Scaling Graph Connectivity to Hundreds of Billions of Edges on Hundreds of GPUs

George M. Slota and Michael Mandulak*

1 Introduction

Graph connectivity is a basic but important graph problem, used as a graph analytic, a preprocessing step for more complex algorithms, and within related fields such as linear algebra. This broad applicability has made graph connectivity one of the most studied problems in the discrete algorithms field. As such, there have been a plethora of parallel algorithms, variants, and optimizations proposed in the literature. We designate these algorithms as falling into four broad classes, and we consider for implementation a representative algorithm from each class within a distributed multi-GPU graph processing framework. We design efficient communication and workload processing methods for these algorithms, and we introduce novel optimizations.

Our approach strong scales all the way to 256 GPUs on graphs ranging from hundreds of millions to hundreds of billions of edges in size. We present the fastest time-to-solution for graph connectivity in the literature for the largest publicly available dataset - the 128 billion-edge Web Data Commons 2012 (WDC12) crawl. We also show performant weak scaling on graphs up to 4× larger than WDC12.

2 Background

Prior algorithms for graph connectivity can be described as falling into four broad categories. These categories include traversal-based algorithms, label propagation algorithms, 'shortcutting' algorithms such as Shiloach-Vishkin, and Union-Find or path-based algorithms. Variants and optimizations of these algorithms have been implemented for shared-memory, GPU, and distributed memory [1, 2, 3, 4]. This work has considered both graph-based and linear-algebraic formulations.

It has also become common in practice to combine several algorithms in a multi-phase approach. Our prior Multistep work [1] utilized direction-optimizing BFS to find the massive component in real networks, before running a label propagation algorithm on any remaining lower-diameter components. This concept was generalized in the ConnectIt framework of Dhulipala et al. [2] with the notion of an initial 'Sampling' and then subsequent 'Finish' phases. 'Sampling' can include algorithms that identify the largest component as with BFS or identify all components of a sparsified graph. The 'Finish' phase finalizes connectivity on the whole graph for all components using a standard algorithm.

3 Methods

We implement algorithms from each of the four categories, as well as a combined multi-phase approach. For all methods, we utilize a 2D graph distribution and associated communication patterns targeted towards a distributed multi-GPU environment. We implement our methods in C/C++, compile with `nvcc`, and use `nccl` for communications. We utilize data-driven queues for efficient computation, update-driven sparse communications, and the *Manhattan Collapse* implicit loop collapse approach for balanced thread computations.

Traversal (BFS): We implement a standard 2D direction-optimizing BFS. The use of only BFS for connectivity decomposition is inefficient, as a single traversal can find at most one component. Hence, we use BFS solely part of the combined approach.

Label Propagation (LP): We implement a minimum-label propagation algorithm, using a push-based approach to simplify the use of data-driven queues and communications within a 2D distribution.

Shortcutting (SV): We implement a variant of the Shiloach-Vishkin algorithm using FastSV [3] as a baseline, modified and optimized for 2D computation and communication patterns. While BFS and label propagation use the standard gather-scatter or reduce-broadcast 2D patterns with relatively simple data-driven queues, our SV implementation requires more advanced methods for pushing non-local grandparent updates as well as for the primary shortcutting phase.

Union-Find (UF): We implement bulk Union-Find [4] within a tunable framework, enabling several variations of Union-Find algorithms. Our implemented variations include differing logic for which direction 'Union' operations occur between trees, how much path compression is utilized within the 'Find' operation, and how often

*Rensselaer Polytechnic Insitute, Troy, NY.

updates are communicated during bulk union/bulk find updates. For these current results, we use limited 3-hop path compression, min-index unions, and communicate after each Union/Find step.

Combined: We further consider a combined approach, where we first run BFS before switching to SV or UF. The other sampling procedures described in ConnectIt are not amenable to a performant distributed implementations. We halt our BFS traversal after $\log n$ iterations, where n is the number of vertices in an input. This maintains a $O(\log n)$ iteration bound when combined with UF and SV, while still possibly taking advantage of the sub-linear work of direction-optimizing BFS. By denoting vertices discovered by BFS with a consistent label/parent pointer, we can directly run LP, SV, or UF in a subsequent step without loss of connectivity information, even if a full component was not discovered.

4 Results

We run strong and weak scaling experiments on the AiMOS (DCS) cluster at RPI. AiMOS has nodes containing 2× Power-9 CPUs and 6× V100 GPUs on an EDR Infiniband network. We run on up to 256 GPUs across 43 nodes. We show results from a selection of algorithms, including LP, SV, UF, and BFS combined with SV (BFS-SV) and UF (BFS-UF).

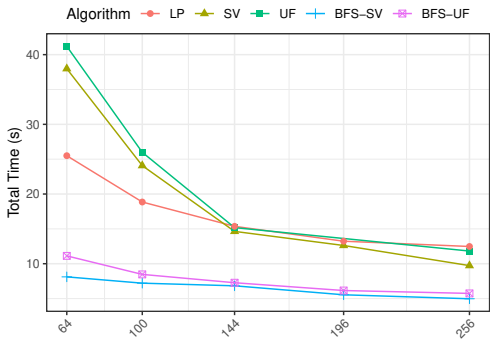


Figure 1: Strong scaling on WDC12.

Strong Scaling: Given in Figure 1 is strong scaling from 64 to 256 GPUs on WDC12 using our algorithm variants. We note that we can solve the connectivity decomposition problem in under 5 seconds, offering the fastest time-to-solution for this dataset in the literature. The fastest prior distributed-memory time known to us was given in Zhang et al. [3], who reported about 30 seconds using 262,144 CPU cores across 4096 nodes.

Weak Scaling: We select the fastest performing algorithm (BFS-SV) and run additional weak scaling on RMAT graphs, Erdös-Renyí graphs, 1D line graphs, and 2D square grid graphs from 1 to 256 GPUs, given in Figure 2. All graphs were generated to have 2^{24} vertices

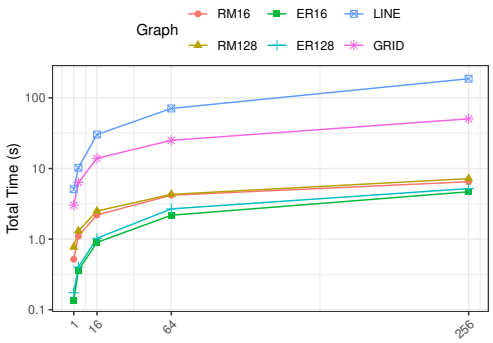


Figure 2: Weak scaling of BFS-SV.

per GPU, and we generate the RMAT and Erdös-Renyí graphs with edge factors of 16 and 128. The largest instances represent graphs over 4× larger than WDC12, with up to 550 billion undirected edges or 1.1 trillion nonzeros in the adjacency matrix. We are able to solve these largest instances in about 8 seconds for the RMAT graph and 5 seconds for the Erdös-Renyí graph. The line graphs are the most difficult inputs due to their massive diameters, requiring up to about 3 minutes. We note that our weak scaling curves are as expected for 2D distributions, given the per-rank work complexity of $O(\frac{n}{\sqrt{p}} + \frac{m}{p})$ with n vertices, m edges, and p ranks.

5 Discussion

Additional experimentation on smaller test graphs demonstrates consistent strong scalability to 256 GPUs, even on graphs small enough to fit in memory of a single GPU. We observe that our BFS-SV and BFS-UF methods offer considerable speedups when compared to FastSV's reported fastest times on each dataset. While FastSV currently runs on CPU, a CUDA implementation of its backend (SuiteSparse:GraphBLAS) is under development, and it will be interesting to perform a direct comparison when it is completed. Future work will investigate other algorithm variants for SV and UF as well as asynchronous techniques. Our current implementations use device synchronization after each kernel and global synchronizations for each communication phase, which can likely be improved upon.

+

References

[1] Slota et al., "BFS and coloring-based parallel algorithms for strongly connected components and related problems," in *IPDPS*, 2014.

[2] Dhulipala et al., "ConnectIt: a framework for static and incremental parallel graph connectivity algorithms," *VLDB*, 2020.

[3] Zhang et al., "FastSV: A distributed-memory connected component algorithm with fast convergence," in *CSC20*, 2020.

[4] Simsiri et al., "Work-efficient parallel union-find," *Concurrency and Computation: Practice and Experience*, 2018.