An Efficient Parallel Algorithm for Dominator Detection

Daniel Giger
University of Massachusetts Lowell

Hang Liu (Advisor)
Stevens Institute of Technology

ABSTRACT

In graph theory, a vertex v dominates a vertex u if all the paths from the entry vertex to u must go through vertex v. This algorithm, called dominator detection, has many applications, including compiler design, circuit testing, and social network analysis. This work introduces an efficient dominator detection algorithm that is massively parallel.

ACM Reference format:
Daniel Giger and Hang Liu (Advisor). 2019. An Efficient Parallel Algorithm for Dominator Detection. In *Proceedings of SC19: The International Conference for High Performance Computing, Networking, Storage, and Analysis, Colorado, Co, November 17 - 22, 2019 (SC '19),* 2 pages.
https://doi.org/

1 INTRODUCTION

Assuming we enter a graph from vertex s, a vertex v dominates a vertex u if all paths from s to u go through v. This algorithm finds a wide range of applications. Particularly, compilers use this algorithm to conduct register assignment and memory analyzers use it to detect memory leakage [10]. Recently, this algorithm finds application in epidemic prediction in social networks [9].

Current dominator detection algorithms use an intermediate step called semidominators to determine the potential dominators for each vertex before finding the dominators [1, 3, 5]. One of the fastest existing algorithms using this approach is SEMI-NCA. This algorithm runs in $O(|V|^2)$ time. This algorithm is typically single threaded because it is difficult to parallelize.

2 BFS-BASED DOMINATOR DETECTION

A BFS algorithm [2, 7, 8] splits a graph into levels, making it possible to determine what each vertex in a given level can reach using one modified BFS for each vertex.

Our key insight is that **vertices can only dominate other vertices from later levels**. Thus it is only necessary to look at later levels when determining what they dominate. If two vertices in the same level can reach the same vertex without going through each other, nothing in that level can dominate that vertex. If only one vertex from that level can reach it, then it dominates it. This is the basis of the proposed algorithm.

The proposed algorithm does one traditional BFS to determine which vertices are in each level, and then one modified BFS for each level. One of the primary modifications to the BFS is that it keeps track of what a vertex is reached by first. This is necessary to determine which vertex is the dominator. The other difference is that each vertex can be activated up to twice. A vertex can be activated for a second time if it is reached by a new vertex during a later iteration.

Although vertices can be activated twice in all but the first BFS, the later searches are much faster than a traditional BFS. In later searches, most of the edges do not need to be examined because it is already known when each vertex will be reached. This makes a bottom-up BFS efficient and means that the number of operations needed for each of the searches (besides the first) tends to be closer to O(n) than O(m).

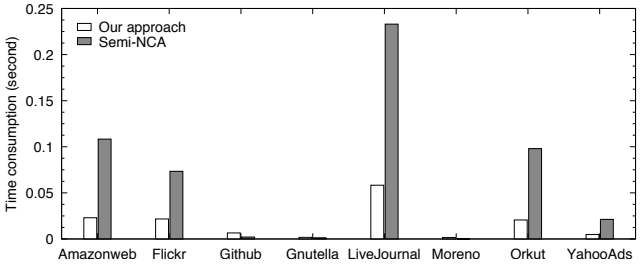

Figure 1: Performance comparison between our approach and the SEMI-NCA approach.

3 EVALUATION

This algorithm was implemented using OpenMP. We study the performance of our approach and SEMI-NCA on eight datasets from SNAP [6] and Konect [4]: Amazonweb, Flickr, Github, Gnutella, Livejournal, Moreno, Orkut and YahooAds with 1,870 - 3,072,441 vertices and 2,277 - and 117,185,083 edges, respectively.

Figure 1 presents the performance difference of our approach and the SEMI-NCA. In general, our approach performs better than SEMI-NCA for large graphs, but falls short on small graphs. The biggest gain of our approach is on the Orkut graph with 4.75× speedup.

4 PROOF OF CORRECTNESS

Since any vertex that dominates another vertex v is on all paths from the source to v, it must also be on the shortest path to it. Thus all vertices which dominate v are on the shortest path to it. The Shortest path to v can only go through one vertex in each prior level; if it goes through more than one vertex in the same level, it fails to be the shortest path. Thus v can only be dominated by one vertex in each depth level. The only remaining problem is determining which one. Suppose a path starts at vertex A and goes through vertex B in the same level before reaching more vertices in later levels. Any of these vertices could be dominated by B, but

SC '19, November 17 - 22, 2019, Colorado, Co

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/

not A. This information can already be determined by examining the paths starting at B, so examining the longer path accomplishes nothing. Thus all paths containing more than one vertex in the starting level can be ignored.

The proposed algorithm takes inspiration from vertex-removal. Vertex-removal finds dominators by determining which vertices cannot be reached without a given vertex. The proposed algorithm does this more efficiently. Since all paths to a vertex must go through all previous levels, paths to later vertices must go through the starting level. Checking whether vertices can be reached without a given vertex is equivalent to checking if there are other paths to other vertices. Because of this and the fact that paths through more than one vertex in the starting level can be ignored, it is sufficient to check whether or not later vertices can be reached by more than one vertex in the starting level.

REFERENCES

[1] K. D. Cooper, T. J. Harvey, and K. Kennedy. A simple, fast dominance algorithm. Technical report, 2006.

[2] A. Gaihre, Z. Wu, F. Yao, and H. Liu. Xbfs: exploring runtime optimizations for breadth-first search on gpus. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pages 121–131. ACM, 2019.

[3] L. Georgiadis, R. F. Werneck, R. E. Tarjan, S. Triantafyllis, and D. I. August. Finding dominators in practice. In *European Symposium on Algorithms*, pages 677–688. Springer, 2004.

[4] J. Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[5] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(1):121–141, 1979.

[6] J. Leskovec and A. Krevl. Snap datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data*, page 49, 2016.

[7] H. Liu and H. H. Huang. Enterprise: breadth-first graph traversal on gpus. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[8] H. Liu, H. H. Huang, and Y. Hu. ibfs: Concurrent breadth-first search on gpus. In *Proceedings of the 2016 International Conference on Management of Data*, pages 403–416. ACM, 2016.

[9] H. Shao, K. Hossain, H. Wu, M. Khan, A. Vullikanti, B. A. Prakash, M. Marathe, and N. Ramakrishnan. Forecasting the flu: designing social network sensors for epidemics. *arXiv preprint arXiv:1602.06866*, 2016.

[10] Wikipedia. Dominator (graph theory). Retrived from https://en.wikipedia.org/wiki/Dominator_(graph_theory). Accessed: 2019, July 30.