How Do Graph Relabeling Algorithms Improve Memory Locality?


Published in:

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
Copyright 2021 IEEE. This work is made available online in accordance with the publisher’s policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
How Do Graph Relabeling Algorithms Improve Memory Locality?

Mohsen Koohi Esfahani  
0000-0002-7465-8003  
LW A  
Peter Kilpatrick  
0000-0003-0818-8979  
Queen’s University Belfast, UK  
{mkooohiesfahani01, p.kilpatrick, h.vandierendonck}@qub.ac.uk  
https://blogs.qub.ac.uk/GraphProcessing/LaganLighter

Abstract—Relabeling algorithms aim to improve the poor memory locality of graph processing by changing the order of vertices. This paper analyses the functionality of three state-of-the-art relabeling algorithms: SlashBurn, GOrder, and Rabbit-Order for real-world graphs.

Index Terms—Graph processing, Memory locality, High performance computing, Graph traversal, Relabeling algorithm, Data analysis, Graph reordering

I. METHODOLOGY

We investigate three relabeling algorithms: GOrder (GO) [1] (commit 7cddfe9), Rabbit-Order (RO) [2] (commit f67a79e), and SlashBurn (SB) [3] (C implementation of basic hub-ordering and 0.02|V|) for Sparse Matrix-Vector (SpMV) multiplication graph traversal using CSC and CSR graph representations [4], with |V| + 1 index values of 8 bytes per index value and |E| neighbour IDs, each requiring 4 bytes. Experiments are performed on a 2-socket machine with 768 GB main memory and each socket has an Intel® Xeon® Gold 6130. Figure 1 and Table III use the last level cache simulator of SimpleScalar [5] with dueling RRIP and SRRIP [6]. The simulation has an average 10.6% error of total misses compared to the real hardware. Table I shows the datasets and their source: Konect (KN) [7]–[9], Network Repository (NR) [10]–[14], or Laboratory of Web Algorithmics (LWA) [9]. [14]–[17].

TABLE I: Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
<th></th>
<th></th>
<th>Bits</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>Twitter MPI</td>
<td>41</td>
<td>1</td>
<td>51</td>
<td>1</td>
<td>Web-Graph</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Friendster</td>
<td>65</td>
<td>1.8</td>
<td>40</td>
<td>2</td>
<td>Social Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SK</td>
<td>90</td>
<td>2</td>
<td>30</td>
<td>2</td>
<td>Web-Graph</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SK-Domain</td>
<td>89</td>
<td>2</td>
<td>30</td>
<td>1</td>
<td>Web-Graph</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CC12</td>
<td>133</td>
<td>5.5</td>
<td>90</td>
<td>4</td>
<td>Web-Graph</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Web-G</td>
<td>105</td>
<td>6.6</td>
<td>21</td>
<td>6</td>
<td>Web-Graph</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CC12</td>
<td>17K</td>
<td>7.9</td>
<td>68</td>
<td>9</td>
<td>Web-Graph</td>
<td></td>
</tr>
</tbody>
</table>

II. CACHE MISS RATE DEGREE DISTRIBUTION

The cache miss rate of a vertex is defined as the percentage of accesses to its in-neighbours’ vertex data that are missed by cache. Figure 1 compares the degree distribution of the last level cache miss rate for a social network (Twtr) and a web graph (UU). It shows that the locality of hubs as very high-degree vertices is not improved by relabeling algorithms as much as other vertices.

Fig. 1: Degree distribution of miss rate (in percent)

Fig. 2: Degree distribution of GCC after SB iterations

SB considers hubs of a graph as the main connector of nodes and detects communities by removing hubs and finding the connected components (communities). This process continues in the next iteration for the giant connected component (GCC) - the community with the largest number of edges. The practicality of this method depends on whether power-law graphs are made/destroyed recursively. Figure 2 investigates this theory by showing the degree distribution of GCC for different iterations of SlashBurn and shows that over different iterations of SlashBurn, the degree distribution of the GCC does not maintain the power-law property. Instead, after a few iterations, the remaining network shows an almost-uniform degree distribution with low degrees. Further iterations of the SlashBurn algorithm separate these low-degree vertices (LDV) from their parents in what are perceived as different communities. As a result, SB reduces the locality of LDV, especially in web graphs (Figure 12). To counter this, we propose a variation on SB (called SB++), that continues the iterations while GCC-max-degree ≥ |V|. SB++ has smaller preprocessing-time, and moreover it has better traversal time and L3 misses than SlashBurn (Table II).
TABLE II: Stopping SB after initial iterations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Pre-processing Time (s)</th>
<th>Traversal Time (ms)</th>
<th>L3 Misses (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twtr</td>
<td>SB</td>
<td>SB++</td>
<td>SB</td>
</tr>
<tr>
<td>Frndstr</td>
<td>46</td>
<td>21</td>
<td>339</td>
</tr>
<tr>
<td>WbCc</td>
<td>75</td>
<td>43</td>
<td>761</td>
</tr>
<tr>
<td>WbCc</td>
<td>81</td>
<td>39</td>
<td>414</td>
</tr>
</tbody>
</table>

IV. GORDER

GO prioritizes neighbours of vertices by defining a “score” function between two vertices that is sum of the sibling score (the number of common in-neighbours) and the neighbourhood score (the number of edges between them). GO starts from the vertex with the highest degree and uses a sliding window (of size 5) to find the vertex (between neighbours of recently assigned IDs) with the maximum score to assign the next ID.

GO considers common neighbours with only a limited number of already-placed vertices. There are numerous LDV in power-law graphs, many of them appearing equally “close” to the 5 last labeled vertices. This is reflected in Figure 1 where GO decreases cache miss rate well on high-degree vertices (HDV) and can not improve locality of the LDV. Table III compares the number of L3 misses for reading the vertex data of HDV. GO and SB have the lowest reloads of HDV. For Twtr and Frndstr, SB has lower reloads of vertices with degree > 2000, but for vertices with degree > 20, GO has the lower reloads. As such, GO increases the reload of HDV for allocating cache space to vertices with lower degree but with more temporal reuse for small durations of processing. This reduces the total misses by reducing the reload of LDV that are exponentially more frequent.

TABLE III: L3 misses (in millions)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Min. Degree</th>
<th>Initial</th>
<th>SB</th>
<th>GO</th>
<th>RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebB</td>
<td>2000</td>
<td>10</td>
<td>21</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Twtr</td>
<td>2000</td>
<td>8</td>
<td>0.4</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Frndstr</td>
<td>2000</td>
<td>2</td>
<td>0.04</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Frndstr</td>
<td>2000</td>
<td>1,177</td>
<td>1,110</td>
<td>818</td>
<td>1,060</td>
</tr>
<tr>
<td>SK</td>
<td>100</td>
<td>8</td>
<td>26</td>
<td>7</td>
<td>11</td>
</tr>
</tbody>
</table>

V. RABBIT-ORDER

RO starts from the vertices with lowest degree and searches for the neighbours where maximum “gain” can be obtained by merging. This process continues while gain is positive: otherwise, the community has been formed. Finally RO applies Depth First Search (DFS) for assigning IDs to vertices of a community. DFS tends to assign spatially close IDs between neighbours LDV. However, HDV have many neighbours that are strongly dispersed through the tree.

To investigate the functionality of RO, we use Average ID Distance (AID) that is defined as the average difference between the ID of a vertex(v) and its neighbours(Nv): AIDv = (Σu∈Nv |v - u|)/|Nv|. The AID is meaningful for assessing locality of clustering relabeling algorithms: if neighbours become closer to each other, it is more likely to have the vertex data of neighbours in cache while processing other neighbours. Figure 3 shows the AID degree distribution of a social network (Twtr) and a web graph (UDD). It shows that RO reduces the AID of LDV and provides better locality for LDV (Figure 1). For HDV in social networks, RO cannot detect distinct communities and AID is not reduced.

VI. PUSH VS. PULL LOCALITY

Push and pull traversals have been studied in [19]; however, it is necessary to consider the CSC and CSR traversals in the first step and then identify how read and write affects the CSC/CSR traversal. Table IV shows there is a fundamental difference between CSC and CSR traversals for different graphs beyond the specific characteristics of graph analytics.

TABLE IV: CSR vs. CSC traversal

<table>
<thead>
<tr>
<th>Dataset</th>
<th>L3 Misses (K)</th>
<th>Traversal Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebB</td>
<td>4,345</td>
<td>3,843</td>
</tr>
<tr>
<td>Twtr</td>
<td>15,706</td>
<td>21,653</td>
</tr>
<tr>
<td>WbCc</td>
<td>5,743</td>
<td>4,619</td>
</tr>
<tr>
<td>UKDls</td>
<td>10,137</td>
<td>9,278</td>
</tr>
<tr>
<td>ClWb9</td>
<td>100,941</td>
<td>194</td>
</tr>
<tr>
<td>SK</td>
<td>100,941</td>
<td>2,221</td>
</tr>
</tbody>
</table>

To explain the difference between CSR and CSC traversals, we consider the number of edges that is covered by selecting different numbers of the highest degree vertices of a graph. Figure 4 shows that if 100K hubs can be kept in cache, the pull traversal of Twtr can process 44% of edges without reloading from memory, but the push traversal can process about 23%. For SK it is vice versa, and pull traversal can process only 4% of edges, while push traversal can process more than 60%. This shows that web graphs benefit from push locality because they have more powerful in-hubs than out-hubs, while social networks can benefit from pull locality because of their more powerful out-hubs.

VII. CONCLUSION

We presented a number of techniques to explain how relabeling algorithms affect locality of vertex classes. This study facilitates enhancing relabeling algorithms by displaying...
their shortcomings and requirements. We also explored differences in locality of push and pull traversals that show the importance and necessity of considering the structure of datasets in interpreting results of graph algorithms.

CODE AVAILABILITY

The additional data and code used for this paper are available in https://blogs.qub.ac.uk/GraphProcessing/how-do-graph-relabeling-algorithms-improve-memory-locality-ispass21/.

ACKNOWLEDGEMENT

We are thankful to our ISPASS reviewers for their informative and constructive feedback. We thank Jordan McComb for SkyLake cache simulation system as his Master project. We also thank Prof. Paolo Boldi, Junya Arai, and Yongsub Lim for giving guidance for WebGraph, Rabbit-Order, and SlashBurn, respectively. This work is partially supported by the High Performance Computing center of QUB and the Kelvin Burn, respectively. First author is supported by a supercomputer (EPSRC grant EP/T022175/1) and by DiPET High Performance Computing center of QUB and the Kelvin Burns, respectively. This work is partially supported by the EPSRC grant EP/T022345/1). We especially thank Tony McHale and John Conway for managing the HPDC cluster, and Vaughan Purnell and James McGroarty for managing the HPC center of QUB.

REFERENCES


