POSTER: Corder: Cache-Aware Reordering for Optimizing Graph Analytics

YuAng Chen
The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
yuangchen@link.cuhk.edu.cn

Yeh-Ching Chung
The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
ychung@cuhk.edu.cn

Abstract

The intrinsic irregular data structure of graphs often causes poor cache utilization thus deteriorates the performance of graph analytics. Prior works have designed a variety of graph reordering methods to improve cache efficiency. However, little insight has been provided into the issue of workload imbalance for multicore systems. In this work, we identify that a major factor affecting the performance is the unevenly distributed computation load amongst cores. To cope with this problem, we propose *cache-aware reordering* (Corder), a lightweight reordering algorithm that facilitates workload balance as well as cache optimization. Comprehensive performance evaluation of Corder is conducted on various graph applications and datasets. We observe that Corder yields speedup of up to 2.59× (on average 1.47×) over original graphs.

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

Keywords: Multicore System, Graph Analytics

1 Introduction

The main reason accounting for the inefficiency of multicore graph processing is the irregular pointer-based data structure of graphs. It frequently incurs random memory accesses. Also, a distinctive property commonly found in natural graph datasets is the *power-law* degree distribution, in which a tiny fraction of vertices contribute to a majority of edges. These vertices are called as *hot* vertices, while the remaining vertices with less connections are named as *cold* vertices. Casting the graph on memory, hot vertices are preferable for caching, because they comprise a large portion of computation but a minor portion of memory usage. As for the cold vertices, they scatter all over the memory and are accessed randomly, thereby causing high cache misses rate.

To improve cache efficiency, a variety of lightweight graph reordering algorithms have been proposed [1, 2, 5]. These approaches, in principle, reorder vertices such that hot vertices and cold ones are separated into different memory locations. In such way, hot vertices are allocated in contiguous memory space and frequently requested by the core. At a result, they have higher chances to be retained within the cache lines, which allows better cache utilization. However, the reordering techniques does not necessarily guarantee performance boost. In de facto, it often leads to a slowdown if an improper strategy is deployed [1]. The gathering of hot vertices aggravates the imbalanced distribution of computation load [5], forcing the main process to wait for the longest thread for synchronization.

To overcome the limitation, we propose *Cache-aware Reordering* (Corder), a reordering algorithm based on the characteristics of the multicore system and its belonging cache hierarchy. Our contributions can be summarized as follows:

- We discover that graph analytics in multicore system suffers from uneven work distribution.
- We present Corder, a novel reordering algorithm aimed to improve workload balance and cache utilization.

2 Corder

The algorithm of Corder consists of three essential steps: (1) The input graph is subdivided into *cacheable* disjoint partitions of size equivalent to the Level-2 (L2) cache. (2) Each partition contains the same ratio of hot/cold vertices as the original graph. (3) Hot vertices and cold vertices are segregated into two segments inside a partition. The time complexity of the algorithm is $O(V + V/C)$, where V is the number of vertices and C is the L2 cache size.

Corder is especially designed according to the characteristics of the multicore system and its belonging cache hierarchy. The graph is segmented into numerous partitions, the sizes of which equal L2 cache size. Hence, each partition can be fitted into the private L2 cache of one core. The design choice of L2 cache achieves a balance between speed and storage, as the storage of Level-1 (L1) cache is too small while the speed of last level cache (LLC) is too low.

Moreover, each partition contains the same number of hot vertices whose degrees are above the average degree of the

PPoPP '21, February 27–March 3, 2021, Virtual Event, Republic of Korea
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8294-6/21/02.
https://doi.org/10.1145/3437801.3441606

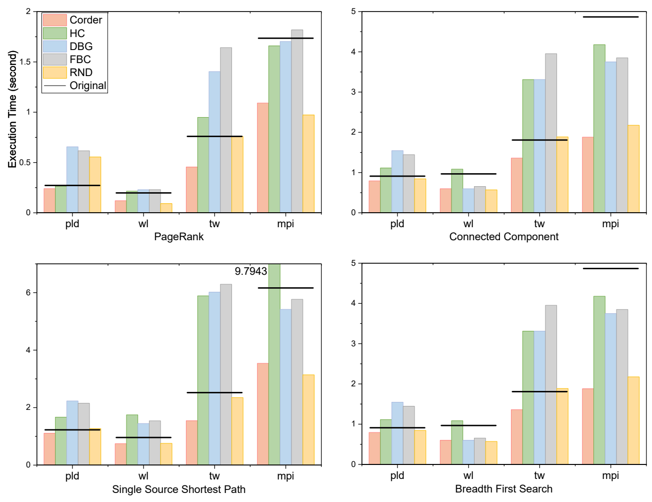

Figure 1. Execution time of graph applications with various reordering methods. PageRank is measured by the time per iteration

graph. Hence, the workloads of partitions are comparable, which facilitates the workload balance amongst the cores. Inside a partition, hot vertices and cold ones are segregated into different memory spaces. The grouping of hot vertices improves cache performance. Meanwhile, the relative orders of hot vertices and cold ones are respectively maintained as original, so that the graph structure is preserved.

3 Evaluation

In our experiments, a dual-processor server is used. Each processor consists of 10 cores with 1 MB L2 cache per core. Corder is implemented in C++ on the basis of GPOP [3], a novel cache-aware graph processing framework designed for multicore systems. The performance of Corder is evaluated on large graphs (e.g., *pld*, *wl*, *tw* and *mpi*) with edges upto 2.1*B* using four representative applications: PageRank, Connected Component, Single Source Shortest Path and Breadth First Search.

Fig. 1 illustrates the execution time of Corder by comparing with three lightweight reordering techniques: Hub-Clustering (HC) [1], Frequency Based Clustering (FBC) [2], Degree Based Grouping (DBG) [5]. These techniques, in general, place hot vertices in contiguous memory space to reduce cache misses. In addition, we implement random reordering (RND) as a supplementary benchmark.

Corder outperforms other lightweight reordering algorithms significantly. It achieves the highest speedup on all graphs and applications, which is on average 1.47×. The deployment of other reordering algorithms, including HC, FBC and DBG, often leads to considerable slowdown, the averages of which are 0.85×, 0.78×, and 0.74× respectively. This phenomenon contradicts with the statement reported by [1, 2, 5] that speedup is expected. The contradiction sources from the "processing paradigm".

The *vertex-centric* paradigm [4], as implemented underneath HC, FBC and DBG, processes the graph at the granularity of vertex per thread. Therefore, the issue of workload imbalance is relatively trivial. The *partition-centric* paradigm adopted in our work by using GPOP, however, treats the graph at a much coarser granularity. A partition containing over ten thousand vertices (e.g., 262144 vertices in the experiment) acts as the basic unit for single thread to process. Hence with the use of HC, FBC and DBG, the gathering of hot vertices heavily exacerbates the imbalance issue and consequently becomes the primary reason for performance degradation.

Random reordering is occasionally beneficial. From the perspective of statistics, random reordering can be interpreted as distributing hot vertices evenly across the partitions. Through randomization, all partitions share the same number of hot vertices and similar workloads. However, the benefit comes at the cost of poor cache utilization as hot vertices are decentralized. The overall performance is undermined when the loss from cache utilization outweighs the gain from workload balance.

4 Conclusion

In this work, we identify that the workload imbalance amongst cores degrades the performance of graph analytics in multicore systems. To address this issue, a cache-aware reordering (Corder) methods is proposed. Corder facilitates a fair sharing of computation load in the multicore system by concentrating the same number of hot vertices on all cores.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1003505).

References

[1] V. Balaji and B. Lucia. 2018. When is Graph Reordering an Optimization? Studying the Effect of Lightweight Graph Reordering Across Applications and Input Graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 203–214.

[2] Priyank Faldu, Jeff Diamond, and Boris Grot. 2019. A closer look at lightweight graph reordering. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–13.

[3] Kartik Lakhotia, Rajgopal Kannan, Sourav Pati, and Viktor Prasanna. 2020. GPOP: A Scalable Cache-and Memory-efficient Framework for Graph Processing over Parts. *ACM Transactions on Parallel Computing (TOPC)* 7, 1 (2020), 1–24.

[4] Julian Shun and Guy E Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*. 135–146.

[5] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Saman Amarasinghe, and Matei Zaharia. 2017. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 293–302.