Hash Joins for Multi-core CPUs

Benjamin Wagner
Joins

- fundamental operator in query processing
- variety of different algorithms
- many papers publishing different results
- main question: is tuning to modern hardware worth it?
- goal: perform own benchmarks on these algorithms
- only main memory hash joins are considered
Problem Statement

- two relations \( R = [\text{value, ID}] \), \( S = [\text{value, ID}] \); usually \( |R| \leq |S| \)
- \( R \) is build relation, \( S \) is probe relation
- joining the relations on "value" to produce triples \([\text{value, idR, idS}]\)
- \( R \bowtie \bowtie p S = \{ x \circ y | x \in R \land y \in S \land p(x, y) \} \) with \( p = "R.\text{value}=S.\text{value}" \)
- performance: bag instead of set semantics
No Partitioning Join (NOP)

- build shared hash table with relation R
- probe the table with tuples from S
Radix Join (RPJ)

- partition R and S in one or more runs
- run regular NOP Join on separate partitions

Radix Join, taken from [1]
Radix Join (RPJ)

- partitioning happens on the least significant bits of the hash
- simple example with hash(x)=x and 2 pass partitioning
- pass 1 using 3 Bits

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash</th>
<th>Bucket (Pass 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1100</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>121</td>
<td>1111001</td>
<td>1</td>
</tr>
<tr>
<td>412</td>
<td>11001100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>523</td>
<td>1000001011</td>
<td>3</td>
</tr>
<tr>
<td>672</td>
<td>10101000000</td>
<td>0</td>
</tr>
</tbody>
</table>
Radix Join (RPJ)

- partitioning happens on the least significant bits of the hash
- simple example with hash(x)=x and 2 pass partitioning
- pass 2 using 3 Bits

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash</th>
<th>Bucket (Pass 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>001100</td>
<td>4.1</td>
</tr>
<tr>
<td>16</td>
<td>010000</td>
<td>0.2</td>
</tr>
<tr>
<td>121</td>
<td>1111001</td>
<td>1.7</td>
</tr>
<tr>
<td>412</td>
<td>11001100</td>
<td>4.3</td>
</tr>
<tr>
<td>2</td>
<td>000010</td>
<td>2.0</td>
</tr>
<tr>
<td>523</td>
<td>100001011</td>
<td>3.1</td>
</tr>
<tr>
<td>672</td>
<td>1010100000</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Radix Join (RPJ)

- several knobs influencing performance
- can lead to improved data locality during the join
- parallelization is rather involved

⇒ overhead of partitioning vs data locality
Literature

- idea to use radix partitioning is fairly old (1999) [4]
- since then: variety of papers claiming different things
- some say: algorithms should be hardware conscious [1, 2, 5]
- others: modern CPUs can hide cache miss latencies [3]
Implementation & Benchmarks

- implemented single and multi-threaded versions
- multi-threaded RPJ utilizes single-threaded NOP
- benchmarks taking several parameters into account
- radix join times always under optimal parameters

<table>
<thead>
<tr>
<th>CPU:</th>
<th>Intel i9-7900X</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Cores</td>
<td>10</td>
</tr>
<tr>
<td># of Threads</td>
<td>20</td>
</tr>
<tr>
<td>Base Frequency</td>
<td>3.30 GHz</td>
</tr>
<tr>
<td>L1 Data Cache (per core)</td>
<td>32 KiB</td>
</tr>
<tr>
<td>L2 Cache (per core)</td>
<td>1 MiB</td>
</tr>
</tbody>
</table>
Benchmarks - RPJ vs NOP

S uniformly distributed, $|R| = |S| \approx 16.8M$
S uniformly distributed, $|R| \approx 65k; |S| \approx 33.6M$
Benchmarks - RPJ resilience

S uniformly distributed, $|R| = |S| \approx 16.8M$
Benchmarks - High Skew

S zipf distributed, $|R| \approx 65k; |S| \approx 33.6M$
Benchmarks - High Skew

S zipf distributed, $|R| = |S| \approx 16.8M
References

- Balkesen et al. “Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underlying Hardware”. In: ICDE (2013).
- Dittrich Schuh Chen. “An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory”. In: SIGMOD (2016).
Questions?