PARALLEL COMPUTATION OF CLOSENESS CENTRALITY USING APSPS

CS 5990 Computational Social Systems

Team Members:

Aumkaareshwar, Jeremy Anunwah,Yurii Lebid, Bill Kim

Closeness Centrality

INTRODUCTION

Objective: Efficiently compute closeness centrality for nodes in large graphs using parallel computing.

Motivation: Closeness centrality is vital for analyzing social networks, but its computation is expensive for large graphs.Parallel computation reduces runtime by distributing workload across processors.

PROBLEM DESCRIPTION

Compute closeness centrality for a graph:

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

Input:

- Social network datasets (e.g., Facebook, Twitter).

Output:

- Centrality values for all nodes.
- Top 5 nodes with highest centrality.
- Average centrality value.

Algorithm Overview

Key Steps:

- Load and distribute graph data across processors.
- Use Dijkstra's algorithm to compute shortest paths for Subset of nodes.
- Compute closeness centrality locally on each processor.
- Gather and combine results at the root processor.
- Output the centrality measures and performance results.

Pseudocode

Main Process:

- Divide nodes into P subsets.
- Broadcast graph to all processors.
- Compute shortest paths using Dijkstra's algorithm.
- Calculate closeness centrality for assigned nodes.
- Gather results and compute top nodes & average centrality.

Dijkstra's Algorithm:

- Initialize distances as infinity.
- Use priority queue to compute shortest paths.
- Return distances to all other nodes.

Data Structures

Graph Representation:

- Adjacency list (efficient memory usage).

Distance Storage:

- Dictionary for shortest paths.
- Per-processor subset reduces memory usage.

Priority Queue:

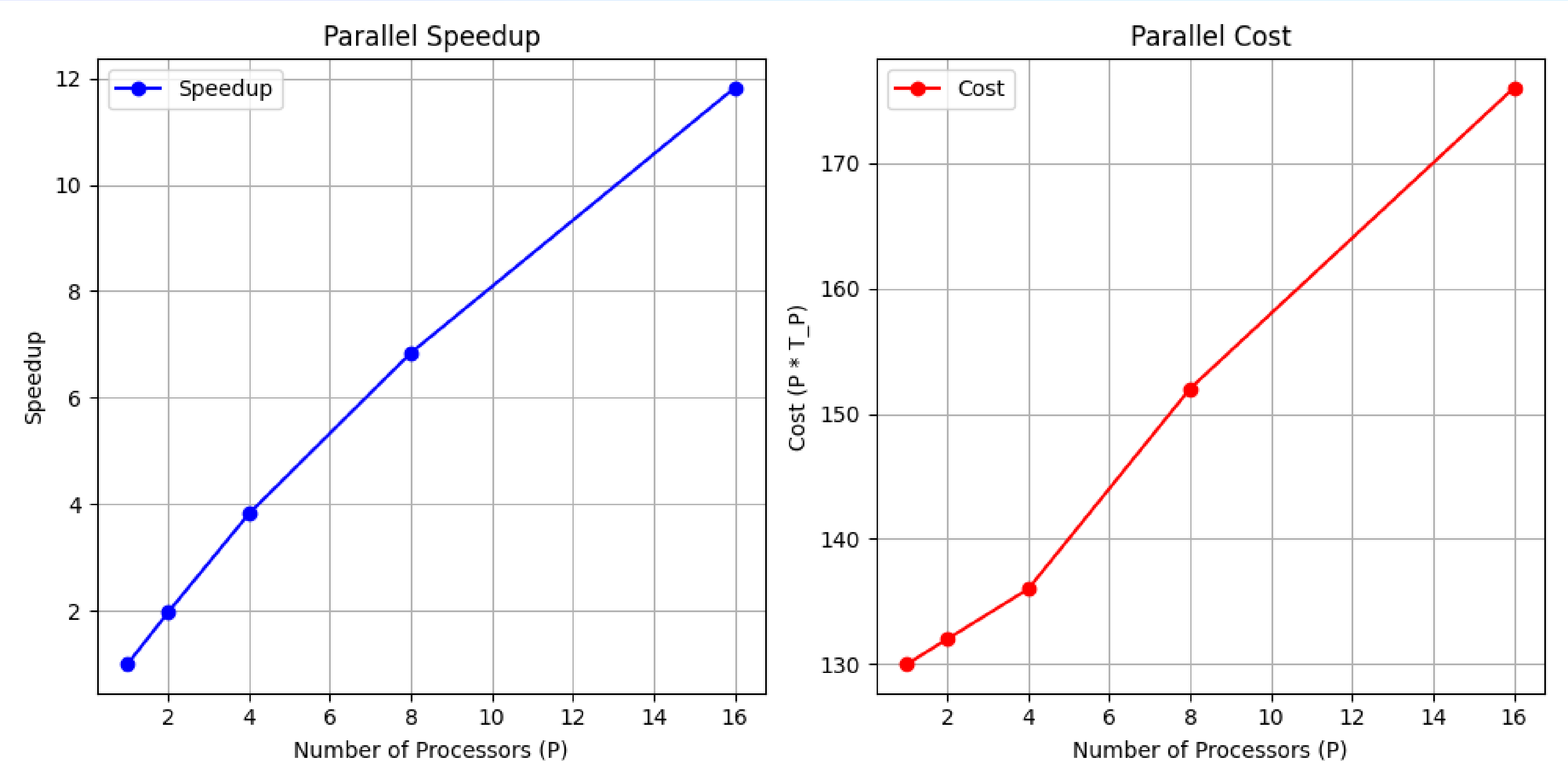
- Implements Dijkstra's algorithm efficiently.

Performance Study Results

Runtime Results:

Processors (PP)	Runtime (TPT P)	Speedup (T1/TP)	Cost ($P \cdot TP$)
1	130 Seconds	1.0	130
2	66 Seconds	1.9	132
4	34 Seconds	3.8	136
8	19 Seconds	6.8	152
16	11 Seconds	11.8	176

Performance Plots Facebook Data



Performance Plots for Twitter data

Complexity Analysis

Time Complexity:

- Single Processor: $O(n \cdot (n+m) \log n)$
- Parallel Version:O(n/P ·(n+m)logn)+O(n/P)

Space Complexity:

- Adjacency list: $O(n+m)$
- Per-processor storage: $O(n/P)$

Strengths and Limitations

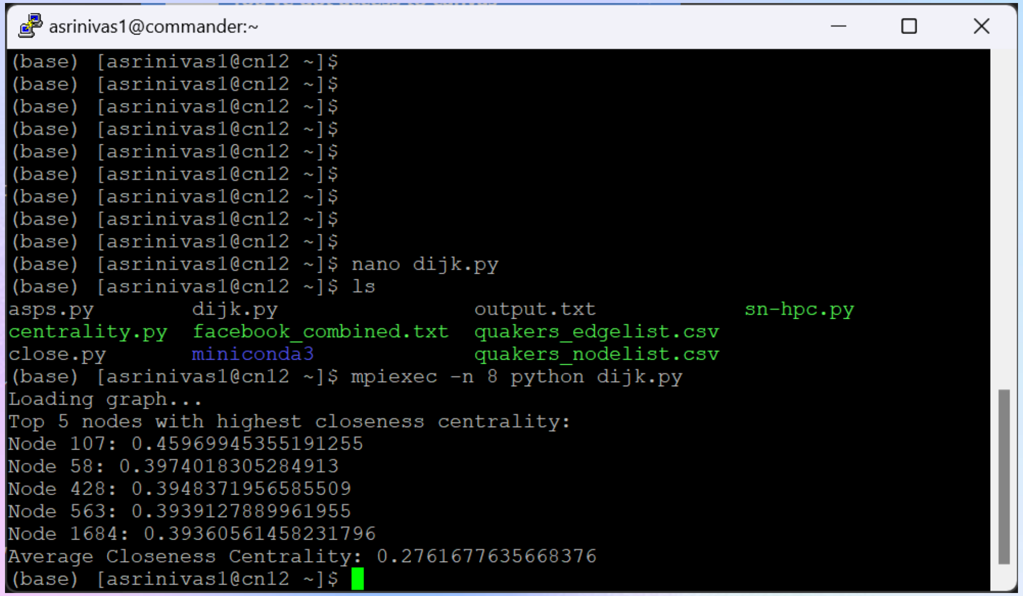
Strengths:

- Efficient for sparse graphs.
- Scalable for small to moderate P.

Limitations:

- Diminishing returns at high processor counts due to communication overhead.
- Load imbalance for graphs with non-uniform node degrees.

Output for Facebook and Twitter Data



```
asrinivas1@commander:~
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$
(base) [asrinivas1@cn12 ~]$ nano dijk.py
(base) [asrinivas1@cn12 ~]$ ls
asps.py         dijk.py                output.txt            sn-hpc.py
centrality.py   facebook_combined.txt  quakers_edgelist.csv
close.py        miniconda3             quakers_nodelist.csv
(base) [asrinivas1@cn12 ~]$ mpiexec -n 8 python dijk.py
Loading graph...
Top 5 nodes with highest closeness centrality:
Node 107: 0.45969945355191255
Node 58: 0.3974018305284913
Node 428: 0.3948371956585509
Node 563: 0.3939127889961955
Node 1684: 0.39360561458231796
Average Closeness Centrality: 0.2761677635668376
(base) [asrinivas1@cn12 ~]$
```

Facebook Data

References

- SNAP Datasets: Facebook Dataset, Twitter Dataset.
- MPI Documentation: mpi4py.

Tasks

- Jeremy Anunwah: presentation, simulation experiments, contributed to research, algorithm design, MPI communication.
- Yurii Lebid: Presentation, Prepared datasets, optimized algorithm design, and conducted performance analysis.
- Bill Kim: optimization, debugging, and presentation preparation.
- Aumkaareshwar: presentation, algorithm implementations, HPC setup, testing, and generating performance plots.

THANK YOU