Motivation

- High Level Engines – SQL like, easy to use
- Low Level Engines – Faster/optimized, harder to write/use
- EmptyHeaded – Create an engine with the simplicity of high level engine yet the speed of low level engine
Definitions

• SIMD – Single Instruction Multiple Data
• GHD – Generalized Hypertree Decomposition
• Multiway Join - join multiple tables at same time
• Worst Case Optimal Join – optimal algorithm with worst case usage (output size of join)
Overview
Preliminaries

Compiler
Execution
Results
Worst Case Optimal Join – Fractional Cover

• Theoretical tight bound of worst case optimal join
• Hypergraph (V,E)
  • V – attribute of query
  • E – relation
• Define vector X with a component for each edge in the graph

\[ X = \langle e_1, e_2, e_3 \rangle \]
Feasible Cover

- Feasible Cover - Each vertex $v$, $\sum_{e \in E} X_e \geq 1$
- Upper bound, $OUT \leq \prod R_e^{x_e}$
- Triangle Query – A,B,C
  - $(1,1,0) \rightarrow O(N^2)$
  - $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}) \rightarrow O(N^{3/2})$

$X = <e_1, e_2, e_3>$
Input Data

<table>
<thead>
<tr>
<th>Original Relation</th>
<th>Dictionary Encoding</th>
<th>Trie Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manages</td>
<td>ID Map</td>
<td>0</td>
</tr>
<tr>
<td>managerID</td>
<td>employeeID</td>
<td>employeeRating</td>
</tr>
<tr>
<td>10</td>
<td>543</td>
<td>1.7</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>3.8</td>
</tr>
<tr>
<td>10</td>
<td>300</td>
<td>9.5</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>6.4</td>
</tr>
<tr>
<td>543</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Queries

- Triangle
- 4 Clique
- Lollipop
- Barbell
Preliminaries

Compiler

Execution

Results
GHD

(a) Hypergraph

(b) LogicBlox GHD

(c) EmptyHeaded GHD

$(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2})$

$O(N^3)$

$O(N^{3/2} + \text{OUT})$
Push Down

• Within Node
  • Reorder attributes to allow early termination in trie \((x,x')\)  
    \(\rightarrow (x',x)\)

• Across Node
  • High selectivity nodes at bottom
  • Choose lowest width GHD’s – fractional hypertree width
    • \(O(N^{fhw})\)
  • If A covers unselected attributes of B, add B as child of A
  • Maximize the depth (sum of heights) of \(fhw\) GHD trees

• Up to around \(10^4\) speedup
Redundancies

• 2 Nodes equivalent if
  • do the same join on same input
  • do same aggregation, selection, projection
  • have same subtree result

• 2x increase in Barbell Query
Layouts

• Uint – efficient sparse data representation
• BitSet – good parallelism for dense data

• Pshort
• Varint
• Bitpacked
Bitset

• Set of pairs (offset, bit vector)
• Offset is index of the smallest value in the vector

• High parallelism
  • Intersect 2 bitsets
    • uint intersection of offsets to find potential block match
    • SIMD intersection of blocks
  • Instead of 4 element in SIMD reg, up to 256 elements
Pshort

- Values close to each other share prefix

- 3 values share prefix 0x10000
  - 96 bits vs 80 bits

![Table]

\[S = \{65536, 65636, 65736\}\]
Varint

• Value differences encoded
  • Bottom 7 bits: store data
  • 8th bit: data extends to next byte or not

• Good for dense, large data

\[
S = \{0, 2, 4\} \quad Diff = \{0, 2, 2\}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>31</th>
<th>32</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>54</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
|S| \delta_{1}[6..0] \quad c \quad \delta_{2}[6..0] \quad c \quad \delta_{3}[6..0] \quad c
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
\]
Bitpacked

- Partition into blocks, compress each block
- Can compute differences in parallel SIMD
- Pack the difference into minimum block width

\[ S = \{0, 2, 8\}, \quad \text{Diff} = \{0, 2, 6\} \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>31</th>
<th>32</th>
<th>39</th>
<th>40</th>
<th>42</th>
<th>43</th>
<th>45</th>
<th>46</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>S</td>
<td>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{bits/ele}m)</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\delta_1[2..0])</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\delta_2[2..0])</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\delta_3[2..0])</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Density Skew

• Varint and Bitpacked decoding takes too long
• Pshort hard to convert/not compatible with other representation

• Relation (Graph) Level
  • Sparse – uint

• Set Level (Vertex)
  • Sparse – uint, dense – bitset

• Block Level (Blocks in set)
  • Sparse – uint, dense – bitset
Density Skew – Optimize

• Relation Level – doesn’t optimize for density at all, 7.3x slower
• Set Level – at most 1.6x slower than optimal
• Block Level – at most 3.2x slower
  • Need to call more intersections and merge
  • 2.5x overhead

• Set Optimizer
  • Dense – Bitset if each value fits into SIMD register space
  • Sparse – Uint if values greater than that
Intersections – uint*uint

• SIMDShuffling – compare pairs of blocks in sets
• V1 – iterate through smaller set, SIMD comparison with larger
• V3 – V1 but do binary search on 4 blocks
• SIMDGalloping – V1 but do scalar binary search
• BMiss – SIMD to compare parts, then scalar comparison for full match
SIMD Shuffling

\[ O(N^2) \]
V1

- A, B sorted
- Uint a, Block b
- Find block where last element in b greater than a
- SIMD Comparison to find match
V3

- A, B sorted
- Uint a, Block b
- Find group of 4 blocks where last element greater than a
- Binary Search the 4 blocks
- SIMD Comparison to find match
SIMD Galloping

• A,B sorted
• Uint a, Block b
• Check block groups of exponential size (1,2,4,...)
• Binary search group
• SIMD Comparison
Cardinality Skew

• Set Cardinality difference – difference in size between sets
• Galloping algorithms work well when one set much smaller than the other
• Have inherent overhead over normal algorithms
• Use SIMDShuffling by default, and SIMDGalloping if cardinality ratio over 1:32
Node Ordering

- Random
- BFS
- Strong Runs – BFS starting at highest degree node
- Degree
- Rev-Degree
- Shingle – order by similar neighbors

- Selecting Intersection and Layout has greater effect, don’t care about ordering
Summary of EmptyHeaded Optimizations

- GHD Ordering
  - Attribute (within GHD node)
  - GHD (across GHD node)
- Layout (Dense vs. Sparse)
- Intersection (Shuffling vs. Galloping)
Experiment

• Dataset
  • Low Density Skew – LiveJournal, Orkut, Patents
  • Medium Density – Twitter, Higgs
  • High Density – Google+

• Low-Level Engines – PowerGraph, CGT-X, Snap-R
• High Level Engines – LogicBlox, SocialLite
## Results – Triangle Counting

<table>
<thead>
<tr>
<th>Dataset</th>
<th>EmptyHeaded</th>
<th>Low-Level</th>
<th></th>
<th>High-Level</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PowerGraph</td>
<td>CGT-X</td>
<td>Snap-Ringo</td>
<td>SociaLite</td>
</tr>
<tr>
<td>Google+</td>
<td><strong>0.31</strong></td>
<td>8.40×</td>
<td>62.19×</td>
<td>4.18×</td>
<td>1390.75×</td>
</tr>
<tr>
<td>Higgs</td>
<td><strong>0.15</strong></td>
<td>3.25×</td>
<td>57.96×</td>
<td>5.84×</td>
<td>387.41×</td>
</tr>
<tr>
<td>LiveJournal</td>
<td><strong>0.48</strong></td>
<td>5.17×</td>
<td>3.85×</td>
<td>10.72×</td>
<td>225.97×</td>
</tr>
<tr>
<td>Orkut</td>
<td><strong>2.36</strong></td>
<td>2.94×</td>
<td>-</td>
<td>4.09×</td>
<td>191.84×</td>
</tr>
<tr>
<td>Patents</td>
<td><strong>0.14</strong></td>
<td>10.20×</td>
<td>7.45×</td>
<td>22.14×</td>
<td>49.12×</td>
</tr>
<tr>
<td>Twitter</td>
<td><strong>56.81</strong></td>
<td>4.40×</td>
<td>-</td>
<td>2.22×</td>
<td>t/o</td>
</tr>
</tbody>
</table>
## Results – Optimizations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>-SIMD</th>
<th>-Representation</th>
<th>-SIMD &amp; Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google+</td>
<td>1.0×</td>
<td>3.0×</td>
<td>7.5×</td>
</tr>
<tr>
<td>Higgs</td>
<td>1.5×</td>
<td>3.9×</td>
<td>4.8×</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>1.6×</td>
<td>1.0×</td>
<td>1.6×</td>
</tr>
<tr>
<td>Orkut</td>
<td>1.8×</td>
<td>1.1×</td>
<td>2.0×</td>
</tr>
<tr>
<td>Patents</td>
<td>1.3×</td>
<td>0.9×</td>
<td>1.1×</td>
</tr>
</tbody>
</table>
Galois Results – PageRank, SSSP

• Galois
  • PageRank
    • Around 2-3x faster, 5x on Google+
    • 271 lines vs. EmptyHeaded 3
  • SSSP
    • 2-30x faster
    • 172 lines vs. EmptyHeaded 2
RDF

• Subject -> Predicate -> Object
• Extra Optimization - Pipelining
  • Since triples may share many common subject prefixes etc.
  • Can Pipeline GHD
## Performance

<table>
<thead>
<tr>
<th>Query</th>
<th>Best</th>
<th>EmptyHeaded</th>
<th>TripleBit</th>
<th>RDF-3X</th>
<th>MonetDB</th>
<th>LogicBlox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>4.00</td>
<td>1.51×</td>
<td>3.45×</td>
<td>1.00×</td>
<td>174.58×</td>
<td>8.62×</td>
</tr>
<tr>
<td>Q2</td>
<td>973.95</td>
<td>1.00×</td>
<td>2.38×</td>
<td>1.92×</td>
<td>8.79×</td>
<td>1.52×</td>
</tr>
<tr>
<td>Q3</td>
<td>0.47</td>
<td>1.00×</td>
<td>92.61×</td>
<td>8.44×</td>
<td>283.37×</td>
<td>83.41×</td>
</tr>
<tr>
<td>Q4</td>
<td>3.39</td>
<td>4.62×</td>
<td>1.00×</td>
<td>1.77×</td>
<td>2093.78×</td>
<td>116.32×</td>
</tr>
<tr>
<td>Q5</td>
<td>0.44</td>
<td>1.00×</td>
<td>99.21×</td>
<td>9.15×</td>
<td>303.11×</td>
<td>81.44×</td>
</tr>
<tr>
<td>Q7</td>
<td>6.00</td>
<td>3.18×</td>
<td>8.53×</td>
<td>1.00×</td>
<td>573.33×</td>
<td>6.52×</td>
</tr>
<tr>
<td>Q8</td>
<td>78.50</td>
<td>9.83×</td>
<td>1.00×</td>
<td>3.07×</td>
<td>206.62×</td>
<td>5.03×</td>
</tr>
<tr>
<td>Q9</td>
<td>581.37</td>
<td>1.00×</td>
<td>3.53×</td>
<td>6.63×</td>
<td>24.29×</td>
<td>1.35×</td>
</tr>
<tr>
<td>Q11</td>
<td>0.45</td>
<td>1.00×</td>
<td>6.07×</td>
<td>11.03×</td>
<td>58.63×</td>
<td>73.76×</td>
</tr>
<tr>
<td>Q12</td>
<td>3.05</td>
<td>2.22×</td>
<td>1.00×</td>
<td>7.86×</td>
<td>118.94×</td>
<td>50.23×</td>
</tr>
<tr>
<td>Q13</td>
<td>0.87</td>
<td>1.00×</td>
<td>48.90×</td>
<td>35.49×</td>
<td>86.18×</td>
<td>102.77×</td>
</tr>
<tr>
<td>Q14</td>
<td>3.00</td>
<td>1.90×</td>
<td>54.47×</td>
<td>1.00×</td>
<td>313.47×</td>
<td>325.02×</td>
</tr>
</tbody>
</table>
## Optimizations

<table>
<thead>
<tr>
<th>Query</th>
<th>+Layout</th>
<th>+Attribute</th>
<th>+GHD</th>
<th>+Pipelining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2.10×</td>
<td>129.85×</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q2</td>
<td>8.22×</td>
<td>1.03×</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q4</td>
<td>2.02×</td>
<td>12.88×</td>
<td>69.94×</td>
<td>-</td>
</tr>
<tr>
<td>Q7</td>
<td>4.35×</td>
<td>95.01×</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Q8</td>
<td>2.24×</td>
<td>1.99×</td>
<td>1.5×</td>
<td>4.67×</td>
</tr>
<tr>
<td>Q14</td>
<td>7.92×</td>
<td>234.49×</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>