Parallel Sort-Based Matching for Data Distribution Management on Shared-Memory Multiprocessors

Moreno Marzolla (moreno.marzolla@unibo.it)  
Gabriele D'Angelo (g.dangelo@unibo.it)

Dept. of Computer Science and Engineering  
University of Bologna
Data Distribution Management

• DDM services are part of the IEEE 1516 “High Level Architecture” (HLA) specification

• Given
  – Sets of subscription and update regions in a $d$-dimensional space
  – Update regions (extents) generate events
  – Subscription regions must receive events generated by overlapping update regions

• Goal
  – Find all update/subscription pairs that overlap
Example in $d = 2$ dimensions

Intersections:
- $(S_1, U_1)$, $(S_2, U_2)$, $(S_3, U_1)$, $(S_3, U_2)$
The Region Matching Problem

• Can be solved using **spatial data structures** and related algorithms
  - e.g., $k$-$d$-trees, $R$-trees, Quad-trees, ...

• However, simpler algorithms are generally preferred for DDM implementations
  - Brute-Force
  - Grid-Based [Boukerche and Dzermajko 2001]
  - Sort-Based [Raczy, Tan and Yu 2005]
  - Interval-Tree [Marzolla, D'Angelo and Mandrioli 2013]
The Region Matching Problem

- The Region Matching Problem in $d > 1$ dimensions can be reduced to $d$ instances on 1D intervals
The Region Matching Problem

- The Region Matching Problem in $d > 1$ dimensions can be reduced to $d$ instances on 1D intervals
The Region Matching Problem

- The Region Matching Problem in $d > 1$ dimensions can be reduced to $d$ instances on 1D intervals.
Sort-Based Matching

- Sort endpoints
- Scan endpoints in sorted order
  - Let $\text{SubSet}$ and $\text{UpdSet}$ be the sets of currently active subscription and update intervals, resp.
  - For each endpoint $t$
    - If $t$ marks the beginning of a subs/upd interval $X$, then
      - add $X$ to $\text{SubSet}$ or $\text{UpdSet}$
    - Else
      - remove $X$ from $\text{SubSet}$ or $\text{UpdSet}$
      - $X$ overlaps with all intervals currently in $\text{UpdSet}$ (if $X$ is a subscription extent) or $\text{SubSet}$ (if $X$ is an update extent)
Example

$\text{UpdSet} = \{ \}$

$\text{SubSet} = \{ \}$

$\text{Intersections} = \{ \}$
Example

$$\text{UpdSet} = \{ \}$$
$$\text{SubSet} = \{ S_1 \}$$
$$\text{Intersections} = \{ \}$$
Example

\[ \text{UpdSet} = \{ U_1 \} \]

\[ \text{SubSet} = \{ S_1 \} \]

\[ \text{Intersections} = \{ \} \]
Example

\[ \text{UpdSet} = \{ U_1, U_2 \} \]
\[ \text{SubSet} = \{ S_1 \} \]
\[ \text{Intersections} = \{ \} \]
Example

\[ \text{UpdSet} = \{ U_1, U_2 \} \]
\[ \text{SubSet} = \{ \} \]
\[ \text{Intersections} = \{ (S_1, U_1), (S_1, U_2) \} \]
Example

\[ \text{UpdSet} = \{ U_1, U_2 \} \]
\[ \text{SubSet} = \{ S_2 \} \]
\[ \text{Intersections} = \{ (S_1, U_1), (S_1, U_2) \} \]
Example

\begin{itemize}
\item \textbf{UpdSet} = \{ U_1 \}
\item \textbf{SubSet} = \{ S_2 \}
\item \textbf{Intersections} = \{ (S_1, U_1), (S_1, U_2), (S_2, U_2) \}
\end{itemize}
Example

UpdSet = \{ U_1 \}
SubSet = \{ \}
Intersections = \{ (S_1, U_1), (S_1, U_2), (S_2, U_2), (S_2, U_1) \}
Example

\[ \text{UpdSet} = \{ U_1 \} \]
\[ \text{SubSet} = \{ S_3 \} \]
\[ \text{Intersections} = \{ (S_1, U_1), (S_1, U_2), (S_2, U_2), (S_2, U_1) \} \]
Example

$\text{UpdSet} = \{ \}$

$\text{SubSet} = \{ S_3 \}$

$\text{Intersections} = \{ (S_1, U_1), (S_1, U_2), (S_2, U_2), (S_2, U_1), (S_3, U_1) \}$
Example

UpdSet = \{ \}
SubSet = \{ \}
Intersections = \{ (S_1, U_1), (S_1, U_2), (S_2, U_2), (S_2, U_1), (S_3, U_1) \}
Parallel Sort-Based Matching on Shared-Memory Systems

- Sort endpoints
- Scan endpoints
  - Let $SubSet$ and $UpdSet$ be the sets of currently active subscription and update intervals, resp.
  - For each endpoint $t$
    - If $t$ marks the beginning of a subs/upd interval $X$, then
      - add $X$ to $SubSet$ or $UpdSet$
    - Else
      - remove $X$ from $SubSet$ or $UpdSet$
      - $X$ overlaps with all intervals currently in $UpdSet$ (if $X$ is a subscription extent) or $SubSet$ (if $X$ is an update extent)
Parallel Sort-Based Matching on Shared-Memory Systems

• Sort endpoints in parallel 🎉

• Scan endpoints
  – Let $SubSet$ and $UpdSet$ be the sets of currently active subscription and update intervals, resp.
  – For each endpoint $t$
    • If $t$ marks the beginning of a subs/upd interval $X$, then
      – add $X$ to $SubSet$ or $UpdSet$
    • Else
      – remove $X$ from $SubSet$ or $UpdSet$
      – $X$ overlaps with all intervals currently in $UpdSet$ (if $X$ is a subscription extent) or $SubSet$ (if $X$ is an update extent)
Parallel Sort-Based Matching on Shared-Memory Systems

- Sort endpoints in parallel 😊
- Scan endpoints in parallel?? 😞
  - Let $SubSet$ and $UpdSet$ be the sets of currently active subscription and update intervals, resp.
  - For each endpoint $t$
    - If $t$ marks the beginning of a subs/upd interval $X$, then
      - add $X$ to $SubSet$ or $UpdSet$
    - Else
      - remove $X$ from $SubSet$ or $UpdSet$
      - $X$ overlaps with all intervals currently in $UpdSet$ (if $X$ is a subscription extent) or $SubSet$ (if $X$ is an update extent)

Loop-carried dependencies
Intervals for which the **left** endpoint only has been encountered by this processor

Intervals for which the **right** endpoint only has been encountered by this processor
Which intervals are still “open” at the end of each sequential scan
SubSet (UpdSet) can now be computed concurrently by all processors.
Performance Evaluation

- Parallel SBM implemented in C++/OpenMP
- Testing according to the methodology used in [Raczy et al. 2005]
- Instances with a single dimension
- Parameters:
  - $N$ = number of intervals
  - $\alpha$ = overlapping degree = $\frac{\sum \text{Area of intervals}}{\text{Total area of the routing space}}$
# Execution platforms

<table>
<thead>
<tr>
<th></th>
<th>Solaris</th>
<th>Titan</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Xeon E5-2640</td>
<td>Intel Core i7-5820K</td>
</tr>
<tr>
<td><strong>Clock freq.</strong></td>
<td>2 GHz</td>
<td>3.3 GHz</td>
</tr>
<tr>
<td><strong>Processors</strong></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Tot n. of cores</strong></td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td><strong>Hyperthreading?</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>128GB</td>
<td>64GB</td>
</tr>
</tbody>
</table>

IEEE/ACM DS-RT 2017
Wall-Clock Time

Wall-Clock Time (WCT), $\alpha=100$, $10^6$ extents

- solaris, parallel BF
- solaris, parallel ITM
- solaris, parallel SBM
- titan, parallel BF
- titan, parallel ITM
- titan, parallel SBM

Parallel Brute Force

Parallel Interval Tree

Parallel SBM
Wall-Clock Time

Wall-Clock Time (WCT), $\alpha=100$, $10^6$ extents

- solaris, parallel BF
- solaris, parallel ITM
- solaris, parallel SBM
- titan, parallel BF
- titan, parallel ITM
- titan, parallel SBM

**Parallel Brute Force**

Two orders of magnitude

Parallel Interval Tree

One order of magnitude

Parallel SBM
Speedup

Speedup, $\alpha=100$, $10^8$ extents

- solaris, parallel ITM
- solaris, parallel SBM
- titan, parallel ITM
- titan, parallel SBM

Speedup

Number of threads

Parallel SBM

Parallel Interval Tree
Strong Scaling Efficiency

Strong Scaling Efficiency, $\alpha=100, 10^8$ extents

$E(p) = \frac{S(p)}{p}$

Number of threads

Strong Scaling Efficiency

solaris, parallel ITM
solaris, parallel SBM
titan, parallel ITM
titan, parallel SBM

Parallel Interval Tree
Parallel SBM
Weak Scaling Efficiency

Weak Scaling Efficiency, \(\alpha=100, 10^7\) extents per thread

\[ W(p) = \frac{\text{Time to perform } p \text{ units of work on } p \text{ processors}}{\text{Time to perform one unit of work on one processor}} \]
Conclusions

• Parallel SBM improves the already fast SBM algorithm
  – Can take advantage of modern multicore processors

• The speedup is limited by several factors
  – The parallel sorting phase
  – Intrinsecally serial regions
  – The baseline is very fast!

• Future works
  – Improve scaling efficiency
  – Extend the parallel SBM algorithm to cope with moving regions
  – Implement parallel SBM on the GPU (???)
Thanks for your attention

Questions?
Brute-Force Matching
Brute-Force Matching
Grid-Based Matching

Grid-based matching is a method for comparing and aligning sets of grid structures. In this diagram, we have three sets of grid structures labeled $S_1$, $S_2$, $S_3$, and $U_1$, $U_2$. The grid-based matching process aims to find the best alignment between these sets, often by minimizing a distance metric.
Grid-Based Matching
Grid-Based Matching
Grid-Based Matching
Interval-Tree Matching

• Based on the Interval Tree data structure
• Solves the 1D matching problem
• Subscription (Update) intervals are stored in the leaves of an Interval Tree
  – Balanced Search Tree
  – Internal nodes are augmented with auxiliary data used to steer queries towards overlapping intervals
• Intersections can be identified with a tree visit for each Update (Subscription) interval

Moreno Marzolla, Gabriele D'Angelo, Marco Mandrioli, A Parallel Data Distribution Management Algorithm, proc. DS-RT 2013, http://dx.doi.org/10.1109/DS-RT.2013.23
Interval Tree

![Interval Tree Diagram]

- **$S_1$**: Interval [1, 9]
- **$S_2$**: Interval [10, 12]
- **$S_3$**: Interval [2, 3]
- **$S_4$**: Interval [1, 12]
- **$S_5$**: Interval [7, 12]
- **$S_6$**: Interval [1, 9]
- **$S_7$**: Interval [7, 12]

**Min lower**: 1
**Max upper**: 12

IEEE/ACM DS-RT 2017 43