Graph Algorithms and Linear Algebra

Ali Kazmi

ABSTRACT

Graph Algorithms are a central part of computer science and have been used in many applications like Uber for navigating paths or finding relationships in massive social network graphs. This project will explore how graph algorithms (parallel or serial) can run faster using linear algebraic formulations.

BACKGROUND

One key graph problem is the All Pairs Shortest Paths (APSP) problem. In this problem, you take a graph (a collection of nodes and weighted non-directed edges for our purposes) and output how far each vertex is from every other vertex, in a 2D matrix. One method of doing this is the Floyd Warshall Algorithm (Hougardy, 2010), which takes $O(V^3)$ to run.

There has been research work done by Professor Vuduc on this topic, published in PPoPP 2020. (Sao et al., 2020) They took Floyd Warshall and used linear algebraic formulations and parallelism (and other optimizations, like exploiting sparsity) to make a version that runs 123 times faster than a less efficient implementation.

GOALS

The goal of this project is to extend current methods to being more extensible and efficient for real world use. This will be done through several avenues.

One aspect will be exploring how to update the results of APSP. Right now, we know it is computationally expensive to compute these results. If we know that only certain parts of a graph changed, is there a way we can focus that computation on updating the results instead of recomputing from scratch?

Another area of interest is slight variations on this problem, such as finding k shortest paths on a graph. There could potentially be slight algorithmic twists or linear algebraic formulations here that could give us speed ups.

The last interesting part of this project is using randomized algorithms to approximate these shortest path results. Is there a way to get something close to the actual APSP results with less computational power?

METHODS

The main language for this project is Python. In order to achieve speeds closer to lower level languages, we will use the libraries Numba and Cython. Both aim to allow python programmers to have certain types of code compiled into C, which results in massive speedups. These libraries both also allow parallelism by releasing python's built in Global Interpreter Lock. I will use Jupiter Notebooks and Google Colab as an enviornment.

PREVIOUS RESEARCH

Over the course of Summer 2020, I started working on implementing fast solutions to the All Pairs Shortest Paths problem. This was a way for me to learn the intricacies of Numba and Cython, and I have experienced great success. Floyd Warshall on a 5,590 node dense graph takes 59 seconds with the standard Scipy library implementation, but with a parallel Cython implementation it takes around 51 seconds. Furthermore, utilizing a parallel and min,+ semiring implementation in Cython I was able to get it to run in 42 seconds. The min,+ semiring is an example of a linear algebraic technique that helps significantly speed up the computation.

MENTORS, LOCATION, AND TEAM

This work will be done at Georgia Tech in Spring 2021. My graduate mentor is Vijay Thakkar, and my faculty mentor is Professor Richard Vuduc. This project will be completed individually.

REFERENCES

Hougardy, S., 2010, The floyd–warshall algorithm on graphs with negative cycles: Information Processing Letters, **110**, 279 – 281.

Sao, P., R. Kannan, P. Gera, and R. Vuduc, 2020, A supernodal all-pairs shortest path algorithm: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Association for Computing Machinery, 250–261.