CS 406 - Proposal Report

Parallelizing Closeness Measures on Graphs

24071 - Barış Batuhan Topal *Furkan Ergün - 24083*

20588 - Çağhan Köksal *Hakan Ogan Alpar - 23462*

Introduction

In today's world, graphs are being used widely in many different areas. For instance, while *Google* uses this data structure in "Google Maps" by representing the roads that connect different places as edges, *Facebook* analyzes their social network by representing each person as a vertex and their relations as edges [1]. Since graphs have a variety of use cases, measures and heuristics chosen to analyze these graph structures also vary. Programs and algorithms to analyze these graphs require lots of computation power. Moreover, because of the increasing data sizes, one may gain a significant time difference by parallelizing these heuristics.

We recognize that in the heart of most these network analysis are *Breadth First Search (BFS)*, and if an end user decides to use different heuristics he or she would have to run multiple BFS algorithms to get the results for analysis. Our purpose is to unify multiple centrality measures under one program such that our program can give the results faster for cases where multiple centrality measures are queried and parallelize the overall *BFS* algorithm with the hope that it can give faster results for even a single centrality measure query. For this project, we specifically focus on the following centrality measures: *degree 1&2 centrality, closeness centrality and betweenness centrality*.

A Detailed Analysis of Our Model

In our model, we will focus on four different centrality measures:

- *Degree One Centrality* [2]*:*

  For each single node, number of links held by this node is calculated.

- *Degree Two Centrality:*

For all vertices, the number of their neighbors and their neighbors' neighbors are found. In other words, for each single vertex, *Breadth-First Search (BFS)* algorithm is run and total count of nodes is computed, which have a distance closer than or equal to 2.

$$deg2(v) = \{u \in V : dist(v, u) \leq 2\}$$

- *Closeness Centrality* [2, 3]*:*

$$clos(v) = \frac{1}{\sum_{u \in V} dist(u, v)}$$

A score is assigned to each node, based on its closeness to other nodes. The closeness of each node to other nodes is found by running *BFS* and adding all the distances found as a result.

- *Betweenness Centrality* [2, 3]*:*

Measures the number of shortest paths between 2 different nodes (s and t), in which our vertex (v) lies on. The more our node is between these 2 other nodes, the higher will be the value for our node. The shortest paths between 2 nodes are determined by *BFS* algorithms.

$$b(v) = \sum_{s \neq v,\, t \neq v,\, (s,\, t)\, \varepsilon\, V} \frac{\sigma_{s,\, t}(v)}{\sigma_{s,\, t}} \quad ;\ \sigma = \textit{number of shortest paths}$$

The reason behind selecting these four heuristics is that all of them can be computed with *BFS*. This means that if Breadth-First Search runs on a graph we choose, its result can be used for finding solutions to all of the heuristics mentioned above. Therefore, our aim is to build one single method that computes all these measurements at the same time (or combinations of these measurements with respect to users' requests). For this purpose, with each BFS algorithm, parameters for each measurement will be updated and extra work of running BFS for each single heuristics separately will be avoided. To fasten this process, especially BFS algorithm should be parallelized by using both CPU and GPU parallelization.

We will test our runtimes using 50 sparse and small-sized graphs & 20 sparse and large-sized graphs. All of them are obtained from *Suite-Sparse Matrix Library* [5] and for simplicity, graphs will be assumed as undirected and unweighted but if enough time is left, an analysis on directed graphs will also be made. If we talk more about the specifications of small and large graphs:

- *Small-sized Graphs* *:* vertex count between 100-500, symmetrical, sparse
- *Large-sized Graphs* *:* vertex count between 10000-100000, symmetrical, sparse

Literature Survey

- ***Scalable Graph Exploration on Multicore Processors [8]:***

The paper introduces a modified parallel BFS algorithm that is quite competitive with the supercomputing result of literature. The algorithm's first improvement is the use of bitmap in order to mark the vertices during the visit by improving the process rate by more than a factor of four. The second optimization is a race-free and lightweight communication way that eliminates memory pipelining due to multiple sockets. The last improvement is the use of batching for operations in the algorithm's inter-socket communication channel. Overall the paper increases millions of edges processed per second dramatically with these three simple improvements.

- ***Direction-Optimizing Breadth-First Search [9]:***

The paper examines a hybrid approach for BFS that is feasible in the graphs with low-diameters which combines both both-down and top-down approaches. Unvisited child nodes search for parents in the bottom-up part, while the parents are searching for unvisited childs in the top-down part. This approach offers improvements in speed when the active frontier of the graph is an important fraction of the total graph. The choice of whether top-down or bottom-up approach will be

used in any step is made by using a heuristic approach. It should also be mentioned that this hybrid approach ranked as 17th in the November 2011 Graph500 rankings.

- ***Efficient BFS on Massively Parallel and Distributed-Memory Machines [10]***

This paper also uses hybrid BFS by extending it to top-tier supercomputers from small-diameter graphs. The algorithm spatially partitions the graphs with the aim of parallelizing the BFS algorithm to over distributed-memory machines. The paper also uses bitmap based sparse matrix in bottom-up search with the reason that the bitmap approach enables the matrix to be significantly compact without being inefficient in retrieving edges between vertices. The paper's proposed methods offer an increase in speed between 3.0 and 5.5 for graph reading and manipulation.

- ***Regularizing Graph Centrality Computations [11]:***

The paper explains how centrality computations can be scaled up to reach higher performance. The paper makes an important difference in calculating betweenness centrality by allowing a GPU to process more than one BFS at a time rather than pure fine-grain parallelism which cannot fully utilize the GPU. Furthermore, the paper uses vectorization for simultaneous graph traversal which is not commonly used. By utilizing vectorization for graph kernels that require more than one BFS traversals it is shown that the experiments are remarkably faster than the existing solutions without the usage of any additional hardware.

Tools & Infrastructures We Plan to Use

In this project we are planning to use OpenMP and CUDA in order to parallelize our graph search algorithms and we will use C++ as the programming language.

OpenMP **:** OpenMP is an API which supports multi-platform shared memory multiprocessing programming in programming languages such as C, C++, and Fortran. In OpenMP, multithreading is achieved by forking a given number of slave threads in parallel regions by master thread. The task is divided among the threads and threads run concurrently on different processors. While CPUs have fewer cores with higher frequency and have more various instruction sets, GPUs have hundreds, thousands of less powerful cores which can handle more threads simultaneously [4, 6, 7].

CUDA **:** CUDA is a parallel computing platform and programming model which is created by Nvidia. We are planning to increase the speed of our algorithm by using the advantage of many cores of GPUs on Gandalf by using CUDA on C++ .

Gandalf **:** Gandalf is a HPC cluster which is a collection of many separate servers that are called nodes. Nodes are called via a fast interconnect. It has ***60 Intel(R) Xeon(R) E7-4870 v2 @ 2.30GHz*** as CPU and ***Nvidia Tesla K40c*** as GPU which has 2880 CUDA cores. Cache Size of the Gandalf is as follows : ***32K L1d cache, 32K L1i cache, 256K L2 cache, 30720K L3 cache***.

References

[1] Geeks For Geeks 'Applications of Graph Data Structure' [online]. Available at: https://www.geeksforgeeks.org/applications-of-graph-data-structure/

[2] Disney A. (2020) 'Social Network Analysis 101:Centrality Measures Explained' [online]. Available at: https://cambridge-intelligence.com/keylines-faqs-social-network-analysis/

[3] Marino A. (2018) 'Centrality Measures' [PowerPoint presentation]. Available at: http://pages.di.unipi.it/marino/centrality18.pdf

[4] OpenMP Application Programming Interface, OpenMp, 2018

[5] Suite-Sparse 'A Suite of Sparse Matrix Software' [online]. Available at: http://faculty.cse.tamu.edu/davis/suitesparse.html

[6] Caulfield B. (2009) 'What's the Difference Between a CPU and a GPU?' [online] Available at : https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

[7] THE MANYCORE SHIFT: Microsoft Parallel Computing Initiative Ushers Computing into the Next Era [Brochure] Available at : https://www.intel.com/pressroom/kits/upcrc/ParallelComputing_backgrounder.pdf

[8] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader (2010) 'Scalable Graph Exploration on Multicore Processors in SC'.

[9] Beamer S., Asanovíc K., Patterson D. (2012) 'Direction Optimizing Breadth-First Search. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis' (SC '12). IEEE Computer Society Press, Los Alamitos, CA, USA, Article 12, 10 pages.

[10] Ueno K., Suzumura T., Maruyama N. et al. (2017) 'Efficient Breadth-First Search on Massively Parallel and Distributed-Memory Machines'. Data Sci. Eng. 2, 22–35 . https://doi.org/10.1007/s41019-016-0024-y

[11] Sarıyüce A. E., Saule E., Kaya K., Çatalyürek Ü. V. (2015) 'Regularizing Graph Centrality Computations'. Available at: http://sariyuce.com/papers/jpdc15.pdf