CS 6230 Project Report

Parallel Computation of Centrality

Rui Dai, Sam Olds

May 4, 2017

1 Introduction

2 Related Work

3 Implementation

3.1 Challenges

In our work, we represent graph in sparse adjacency matrix. The largest challenge was handling a graph distributed across processors. Trying to partition the graph evenly among the workers could be a project on its own. This problem was solved simply by putting the whole graph in shared memory.

The next challenge that arose was getting the graph data. During initial development, a synthetic graph was used by looping through every vertex for each vertex and creating and edge between them with a 25% probability. Once we were comfortable with our implementation we found a graph of Facebook friend circles used in a Stanford paper [2] with 4039 vertices and 88234 edges. We then found an even larger dataset of Google+ users used in the same paper. This graph has 107614 vertices and 13673453 edges.

3.2 Graph Representation

Initially, the underlying representation we used was a sparse adjacency matrix. We thought that this would allow for easier parallelization when performing the matrix multiplication. However, it presented a few challenges during implementation. First, an $n \times n$ matrix becomes memory demanding, even thought most of the values are simply 0. Second, we used the vertex ids as the indices into the n^2 array, which didn't work for the Google+ dataset. This dataset used 21 digit long values, which means a separate data structure would have been needed to map the vertex ids back to the matrix indices.

Instead, we switched to using an adjacency list to decrease the memory footprint and simplify the vertex id handling. This had a few additional advantages. With an adjacency list, we could easily figure out the total number of vertices that exist in the graph by just getting the size of the list. This also made it easy to split work among processors by making each processor handle some chunk of the list. This made implementing the Brandes algorithm more straightforward as this was the underlying graph representation used in that paper.

3.3 Algorithms

Calculating betweenness centrality can be naively accomplished by calculating the shortest path from every vertex to every other vertex. This was our initial approach. We

In this implementation, each shortest path calculation can be performed in parallel if the entire graph is in memory.

Next we implemented the Brandes

Ultimately, we implemented the parallel version of Brandes' algorithm presented as *Algorithm 1* in a paper by Bader and Madduri [1]. Fortunately, this algorithm requires very few modifications from the original. There was a number of directives, such as `#pragma omp parallel for`, and the associated declaration of critical sections and shared memory management.

4 Experimentation

4.1 Simple Solution

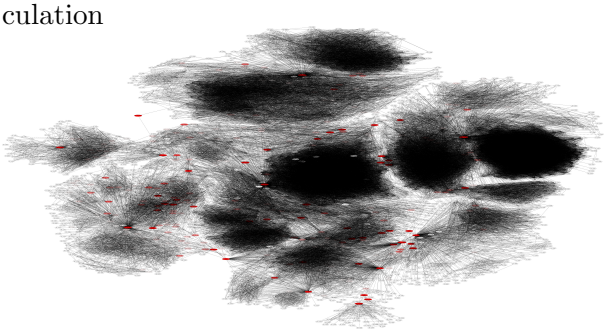
4.2 Method Based on Brandes'

Sam

5 Results

Sam

Figure 1: Facebook betweenness centrality calculation



6 Conclusion

7 Future Work

References

[1] Bader, D. A., and Madduri, K. Parallel algorithms for evaluating centrality indices in real-world networks. In *Parallel Processing, 2006. ICPP 2006. International Conference on* (2006), IEEE, pp. 539–550.

[2] Leskovec, J., and Mcauley, J. J. Learning to discover social circles in ego networks. In *Advances in neural information processing systems* (2012), pp. 539–547.