An Adaptive Parallel Algorithm for Computing Connectivity

Chirag Jain*, Patrick Flick*, Tony Pan*, Oded Green*, Srinivas Aluru*

1 Introduction

Computing connected components in undirected graphs is a fundamental problem in graph analytics. The size of graph data collections continues to grow in many different scientific domains, which motivates the need for high performance distributed memory parallel graph algorithms, especially for large networks that cannot fit into the memory of a single compute node. For a graph $G(V, E)$ with n vertices and m edges, two vertices belong to the same *connected component* iff there is a path between the two vertices in G. Sequentially, this problem can be solved in linear $O(m)$ time, e.g. by using one of the following two approaches. One approach is to use either Breadth First (BFS) or Depth First Search (DFS) for each component. Another technique is to use a union-find based algorithm, where each vertex is initially assumed to be a different graph component and components connected by an edge are iteratively merged.

There are known work-optimal and practical parallel solutions for computing BFS traversals on distributed memory systems. While parallel BFS algorithms have been optimized for traversing a short diameter big graph component, they can still be utilized for finding connectivity using multiple executions, one per connected component. However, for graphs with large number of small components, parallel BFS needs to be executed one after another, because unless a component is identified, a vertex not in the component cannot be chosen to initiate search for the next connected component. On the contrary, the classic Shiloach-Vishkin (SV) algorithm [1], a widely known PRAM algorithm for computing connectivity, simultaneously computes connectivity of all vertices and promises convergence in logarithmic iterations making it suitable for components with large diameter, as well as for graphs with a large number of small sized components. Compared to the simple label propagation techniques, the SV algorithm bounds the number of iterations to $O(\log n)$ instead of $O(n)$, where each iteration is equivalent to $O(m)$ work.

*School of Computational Science and Engineering, Georgia Institute of Technology

2 Contribution

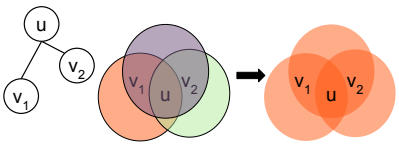
In this work, we propose a novel edge-based parallel algorithm based on the SV approach for distributed memory systems. We also suggest optimizations to reduce the data volume and balance the load as the iterations progress.

Small world scale-free networks, for instance social networks or web crawls, have the property of containing a single large connected component. For this single massive connected component, BFS is more efficient than SV is for identifying all its members. To achieve the best performance on different graph topologies, we introduce a dynamic pre-processing stage to our algorithm that guides the algorithm selection at runtime.

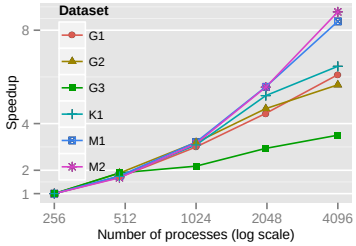
For evaluation, we cover a diverse set of graphs, both small world and large world, to highlight that our algorithm can serve as a general solution to compute connected components for undirected graphs. For the experiments, we include de Bruijn graphs from publicly available metagenomic samples, road networks, scale free networks (social networks and web links), as well as Kronecker graphs from the Graph500 benchmark. Our approach achieves a runtime of 215 seconds using 32K cores on Cray XC30 for a large metagenomic graph with 54 billion edges. When compared against the previous state of the art, we observe performance improvements up to 24x. This work extends our previous algorithm, which was specific to the metagenomic applications [2], to a general connectivity solution for any undirected graphs.

3 Algorithm

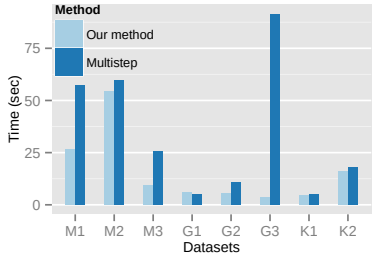
Parallel SV Algorithm Our algorithm is structured similar to the *Shiloach-Vishkin* algorithm. While it is implemented differently, our algorithm includes the two short-cutting steps in the SV algorithm that ensure logarithmic convergence to the solution. Initially, every vertex is in its own *partition*, and partitions are connected via edges of the graph. Iteratively, we join each partition to the minimum numbered partition in its *neighborhood*, until the partitions coalesce into the required connected components of the graph (Figure 1a). In order to resolve large diameter components quickly, we utilize a pointer doubling technique.



(a) Partitions at the beginning, are highlighted using different shades. Desired solution, assuming $v_1 < u, v_2$, shown on the right will be to have all three vertices in a single component v_1.



(b) Strong scalability results of our algorithm on different graphs using 4096 processes. Speedups are computed relative the runtime on 256 cores.



(c) Performance comparison of our algorithm against the Multistep method [3] using multiple graphs with 2K processes

Figure 1: Description and experimental evaluation of our parallel connectivity algorithm

Internally, our algorithm works with a distributed array of 3-tuples. During the execution, global sorting of the tuples plays an essential role to establish data locality. Each iteration of the algorithm involves four global sorts of the tuple array. The first two sorts allow each partition to compute and join its minimum partition neighbor. The next two sorts facilitate the pointer doubling mechanism. The algorithm terminates when none of the partitions have any more neighboring partitions.

The number of iterations required for convergence of a component depends on its diameter, hence small sized components tend to complete much before large components. We improve the performance of our algorithm by detecting the early convergence of *completed* partitions, and removing the tuples associated with the completed partitions from the local working set. However, removing completed partitions creates a growing load imbalance of active tuples. We counteract this by evenly redistributing all active tuples at the end of each iteration.

Hybrid Approach using BFS Parallel BFS implementations can be adapted to achieve the same objective as our parallel SV algorithm, namely to compute all the connected components in a graph. For a large component of a small world graph, the large number of vertices at each level of the traversal yields enough data parallelism for parallel BFS methods. In this case, the parallel BFS application is bandwidth bound and close to work efficient. Small world scale-free networks have the property of containing a single large connected component. These graphs are characterized by a power-law degree distribution. We classify a given graph as scale-free by estimating the goodness of fit to a power-law model. Only if the classification is positive, we choose to execute a BFS iteration to identify the first component before using our parallel SV algorithm.

4 Experiments

We use Edison, a Cray XC30 machine for the experimental evaluation. Our data-sets include 9 graphs: four metagenomic graphs (M1-M4), a twitter network (G1), a web crawl graph (G2), a road network(G3) and two kronecker graphs (K1-K2).

We conduct strong scaling experiments for our algorithm on 256-4096 cores (Figure 1b). Ideal speedup relative to 256 processes is 16. We achieve a maximum speedup of more than 8x for the metagenomic graphs M1 and M2 and about 6x speedup for small world graphs G1, G2 and K1.

Further, we compare the performance of our algorithm against the state of the art *Multistep* algorithm [3] using 2000 cores (Figure 1c). Our algorithm outperforms the *Multistep* method for all graphs except G1. The speedup we achieve over the *Multistep* algorithm ranges from 1.1x for K2 up to 24.5x for G3, more than an order magnitude faster.

References

[1] Shiloach and Vishkin *An O(logn) Parallel Connectivity Algorithm*, Journal of Algorithms, 3 (1982), pp. 57–67.

[2] Flick et al. *A Parallel Connectivity Algorithm for de Bruijn Graphs in Metagenomic Applications*, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. (2015) pp. 15.

[3] Slota et al. *A Case Study of Complex Graph Analysis in Distributed Memory: Implementation and Optimization*, In Parallel and Distributed Processing Symposium (2016).