Speedup Graph Processing by Graph Ordering

Presenter: Haonan Wang

haonanw@mit.edu

April 23, 2019
Overview

1 Background
   - Motivation
   - Graph Access Patterns

2 Algorithm
   - The GO Algorithm
   - GO-PQ

3 Evaluation
Section 1

Background
Subsection 1

Motivation
Graphs are important

- Graph is a popular data model for Big Data
  social graph, web graph, knowledge graph
- Large graph is emerging with increasing size
  Friendship connection in Facebook, Twitter, .......

CPU cache performance is key issue in efficiency in DBS

- cache miss latency takes a half of the execution time in database systems

Design general optimization approach for graph model
The cost of cache miss

(a) The original order

(b) Gorder
Subsection 2

Graph Access Patterns
Common Ideas About Graph Access

1: for each node \( v \in N_O(u) \) do
2: the program segment to compute/access \( v \)

Common Relationships: Neighbours / Sibling
Key Idea here is that sibling relationship is a dominating factor.

\[
\left( \frac{d_O(u)}{2} \right) \gg d_O(u)
\]
Measure the Closeness of two points:

\[ S(u, v) = S_s(u, v) + S_n(u, v) \]

Our problems turn into a permutation problem!
Section 2

Algorithm
Some Key Ideas About The Algorithms

Theorem 2.1: Maximizing $F(\phi)$ to obtain an optimal permutation $\phi(\cdot)$ for a directed graph $G$ is \textit{NP}-hard.

The proof sketch is given in Appendix.

- All algorithms here are all approximate algorithms.
- If we set $w = 1$, the problem here is actually equivalent to the maximum traveling salesman problem, denoted as $\text{maxTSP}$ for short.
Algorithm 1 GO ($G, w, S(\cdot, \cdot)$)

1: select a node $v$ as the start node, $P[1] \leftarrow v$;
2: $V_R \leftarrow V(G) \setminus \{v\}$, $i \leftarrow 2$;
3: while $i \leq n$ do
4: \hspace{1em} $v_{\max} \leftarrow \emptyset$, $k_{\max} \leftarrow -\infty$;
5: \hspace{1em} for each node $v \in V_R$ do
6: \hspace{2em} $k_v \leftarrow \sum_{j=\max\{1, i-w\}}^{i-1} S(P[j], v)$;
7: \hspace{1em} if $k_v > k_{\max}$ then
8: \hspace{2em} $v_{\max} \leftarrow v$, $k_{\max} \leftarrow k_v$;
9: \hspace{2em} $P[i] \leftarrow v_{\max}$, $i \leftarrow i + 1$;
10: $V_R \leftarrow V_R \setminus \{v_{\max}\}$;
**Theorem 3.1:** The algorithm GO gives $\frac{1}{2w}$-approximation for maximizing $F(\phi)$ to determine the optimal graph ordering.

<table>
<thead>
<tr>
<th></th>
<th>$w = 3$</th>
<th></th>
<th>$w = 5$</th>
<th></th>
<th>$w = 7$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$F_{go}$</td>
<td>$\overline{F}_w$</td>
<td>$F_{go}$</td>
<td>$\overline{F}_w$</td>
<td>$F_{go}$</td>
<td>$\overline{F}_w$</td>
</tr>
<tr>
<td>Facebook</td>
<td>149,073</td>
<td>172,526</td>
<td>231,710</td>
<td>275,974</td>
<td>308,091</td>
<td>373,685</td>
</tr>
<tr>
<td>AirTraffic</td>
<td>2,420</td>
<td>3,468</td>
<td>2,993</td>
<td>4,697</td>
<td>3,465</td>
<td>5,545</td>
</tr>
</tbody>
</table>

**Table 1:** $F_{go}$ and $\overline{F}_w$
The Priority Queue based Algorithm

- Go is expensive.
  Time complexity:
  \[ O(w \cdot d_{max} \cdot n^2) \]

- Reasons for the inefficiency
  1. Repeatedly computes score function \( w \) times for the same pair \((v_j, v)\) while \( v_j \) is in the window of size \( w \)
  2. It scans every node \( v \) in the set of remaining nodes \( V_R \) in every iteration
Explain For What we do in the new algorithm
Algorithm 2 GO-PQ \((G, w, S(\cdot, \cdot))\)

1: for each node \(v \in V(G)\) do
2: \hspace{1em} insert \(v\) into \(Q\) such that \(\text{key}(v) \leftarrow 0\);
3: select a node \(v\) as the start node, \(P[1] \leftarrow v\), delete \(v\) from \(Q\);
4: \(i \leftarrow 2\);
5: while \(i \leq n\) do
6: \hspace{1em} \(v_e \leftarrow P[i - 1]\);
7: \hspace{2em} for each node \(u \in N_O(v_e)\) do
8: \hspace{3em} if \(u \in Q\) then \(Q.\text{incKey}(u)\);
9: \hspace{2em} for each node \(u \in N_I(v_e)\) do
10: \hspace{3em} if \(u \in Q\) then \(Q.\text{incKey}(u)\);
11: \hspace{3em} for each node \(v \in N_O(u)\) do
12: \hspace{4em} if \(v \in Q\) then \(Q.\text{incKey}(v)\);
13: \hspace{2em} if \(i > w + 1\) then
14: \hspace{3em} \(v_b \leftarrow P[i - w - 1]\);
15: \hspace{2em} for each node \(u \in N_O(v_b)\) do
16: \hspace{3em} if \(u \in Q\) then \(Q.\text{decKey}(u)\);
17: \hspace{2em} for each node \(u \in N_I(v_b)\) do
18: \hspace{3em} if \(u \in Q\) then \(Q.\text{decKey}(u)\);
19: \hspace{3em} for each node \(v \in N_O(u)\) do
20: \hspace{4em} if \(v \in Q\) then \(Q.\text{decKey}(v)\);
21: \hspace{3em} \(v_{max} \leftarrow Q.\text{pop}()\);
22: \hspace{1em} \(P[i] \leftarrow v_{max}, i \leftarrow i + 1\);
Figure 7: The Priority Queue: $Q$, $Q_h$, and $\text{top}$
Lazy Update, use key and update instead of key
Maintain the double link list until there should be changes
Use the head and end to get the position for the first element and the last element of the subqueue with same key
Algorithm 3 $\text{decKey} (v_i)$

1: $\text{update}(v_i) \leftarrow \text{update}(v_i) - 1;$
Algorithm 4 incKey \((v_i)\)

1: \(\text{update}(v_i) \leftarrow \text{update}(v_i) + 1;\)
2: \(\textbf{if} \ \text{update}(v_i) > 0 \ \textbf{then}\)
3: \(\text{update}(v_i) \leftarrow 0, x \leftarrow \text{key}(v_i), \text{key}(v_i) \leftarrow \text{key}(v_i) + 1;\)
4: \(\text{delete } v_i \text{ from } Q;\)
5: \(\text{insert } v_i \text{ into } Q \text{ in the position just before } \text{head}[x];\)
6: \(\text{update the head } Q_h \text{ array accordingly};\)
7: \(\textbf{if} \ \text{key}(v_i) > \text{key}(\text{top}) \ \textbf{then}\)
8: \(\text{top } \leftarrow v_i;\)
Algorithm 5 pop ()

1: while update(top) < 0 do
2:     \( v_t \leftarrow \text{top}; \)
3:     key(\( v_t \)) \leftarrow \text{key}(\( v_t \)) + \text{update}(\( v_t \));
4:     update(\( v_t \)) \leftarrow 0;
5:     if key(top) \leq \text{key}(\text{next}(\text{top})) then
6:         adjust the position of \( v_t \) and insert \( v_t \) just after \( u \) in \( Q \), such that \( \text{key}(u) \geq \text{key}(\text{top}) \) and \( \text{key}(\text{next}(u)) < \text{key}(\text{top}) \);
7:     top \leftarrow \text{next}(\text{top});
8:     update the head array;
9:     \( v_t \leftarrow \text{top}; \)
10: remove the node pointed by top from \( Q \) and update \( \text{top} \leftarrow \text{next}(\text{top}); \)
11: return \( v_t \);
Section 3
Evaluation
(a) $F(\cdot)$ by Different Orderings
<table>
<thead>
<tr>
<th>Order</th>
<th>L1-ref</th>
<th>L1-mr</th>
<th>L3-ref</th>
<th>L3-r</th>
<th>Cache-mr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>11,109M</td>
<td>52.1%</td>
<td>2,195M</td>
<td>19.7%</td>
<td>5.1%</td>
</tr>
<tr>
<td>MINLA</td>
<td>11,110M</td>
<td>58.1%</td>
<td>2,121M</td>
<td>19.0%</td>
<td>4.5%</td>
</tr>
<tr>
<td>MLOGA</td>
<td>11,119M</td>
<td>53.1%</td>
<td>1,685M</td>
<td>15.1%</td>
<td>4.1%</td>
</tr>
<tr>
<td>RCM</td>
<td>11,102M</td>
<td>49.8%</td>
<td>1,834M</td>
<td>16.5%</td>
<td>4.1%</td>
</tr>
<tr>
<td>DegSort</td>
<td>11,121M</td>
<td>58.3%</td>
<td>2,597M</td>
<td>23.3%</td>
<td>5.3%</td>
</tr>
<tr>
<td>CHDFS</td>
<td>11,107M</td>
<td>49.9%</td>
<td>1,850M</td>
<td>16.7%</td>
<td>4.4%</td>
</tr>
<tr>
<td>SlashBurn</td>
<td>11,096M</td>
<td>55.0%</td>
<td>2,466M</td>
<td>22.2%</td>
<td>4.3%</td>
</tr>
<tr>
<td>LDG</td>
<td>11,112M</td>
<td>52.9%</td>
<td>2,256M</td>
<td>20.3%</td>
<td>5.4%</td>
</tr>
<tr>
<td>METIS</td>
<td>11,105M</td>
<td>50.3%</td>
<td>2,235M</td>
<td>20.1%</td>
<td>5.2%</td>
</tr>
<tr>
<td>Gorder</td>
<td>11,101M</td>
<td>37.9%</td>
<td>1,280M</td>
<td>11.5%</td>
<td>3.4%</td>
</tr>
</tbody>
</table>

Table 3: Cache Statistics by PR over Flickr (M = Millions)
<table>
<thead>
<tr>
<th>Order</th>
<th>NQ</th>
<th>BFS</th>
<th>DFS</th>
<th>SCC</th>
<th>SP</th>
<th>PR</th>
<th>DS</th>
<th>Kcore</th>
<th>Diam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>50.8</td>
<td>15.3</td>
<td>5.4</td>
<td>7.8</td>
<td>21.5</td>
<td>52.1</td>
<td>21.9</td>
<td>20.8</td>
<td>14.9</td>
</tr>
<tr>
<td>MINLA</td>
<td>51.8</td>
<td>18.0</td>
<td>5.5</td>
<td>8.1</td>
<td>24.6</td>
<td>58.1</td>
<td>22.1</td>
<td>21.5</td>
<td>17.9</td>
</tr>
<tr>
<td>MLOGA</td>
<td>41.7</td>
<td>16.3</td>
<td>5.1</td>
<td>7.2</td>
<td>21.9</td>
<td>53.1</td>
<td>21.1</td>
<td>20.6</td>
<td>16.4</td>
</tr>
<tr>
<td>RCM</td>
<td>49.1</td>
<td>12.1</td>
<td>4.6</td>
<td>6.6</td>
<td>15.9</td>
<td>49.7</td>
<td>20.3</td>
<td>20.2</td>
<td>12.4</td>
</tr>
<tr>
<td>DegSort</td>
<td>45.7</td>
<td>16.7</td>
<td>4.8</td>
<td>7.0</td>
<td>24.9</td>
<td>58.3</td>
<td>21.4</td>
<td>18.6</td>
<td>17.0</td>
</tr>
<tr>
<td>CHDFS</td>
<td>42.1</td>
<td>12.3</td>
<td>4.1</td>
<td>5.8</td>
<td>18.5</td>
<td>49.9</td>
<td>21.1</td>
<td>20.6</td>
<td>12.9</td>
</tr>
<tr>
<td>SlashBurn</td>
<td>46.2</td>
<td>16.0</td>
<td>4.5</td>
<td>6.2</td>
<td>22.1</td>
<td>55.0</td>
<td>20.7</td>
<td>21.3</td>
<td>15.8</td>
</tr>
<tr>
<td>LGD</td>
<td>50.7</td>
<td>15.9</td>
<td>5.8</td>
<td>8.2</td>
<td>21.8</td>
<td>52.9</td>
<td>22.4</td>
<td>21.2</td>
<td>14.9</td>
</tr>
<tr>
<td>METIS</td>
<td>63.0</td>
<td>18.2</td>
<td>7.7</td>
<td>10.1</td>
<td>20.8</td>
<td>50.3</td>
<td>23.0</td>
<td>21.7</td>
<td>16.7</td>
</tr>
<tr>
<td>Gorder</td>
<td>35.4</td>
<td>11.1</td>
<td>3.6</td>
<td>5.2</td>
<td>12.8</td>
<td>37.9</td>
<td>18.7</td>
<td>18.1</td>
<td>10.9</td>
</tr>
</tbody>
</table>

**Table 6: L1 Cache Miss Ratio on Flickr (in percentage %)**
## Table 10: Running Time of *Diam* (in second)

<table>
<thead>
<tr>
<th>Order</th>
<th>Pokec</th>
<th>Flickr</th>
<th>LiveJ</th>
<th>wiki</th>
<th>G+</th>
<th>pld</th>
<th>twitter</th>
<th>sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1,187</td>
<td>750</td>
<td>3,040</td>
<td>10,503</td>
<td>23,128</td>
<td>21,961</td>
<td>48,238</td>
<td>50,784</td>
</tr>
<tr>
<td>MINLA</td>
<td>1,176</td>
<td>843</td>
<td>3,471</td>
<td>10,322</td>
<td>25,543</td>
<td>20,698</td>
<td>44,536</td>
<td>48,218</td>
</tr>
<tr>
<td>MLOGA</td>
<td>1,181</td>
<td>787</td>
<td>3,427</td>
<td>10,121</td>
<td>24,229</td>
<td>20,829</td>
<td>45,698</td>
<td>48,899</td>
</tr>
<tr>
<td>RCM</td>
<td>1,091</td>
<td>673</td>
<td>2,883</td>
<td>7,272</td>
<td>20,371</td>
<td>18,657</td>
<td>40,730</td>
<td>36,334</td>
</tr>
<tr>
<td>DegSort</td>
<td>1,188</td>
<td>815</td>
<td>3,281</td>
<td>9,500</td>
<td>24,799</td>
<td>21,278</td>
<td>44,228</td>
<td>47,723</td>
</tr>
<tr>
<td>CHDFS</td>
<td>1,107</td>
<td>690</td>
<td>2,934</td>
<td>7,600</td>
<td>21,150</td>
<td>17,732</td>
<td>39,517</td>
<td>36,585</td>
</tr>
<tr>
<td>SlashBurn</td>
<td>1,219</td>
<td>810</td>
<td>3,452</td>
<td>10,031</td>
<td>24,616</td>
<td>21,564</td>
<td>44,261</td>
<td>45,134</td>
</tr>
<tr>
<td>LDG</td>
<td>1,168</td>
<td>793</td>
<td>2,940</td>
<td>10,137</td>
<td>23,569</td>
<td>22,740</td>
<td>47,841</td>
<td>55,234</td>
</tr>
<tr>
<td>METIS</td>
<td>1,131</td>
<td>843</td>
<td>3,232</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gorder</td>
<td>1,003</td>
<td>620</td>
<td>2,556</td>
<td>5,932</td>
<td>17,936</td>
<td>14,389</td>
<td>32,808</td>
<td>30,202</td>
</tr>
</tbody>
</table>
The End