A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins

Panagiotis Bouros\textsuperscript{1} and Nikos Mamoulis\textsuperscript{2}

\textsuperscript{1} Aarhus University, Denmark
\textsuperscript{2} University of Ioannina, Greece
## Interval Joins

<table>
<thead>
<tr>
<th>Employee</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1994</td>
<td>2006</td>
</tr>
<tr>
<td>Mary</td>
<td>1992</td>
<td>2002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1990</td>
<td>1993</td>
</tr>
<tr>
<td>Bob</td>
<td>1995</td>
<td>1996</td>
</tr>
<tr>
<td>Hugo</td>
<td>1997</td>
<td>2003</td>
</tr>
<tr>
<td>Helen</td>
<td>2005</td>
<td>2007</td>
</tr>
<tr>
<td>Tom</td>
<td>2006</td>
<td>2008</td>
</tr>
</tbody>
</table>
Interval Joins

<table>
<thead>
<tr>
<th>employee</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1994</td>
<td>2006</td>
</tr>
<tr>
<td>Mary</td>
<td>1992</td>
<td>2002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employee</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1990</td>
<td>1993</td>
</tr>
<tr>
<td>Bob</td>
<td>1995</td>
<td>1996</td>
</tr>
<tr>
<td>Hugo</td>
<td>1997</td>
<td>2003</td>
</tr>
<tr>
<td>Helen</td>
<td>2005</td>
<td>2007</td>
</tr>
<tr>
<td>Tom</td>
<td>2006</td>
<td>2008</td>
</tr>
</tbody>
</table>
Interval Joins

Find all pair of employees whose period of work on D1 and D2 intersect
## Interval Joins

<table>
<thead>
<tr>
<th>Employee</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1994</td>
<td>2006</td>
</tr>
<tr>
<td>Mary</td>
<td>1992</td>
<td>2002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1990</td>
<td>1993</td>
</tr>
<tr>
<td>Bob</td>
<td>1995</td>
<td>1996</td>
</tr>
<tr>
<td>Hugo</td>
<td>1997</td>
<td>2003</td>
</tr>
<tr>
<td>Helen</td>
<td>2005</td>
<td>2007</td>
</tr>
<tr>
<td>Tom</td>
<td>2006</td>
<td>2008</td>
</tr>
</tbody>
</table>

Find all pair of employees whose period of work on D1 and D2 intersect

- **Applications**
  - Temporal databases
  - Multidimensional data management
  - Uncertain data management
Our focus

• **Efficient evaluation** of interval joins
  – **Single-threaded** processing
    • **Simple** plane sweep based method
    • **Competitive** to state-of-the-art
  – **Parallel** processing
    • **Partitioning-based** join
    • **Share nothing**
SINGLE-THREADED PROCESSING
Related work

**Nested loops & Sort-merge join**
- [Segev and Gunadhi, VLDB’89]
- [Gunadhi and Segev, ICDE’91]

**Index-based**
- [Zhang et al., ICDE’02]
- [Enderle et al., SIGMOD’04]

**Partitioning-based**
- [Soo et al., ICDE’94]
- [Dignös et al., SIGMOD’14]
- [Cafagna and Böhlen, VLDBJ’17]

**Plane-sweep based**
- [Brinkhoff et al., SIGMOD’93]
- [Arge et al., VLDB’98]
- [Piatov et al., ICDE’16]
Related work

**Nested loops & Sort-merge join**
- [Segev and Gunadhi, VLDB’89]
- [Gunadhi and Segev, ICDE’91]

**Index-based**
- [Zhang et al., ICDE’02]
- [Enderle et al., SIGMOD’04]

**Partitioning-based**
- [Soo et al., ICDE’94]
- [Dignös et al., SIGMOD’14]
- [Cafagna and Böhlen, VLDBJ’17]

**Plane-sweep based**
- [Brinkhoff al., SIGMOD’93]
- [Arge et al., VLDB’98]
- [Piatov et al., ICDE’16]
Plane sweep methods

- **Endpoint-Based Join** (EBI/LEBI)  
  - Sweep line stops both on *start* and *end*  
  - Backwards scan, Gapless hash map **buffers open** intervals

**Pros**
- ✓ No domain-point comparisons
- ✓ Tailored to modern hardware
- ✓ Main memory cache-aware
- ✓ Fast

**Cons**
- ✗ Special structure needed

[ Piatov et al., ICDE’16 ]
Plane sweep methods

• **Endpoint-Based Join (EBI/LEBI)**
  - Sweep line stops both on *start* and *end*
  - Backwards scan, Gapless hash map **buffers open** intervals

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ No domain-point comparisons</td>
<td>✓ Special structure needed</td>
</tr>
<tr>
<td>✓ Tailored to modern hardware</td>
<td></td>
</tr>
<tr>
<td>✓ Main memory cache-aware</td>
<td></td>
</tr>
<tr>
<td>✓ Fast</td>
<td></td>
</tr>
</tbody>
</table>

• **Forward Scan based (FS)**
  - Sweep lines stops only on *start*

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Simple</td>
<td>✓</td>
</tr>
<tr>
<td>✓ No special structure needed</td>
<td></td>
</tr>
</tbody>
</table>

[Piatov et al., ICDE’16]
[Brinkhoff et al., SIGMOD’93]
Plane sweep methods

• **Endpoint-Based Join** (EBI/LEBI)  
  - Sweep line stops both on *start* and *end*  
  - Backwards scan, Gapless hash map buffers open intervals

  **Pros**
  ✓ No domain-point comparisons  
  ✓ Tailored to modern hardware  
  ✓ Main memory cache-aware  
  ✓ Fast

  **Cons**
  ✗ Special structure needed

  [Piatov et al., ICDE’16]

• **Forward Scan based** (FS)  
  - Sweep lines stops only on *start*

  **Pros**
  ✓ Simple  
  ✓ No special structure needed

  **Cons**
  ✗ \(|R| + |S| + |R \cap S|\) comparisons in total

  [Brinkhoff et al., SIGMOD’93]
Optimizing FS
Optimizing FS

FS

$r_1$'s scanned area

$(r_1, s_2), (r_2, s_3)$

$r_2$'s scanned area

$(r_2, s_2), (r_2, s_3), (r_2, s_4)$

$s_1, s_2, s_3, s_4, s_5$
Optimizing FS

FS grouping

Sort by end $G^R$

$G^R$

domain

$r_1$'s scanned area

$(r_1, s_2), (r_1, s_3)$

$(r_2, s_2), (r_2, s_3), (r_2, s_4)$

$G^R$

domain

$r_2$'s scanned area

$(r_2, s_2), (r_2, s_3)$

$(r_2, s_4)$

$G^R$

domain

$r_1$'s scanned area

$(r_1, s_2), (r_1, s_3)$
Optimizing FS

FS grouping

(r₁,s₂),(r₁,s₃)  
(r₂,s₂),(r₂,s₃),(r₂,s₄)

Sort by end $G^R$

(r₁,s₂),(r₁,s₃)  
(r₂,s₂),(r₂,s₃)

Sort by end $G^R$

(r₂,s₂),(r₂,s₃),(r₂,s₄)

(r₂,s₄)
Optimizing FS

FS grouping

FS grouping bucketing

Sort by end $G^R$

Sort by end $G^R$

Sort by end $G^R$

Sort by end $G^R$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_2,s_2),(r_2,s_3),(r_2,s_4)$

$(r_2,s_2),(r_2,s_3),(r_2,s_4)$

$(r_2,s_2),(r_2,s_3),(r_2,s_4)$

$(r_2,s_2),(r_2,s_3),(r_2,s_4)$

$(r_2,s_2)$

$(r_2,s_2)$

$(r_2,s_2)$

$(r_2,s_2)$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_1,s_2),(r_1,s_3)$

$(r_2,s_2)$

$(r_2,s_2)$

$(r_2,s_2)$

$(r_2,s_2)$

$B^S$

$B^S$

$B^S$

$B^S$

$(s_1)$

$(s_2,s_3)$

$(s_4,s_5)$

$(s_1)$

$(s_2,s_3)$

$(s_4,s_5)$

$(s_1)$

$(s_2,s_3)$

$(s_4,s_5)$
Optimizing FS

FS grouping

FS grouping bucketing

Sort by end $G^R$

Sort by end $G^R$

Sort by end $G^R$

Sort by end $G^R$

$\{s_1\} \cup \{s_2, s_3\} \cup \{s_4, s_5\}$

$\{s_1\} \cup \{s_2, s_3\} \cup \{s_4, s_5\}$

$\{s_1\} \cup \{s_2, s_3\} \cup \{s_4, s_5\}$

$\{s_1\} \cup \{s_2, s_3\} \cup \{s_4, s_5\}$
PARALLEL PROCESSING
Hash-based partitioning

- Idea
  - Randomly split each input into $k$ partitions using hash function $h$
  - Evaluate $k^2$ independent partition joins

[Piatov et al., ICDE’16]
Hash-based partitioning

- **Idea**
  - Randomly split each input into \( k \) partitions using hash function \( h \)
  - Evaluate \( k^2 \) independent partition joins

**Pros**
- ✓ Simple
- ✓ Load balancing

[August 31, 2017] 43rd International Conference on Very Large Data Bases
Hash-based partitioning

- Idea
  - Randomly split each input into $k$ partitions using hash function $h$
  - Evaluate $k^2$ independent partition joins

**Pros**
- ✓ Simple
- ✓ Load balancing

**Cons**
- ✗ Domain-point comparisons rise
  - $2k(|R|+|S|)$ for EBI/LEBI, $k(|R|+|S|)$ for FS
- ✗ Degree of parallelism
  - $n$ CPU cores $\rightarrow k = \sqrt{n}$ partitions

[Piatov et al., ICDE’16]
Domain-based partitioning

**Idea**
- Split domain into $k$ tiles
- Replicate intervals
- Evaluate $k$ independent partition joins
Domain-based partitioning

**Pros**

- **Idea**
  - Split domain into $k$ tiles
  - Replicate intervals
  - Evaluate $k$ independent partition joins

**Pros**

- ✓ Degree of parallelism
  - $n$ CPU cores $\Rightarrow k = n$ partitions
- ✓ Automatic duplicate elimination
**Domain-based partitioning**

- **Idea**
  - Split domain into $k$ tiles
  - Replicate intervals
  - Evaluate $k$ independent partition joins

**Pros**
- Degree of parallelism
  - $n$ CPU cores $\rightarrow k = n$ partitions
- Automatic duplicate elimination

**Cons**
- Replication
- Load balancing
Domain-based partitioning

- **Idea**
  - Split domain into $k$ tiles
  - Replicate intervals
  - Evaluate $k$ independent partition joins

**Pros**
- ✓ Degree of parallelism
  - $n$ CPU cores $\rightarrow$ $k = n$ partitions
- ✓ Automatic duplicate elimination

**Cons**
- ✗ Replication
- ✗ Load balancing
Mini-joins break down

• 3 types of intervals $\rightarrow$ 9 types of mini-tasks
Mini-joins break down

• 3 types of intervals $\rightarrow$ 9 types of mini-tasks

Duplicate results elimination
Mini-joins break down

• 3 types of intervals $\rightarrow$ 9 types of mini-tasks

(1) Same complexity as original join
(2) Sweep only one input
(3) No comparisons – cross product
(4) (6) Duplicate results elimination
(5) (7)
(8) (9)

Aug 31, 2017
43rd International Conference on Very Large Data Bases
Greedy scheduling

• Idea
  – Distribute $1 + 5 \cdot (k-1)$ mini-joins to different cores
  – Evenly distribute load $\rightarrow$ minimize max load
  – NP-hard problem
    • Greedy approximation algorithm
Greedy scheduling

- **Idea**
  - Distribute $1 + 5 \cdot (k-1)$ mini-joins to different cores
  - Evenly distribute load $\rightarrow$ minimize max load
  - NP-hard problem
    - Greedy approximation algorithm
Adaptive partitioning

• Idea
  – Create an initial uniform partitioning
  – Employ a very fine tiling – granules
  – Move load between neighboring tiles
    • Move granules between neighboring tiles
    • Reposition borders of tiles
Adaptive partitioning

• Idea
  – Create an initial uniform partitioning
  – Employ a very fine tiling – granules
  – Move load between neighboring tiles
    • Move granules between neighboring tiles
    • Reposition borders of tiles
EXPERIMENTAL ANALYSIS
Setup

- **Hardware**
  - dual 10-core Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10 GHz with 128 GBs of RAM
  - Hyper-threading enabled, up to 40 threads

- **Software**
  - Workload [ICDE’16] → XOR of start
  - Loop unrolling forced, OpenMP for multi-threading

- **Datasets**
  - WEBKIT git repo, interval = period of time file unchanged
  - BOOKS Aarhus libraries, interval = period of time book lent
  - Synthetic
    - Interval duration follows exponential distribution, uniformly distributed start plus peaks

- **Experiments**
  - Execution time, # comparisons, memory footprint
  - Both self joins and non-self joins
  - Vary |R|/|S|, # cores (threads)
Optimizing FS

WEBKIT

Execution time [secs]

Endpoint comparisons [%]

|IR|/|SI| [1 core]
Single-threaded processing

WEBKITT

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimizing Domain-based Paradigm

![Bar charts showing the execution time and average idle time ratio for different partitioning strategies and core numbers.](image)

- Atomic/uniform
- \( mj+\text{atomic/uniform} \)
- Atomic/adaptive
- \( mj+\text{greedy/uniform} \)
- \( mj+\text{greedy/adaptive} \)

### Execution Time [secs]

- \(|R|/|S| [16 cores] = 0.25\)
- \(|R|/|S| [16 cores] = 0.5\)
- \(|R|/|S| [16 cores] = 0.75\)
- \(|R|/|S| [16 cores] = 1\)

### Average Idle Time Ratio [%]

- \(|R|/|S| [16 cores] = 0.25\)
- \(|R|/|S| [16 cores] = 0.5\)
- \(|R|/|S| [16 cores] = 0.75\)
- \(|R|/|S| [16 cores] = 1\)
Parallel processing

![Graph 1: Speedup vs. # cores](image1)

![Graph 2: Execution time vs. # cores](image2)

![Graph 3: Execution time vs. [IR/ISI [16 cores]]](image3)
To sum up

• Contributions
  – Efficient evaluation of interval joins
  – Single-threaded processing
    • Optimized bgFS, competitive to state-of-the-art EBI/LEBI
    • Lower memory footprint
  – Parallel processing
    • Novel domain-based partitioning paradigm
    • Higher speedup

• Future work
  – Other types of temporal joins
  – Other types of temporal operators
  – Parallel processing
    • Data-level parallelism, share data between threads
Questions ?
EXTRAS
Endpoint-Based Join (EBI/LEBI)

[Piatov et al., ICDE’16]

Endpoint indices
EI\(^R\) = \{r_1.start, r_2.start, r_1.end, r_2.end\}
EI\(^S\) = \{s_1.start, s_1.end, s_2.start, s_2.end, s_3.start, s_3.end, s_4.start, s_5.start, s_4.end, s_5.end\}

Active sets
A\(^R\) = \{\}
A\(^S\) = \{\}

Result
\{\}

Aug 31, 2017
Endpoint-Based Join (EBI/LEBI)

[Пiatov et al., ICDE’16]

Endpoint indices
EI<sup>R</sup> = \{r<sub>1</sub>.start, r<sub>2</sub>.start, r<sub>1</sub>.end, r<sub>2</sub>.end\}
EI<sup>S</sup> = \{s<sub>1</sub>.start, s<sub>1</sub>.end, s<sub>2</sub>.start, s<sub>2</sub>.end, s<sub>3</sub>.start, s<sub>3</sub>.end, s<sub>4</sub>.start, s<sub>5</sub>.start, s<sub>4</sub>.end, s<sub>5</sub>.end\}

Active sets
A<sup>R</sup> = \{
A<sup>S</sup> = \{s<sub>1</sub>\}

Result
\{

Endpoint-Based Join (EBI/LEBI)

Endpoint indices
\( E^R = \{r_1.\text{start}, r_2.\text{start}, r_1.\text{end}, r_2.\text{end}\} \)
\( E^S = \{s_1.\text{start}, s_1.\text{end}, s_2.\text{start}, s_2.\text{end}, s_3.\text{start}, s_3.\text{end}, s_4.\text{start}, s_5.\text{start}, s_4.\text{end}, s_5.\text{end}\} \)

Active sets
\( A^R = \{\} \)
\( A^S = \{s_1\} \)

Result
\( \{(r_1, s_1)\} \)

[Refer to Piatov et al., ICDE’16]
Endpoint-Based Join (EBI/LEBI)

[Novak et al., ICDE’16]

Endpoint indices
\[ \text{EI}^R = \{r_1.start, r_2.start, r_1.end, r_2.end\} \]
\[ \text{EI}^S = \{s_1.start, s_1.end, s_2.start, s_2.end, s_3.start, s_3.end, s_4.start, s_5.start, s_4.end, s_5.end\} \]

Active sets
\[ \text{AR} = \{\} \]
\[ \text{AS} = \{s_1\} \]

Result
\[ \{(r_1,s_1)\} \]
Endpoint-Based Join (EBI/LEBI)

Pros
✓ No domain-point comparisons when producing results
✓ Tailored to modern hardware
✓ Main memory cache-aware
✓ Fast

Cons
✗ Special data structure needed for active sets, support for efficient updates and scans
Forward Scan based (FS)

Sorted inputs
R = \{r_1, r_2\}
S = \{s_1, s_2, s_3, s_4, s_5\}

Result
\{
\}
Forward Scan based (FS)

[Brinkhoff et al., SIGMOD’93]

Sorted inputs
R = \{r_1, r_2\}
S = \{s_1, s_2, s_3, s_4, s_5\}

Result
\{(r_1,s_1)\}
Forward Scan based (FS)

[Brinkhoff et al., SIGMOD’93]

Sorted inputs
R = \{r_1, r_2\}
S = \{s_1, s_2, s_3, s_4, s_5\}

Result
\{(r_1,s_1), (r_1,s_2), (r_1,s_3)\}
Forward Scan based (FS)

[Brinkhoff et al., SIGMOD’93]

Pros

✓ Simple
✓ No special structure needed

Cons

✗ Each join result requires a domain-point comparison, $|R| + |S| + |R \bowtie S|$ comparisons in total
Forward Scan based (FS)

Pros
✓ Simple
✓ No special structure needed

Cons
✗ Each join result requires a domain-point comparison,
\[ |R| + |S| + |R \bowtie S| \] comparisons in total