Parallelising Betweenness Centrality using Brande's Algorithm

Akashdeep S 181IT203
Sujan Reddy 181IT147
Harshvardhan 181IT217

Introduction

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.

It 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.

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

σst : number of shortest paths between vertices s and t and
σst (v) : number of those shortest paths that pass through v.

Relevance of Problem Statement

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.

Literature Survey

Sl No	Paper Title & Authors	Work Done	Advantages	Disadvantages
1	Algorithm 97: Shortest path (1962) by R. W. Floyd [1]	Algorithm for all pairs shortest paths	Provides a simple algorithm to find the BC of every vertex	Time complexity is O(n^3)
2	A Faster Algorithm for Betweenness Centrality (2001) by U. Brandes [2]	Reuses 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.	Optimises the naive algorithm and has a time complexity of O(mn)	Examination of large graphs take time

Literature Survey

Sl No	Paper Title & Authors	Work Done	Advantages	Disadvantages
3	Edge v. Node Parallelism for Graph Centrality Metrics (2011) by Y. Jia, V. Lu, J. Hoberock, M. Garland, and J. C. Hart [3]	Parallelising by 2 methods: 1) vertex-parallel 2) edge-parallel	Time taken to perform computations is lesser	Causes load imbalance as some edges or vertices are unnecessarily assigned a thread.
4	Scalable and High Performance Betweenness Centrality on the GPU. (2014) by Adam McLaughlin, David A. Bader [4]	Parallelising by 2 methods: 1) Work efficient (fine grained) 2) Work distributed (coarse grained)	Parallelised based on the dependence and the BFS	The execution time in case of smaller graphs increases because to the overhead of assigning threads.

Formulae

Partial dependency :

$$\delta(s|v) = \sum_{t \in V} \delta(s,t|v)$$

$$\delta(s,t|v) = \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

$$C_B(v) = \sum_{s \in V} \delta(s|v)$$

MAGIC Equation :

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

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

Methodology

Graph representation

The graph is stored in the Compressed Sparse Row format.

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

Methodology

Serial Algorithm:

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
```

Methodology

Parallel

Vertex-parallel method assigns a thread to each vertex of the graph and that thread traverses all of the outgoing edges from that vertex.

Edge-parallel approach assigns a thread to each edge of the graph and that thread traverses that edge only.

Work efficient parallelism by running a parallel BFS using optimal strategies for achieving fine grained parallelism.

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

Individual Contribution

Akashdeep S (181IT203): Work Efficient parallelism + Serial Algorithm

Sujan Reddy A (181IT147): Vertex based parallelism + Edge based parallelism

Harshvardhan R(181IT217) : Work distributed parallelism + Graph Data Structure

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