Parallel Execution Approaches on Data and Index Structures in the Context of Semantic Web Database Management Systems

Dennis Heinrich

Institute of Information Systems

September 19th, 2018
Motivation

[4] EDB, Busan (SKR), 2017
[5] SUPE, 2018
[9] CIT, Xi’an (CN), 2014
[14] ReConFig, Cancun (MEX), 2015
[15] ARCS, Vienna (AUT), 2017
Motivation

Focus of thesis

[4] EDB, Busan (SKR), 2017
[5] SUPE, 2018
[9] CIT, Xi’an (CN), 2014
[14] ReConFig, Cancun (MEX), 2015
[15] ARCS, Vienna (AUT), 2017
Motivation

What’s the problem?

▶ Continuous data growth leads to necessary improvement
▶ Improving sequential executions becomes challenging

... and the solution?

▶ Parallel executions to fit multi-core systems and specialized hardware accelerators (like FPGAs)
Semantic Web Database: Data

Triples (String)

Dog | type | Animal
Cat | type | Animal
Semantic Web Database: Dictionary

Triples (String)

Dog
Cat
type
Animal

Dictionary

1
Animal

4
Cat

7
Dog

9
type
Semantic Web Database: Data revisited
Semantic Web Database: B⁺-tree as Index Structure
Semantic Web Database: Example Query

- Triples (String)
  - Dog type Animal
  - Cat type Animal

- Triples (Integer)
  - 7 9 1
  - 4 9 1

- B+ Tree
  - (1)
  - (2)

- Dictionary
  - 1 Animal
  - 4 Cat
  - 7 Dog
  - 9 type

- (3)
- (4)
Semantic Web Database: Example Query

D. Heinrich Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
Semantic Web Database: Example Query

![Diagram of a Semantic Web Database Example Query]

- **Triples (String):**
  - Dog type Animal
  - Cat type Animal

- **Triples (Integer):**
  - 7 9 1
  - 4 9 1

- **Dictionary:**
  - 1 Animal
  - 4 Cat
  - 7 Dog
  - 9 type

- **B+ Tree:**
  - 2 4 4 9 8 9 1 7 1
  - 6 7 7 8 9 8 9 9 3 4 1 3

- **Dog? type? 1?**
  - Animal
A short recapitulation on $B^+$-Trees
A short recapitulation on B⁺-Trees - root
A short recapitulation on $\text{B}^+$-Trees - inner nodes

inner node order $k \rightarrow$ from $k$ to $2 \cdot k$ keys
A short recapitulation on $B^+$-Trees - leaves

inner node order $k \rightarrow$ from $k$ to $2 \cdot k$ keys

leaf order $k' \rightarrow$ from $k'$ to $2 \cdot k'$ keys
A short recapitulation on $B^+$-Trees - values with data or references

inner node order $k \rightarrow$ from $k$ to $2 \cdot k$ keys

leaf order $k' \rightarrow$ from $k'$ to $2 \cdot k'$ keys
A short recapitulation on B⁺-Trees - edges

inner node order $k \rightarrow$ from $k$ to $2 \cdot k$ keys

leaf order $k' \rightarrow$ from $k'$ to $2 \cdot k'$ keys
A short recapitulation on B\(^+\)-Trees - example search

search for value from key 48
A short recapitulation on $B^+$-Trees - example search

search for value from key 48
A short recapitulation on B$^+$-Trees - example search

search for value from key 48
Searching inside a $B^+$-Tree node

linear search

13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93

binary search

13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93
Searching inside a $B^+$-Tree node - linear search

linear search

13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93

binary search

13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93
Searching inside a $\mathbf{B}^+$-Tree node - binary search

linear search

```
13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93
```

binary search

```
13 24 32 36 44 48 52 55 57 63 69 71 76 82 88 93
```
Searching inside a B\textsuperscript{+}-Tree node - approach: parallel search

**Linear Search**

**Binary Search**

**Parallel Search**
Requirements & Challenges

What we need to consider:

- Parallel memory access
  - FPGA: independent Block RAM (distributed over chip area)
  - CPU/GPU: sequential data access
- 96 Bit Integer triples
  - FPGA: individual data width
  - CPU/GPU: word width of 32 or 64 Bits

Further we want:

- Fully functional Semantic Web system (insert, update, delete)
  ⇒ use a hybrid system
Our approach
Our approach
Experimental setup

Hardware:

- Xilinx Virtex-6 XC6VHX380T
- Hosted on a board
- Connected via PCIe

Software:

- Dell Precision T3610 workstation
- 40 GB of RAM
- Intel Xeon E5-1600 v2 processor with 3.0 GHz
- LUPOSDATE
Measured tree groups

$L_{FPGA}$:
- Variable order from 3 to 9
- Software + FPGA

$L_{B^+}$:
- Typical parameters for disk use
- Order is 500, just entered triple amount differs
- Software only
What was achieved?

Search:

- Speed up of over 2
- Halved the time per search operation
What was achieved?

**Search:**
- Speed up of over 2
- Halved the time per search operation

**What about update operations (insert/delete)?**
- Every update needs a search
What was achieved?

Search:
- Speed up of over 2
- Halved the time per search operation

What about update operations (insert/delete)?
- Every update needs a search

**Problem:** *Is it worth to transfer our tree to the FPGA if the structure of the tree is often changing?*
Worst Case and Best Case for insert and delete operations

![Diagram showing worst and best cases for insert and delete operations in a data structure.]

Worst Case for Insert
Best Case for Delete

Best Case for Insert
Worst Case for Delete
Worst Case and Best Case for insert and delete operations

Best Case:

possible Number Of Updates = \( (\text{Updates In Leaves} \cdot \text{Updates In Inner Nodes Software} \cdot \text{Entry Points FPGA to Host}) \) (1)
Worst Case and Best Case for insert and delete operations

Best Case:

\[ \text{possible Number Of Updates} = (\text{Updates In Leaves} \cdot \text{Updates In Inner Nodes Software} \cdot \text{Entry Points FPGA to Host}) \] (1)

Worst Case:

- only one operation
Idea: Scheduler

Let a scheduler decide:

\[ t_{\text{gain}} = \text{possible Number Of Updates} \cdot t_{\text{Op}} \cdot r_{\text{Op}} - t_{\text{setup}}(x) \]  \hspace{1cm} (2)

\( t_{\text{Op}} \): time saved per operation

\[ r_{\text{Op}} = \frac{\text{number of searches}}{\text{number of updates}} \]  \hspace{1cm} (3)

\( t_{\text{setup}}(x) \): time for setting up the tree with \( x \) triples
Setup time $t_{\text{setup}}(x)$ for a new tree

$$t(x) = 0.00000027501701520384x$$
### Minimum ratio $r_{Op}$ for best and worst case

<table>
<thead>
<tr>
<th>order</th>
<th>$r_{Op_{\text{worst}}}$</th>
<th>$r_{Op_{\text{best}}}$</th>
<th>Possible Number Of Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>30,112.9</td>
<td>0.0148414</td>
<td>1,200,500</td>
</tr>
<tr>
<td>4</td>
<td>34,210.9</td>
<td>0.0107322</td>
<td>3,280,500</td>
</tr>
<tr>
<td>5</td>
<td>73,021.3</td>
<td>0.0082955</td>
<td>7,320,000</td>
</tr>
<tr>
<td>6</td>
<td>125,228.9</td>
<td>0.0083823</td>
<td>14,280,000</td>
</tr>
<tr>
<td>7</td>
<td>205,270.3</td>
<td>0.0077481</td>
<td>25,312,000</td>
</tr>
<tr>
<td>8</td>
<td>280,292.6</td>
<td>0.0064449</td>
<td>41,760,500</td>
</tr>
<tr>
<td>9</td>
<td>486,721.4</td>
<td>0.0066672</td>
<td>65,160,000</td>
</tr>
</tbody>
</table>
Average Case

![Graph showing the average case for different order of the tree and number of triples, with symbols representing best case and various quantities of triples.](image)

- Best case
- 500 triples
- 1,000 triples
- 2,000 triples
- 4,000 triples

$r_{OP}$ vs order of the tree
Summary

- Hybrid use of a $B^+$-Tree to search all keys of a node in parallel on the FPGA
- Evaluation: system setup vs. search acceleration

Questions?
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS

D. Heinrich
Appendix

D. Heinrich: Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
Appendix

D. Heinrich Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
Appendix

Addressnumber | AddressL0 | ⋮ | AddressLn
--- | --- | ⋮ | ---
10 | key0-0 | key0-1
10 | key1-0 | key1-1
00 | key2-0 | key2-1 | key2-2 | key2-3
10 | key3-0 | key3-1
01 | key4-0 | key4-1 | key4-2
⋯
Execution times

<table>
<thead>
<tr>
<th>order</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>343,001</td>
<td>729,001</td>
<td>1,331,001</td>
<td>2,197,001</td>
<td>3,375,001</td>
<td>4,913,001</td>
<td>6,859,001</td>
</tr>
<tr>
<td>B</td>
<td>857,500</td>
<td>2,187,000</td>
<td>4,658,250</td>
<td>8,787,750</td>
<td>15,187,250</td>
<td>24,565,000</td>
<td>37,723,000</td>
</tr>
<tr>
<td>C</td>
<td>1,372,000</td>
<td>3,645,000</td>
<td>7,985,500</td>
<td>15,378,500</td>
<td>26,999,500</td>
<td>44,217,000</td>
<td>68,588,000</td>
</tr>
<tr>
<td>D</td>
<td>1,886,500</td>
<td>5,103,000</td>
<td>11,312,750</td>
<td>21,969,250</td>
<td>38,811,750</td>
<td>63,869,000</td>
<td>99,453,000</td>
</tr>
<tr>
<td>E</td>
<td>2,401,000</td>
<td>6,561,000</td>
<td>14,640,000</td>
<td>28,560,000</td>
<td>50,624,000</td>
<td>83,521,000</td>
<td>130,320,000</td>
</tr>
</tbody>
</table>
Speed up - $L_{MM}$ against $L_{FPGA}$

<table>
<thead>
<tr>
<th>Order</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.05</td>
</tr>
<tr>
<td>4</td>
<td>1.1</td>
</tr>
<tr>
<td>5</td>
<td>1.15</td>
</tr>
<tr>
<td>6</td>
<td>1.2</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the speed up for different orders of triples, with $L_{MM}$ against $L_{FPGA}$.
Speed up $- L_{B+}$ against $L_{FPGA}$

D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
18/18
An example $B^+$-Tree ...
... and as a Cache Sensitiv B\textsuperscript{+}-Tree (CSB\textsuperscript{+}-Tree\textsuperscript{1})

\[\text{address} + \text{offset} \cdot \text{offset} + 2 \cdot \text{offset} \text{ address} + \text{offset}\]

\[\begin{array}{c}
1 \cdot 5 \\
20 \cdot 24 \\
37 \cdot 38 \cdot 42 \\
44 \cdot 45 \\
47 \cdot 48 \\
135 \cdot 166 \\
\end{array}\]

\[\begin{array}{c}
15 \cdot 25 \\
46 \cdot 128 \\
\end{array}\]

\[\text{node group}\]

\[\text{+offset}\]

\[\text{+2.offset}\]

---

\textsuperscript{1} from Jun Rao and Kenneth A. Ross, Making B\textsuperscript{+}-Trees Cache Conscious in Main Memory, ACM 2000
Parallel search for a triple inside the FPGA

Current Address

\[\text{Triple}_1 \geq \text{Triple}_2 \geq \ldots \geq \text{Triple}_n\]

Next Address

Position

Searched Triple
Parallel search for a triple inside the FPGA - load node
Parallel search for a triple inside the FPGA - compare with searched key

Current Address

Triple$_1$  Triple$_2$  ...  Triple$_n$

Position

Searched Triple

Next Address

+
Parallel search for a triple inside the FPGA - calculate new address

Current Address

Triple_1

Triple_2

\ldots

Triple_n

\geq

\geq

\ldots

\geq

Next Address

Position

\geq

Searched Triple
Entered number of triples

- Generated trees have 4 inner levels (except $L_{B^+}$)
- Leaves with order 500 always in software
- Gradually filling the inner nodes with keys from A to E (max. keys)

<table>
<thead>
<tr>
<th>order</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>343,001</td>
<td>857,500</td>
<td>1,372,000</td>
<td>1,886,500</td>
<td>2,401,000</td>
</tr>
<tr>
<td>4</td>
<td>729,001</td>
<td>2,187,000</td>
<td>3,645,000</td>
<td>5,103,000</td>
<td>6,561,000</td>
</tr>
<tr>
<td>5</td>
<td>1,331,001</td>
<td>4,658,250</td>
<td>7,985,500</td>
<td>11,312,750</td>
<td>14,640,000</td>
</tr>
<tr>
<td>6</td>
<td>2,197,001</td>
<td>8,787,750</td>
<td>15,378,500</td>
<td>21,969,250</td>
<td>28,560,000</td>
</tr>
<tr>
<td>7</td>
<td>3,375,001</td>
<td>15,187,250</td>
<td>26,999,500</td>
<td>38,811,750</td>
<td>50,624,000</td>
</tr>
<tr>
<td>8</td>
<td>4,913,001</td>
<td>24,565,000</td>
<td>44,217,000</td>
<td>63,869,000</td>
<td>83,521,000</td>
</tr>
<tr>
<td>9</td>
<td>6,859,001</td>
<td>37,723,000</td>
<td>68,588,000</td>
<td>99,453,000</td>
<td>130,320,000</td>
</tr>
</tbody>
</table>
Patricia Trie Merge - 2 Tries

Legend:
- \( k_x \) node with label \( k_x \)
- edge for \( c_1 \ldots c_y \) string

1. Step
2. Step
3. Step
4. Step
5. Step
Patricia Trie Merge - Multiple Tries

Legend:
- \( k_x \) node with label \( k_x \)
- edge for \( c_1...c_y \) "\( c_1...c_y \)"
- string

1. Step
2. Step
3. Step
4. Step
5. Step
6. Step
7. Step
8. Step
9. Step
PatTrieSort - Total Time

D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS

PatTrieSort - Bytes Read

- PatTrieSort
- String Merging
- External Merge Sort
- Replacement Selection

Bytes Read

- 0
- 50,000,000,000
- 100,000,000,000
- 150,000,000,000
- 200,000,000,000
- 250,000,000,000
- 300,000,000,000
- 350,000,000,000
- 500,000,000
- 1,000,000,000
- 2,000,000,000
- 4,000,000,000
- 8,000,000,000
- 16,000,000,000
- 32,000,000,000

D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
PatTrieSort - Bytes Written
PatTrieSort - Number of Runs

![Bar chart showing the number of runs for different data sizes for PatTrieSort and External Merge Sort. The x-axis represents data sizes in steps of 500,000 up to 32,000,000. The y-axis represents the number of runs ranging from 1 to 10,000. The bars for PatTrieSort and External Merge Sort show the number of runs for each data size.]

- PatTrieSort/String Merging
- External Merge Sort
PatTrieSort - Number of Runs - Replacement Selektion
PatTrieSort - Total IO

D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS
18/18
1. Build patricia trie and map triples to temporary IDs

2. Map to local IDs

3. Roll out patricia trie

4. Sort 6 times and store runs

5. Merge patricia tries

6. Generate dictionary

7. Determine mapping from local to global IDs

8. Map runs from local to global IDs

9. Merge runs and generate evaluation indices
Index construction single times

- Building patricia tries
- Mapping to local IDs
- Local sorting
- Merging tries
- Generating global dictionary
- Mapping initial runs to global ids
- Merging initial runs
- Generating evaluation indices

Time in Seconds vs. Size of RDF Blocks (Number of Triples)
Index construction Total Time

![Index construction Total Time diagram]

- Building patricia tries
- Mapping to local IDs
- Local sorting
- Generating global dictionary
- Mapping initial runs to global ids
- Merging initial runs
- Generating evaluation indices

Size of RDF Blocks (Number of Triples):
- 1 M
- 5 M
- 10 M
- 25 M
- 50 M
- 100 M

Construction Time in Seconds:
- 0
- 5000
- 10000
- 15000
- 20000
- 25000
- 30000
- 35000
- 40000
- 45000

D. Heinrich
Parallel Execution Approaches on Data and Index Structures in the Context of SW DBMS

18/18