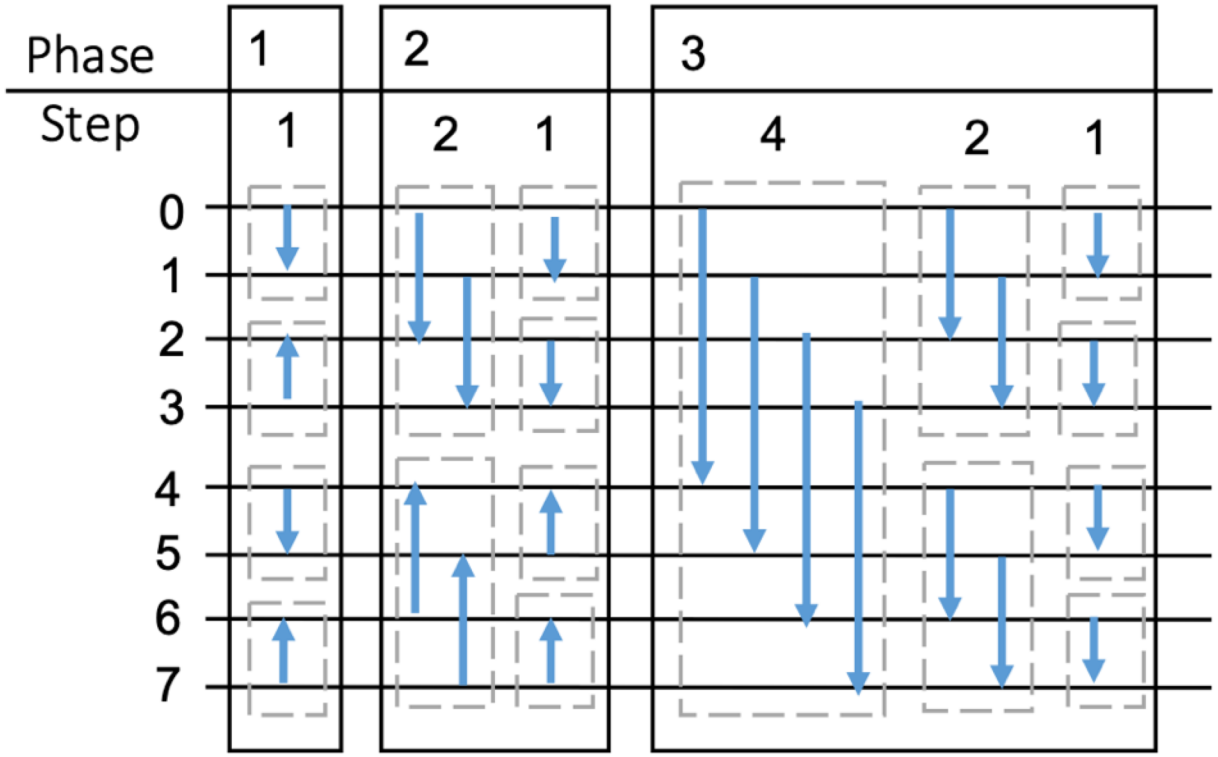


After Phase 1

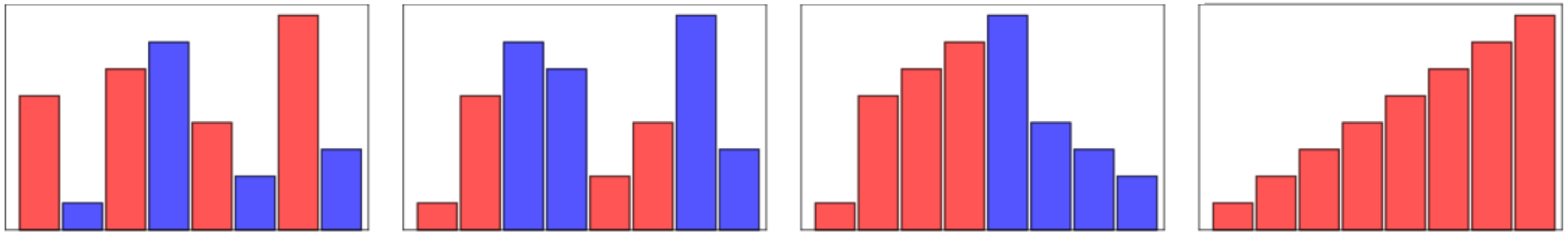


After Phase 2

Bitonic Merge Sort

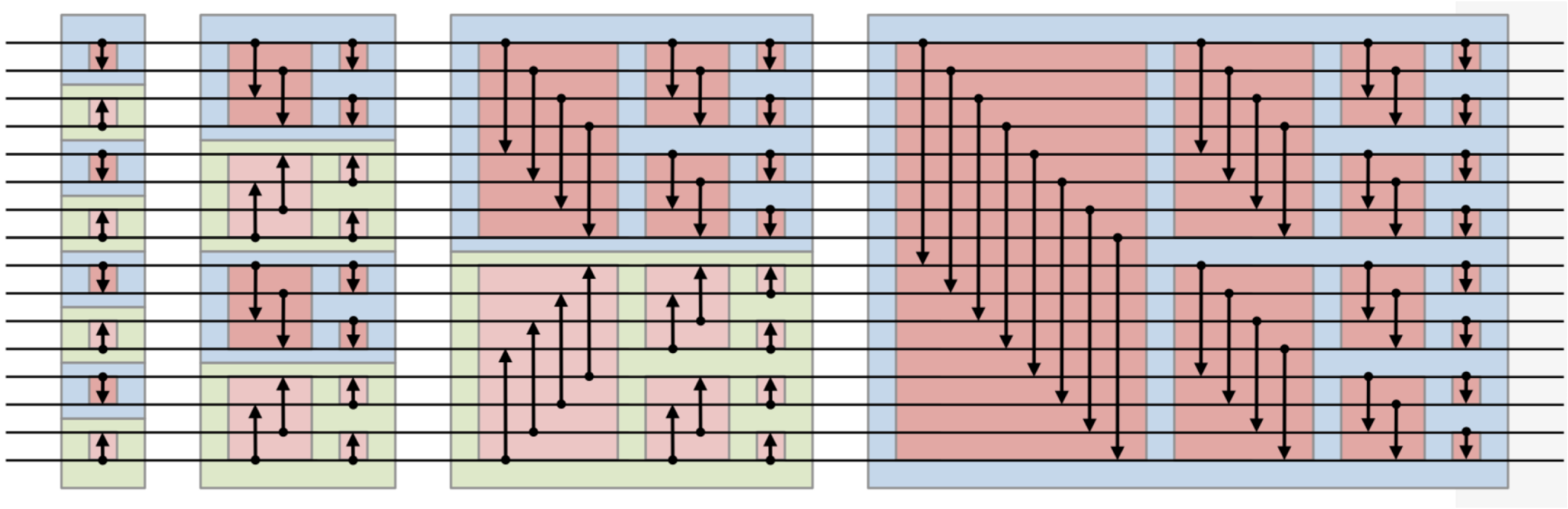


- Merge ascending array with descending array



Unsorted Input | After Phase 1 | After Phase 2 | After Phase 3

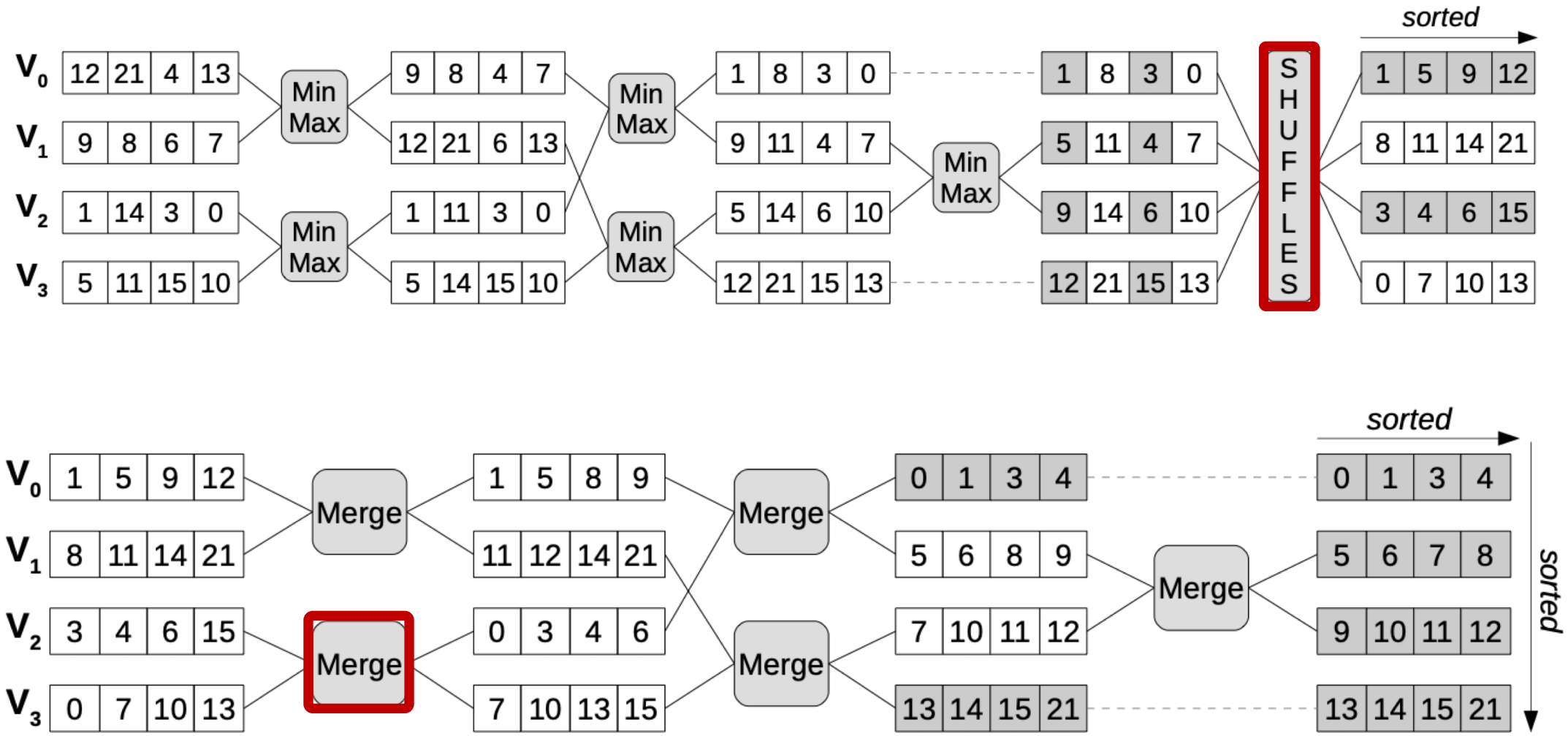
Bitonic Merge Sort



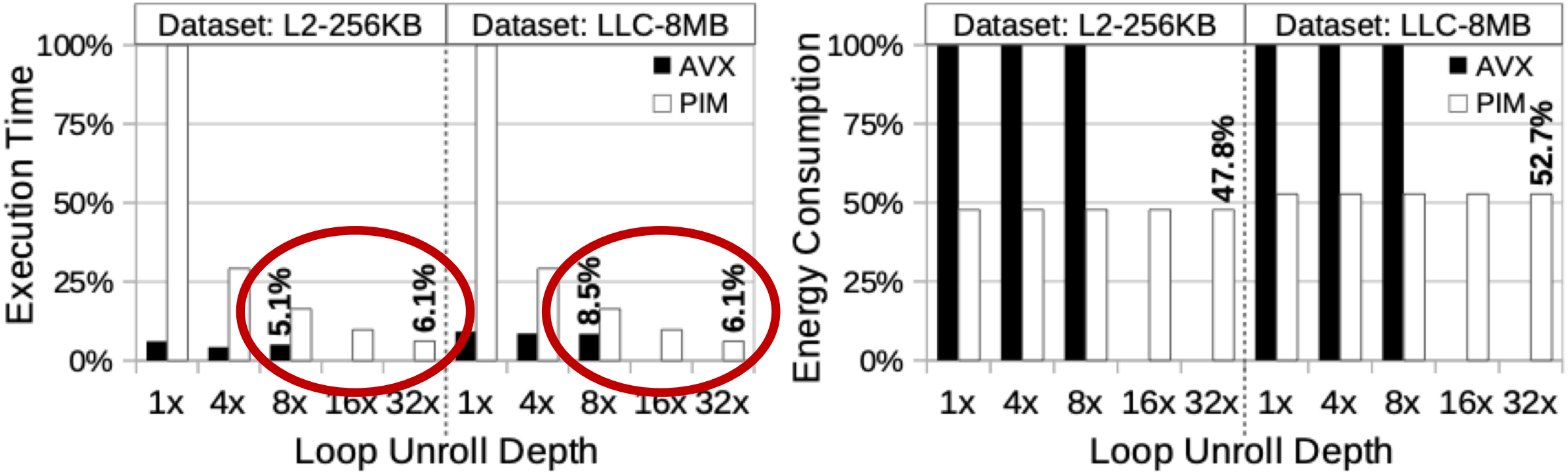
Comparators: $O(n \log^2 n)$

Runtime: $O(\log^2 n)$

SIMD-Based Bitonic Sorting

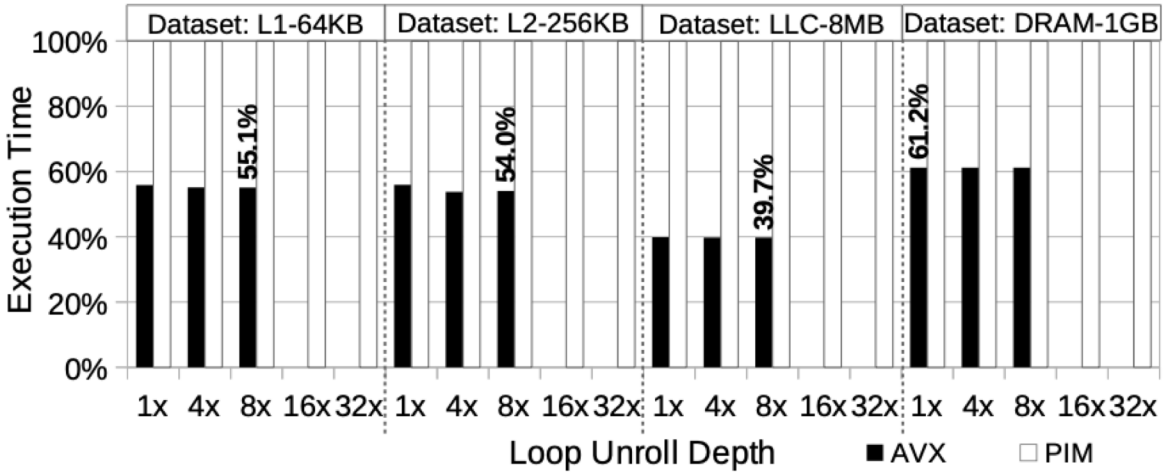


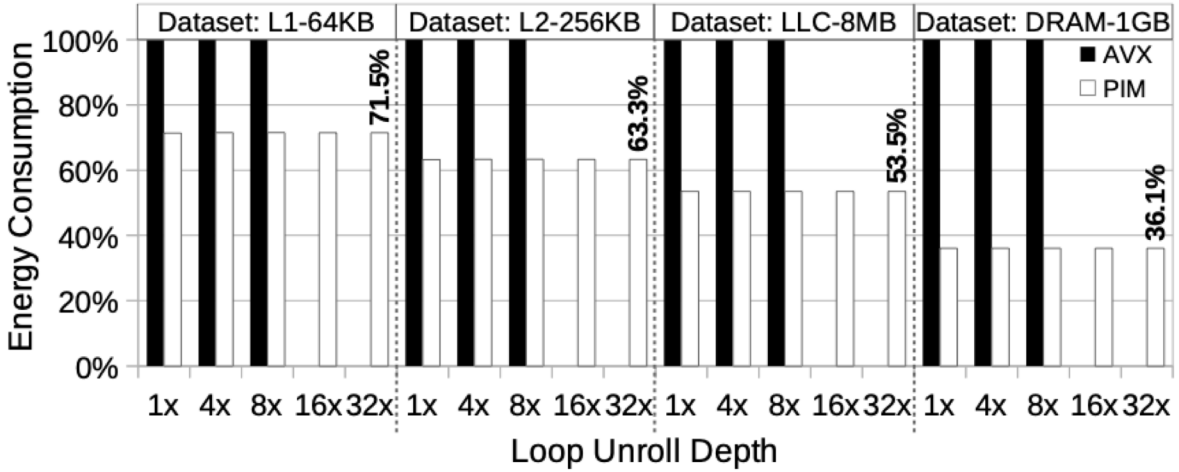
Nested Loop Join (NLJ)



- AVX outperforms PIM when inner relation fits in cache
- PIM reduces energy by 2x

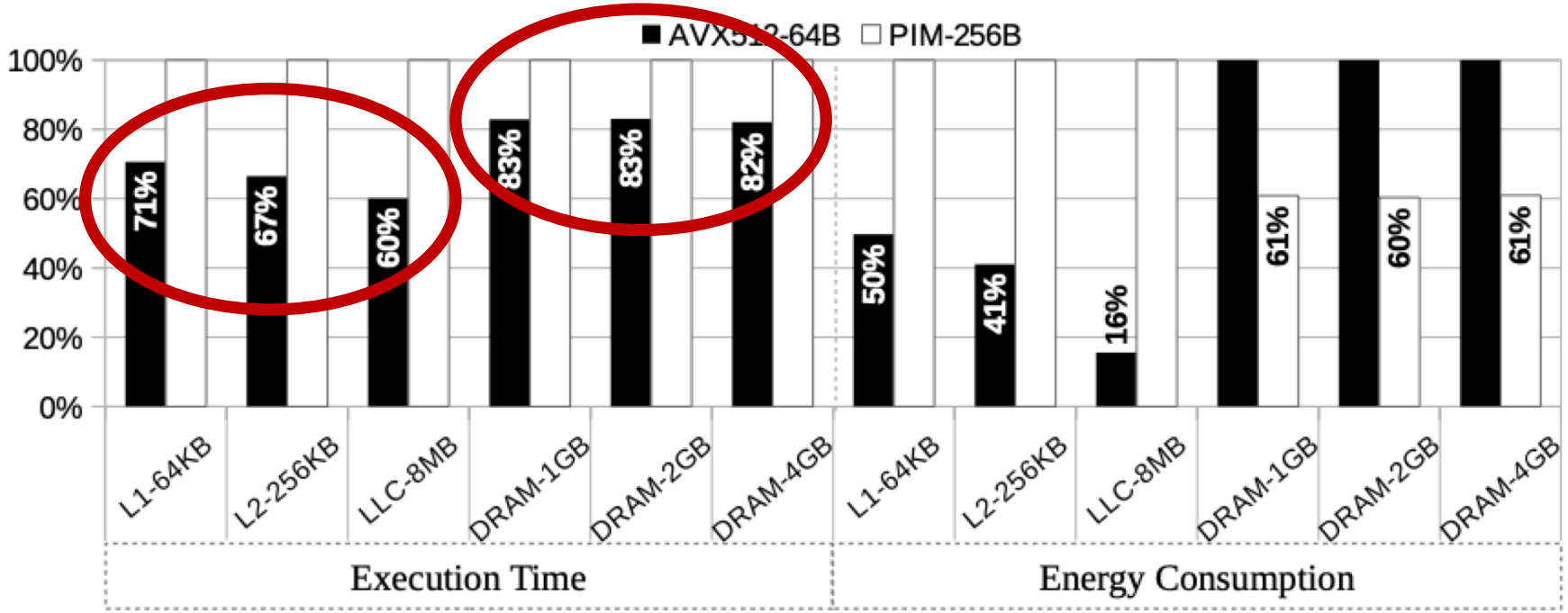
Hash Join





- PIM performs worse than AVX due to excessive random accesses
- PIM reduces energy (from 30% to 3x depending on the dataset size)

Sort-Merge Join



Unroll depth = 8x

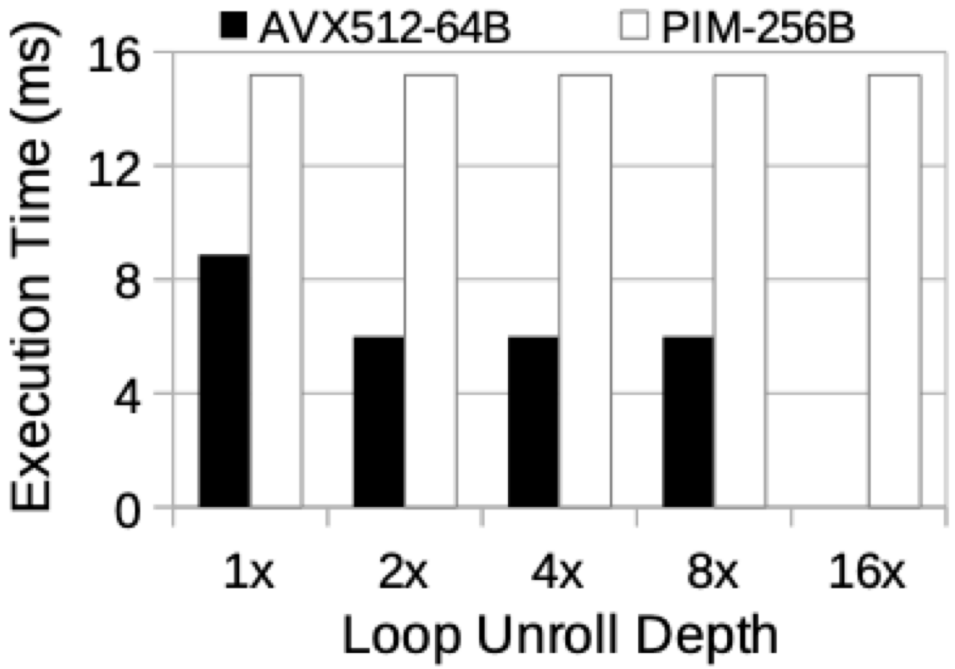
AVX outperforms PIM

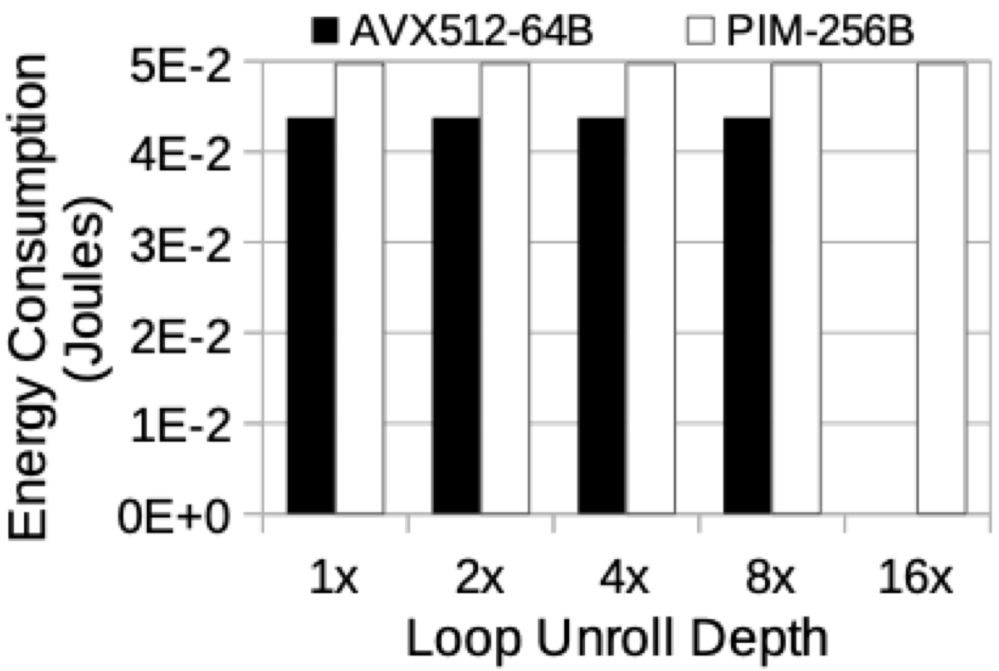
Aggregation – Query 1

```
SELECT
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc, count(*) as count_order
FROM
        lineitem
WHERE
        l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY
        l_returnflag, l_linestatus
ORDER BY
        l_returnflag, l_linestatus;
```

Aggregation with group by

Aggregation – Query 1 Evaluation

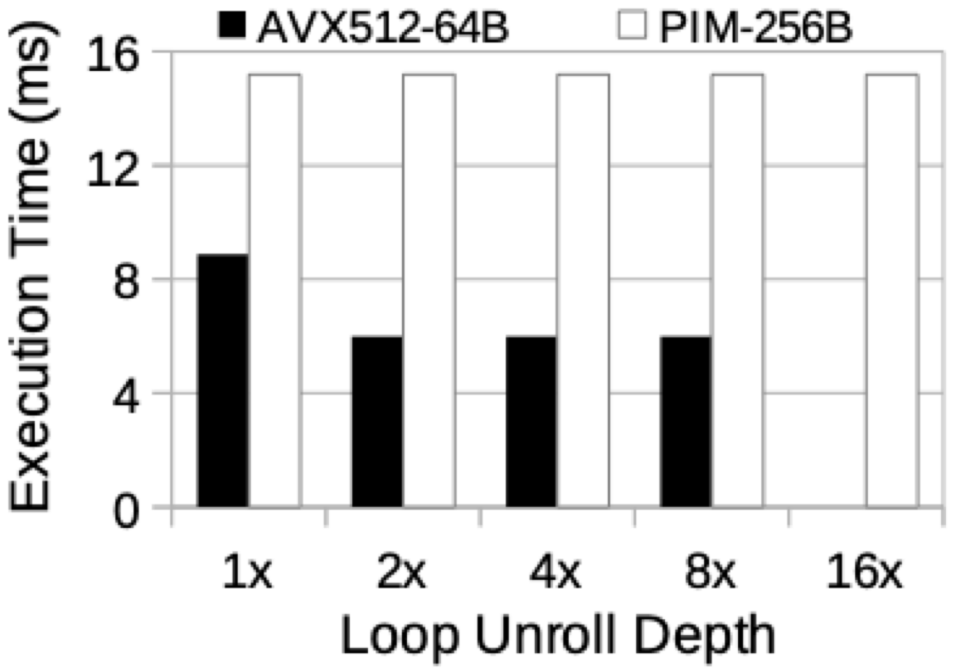


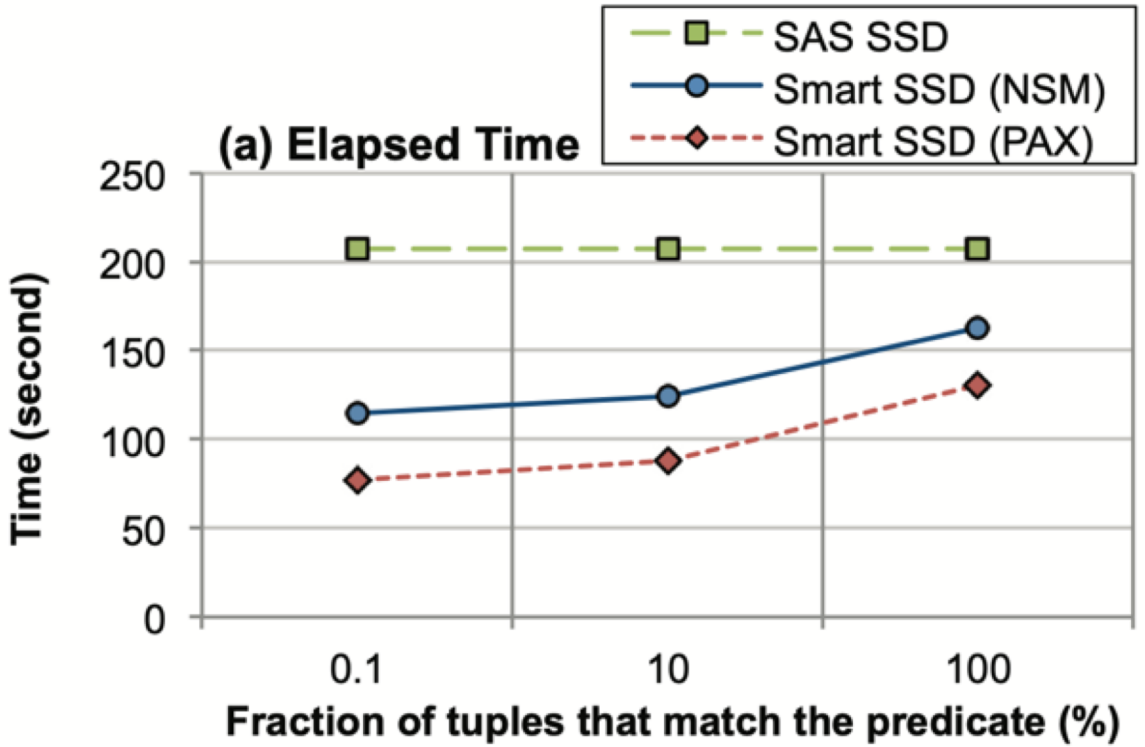


PIM worse than AVX due to random accesses to hash table

Why scatter to hash table?

Aggregation – PIM vs Smart SSD





Solutions to improve aggregation performance in PIM?

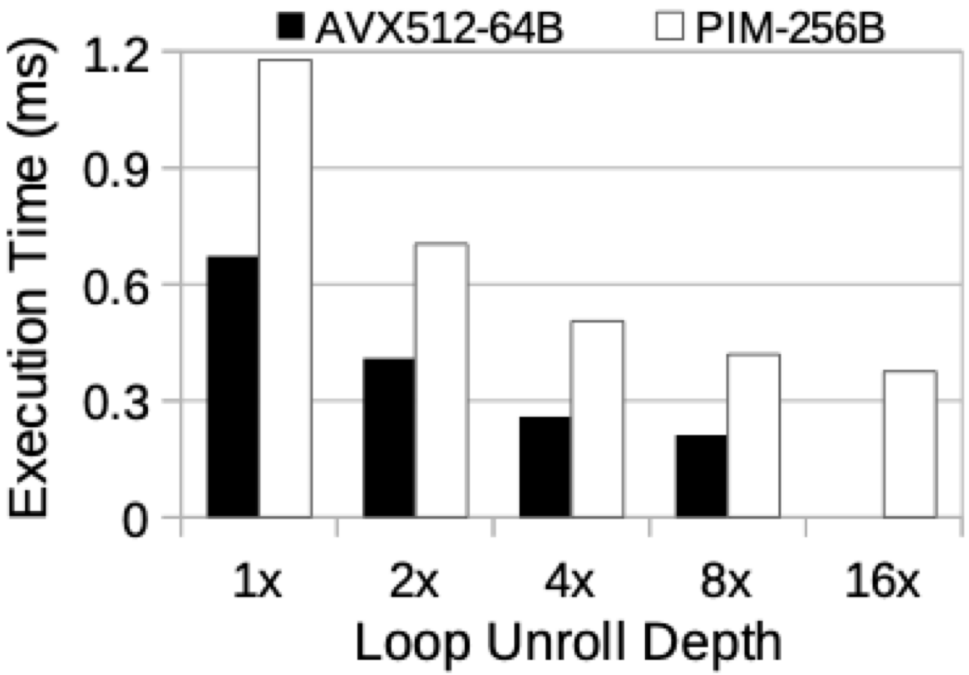
Aggregation – Query 3

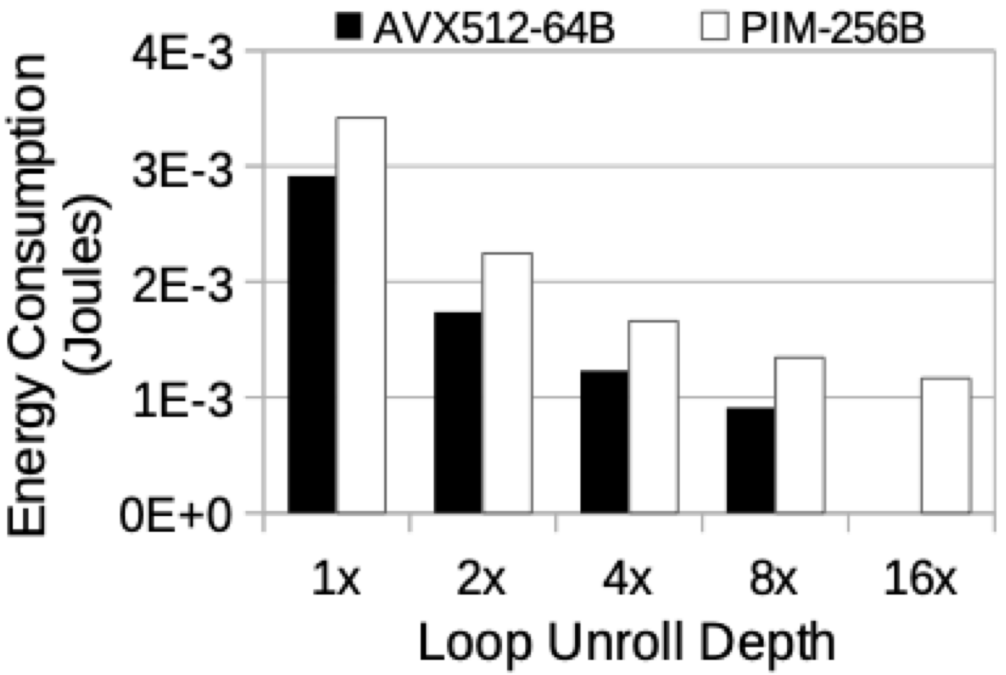
```
SELECT
        l_orderkey,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate,
        o_shippriority
FROM
        customer,
        orders,
        lineitem
WHERE
        c_mktsegment = 'BUILDING'
        AND c_custkey = o_custkey
        AND l_orderkey = o_orderkey
        AND o_orderdate < date '1995-03-15'
        AND l_shipdate > date '1995-03-15'
GROUP BY
        l_orderkey,
        o_orderdate,
        o_shippriority
ORDER BY
        revenue desc,
        o_orderdate
LIMIT 20;
```

Join

Aggregation with group by

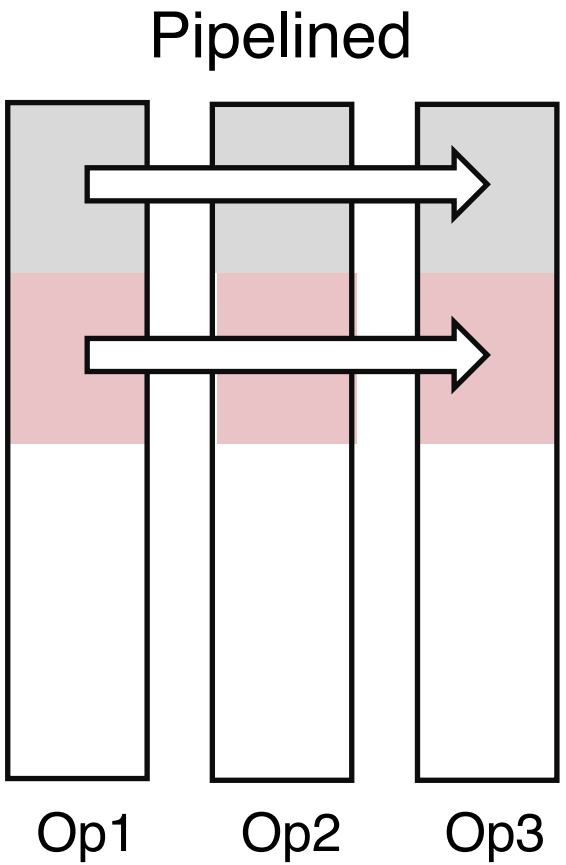
Aggregation – Query 3 Evaluation

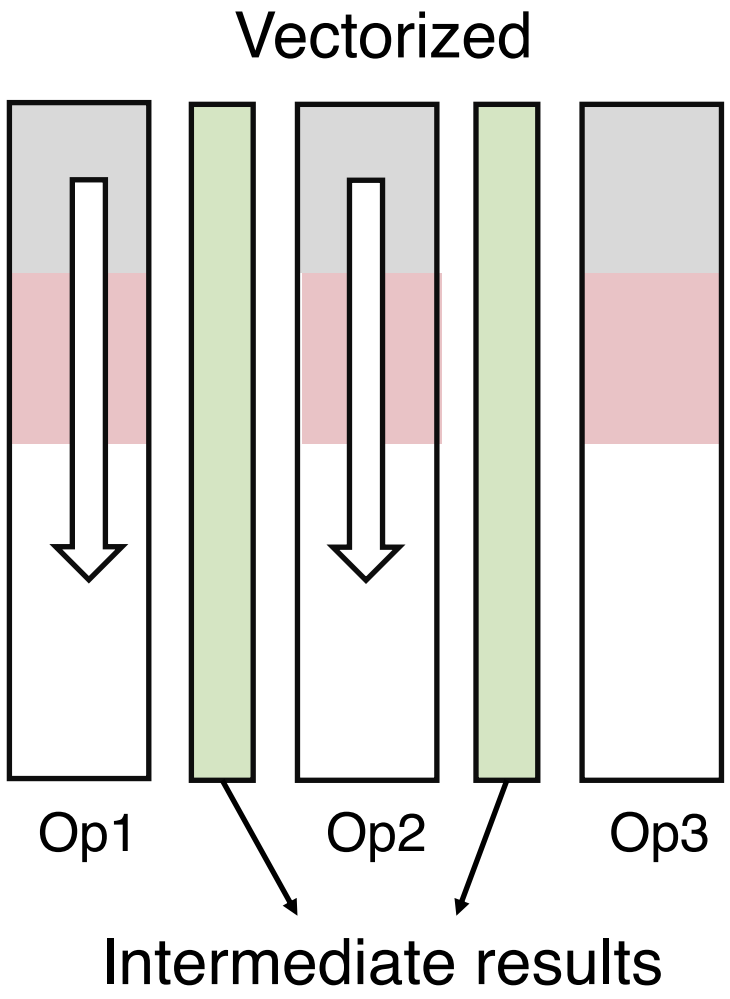




- Number of entries in hash table: a few hundreds (fit in L2)
- AVX outperforms PIM

Pipelined vs. Vectorized

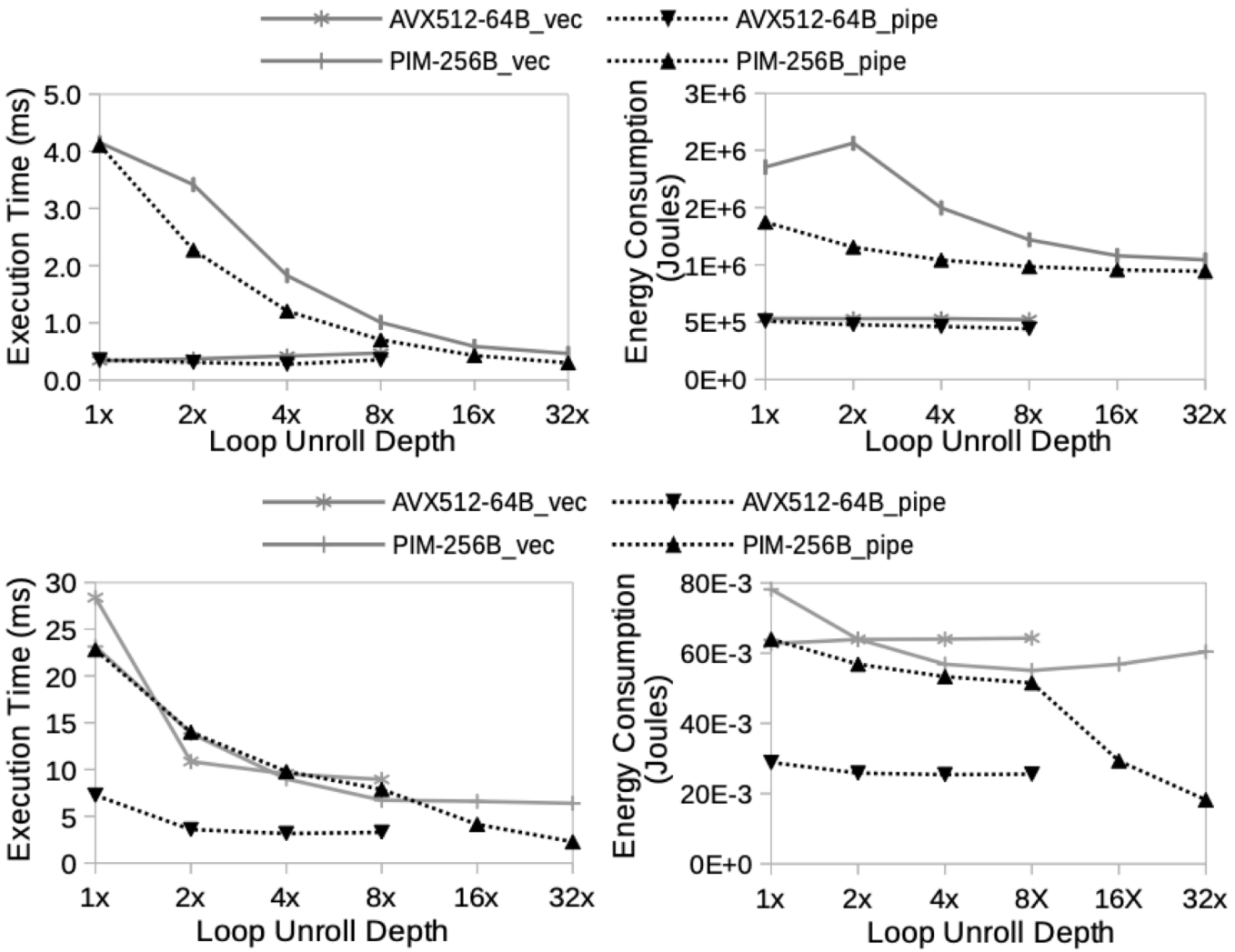




Pipelined vs. Vectorized – Evaluation

TPC-H Q3 selection followed by building

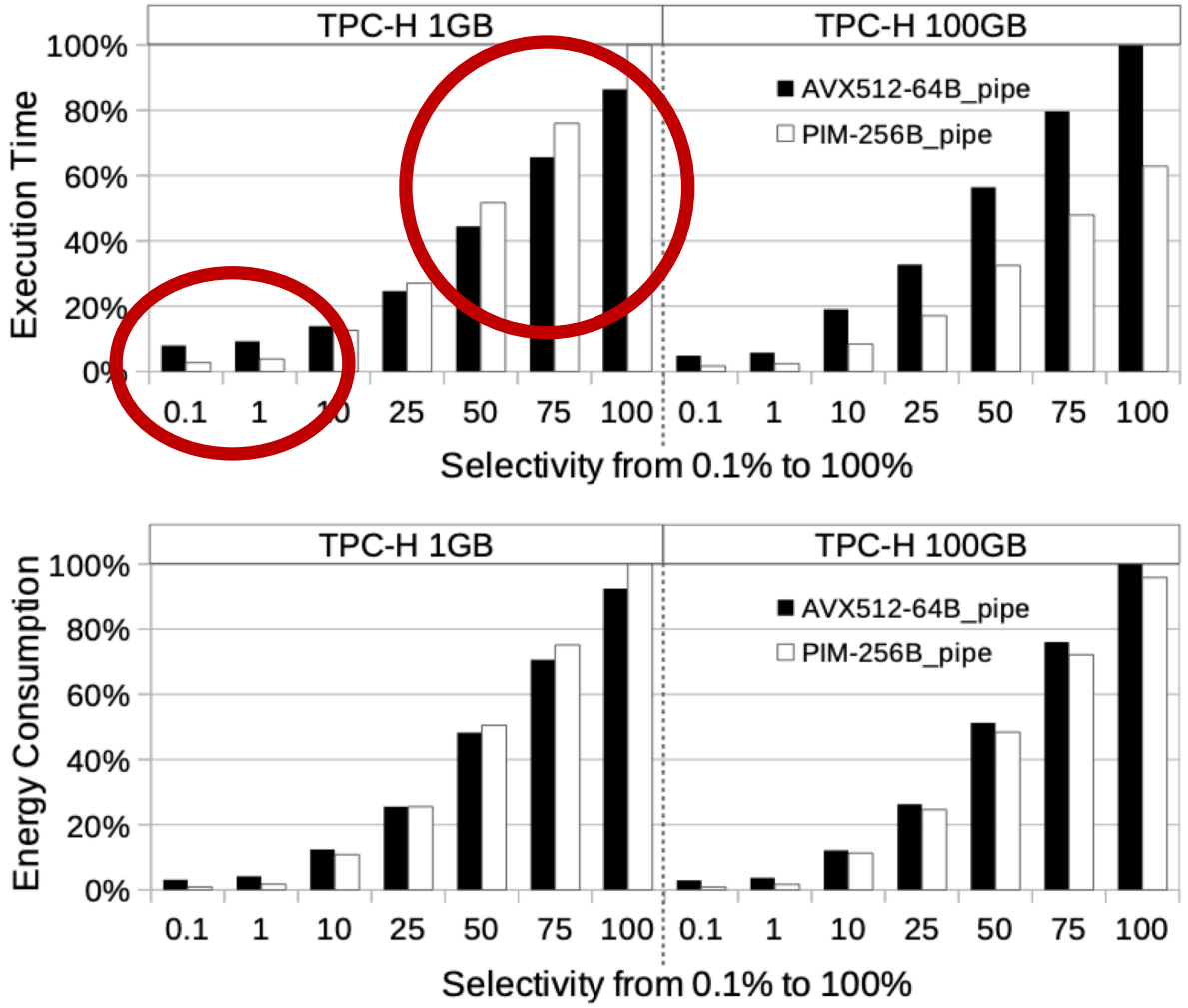
TPC-H Q1 selection followed by aggregation



Selectivity

TPC-H Query 3, pipelined

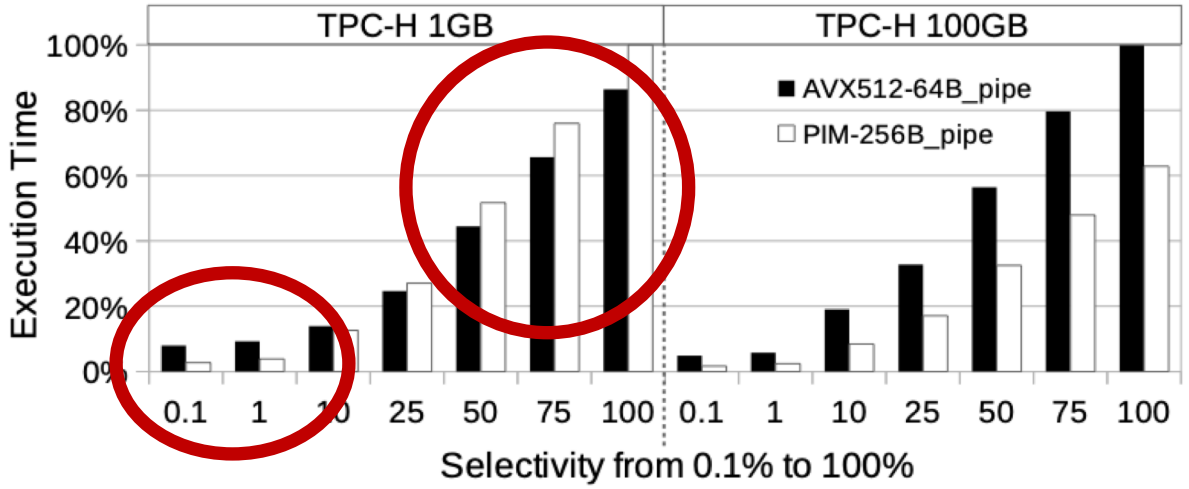
Selectivity on `c_mktsegment` ranges from 0.1% to 100%

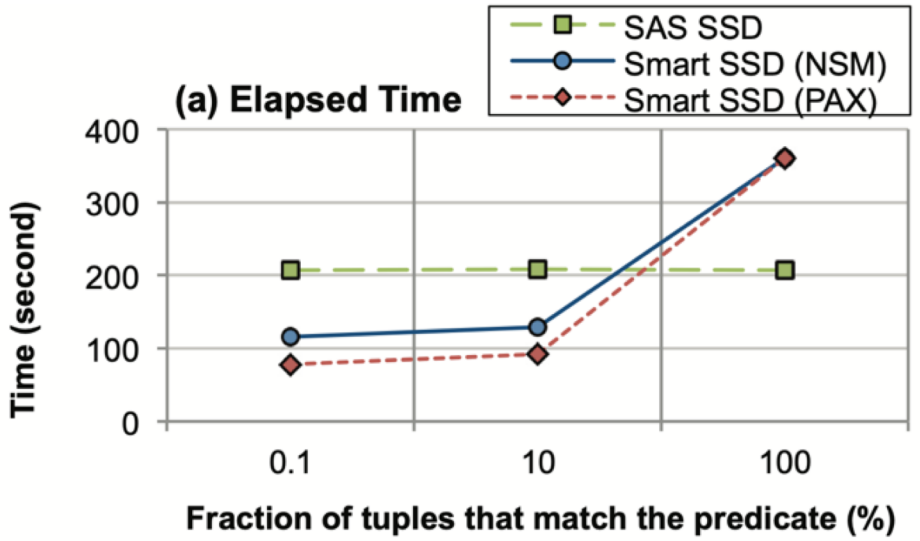


Selectivity

TPC-H Query 3, pipelined

Selectivity on `c_mktsegment` ranges from 0.1% to 100%

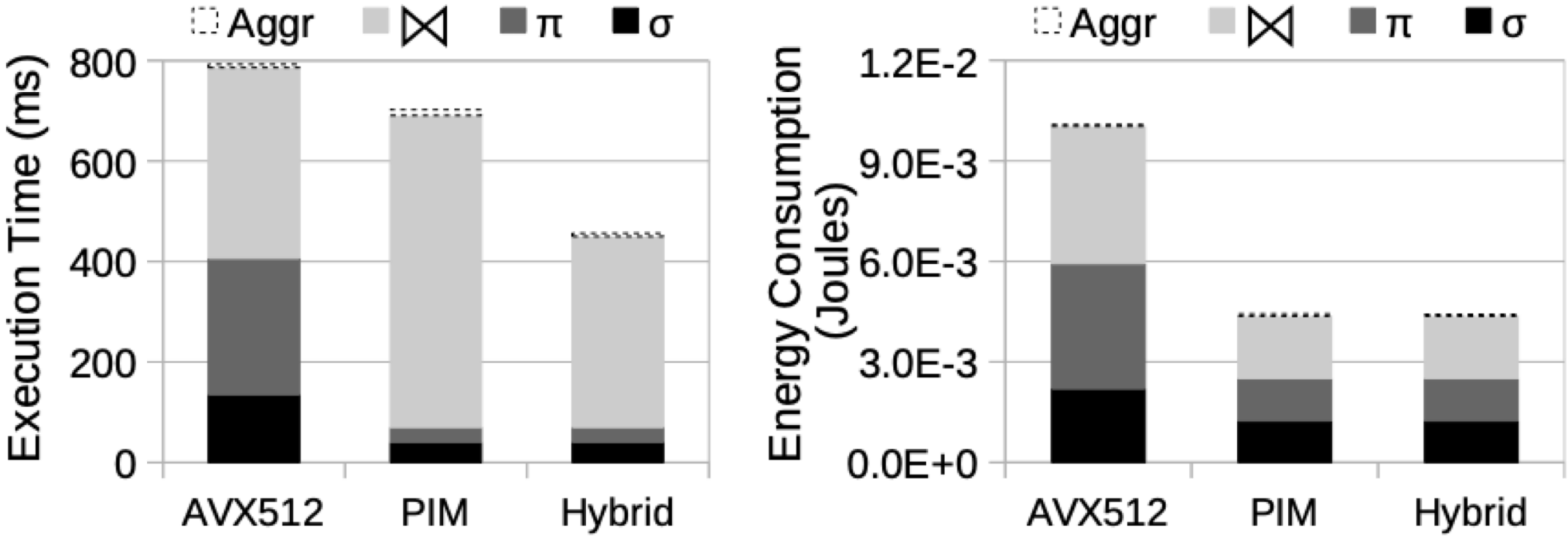




PIM vs. AVX512

Operator		Dataset **Fit in cache?**	Performance **Metrics**	Processing **Architectures**
Selection		no/yes	time/energy	**PIM**
Projection		no/yes	time/energy	**PIM**
Join	Nested Loop	L1/L2	time	**AVX512**
		LLC	time	**PIM**
		yes	energy	**PIM**
	Hash Join	no/yes	time	**AVX512**
		no/yes	energy	**PIM**
	Sort Merge	no/yes	time	**AVX512**
		yes	energy	**AVX512**
		no	energy	**PIM**
Aggregation		no/yes	time/energy	**AVX512**

Hybrid Execution



Hybrid query plan is 35% faster than PIM and 45% faster than AVX512

Summary

Operator		Dataset **Fit in cache?**	Performance **Metrics**	Processing **Architectures**
Selection		no/yes	time/energy	**PIM**
Projection		no/yes	time/energy	**PIM**
Join	Nested Loop	L1/L2	time	**AVX512**
		LLC	time	**PIM**
		yes	energy	**PIM**
	Hash Join	no/yes	time	**AVX512**
		no/yes	energy	**PIM**
	Sort Merge	no/yes	time	**AVX512**
		yes	energy	**AVX512**
		no	energy	**PIM**
Aggregation		no/yes	time/energy	**AVX512**

HMC Today?

Micron Announces Shift in High-Performance Memory Roadmap Strategy
By Andreas Schlapka - 2018-08-28

Now, as the volume projects that drove HMC success begin to reach maturity, **at Micron we are now turning our attention to the needs of the next generation of high-performance compute and networking solutions**. We continue to leverage our successful Graphics memory product line **(GDDR)** beyond the traditional graphics market and for extreme performance applications, Micron is investing in **HBM** (High-Bandwidth Memory) development programs which we recently made public.

HMC vs. HBM

	HBM2	HMC Gen3
Density	8 GB (4GB)	8 GB (4GB)
Bandwidth	256 GB/s	480 GB/s (320 GB/s)
IO	Parallel (1G – 2G), 8 channels, 128b per channel	SerDes (up to 30G), 4(2) links per HMC, 16 lanes/link
Package Type	Si-interposer	Discrete (SerDes)
Expansion Capability	No	Yes, via chaining
Memory Access	DDR	Packet based
Power	Lower	Higher
Memory Suppliers	SK Hynix and Samsung	Only Micron
Thermal Dissipation Req.	High (Logic + DRAM in single 2.5D ASIC package)	Lower (discrete ICs)
Ideal target markets	Graphics, Networking, Less frequently accessed memory, Small form-factor	High-performance Computing, Networking

PIM – Q/A

Why scatter to hash table in aggregation?

How to make a hardware design popular? (Wide application area and general purpose)

Current state of research

Combine these operators in a full-fledged database?

- IBM Netezza and Oracle Exadata

Concurrency control?

PIM in other memory technologies?

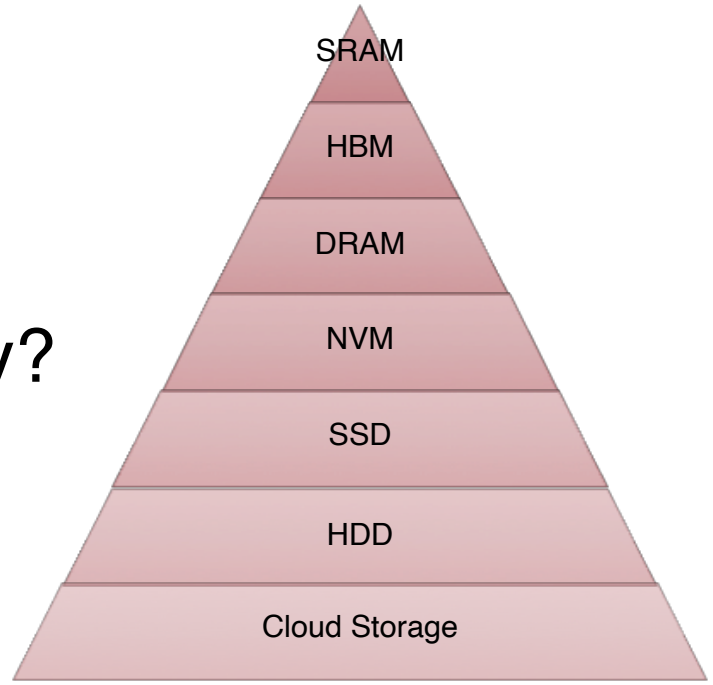
Cost analysis

Group Discussion

How to improve the performance of group-by aggregation in PIM?

How does smart SSD/memory affect transaction processing?

Looking at the bigger picture, where will PIM most likely to succeed in the storage hierarchy?



Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- **Deadline: Friday 11:59pm**

Submit review for

- The End of Slow Networks: It's Time for a Redesign
- [Optional] The End of a Myth: Distributed Transaction Can Scale