How Do Graph Relabeling Algorithms Improve Memory Locality?

Mohsen Koohi Esfahani
0000-0002-7465-8003

Peter Kilpatrick
0000-0003-0818-8979

Hans Vandierendonck
0000-0001-5868-9259

Queen's University Belfast, UK
{mkoohiesfahani01, 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 `7ccdfe9`), 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

Dataset	**Name**	**Source**	$\|V\|$ (M)	$\|E\|$ (B)	**Type**
WebB	WebBase-2001	LWA	118	1	Web Graph
Twtr	Twitter MPI	NR	41	1.5	Social Network
Frndstr	Friendster	NR	65	1.8	Social Network
SK	SK-Domain	LWA	50	2	Web Graph
WbCc	Web-CC12	NR	89	2	Web Graph
UKDls	UK-Delis	LWA	110	4	Web Graph
UU	UK-Union	LWA	133	5.5	Web Graph
UKDmn	UK-Domain	KN	105	6.6	Web Graph
ClWb9	ClueWeb09	NR	1.7K	7.9	Web Graph

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**.

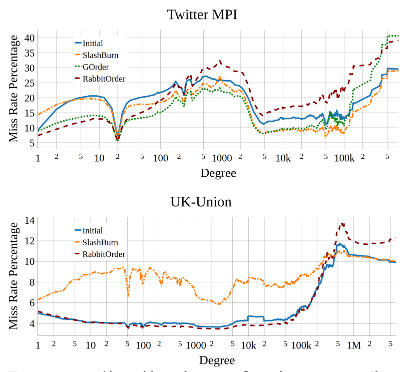

Fig. 1: Degree distribution of miss rate (in percent)

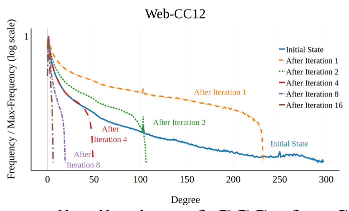

Fig. 2: Degree distribution of GCC after SB iterations

III. SlashBurn

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**) [18] from their parents in what are perceived as different communities. As a result, **SB reduces the locality of LDV, especially in web graphs** (Figure 1). To counter this, we propose a variation on SB (called SB++), that continues the iterations while GCC-max-degree $\geq \sqrt{|V|}$. SB++ has smaller preprocessing-time, and moreover it has better traversal time and L3 misses than SlashBurn (Table II).

978-1-7281-8643-6/18/©2021 IEEE
https://doi.org/10.1109/ISPASS51385.2021.00023

TABLE II: Stopping SB after initial iterations

Dataset	**Pre-processing Time** (s)		**Traversal Time** (ms)		**L3 Misses** (K)	
	SB	SB++	SB	SB++	SB	SB++
Twtr	46	21	339	328	14,218	13,607
Frndstr	75	43	761	700	39,200	36,020
WbCc	81	39	414	334	19,337	14,631

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)

Dataset	**Min. Degree**	**Initial**	**SB**	**GO**	**RO**
WebB	2000	10	21	2	10
Twtr	2000	8	0.4	4	11
Twtr	20	345	260	230	377
Frndstr	2000	2	0.04	1	3
Frndstr	20	1,177	1,110	818	1,060
SK	100	8	26	7	11

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 neighbouring LDV. However, HDV have many neighbours that are strongly dispersed through the tree.

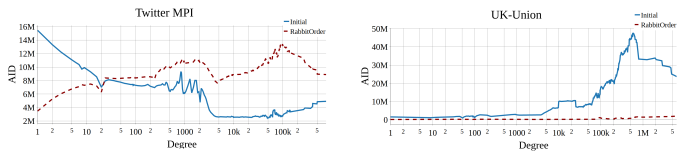

Fig. 3: AID degree distribution

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(N_v): $AID_v = (\sum_{u \in N_v} |v - u|)/|N_v|$. 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 (`UU`). 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

Dataset	**L3 Misses** (K)		**Traversal Time** (ms)	
	CSC	CSR	CSC	CSR
WebB	4,345	3,843	90	81
Twtr	15,706	21,653	354	439
SK	5,743	4,619	117	88
UKDls	10,137	9,278	194	177
ClWb9	100,941	96,490	2,221	2,129

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**.

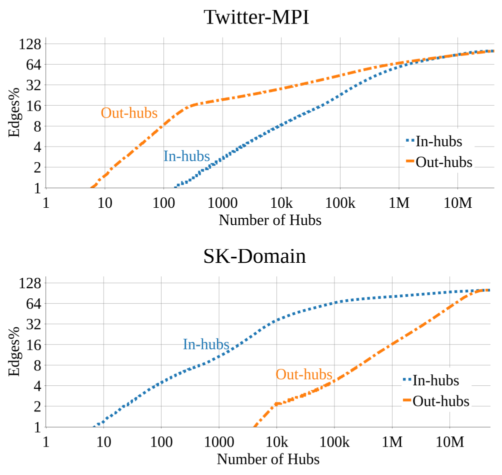

Fig. 4: Cumulative edges of in-hubs and out-hubs

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 supercomputer (EPSRC grant EP/T022175/1) and by DiPET (EPSRC grant EP/T022345/1). First author is supported by a scholarship of Queen's University Belfast and the Department for the Economy, Northern Ireland. 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

[1] H. Wei, J. X. Yu, C. Lu, and X. Lin, "Speedup graph processing by graph ordering," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. NewYork, NY, USA: ACM, 2016, pp. 1813–1828. [Online]. Available: http://doi.acm.org/10.1145/2882903.2915220

[2] J. Arai, H. Shiokawa, T. Yamamuro, M. Onizuka, and S. Iwamura, "Rabbit order: Just-in-time parallel reordering for fast graph analysis," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 22–31.

[3] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3077–3089, Dec 2014.

[4] Y. Saad, "Sparskit: a basic tool kit for sparse matrix computations - version 2," 1994.

[5] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, no. 3, p. 13–25, Jun. 1997. [Online]. Available: https://doi.org/10.1145/268806.268810

[6] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 60–71, Jun. 2010. [Online]. Available: https://doi.org/10.1145/1816038.1815971

[7] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proc. Int. Conf. on World Wide Web Companion*, 2013, pp. 1343–1350. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488173

[8] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 29–42. [Online]. Available: https://doi.org/10.1145/1298306.1298311

[9] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[10] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: http://networkrepository.com

[11] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in twitter: The million follower fallacy," in *ICWSM*, Washington DC, USA, May 2010.

[12] F. social network, "Friendster: The online gaming social network," https://archive.org/details/friendster-dataset-201107.

[13] C. L. Clarke, N. Craswell, and I. Soboroff, "Overview of the trec 2009 web track," DTIC Document, Tech. Rep., 2009.

[14] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proceedings of the 13th International Conference on World Wide Web*, ser. WWW '04. New York, NY, USA: ACM, 2004, pp. 595–602. [Online]. Available: http://doi.acm.org/10.1145/988672.988752

[15] P. Boldi, A. Marino, M. Santini, and S. Vigna, "BUbiNG: Massive crawling for the masses," in *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2014, pp. 227–228.

[16] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 587–596. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963488

[17] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 591–600. [Online]. Available: https://doi.org/10.1145/1772690.1772751

[18] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, pp. 354–364, 1997.

[19] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, "To push or to pull: On reducing communication and synchronization in graph computations," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 93–104. [Online]. Available: https://doi.org/10.1145/3078597.3078616