When Label Propagation Outperforms BFS in Breadth-First Graph Traversal

Kalsuda Lapborisuth
Georgia Institute of Technology
Atlanta, GA, USA
klapborisuth3@gatech.edu

Srinivas Aluru
Georgia Institute of Technology
Atlanta, GA, USA
aluru@cc.gatech.edu

Abstract

We tackle the challenge of breadth-first traversal (BFT) on sparse graphs with a high number of connected components. We propose a novel distributed-memory parallel algorithm that uses the label propagation (LP) algorithm to perform BFT on all connected components of the graph simultaneously. In synthetic benchmarks with RMAT-like graphs, we show that our LP-based algorithm can be up to 77x faster compared to the parallel direction-optimized BFS in CombBLAS [4], while scaling up to 1.5k CPU cores.

CCS Concepts

• **Theory of computation** → **Massively parallel algorithms**; **Shortest paths**.

Keywords

Label Propagation, Breadth-First Graph Traversal, BFS, Connected Components, Distributed-Memory Algorithm

ACM Reference Format:
Kalsuda Lapborisuth and Srinivas Aluru. 2025. When Label Propagation Outperforms BFS in Breadth-First Graph Traversal. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '25).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

We study breadth-first traversal of graphs with multiple connected components in distributed-memory systems. Graph traversal explores all vertices and edges in a defined order. These orderings, particularly BFS, are fundamental to numerous graph problems, such as graph compression [1], triangle counting [3], and connected components [9]. Existing parallel BFS algorithms are optimized for single-source traversal [2, 4], especially for the Graph500 benchmark[5].

However, breadth-first traversal is a common subroutine in several real-world applications, which often involve graphs with many connected components. SNAP datasets[1] [7] average approximately 2,700 components, with some (e.g., web-Google [8]) having up to 43,461. Performing breadth-first traversal using BFS requires running the algorithm sequentially for each connected component, which limits its scalability. We propose adapting the label propagation (LP) algorithm to compute the breadth-first ordering over all components simultaneously. Although BFS has work-optimal theoretical complexity, traversal via LP achieves better execution time and scalability for multi-component graphs due to reduced synchronization overhead and increased parallelism. We demonstrate these performance advantages through an experimental comparison of our approach against the BFS implementation of CombBLAS [4], using up to 1521 CPU cores of the Phoenix Cluster at Georgia Tech.

[1] Randomly selected 48 out of 71 SNAP graphs available in SuiteSparse Matrix Collection.

SC '25, St. Louis, MO

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

2 Breath-First Traversal via Label Propagation

Given an unweighted, undirected graph $G=(V,E)$, the BFT decomposes vertices into levels according to their shortest path distance from a designated source vertex of their respective connected components [3]. Let $n=|V|$ and $m=|E|$ denote the number of vertices and edges in G. Two vertices $u,v \in V$ are *neighbors* if both $u,v \in E$. The *neighborhood* of v is $Nbr(v) = \{u \in V : \{u,v\} \in E\}$. A *path* from vertex v_1 to v_k refers to a sequence of distinct vertices $v_1,...,v_k$ such that $\forall i \in 1,...,k-1$, v_i and v_{i+1} are neighbors. The *diameter* $d=\max\{d_1,d_2,...,d_n\}$ is the maximum shortest path length among all connected components with diameters $d_1,d_2,...,d_n$. Two vertices belong to the same *connected component* if and only if there exists a path between them.

2.1 Parallel Algorithm

Algorithm 1: Parallel Label Propagation Algorithm

Input: Undirected Graph $G(V,E)$
Output: Partition label array ID, Level array L, Parent array π

```
1  for each v ∈ V do
2  |  ID[v] ← v; ID_prev[v] ← v; π[v] ← −1; L[v] ← 0;
3  converged ← FALSE;
4  while converged ≠ TRUE do
5  |  converged ← TRUE; ID ← ID_prev
6  |  Exchange ID and L with neighboring PEs
7  |  /*Examining all neighbors of each vertex in O(n+m)*/
8  |  for each v ∈ V do
9  |  |  for each u ∈ N(v) do
10 |  |  |  if ID_prev[u] < ID[v] then
11 |  |  |  |  ID[v] ← ID_prev[u]; π[v] ← u; L[v] ← L[u]+1
12 |  |  |  |  converged ← FALSE
13 |  converged ← AllReduce(converged, i, p);
```

We consider the LP algorithm formulation based on the Shiloach-Vishkin (SV) algorithm for connected components [10]. The algorithm begins by initializing each vertex as a singleton partition with itself as both the partition label and parent (lines 1-2 of Algorithm 1). The *partitions* represent intermediate vertex groupings that eventually consolidate into one set per connected component.

Throughout the algorithm's execution, each vertex u maintains its partition label $ID[u]$, level $L[u]$, and parent vertex $\pi[u]$ from which it received the label. The parent array π traces the propagation path of length $L[u]$ from the vertex u to the root $ID[u]$. With each vertex only pointing to one parent, these arrays form implicit tree structures rooted at the vertex with the minimum label among all vertices in each partition. Naturally, any given vertex can be made the root by assigning it the smallest label value before executing the algorithm.

In each iteration, the algorithm performs a *hooking* operation (lines 8-12, Alg 1), where each vertex examines its neighborhood to find the minimum partition label among all neighbors. Each vertex v then updates its partition label $ID[v] = \min_{u \in N(v)} ID_{prev}[u]$, adopts the corresponding neighbor as its parent, and sets its level to one greater than the parent's level (line 11, Alg 1). This process terminates when no vertex changes its partition label in a complete iteration, resulting in trees that each span one connected component.

Table 1: Communication comparisons of LP and BFS

	Synchronization	Volume
LP	$\max_i d_i$	$O(d\|E\|)$
BFS	$\sum_i d_i$	$O(\|V\|+\|E\|)$

2.2 Label Propagation Spanning Tree

A valid BFS tree requires each vertex to have a level equal to its shortest distance from the root. We prove that the LP algorithm produces spanning trees that satisfy this property for each component

Theorem 2.1. *. $L[u]$ is the shortest path distance from vertex u to vertex $ID[u]$ in G.*

Proof. *Level 0:* If a vertex u is a *root*, u has never joined another partition in previous iterations and maintains $ID[u] = u$ and $L[u] = 0$.

Level 1: Let a vertex u adopt its partition label from a root $r \in Nbr(u)$, as defined above. Alg 1 sets $ID[u] = r$ and $L[u] = 1 + L[r] = 1$.

Induction: Let vertex u have $ID[u] = r$ and level $L[u] = l$. Suppose that in some iteration, a vertex $v \in Nbr(u)$ adopts the partition label r from u, resulting in $ID[v] = r$ and $L[v] = l + 1$. By the inductive hypothesis, $L[u] = l$ is the shortest path distance from u to the root r. Since $v \in Nbr(u)$, there exists an edge (v,u), and the shortest path from v to r is at most $l + 1$ through u. Suppose that there exists a shorter path from v to r of length $k < l+1$. Then there would be a path from u to r of length $k+1 \leq l$, contradicting the inductive hypothesis that $L[u] = l$ is the shortest distance. Therefore, $L[v] = l + 1$ is the shortest path distance from v to the root r. □

3 Results and Discussion

We compare the execution time and scaling of our MPI-based LP implementation against CombBLAS's Direction-Optimizing BFS [4] on Georgia Tech's Phoenix cluster, using up to 64 nodes (dual-socket Xeon Gold 6226, 2.7 GHz, 24 cores). For synthetic benchmarks, we

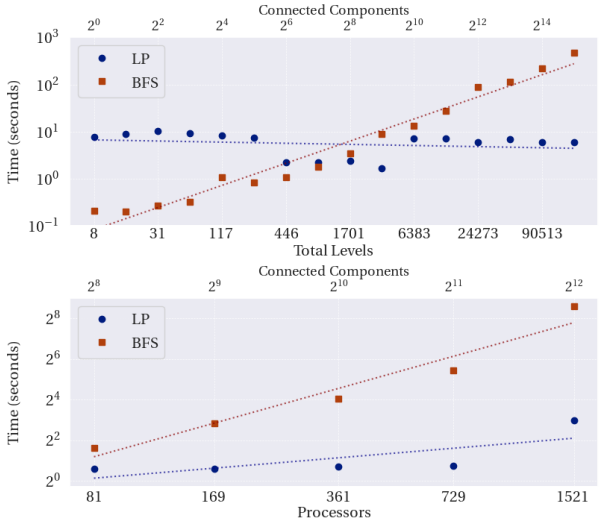

Figure 1: (Top) Runtime of LP and BFS on scale 27 GRMAT graph with increasing connected components at 1521 processors. (Bottom) Weak scaling of LP and BFS on GRMAT graphs.

incorporated Generalized-RMAT (GRMAT), an expanded version of RMAT graphs. We define a (c,s)-RMAT graph to contain 2^s vertices and 2^c almost equally-sized weakly connected components. We used the linear-work RMAT generator library [6] to generate edges for our synthetic graphs. The number of synchronizations of LP is up to 30,000x fewer than BFS in Figure 1, which makes Alg 1 up to 77x faster than direction-optimizing BFS.

4 Conclusion and Future Direction

Parallel BFS remains effective for traversing graphs with few large, short-diameter connected components, as each traversal level contains sufficient vertices for parallelism while requiring minimal synchronization. However, as the number of connected components increases, sequential component traversal causes synchronization costs to dominate, limiting parallel BFS scalability. Although LP excels in practice, its weaker theoretical complexity motivates further algorithmic optimizations and communication strategies to reduce communication volume and synchronization.

Acknowledgments

We thank Souvadra Hati and Richard Vuduc for their insightful discussions and for sharing their implementation of the Generalized-RMAT generator. This research was supported in part through research cyberinfrastructure resources and services provided by the Partnership for an Advanced Computing Environment (PACE) at the Georgia Institute of Technology, Atlanta, Georgia, USA.

References

[1] Alberto Apostolico and Guido Drovandi. 2009. Graph Compression by BFS. *Algorithms* 2, 3 (2009), 1031–1044. doi:10.3390/a2031031

[2] Junya Arai, Masahiro Nakao, Yuto Inoue, Kanto Teranishi, Koji Ueno, Keiichiro Yamamura, Mitsuhisa Sato, and Katsuki Fujisawa. 2024. Doubling Graph Traversal Efficiency to 198 TeraTEPS on the Supercomputer Fugaku. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. doi:10.1109/SC41406.2024.00107

[3] David A. Bader, Fuhuan Li, Anya Ganeshan, Ahmet Gundogdu, Jason Lew, Oliver Alvarado Rodriguez, and Zhihui Du. 2022. Triangle Counting Through Cover-Edges. *2023 IEEE High Performance Extreme Computing Conference (HPEC)* (2022), 1–7. https://api.semanticscholar.org/CorpusID:261882426

[4] Aydin Buluç, Erika Duriakova, Armando Fox, John R. Gilbert, Shoaib Kamil, Adam Lugowski, Leonid Oliker, and Samuel Williams. 2013. High-Productivity and High-Performance Analysis of Filtered Semantic Graphs. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 237–248. doi:10.1109/IPDPS.2013.52

[5] Graph 500 Benchmark 2014. http://www.graph500.org/.

[6] Lorenz Hübschle-Schneider and Peter Sanders. 2020. Linear work generation of R-MAT graphs. *Network Science* 8, 4 (2020), 543–550.

[7] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[8] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2009. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[9] Robert McColl, Oded Green, and David A. Bader. 2013. A new parallel algorithm for connected components in dynamic graphs. In *20th Annual International Conference on High Performance Computing*. 246–255. doi:10.1109/HiPC.2013.6799108

[10] Yossi Shiloach and Uzi Vishkin. 1982. An O(logn) parallel connectivity algorithm. *Journal of Algorithms* 3, 1 (1982), 57–67. doi:10.1016/0196-6774(82)90008-6