Poster : VEBO: A Vertex- and Edge-Balanced Ordering Heuristic to Load Balance Parallel Graph Processing

Poster : VEBO: A Vertex- and Edge-Balanced Ordering Heuristic to Load Balance Parallel Graph Processing

Jiawen Sun
The Queen's University of Belfast
jsun2@ed.ac.uk

Hans Vandierendonck
The Queen's University of Belfast
h.vandierendonck@qub.ac.uk

Dimitrios S. Nikolopoulos
The Queen's University of Belfast
d.nikolopoulos@qub.ac.uk

Abstract

This work proposes Vertex- and Edge-Balanced Ordering (VEBO): balance the number of edges and the number of unique destinations of those edges. VEBO balances edges and vertices for graphs with a power-law degree distribution, and ensures an equal degree distribution between partitions. Experimental evaluation on three shared-memory graph processing systems (Ligra, Polymer and GraphGrind) shows that VEBO achieves excellent load balance and improves performance by 1.09× over Ligra, 1.41× over Polymer and 1.65× over GraphGrind, compared to their respective partitioning algorithms, averaged across 8 algorithms and 7 graphs. VEBO improves GraphGrind performance with a speedup of 2.9× over Ligra on average.

CCS Concepts • **Computing methodologies → Shared memory algorithms**; • **Computer systems organization → Multicore architectures**;

1 Introduction

Parallel graph processing is prone to workload imbalance due to the skewed interconnection structure of graphs [1, 9, 10]. Most graph processing systems use simple, constant-time [5] or linear-time [4, 6, 8] algorithms to assign work to threads due to the size of the graph. It has been repeatedly documented that these algorithms are prone to introducing load imbalance [1, 9]. Load imbalance mainly results from inappropriately placing the few vertices that have very high connectivity.

This work proposes a solution to the load balance problem in shared memory system by proposing VEBO, a novel algorithm for distributing graph processing across threads. The algorithm leverages graph partitioning, which is increasingly used in shared memory systems [6, 8, 9]. Moreover, the algorithm runs in linear time as function of the size of the graph and logarithmic time as a function of the number of partitions.

VEBO (Vertex- and Edge-Balanced Ordering) balances the number of edges and the number of distinct destination vertices per partition to obtain a well-balanced workload while minimizing the computational complexity of graph partitioning. These optimization goals can be achieved in linear-time. In contrast, prior work often aims to jointly balance the number of edges per partition while minimizing the number of partitions where each vertex appears (vertex replication) [2]. No algorithms are known that solve this problem exactly and heuristics leave significant room for optimization [2].

A key motivation for considering joint vertex and edge balancing is provided by the classification of GraphGrind [6], which distinguishes between *edge-oriented* algorithms and *vertex-oriented* algorithms [6]. Edge-oriented algorithms, like PageRank, strongly favour balanced edge counts. In contrast, vertex-oriented algorithms, like Breadth-First Search (BFS), perform an amount of computation proportional to the number of vertices and strongly favour balanced vertex counts. Performance can be affected by as much as 40% if the wrong partitioning heuristic is used [6]. VEBO achieves both balanced edge and vertex counts, catering for both classes of algorithms.

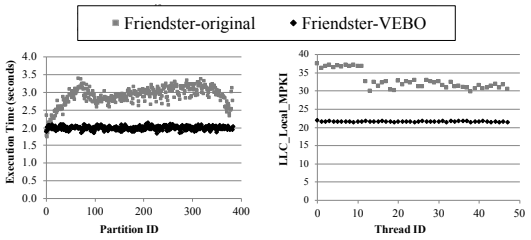
2 The VEBO Algorithm

VEBO follows an approach similar to the multi-processor job scheduling heuristic [3]: place a set of objects in order of decreasing size, for each object selecting the least-loaded partition. We adapt the algorithm to balance both the number of objects (vertices) and their size (degree).

The VEBO algorithm consists of three phases: In the first phase, VEBO assigns vertices with non-zero in-degree in order of decreasing in-degree. This is performed in two steps in order to maintain any spatial locality that may exist in the original vertex IDs. First we determine how many vertices should be assigned to each partition using the multiprocessor scheduling heuristic. Then we place the required number of vertices according to their increasing original IDs. This achieves a near-equal edge count and in-degree distribution in each partition. In the second phase, zero-degree vertices are placed. These vertices do not affect edge balance. As such, we aim to maintain vertex balance during their placement. We follow a similar two-step approach to maintain locality.

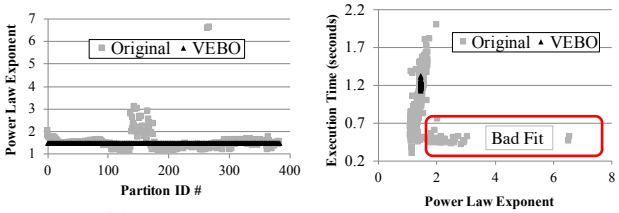
PPoPP '19, February 16–20, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-6225-2/19/02.
https://doi.org/10.1145/3293883.3295703



(a) PR – time **(b)** PR – local misses

Figure 1. Execution time and micro-architectural statistics per partition or per thread for PR with Friendster. Measured on GraphGrind using 384 partitions. Thread t executes partitions $8t$ to $8t+7$. Architectural statistics expressed in misses per thousand instructions (MPKI).



(a) Twitter **(b)** Twitter

Figure 2. Power law exponent of in-degree distribution per partition. Bad Fit means these α points with high sum of squared errors do not fit the power law distribution.

The third phase reorders the vertices. It assigns new sequence numbers to the vertices such that each partition consists of contiguous vertex IDs and rewrites the graph data structures. Further details are provided in [7].

3 Evaluation

We evaluate the performance benefits of VEBO on a 4-socket 2.6GHz Intel Xeon E7-4860 v2 machine, totaling 48 threads and 256 GB of DRAM. We disregard hyperthreading. We compile all codes using the Clang 4.9 compiler with Cilk support.

3.1 Load Balance and Locality

VEBO balances execution at the micro-architectural level, e.g., miss rates for caches (Figure 1). We observed that VEBO improves memory locality for the majority of the graphs. as the cache statistics are reduced (Figure 1b). VEBO improves each partition's locality performance but this is compensated by improved load balance. Compared to original order graph, VEBO ensures each partition has a balanced cache miss. Figure 1 shows that this load balance translates to run-time statistics on the PR algorithm for Friendster graph. Figure 1a shows the execution time for each of the 384 partitions. There is a large variation on the execution time for the original graph, e.g., from 1.73 s per iteration to 3.37 s.

3.2 Balanced Degree Distributions

Besides balancing vertices and edges, VEBO also balances the degree distribution in each partition. We calculated the power law exponent (α) for each partition using a least-squares fit (Figure 2a). A sizeable proportion of partitions in the original graph are a bad fit to the power-law distribution. Partitions of the original graph have widely varying degree distributions, while VEBO ensures partitions have *equal degree distribution.*

The skewedness of the degree distribution impacts on execution time. We plot how execution time varies with α and indicate which partitions are a bad fit to the power-law distribution (Figure 2b). For the partitions that fit well, a general trend emerges that execution time increases with increasing α. Higher α values imply that there are more low-degree vertices, which confirms that low-degree vertices require more processing time than high-degree vertices.

4 Conclusion

The established heuristic to balance the processing time of graph partitions is to create edge-balanced partitions. Edge-balance alone does not create good load balance. Considering vertex-balance along with edge-balance improves load balance significantly. Moreover, our results show that minimizing edge cut or vertex replication is not necessary on shared memory systems. We present VEBO, a vertex reordering algorithm for joint vertex, edge and degree distribution balancing and demonstrate that it achieves excellent load balance.

References

[1] M. Besta, F. Marending, E. Solomonik, and T. Hoefler. 2017. SlimSell: A Vectorizable Graph Representation for Breadth-First Search. In *IPDPS*. 32–41. DOI: https://doi.org/10.1109/IPDPS.2017.93

[2] J. E Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.. In *OSDI*.

[3] R. L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. In *SIAM J. Appl. Math.* 416-429.

[4] A. Kyrola, G. E Blelloch, and C. Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *OSDI*, Vol. 12. 31–46.

[5] J. Shun and G. E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *PPoPP*. 135–146.

[6] J. Sun, H. Vandierendonck, and D.S. Nikolopoulos. 2017. GraphGrind: Addressing Load Imbalance of Graph Partitioning. In *ICS*,. 16:1-16:10.

[7] J. Sun, H. Vandierendonck, and D. S. Nikolopoulos. 2018. VEBO: A Vertex- and Edge-Balanced Ordering Heuristic to Load Balance Parallel Graph Processing. *eprint arXiv:1806.06576* (June 2018).

[8] K. Zhang, R. Chen, and H. Chen. 2015. NUMA-aware graph-structured analytics. In *PPoPP*. 183–193.

[9] Y. Zhang, M. Yang, R. Baghadi, S. Kamil, J. Shun, and A. Amarasinghe. 2018. GraphIt - A High-Performance DSL for Graph Analytics. *eprint arXiv:1805.00923* (June 2018).

[10] X. Zhu, W. Chen, W. Zheng, and X. Ma. 2016. Gemini: A Computation-Centric Distributed Graph Processing System. In *OSDI*. 301–316.