A Design Space Exploration and Evaluation for Main-Memory Hash Joins in Storage Class Memory

Wentao Huang*, Yunhong Ji†, Xuan Zhou‡, Bingsheng He*, Kian-Lee Tan*

*National University of Singapore
†Renmin University of China
‡East China Normal University
The Memory Scaling Challenge

• DRAM scaling wall
  - Scaling DRAM technologies to sub-20nm is challenging

• Growing imbalance between memory demand and supply
  - Increasing memory demand in datacenter applications
  - DRAM capacity per core dropping by 30% biannually

• Approaches to break the wall
  - Manufacturing advancement for scaling DRAM
  - Exploring alternative technologies for DRAM
The Memory Scaling Challenge

• DRAM scaling wall
  ○ Scaling DRAM technologies to sub-20nm is challenging
The Memory Scaling Challenge

• DRAM scaling wall
  o Scaling DRAM technologies to sub-20nm is challenging

• Growing imbalance between memory demand and supply
  o Increasing memory demand in datacenter applications
  o DRAM capacity per core dropping by 30% biannually
The Memory Scaling Challenge

• DRAM scaling wall
  o Scaling DRAM technologies to sub-20nm is challenging

• Growing imbalance between memory demand and supply
  o Increasing memory demand in datacenter applications
  o DRAM capacity per core dropping by 30% biannually

• Approaches to break the wall
  o Manufacturing advancement for scaling DRAM
  o Exploring alternative technologies for DRAM
Storage Class Memories (SCMs)

- Good scaling ability
- Large capacity
- Low cost per byte
- DRAM-scale latency/bandwidth
- Byte addressability
- Data persistence
Storage Class Memories (SCMs)

• Appealing characteristics in SCMs
  o Good scaling ability
  o Large capacity
  o Low cost per byte
Storage Class Memories (SCMs)

• Appealing characteristics in SCMs
  o Good scaling ability
  o Large capacity
  o Low cost per byte
  o DRAM-scale latency/bandwidth
  o Byte addressability
  o Data persistence
Storage Class Memories (SCMs)
Storage Class Memories (SCMs)

- NVDIMM-P
Storage Class Memories (SCMs)

- NVDIMM-P
Storage Class Memories (SCMs)

- NVDIMM-P primitives
  - $P_1$: access granularity mismatch
  - $P_2$: on-DIMM buffer/controller
  - $P_3$: read/write asymmetry
  - $P_4$: costly page fault handling
  - $P_5$: persistent instructions supported
Storage Class Memories (SCMs)

- NVDIMM-P primitives
Storage Class Memories (SCMs)

- NVDIMM-P primitives
  - \( p_1 \): access granularity mismatch
Storage Class Memories (SCMs)

- NVDIMM-P primitives
  - $p_1$: access granularity mismatch
Storage Class Memories (SCMs)

• NVDIMM-P primitives
  - $P_1$: access granularity mismatch
  - $P_2$: on-DIMM buffer/controller
Storage Class Memories (SCMs)

- NVDIMM-P primitives
  - $P_1$: access granularity mismatch
  - $P_2$: on-DIMM buffer/controller
  - $P_3$: read/write asymmetry
Storage Class Memories (SCMs)

• NVDIMM-P primitives
  o $P_1$: access granularity mismatch
  o $P_2$: on-DIMM buffer/controller
  o $P_3$: read/write asymmetry
  o $P_4$: costly page fault handling
  o $P_5$: persistent instructions supported
Main-Memory Hash Joins

Main design objective
- Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
  - Out-of-order execution (OOE)
  - Simultaneous multithreading (SMT)

- Partitioned hash join (PHJ)
  - Cache-sized scale join
Main-Memory Hash Joins

• Main design objective
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty
Main-Memory Hash Joins

- Main design objective
  - Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty

• Non-partitioned hash join (NPHJ)
Main-Memory Hash Joins

- Main design objective
  - Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
Main-Memory Hash Joins

- Main design objective
  - Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
Main-Memory Hash Joins

- Main design objective
  - Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty

• Non-partitioned hash join (NPHJ)
  o Out-of-order execution (OOE)
  o Simultaneous multithreading (SMT)
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty

• Non-partitioned hash join (NPHJ)
  o Out-of-order execution (OOE)
  o Simultaneous multithreading (SMT)

• Partitioned hash join (PHJ)
Main-Memory Hash Joins

- Main design objective
  - Minimize cache thrashing penalty

- Non-partitioned hash join (NPHJ)
  - Out-of-order execution (OOE)
  - Simultaneous multithreading (SMT)

- Partitioned hash join (PHJ)
Main-Memory Hash Joins

- **Main design objective**
  - Minimize cache thrashing penalty

- **Non-partitioned hash join (NPHJ)**
  - Out-of-order execution (OOE)
  - Simultaneous multithreading (SMT)

- **Partitioned hash join (PHJ)**
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty

• Non-partitioned hash join (NPHJ)
  o Out-of-order execution (OOE)
  o Simultaneous multithreading (SMT)

• Partitioned hash join (PHJ)
Main-Memory Hash Joins

• Main design objective
  o Minimize cache thrashing penalty

• Non-partitioned hash join (NPHJ)
  o Out-of-order execution (OOE)
  o Simultaneous multithreading (SMT)

• Partitioned hash join (PHJ)
  o Cache-sized scale join
An Ongoing Debate: PHJ vs. NPHJ
An Ongoing Debate: PHJ vs. NPHJ

Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs
SIGMOD 2011

▷ NPHJ > PHJ
An Ongoing Debate: PHJ vs. NPHJ
An Ongoing Debate: PHJ vs. NPHJ

Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs

Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware

An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory

SIGMOD 2016

▷ PHJ > NPHJ
An Ongoing Debate: PHJ vs. NPHJ

Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs

Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware

An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory

To Partition, or Not to Partition, That is the Join Question in a Real System

SIGMOD 2021

▷ NPHJ > PHJ
An Ongoing Debate: PHJ vs. NPHJ

- Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs
- Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware
- An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory
- To Partition, or Not to Partition, That is the Join Question in a Real System
- Evaluating In-Memory Hash-Joins on Persistent Memory

▷ NPHJ > PHJ

© Copyright National University of Singapore. All Rights Reserved.
An Ongoing Debate: PHJ vs. NPHJ

- Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs
- Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware
- An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory
- To Partition, or Not to Partition, That is the Join Question in a Real System
- Evaluating In-Memory Hash-Joins on Persistent Memory
- A Design Space Exploration and Evaluation for Main-Memory Hash Joins in Storage Class Memory

▶ PHJ > NPHJ
Exploration & Evaluation

- Experiment Setup
  - App-direct mode
  - Pure in-SCM join implementation (cache + SCM)

- Design space exploration
  - 23 × join implementation variants
  - 5 × NVDIMM-P primitives
  - 6 × scaling effects (e.g., thread scalability, prefetching)

- Comprehensive evaluation
  - 18 × synthesized workloads (e.g., zipfian, selectivity)
  - Real benchmarks (e.g., TPC-H)
Exploration & Evaluation

• Experiment Setup
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)

• Design space exploration
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)

• Design space exploration
  o 23× join implementation variants
Exploration & Evaluation

- **Experiment Setup**
  - App-direct mode
  - Pure in-SCM join implementation (cache + SCM)

- **Design space exploration**
  - 23× join implementation variants

### Taxonomy
<table>
<thead>
<tr>
<th>Non-Partitioned Hash Join</th>
<th>Join Notation</th>
<th>Partitioning</th>
<th>Hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPHJ-SC</td>
<td>NPHJ-LP</td>
<td></td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Linear Probing</td>
</tr>
<tr>
<td>SHRJL-SC</td>
<td>SHRJL-LP</td>
<td>Shared Partitioning (linked list)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>SHRJL-HM</td>
<td>Shared Partitioning (linked list)</td>
<td>Linear Chaining</td>
</tr>
<tr>
<td></td>
<td>SHRJL-HM</td>
<td>Shared Partitioning (contiguous memory)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>SHRJL-BC</td>
<td>Shared Partitioning (contiguous memory)</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td></td>
<td>SHRJL-SC</td>
<td>Shared Partitioning (contiguous memory)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>SHRJL-LP</td>
<td>Shared Partitioning (contiguous memory)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>SHRJL-HM</td>
<td>Shared Partitioning (contiguous memory)</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td>INDJL-SC</td>
<td>INDJL-LP</td>
<td>Independent Partitioning (linked list)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>INDJL-HM</td>
<td>Independent Partitioning (linked list)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>INDJL-BC</td>
<td>Independent Partitioning (linked list)</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td></td>
<td>INDJL-SC</td>
<td>Independent Partitioning (linked list)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>INDJL-LP</td>
<td>Independent Partitioning (linked list)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>INDJL-HM</td>
<td>Independent Partitioning (linked list)</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td></td>
<td>INDJL-BC</td>
<td>Independent Partitioning (linked list)</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td></td>
<td>INDJL-SC</td>
<td>Independent Partitioning (linked list)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>INDJL-LP</td>
<td>Independent Partitioning (linked list)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>INDJL-HM</td>
<td>Independent Partitioning (linked list)</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td></td>
<td>RDX-JL-BC</td>
<td>Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td></td>
<td>RDX-JL-SC</td>
<td>Radix Partitioning</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>RDX-JL-LP</td>
<td>Radix Partitioning</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>RDX-JL-HM</td>
<td>Radix Partitioning</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td></td>
<td>ASYM-JL-BC</td>
<td>Asymmetric Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td></td>
<td>ASYM-JL-SC</td>
<td>Asymmetric Radix Partitioning</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td></td>
<td>ASYM-JL-LP</td>
<td>Asymmetric Radix Partitioning</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td></td>
<td>ASYM-JL-HM</td>
<td>Asymmetric Radix Partitioning</td>
<td>Histogram Mechanism</td>
</tr>
</tbody>
</table>
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)

• Design space exploration
  o 23× join implementation variants
  o 5× NVDIMM-P primitives
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)

• Design space exploration
  o 23× join implementation variants
  o 5× NVDIMM-P primitives
  o 6× scaling effects (e.g., thread scalability, prefetching)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Join Notation</th>
<th>Partitioning</th>
<th>Hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Partitioned Hash Join</td>
<td></td>
<td>Separately Chaining</td>
<td></td>
</tr>
<tr>
<td>NPHJ-SC</td>
<td></td>
<td>Separate Chaining</td>
<td>Linear Probing</td>
</tr>
<tr>
<td>NPHJ-LP</td>
<td></td>
<td>Separate Chaining</td>
<td>Linear Probing</td>
</tr>
<tr>
<td>SHRJ-SC</td>
<td></td>
<td>Shared Partitioning (linked list)</td>
<td>Separate Chaining</td>
</tr>
<tr>
<td>SHRJ-LP</td>
<td></td>
<td>Shared Partitioning (linked list)</td>
<td>Linear Probing</td>
</tr>
<tr>
<td>SHRJ-HM</td>
<td></td>
<td>Shared Partitioning (linked list)</td>
<td>Histogram Mechanism</td>
</tr>
<tr>
<td>SHRc-BC</td>
<td></td>
<td>Separate Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>SHRc-SC</td>
<td></td>
<td>Separate Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>SHRc-LP</td>
<td></td>
<td>Separate Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>SHRc-HM</td>
<td></td>
<td>Separate Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>INDC-BC</td>
<td></td>
<td>Independent Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>INDC-SC</td>
<td></td>
<td>Independent Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>INDC-LP</td>
<td></td>
<td>Independent Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>INDC-HM</td>
<td></td>
<td>Independent Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>RDX-BC</td>
<td></td>
<td>Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>RDX-SC</td>
<td></td>
<td>Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>RDX-LP</td>
<td></td>
<td>Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>RDX-HM</td>
<td></td>
<td>Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>ASYM-BC</td>
<td></td>
<td>Asymmetric Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>ASYM-SC</td>
<td></td>
<td>Asymmetric Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
<tr>
<td>ASYM-HM</td>
<td></td>
<td>Asymmetric Radix Partitioning</td>
<td>Bucket Chaining</td>
</tr>
</tbody>
</table>
Exploration & Evaluation

• Experiment Setup
  o App-direct mode
  o Pure in-SCM join implementation (cache + SCM)

• Design space exploration
  o 23× join implementation variants
  o 5× NVDIMM-P primitives
  o 6× scaling effects (e.g., thread scalability, prefetching)

• Comprehensive evaluation
Exploration & Evaluation

- **Experiment Setup**
  - App-direct mode
  - Pure in-SCM join implementation (cache + SCM)

- **Design space exploration**
  - 23× join implementation variants
  - 5× NVDIMM-P primitives
  - 6× scaling effects (e.g., thread scalability, prefetching)

- **Comprehensive evaluation**
  - 18× synthesized workloads (e.g., zipfian, selectivity)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Join Notation</th>
<th>Partitioning</th>
<th>Hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Partitioned Hash Join</td>
<td>NPHJ-SC</td>
<td>Separate Chaining</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>NPHJ-LP</td>
<td>Linear Probing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRJ-SC</td>
<td>Separate Chaining</td>
<td>Linear Probing</td>
</tr>
<tr>
<td></td>
<td>SHRJ-LP</td>
<td>Shared Partitioning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRJ-HM</td>
<td>(linked list)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRc-BC</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRc-SC</td>
<td>Separate Chaining</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRc-LP</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHRc-HM</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDIJ-SC</td>
<td>Separate Chaining</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDIJ-LP</td>
<td>Independent Partitioning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDIJ-HM</td>
<td>(linked list)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDCm-BC</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDCm-SC</td>
<td>Separate Chaining</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDCm-LP</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDCm-HM</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDX-BC</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDX-SC</td>
<td>Separate Chaining</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDX-LP</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDX-HM</td>
<td>histogram mechanism</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASYM-BC</td>
<td>bucket 35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASYM-SC</td>
<td>Separate Chaining</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASYM-HM</td>
<td>Asymmetric Radix Partitioning</td>
<td>histogram mechanism</td>
</tr>
</tbody>
</table>
Exploration & Evaluation

- **Experiment Setup**
  - App-direct mode
  - Pure in-SCM join implementation (cache + SCM)

- **Design space exploration**
  - $23 \times$ join implementation variants
  - $5 \times$ NVDIMM-P primitives
  - $6 \times$ scaling effects (e.g., thread scalability, prefetching)

- **Comprehensive evaluation**
  - $18 \times$ synthesized workloads (e.g., zipfian, selectivity)
  - Real benchmarks (e.g., TPC-H)
PHJ vs. NPHJ: An Overall Comparison
PHJ vs. NPHJ: An Overall Comparison

The diagram shows the elapsed time (in seconds) for different size ratios: 1:16, 1:4, and 1:1. The size ratio is plotted on the x-axis, and the elapsed time is plotted on the y-axis.

Key:
- NPHJ
- INDcm
- SHR1I
- RDX
- SHRcm
- ASYM
- INDII

The bars represent the time taken for each size ratio, with different colors indicating different categories of PHJ and NPHJ.
PHJ vs. NPHJ: An Overall Comparison

PHJs generally outperform NPHJs
PHJ vs. NPHJ: Late Materialization
PHJ vs. NPHJ: Late Materialization

FROM R1, R2
WHERE R1.A = R2.D
PHJ vs. NPHJ: Late Materialization

FROM R1, R2
WHERE R1.A = R2.D
PHJ vs. NPHJ: Late Materialization

FROM R1, R2
WHERE R1.A = R2.D
PHJ vs. NPHJ: Late Materialization

FROM R1, R2
WHERE R1.A = R2.D
PHJ vs. NPHJ: Late Materialization
PHJ vs. NPHJ: Late Materialization

![Graph showing the comparison between PHJ and NPHJ in terms of elapsed time for different tuple sizes. The x-axis represents tuple size in bytes (16, 32, 64, 128, 256, 512), and the y-axis represents elapsed time in seconds. The graph compares NPHJ, RDX, ASYM, NPHJ, and RDX, with NPHJ showing consistently lower elapsed times across all tuple sizes.]
PHJ vs. NPHJ: Late Materialization

<key, pointer>
PHJ vs. NPHJ: Late Materialization

<key, pointer>
PHJ vs. NPHJ: Late Materialization

- Late materialization \(<key, \text{pointer}>\) -based join processing further reinforces PHJ’s superiority against NPHJ.
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning

![Graphs showing Bandwidth Regulation in Partitioning](image-url)
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning
Bandwidth Regulation in Partitioning

- **SHR**
- **INDcm**
- **SHRcm**
- **RDX**
- **INDII**
- **ASYM**

Partition Phase Time (s)

- THR-20
- BW-REG

1.43
1.32
Bandwidth Regulation in Partitioning

20-thread processing for step ① and ②
10-thread processing for step ③
### Bandwidth Regulation in Partitioning

#### Data Table

<table>
<thead>
<tr>
<th>#Thread</th>
<th>THR-20</th>
<th>BW-REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1.43</td>
<td>1.32</td>
</tr>
</tbody>
</table>

#### Histograms

#### Prefix Sums

Parallellism limitation in write-intensive steps further facilitates partitioning.
Bandwidth Regulation in Partitioning

20-thread processing for step ① and ②
10-thread processing for step ③

Parallellism limitation in write-intensive steps further facilitates partitioning
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)

![Diagram of PHJ partitioning](image)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)

\[ \lambda = 4.36 \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>32</td>
<td>1.50</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>1.25</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1.00</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>0.75</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.25</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1.50</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>1.75</td>
</tr>
</tbody>
</table>

R/S partitioning fanout ratio

|S|/|R|

read-to-write bandwidth ratio

The Build Side (R) The Probe Side (S)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)

![Graph showing read-to-write bandwidth ratio and R/S partitioning fanout ratio]

The Build Side (R)  The Probe Side (S)
Read/Write Asymmetry in PHJ

- PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)

ASYM has its own sweet spot, and renders greater gains with increased read/write asymmetry.

The Build Side (R)
The Probe Side (S)

read-to-write bandwidth ratio

\[ \frac{|S|}{|R|} \]

R/S partitioning fanout ratio

Elapsed Time (s)

\[ \lambda = 4.36 \]

|S|/|R|

<table>
<thead>
<tr>
<th>Read-to-Write Bandwidth Ratio</th>
<th>R/S Partitioning Fanout Ratio</th>
<th>Elapsed Time (s)</th>
<th>RDX-1</th>
<th>RDX-2</th>
<th>ASYM</th>
</tr>
</thead>
<tbody>
<tr>
<td>64M:256M (x = 4, k = 2)</td>
<td>128M:1024M (x = 8, k = 2)</td>
<td>256M:4096M (x = 16, k = 2)</td>
<td>512M:16384M (x = 32, k = 2)</td>
<td>1024M:16384M (x = 16, k = 2)</td>
<td>2048M:16384M (x = 8, k = 4)</td>
</tr>
</tbody>
</table>
Read/Write Asymmetry in PHJ

• PHJ (RDX) partitioning suffer from TLB thrashing
  - Mult-pass partitioning (excessive writes)
  - Asymmetric partitioning (ASYM)

ASYM has its own sweet spot, and renders greater gains with increased read/write asymmetry
Conclusion

• Key insights in SCM-based joins
  - PHJ generally over NPHJ
  - PHJ configuration in context: hardware & workloads
  - Bandwidth regulation for SCMs or emerging technologies (e.g., CXL)

• Additional experiments & analyses
  - Technical report: https://nus.edu/3qIVvjJ
  - Source codes: https://github.com/fukien/hashjoin-scm
Conclusion

• Key insights in SCM-based joins
Conclusion

• Key insights in SCM-based joins
  o PHJ generally over NPHJ
Conclusion

• Key insights in SCM-based joins
  o PHJ generally over NPHJ
  o PHJ configuration in context: hardware & workloads
Conclusion

• Key insights in SCM-based joins
  o PHJ generally over NPHJ
  o PHJ configuration in context: hardware & workloads
  o Bandwidth regulation for SCMs or emerging technologies (e.g., CXL)
Conclusion

• Key insights in SCM-based joins
  o PHJ generally over NPHJ
  o PHJ configuration in context: hardware & workloads
  o Bandwidth regulation for SCMs or emerging technologies (e.g., CXL)

• Additional experiments & analyses
  o Technical report: https://nus.edu/3qIVvjJ
  o Source codes: https://github.com/fukien/hashjoin-scm
THANK YOU