# Parallelising Betweenness Centrality using Brande's Algorithm

Akashdeep S
181IT203
*Information Technology*
*National Institute of Technology*
*Karnataka, Surathkal*
akashdeep.181it203@nitk.edu.in

Sujan Reddy A
181IT147
*Information Technology*
*National Institute of Technology*
*Karnataka, Surathkal*
sujan.181it147@nitk.edu.in

Harshvardhan R
181IT217
*Information Technology*
*National Institute of Technology*
*Karnataka, Surathkal*
harshvardhanr.181it217@nitk.edu.in

***Abstract*—Graphs that model COVID-19 transmission, social networks, and the structure of the Internet are enormous and cannot be manually inspected. A popular metric used to analyze these networks is betweenness centrality. The fastest algorithm to calculate betweenness centrality runs in quadratic time. Hence the examination of large graphs poses an issue. To improve the performance we can parallelise the algorithm using CUDA.**

***Keywords—CUDA, Parallel Computing, Graph Analysis, Betweenness Centrality.***

## I. Introduction

Graph analysis is a fundamental tool for diverse domains like social networks, machine learning, computational biology and many more. Betweenness Centrality is a popular analytic that determines vertex influence in a graph. It is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that the number of edges that the path passes through is minimized for undirected graphs. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.Naive implementations of Betweenness Centrality can be solved with the all-pairs shortest-paths problem using the $O(n^3)$ Floyd-Warshall algorithm [1]. Brande's algorithm runs in O(mn) time for unweighted graphs [2] where m is the number of edges , n is the number of nodes. It has applications in analysing COVID-19 transmission. It reflects the role a patient plays in creating a bridge of infectious transmission between patients who would not have had direct contact with each other. Betweenness Centrality also has applications in community detection, power grid contingency analysis, and the study of the human brain, finding the best location of stores within cities.

## II. Literature survey

Betweenness Centrality ( BC ) attempts to distinguish the most influential vertices in a network by measuring the ratio of shortest paths passing through a particular vertex to the total number of shortest paths between all pairs of vertices. This determines how well a vertex connects the pairs of other vertices in the same graph. Mathematically Between Centrality of a vertex ($v$) can be defined as

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (1)$$

where $\sigma_{st}$ is the number of shortest paths between vertices s and t and $\sigma_{st}(v)$ is the number of those shortest paths that pass through $v$. Brandes improved upon the naive approach of using Floyd-Warshall algorithm. In Brande's algorithm we reuse the already calculated shortest distances by using partial dependencies of shortest paths between pairs of nodes for calculating BC of a given vertex with respect to a fixed root vertex. [3]. Brandes's algorithm splits the betweenness centrality calculation into two major steps:

1. Find the number of shortest paths between each pair of vertices ie. partial dependency $\delta_s(v)$

$$\delta_s(v) = \sum_{w:v \in pred(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_s(w)) \qquad (2)$$

2. Sum the dependencies for each vertex

$$BC(v) = \sum_{s \neq v} \delta_s(v) \qquad (3)$$

Jia et al. [4] compared two types of fine-grained parallelism based on the distribution of threads to the graph entities: vertex-parallel and edge-parallel. The vertex-parallel method assigns a thread to each vertex of the graph and that thread traverses all of the outgoing edges from that vertex and, the edge-parallel approach assigns a thread to each edge of the graph and that thread traverses that edge only. Since there will be more number of vertices and edges than the number of threads, the edges or vertices will be sequentially processed by the threads. Both these approaches have a time complexity of $O(n^2 + m)$.

The problem with these two algorithms is that vertices or edges that need not be assigned a thread for a given source vertex are assigned a thread which leads to wasted work. This has been addressed by Adam et al.[3] who came up with a work efficient method with fine-grained and coarse-grained parallelism. Running a parallel BFS using optimal work distribution strategies was used for achieving fine grained parallelism. Coarse grained parallelism was

achieved by using multiple blocks in a grid in CUDA for parallelising the partial dependencies for the respective independent sources.

## III. METHODOLOGY

### A. *Graph input*

The graph is stored in the Compressed Sparse Row format.

It has two arrays:

- Adjacency List array where each adjacency list is concatenated to form a single array.
- Adjacency List Pointers array which is an array which points to where each adjacency list starts and ends within the adjacency list array.

### B. Serial Algorithm

For every source vertex we initialise the dependency, sigma, distance pointers to their respective default values. The source vertex is pushed into a queue and as long as the queue is not empty we dequeue a vertex, push it to the stack and update the distance, sigma and predecessor arguments for its neighbours. When the queue is empty we pop the elements of the stack. For each of the predecessors of the popped vertex the dependency is updated. The Betweenness centrality is found by calculating the summation of all the dependencies.

**Algorithm 1:** Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Figure 1: Brandes' Algorithm [2]

## IV. REFERENCES


[1] R. W. FLOYD, "ALGORITHM 97: SHORTEST PATH," COMMUN. ACM, VOL. 5, NO. 6, PP. 345–, JUN. 1962.

[2] U. BRANDES, "A FASTER ALGORITHM FOR BETWEENNESS CENTRALITY," JOURNAL OF MATHEMATICAL SOCIOLOGY, VOL. 25, PP. 163–177, 2001.

[3] ADAM MCLAUGHLIN, DAVID A. BADER (2014) SCALABLE AND HIGH PERFORMANCE BETWEENNESS CENTRALITY ON THE GPU.

[4] Y. JIA, V. LU, J. HOBEROCK, M. GARLAND, AND J. C. HART, "EDGE V. NODE PARALLELISM FOR GRAPH CENTRALITY METRICS," GPU COMPUTING GEMS, VOL. 2, PP. 15–30, 2011