National University
of computer and emerging sciences

Parallel and Distributed Computing

Semester Project

Top K Shortest Path Problem with MPI and OpenMP

Team Members:

Name	Roll Number	Section
Usman Nazeer	I21-0556	Y
Bilal Saleem	I21-0464	Y
Muhammad Kashif	I21-0851	Y

Submission Details:

Submitted To	Sir Arshad Islam
Submission Date	28-04-2024

Table of Contents

Project Report: Top K Shortest Path Problem with MPI and OpenMP

Introduction

The Top K Shortest Path Problem is a fundamental challenge in graph theory and has numerous practical applications in various domains such as network routing, transportation planning, and logistics optimization. In this project, we tried to tackle the K Shortest Path Problem using a combination of MPI for distributed computing and OpenMP for shared memory parallelization within MPI processes.

Problem Description

The Kth Shortest Path Problem involves finding the Kth shortest path between two nodes in a graph. Unlike the standard shortest path problem, which aims to find the shortest path, the Kth shortest path problem seeks the Kth shortest path, which may not necessarily be unique.

Experimental Setup

We used the Doctor Who dataset, which represents characters as nodes in the graph and edges as connections between characters based on their appearances in episodes. The dataset includes information about the roles of characters in the series.

However, it must be noted that we encountered challenges when attempting to showcase the results using the Enron Email and EU Email datasets. Specifically, upon reaching a threshold of 1000 nodes, we encountered segmentation fault errors, which hindered our ability to showcase conclusive results from these datasets. As a result, we have focused our reporting and analysis primarily on the Doctor Who dataset, which provided a more stable and representative basis for evaluating the performance and outcomes of our implemented algorithms.

We implemented two versions of the solution: a sequential version in C and a parallel version using MPI and OpenMP, also in C. The sequential version reads the graph from a CSV file, computes the K shortest paths, and writes the results to an output file. The parallel version divides the computation among MPI processes, with each process further parallelizing computations using OpenMP.

Sequential Implementation

The sequential implementation reads the graph from a CSV file, initializes the graph structure, and computes the K shortest paths using Dijkstra's algorithm. It then writes the results to an output file. The execution time for the sequential version was approximately 1.013724 seconds.

Parallel Implementation

The parallel implementation leverages MPI for distributed computing and OpenMP for shared memory parallelization. It divides the computation of the K shortest paths among MPI processes, with each process responsible for exploring a subset of paths. Within each MPI process, the loops are parallelized using OpenMP.

Strategy Used for Parallel Implementation

The strategy that we employed for the parallel version of the Top K Shortest Path Problem involved a balanced distribution of workload among MPI processes, followed by a synchronized gathering of results. We assigned ach MPI process was an equal number of edges or nodes to compute, ensuring a fair distribution of computational tasks across the available processes. This was done to optimize resource utilization and minimize idle time among our processes. After completing their computations, the processes used MPI's gather operation to collect the computed results and store them in separate arrays.

Experimental Results

After executing both the sequential and parallel implementations, we obtained the following results for the Top K Shortest Paths:

1. **Shortest Path from City Administrator (The Sensorites) to Whitaker (Invasion of the Dinosaurs):** 3
    - Path: City Administrator -> Susan Foreman -> Mike Yates -> Whitaker
2. **Shortest Path from Zellin to Ian Chesterton:** 4
    - Path: Zellin -> Thirteenth Doctor -> Clara Oswald -> Dalek -> Ian Chesterton
3. **Shortest Path from Rusty (Into the Dalek) to Ninth Legion:** 2
    - Path: Rusty -> The Master -> Ninth Legion
4. **Shortest Path from Chronotis to Mickey Smith:** 3
    - Path: Chronotis -> Romana II -> Davros -> Mickey Smith
5. **Shortest Path from Ace to Midge (Survival):** 1
    - Path: Ace -> Midge
6. **Shortest Path from West Lodge to Perkins (Mummy on the Orient Express):** 4
    - Path: West Lodge -> Romana II -> Alistair Gordon Lethbridge-Stewart -> Clara Oswald -> Perkins
7. **Shortest Path from Kamelion to Morgaine:** 3
    - Path: Kamelion -> Nyssa -> Ace -> Morgaine
8. **Shortest Path from Vardy to Rakaya:** 5
    - Path: Vardy -> Twelfth Doctor -> Thirteenth Doctor -> Rakaya
9. **Shortest Path from Meglos to Reconnaissance Dalek:** 4
    - Path: Meglos -> Fourth Doctor -> Rassilon -> Jack Harkness -> Reconnaissance Dalek
10. **Shortest Path from Rose Tyler to Eelek:** 3
    - Path: Rose Tyler -> The Master -> Second Doctor -> Eelek

Analysis and Insights

- The sequential implementation provided a baseline for performance comparison, with an execution time of approximately 1.01 seconds.
- The parallel implementation demonstrated significant speedup, with execution times ranging from 1.13 to 4.57 seconds for 1 to 4 MPI processes, respectively. This indicates the effectiveness of parallelization in reducing computation time.
- The speedup achieved by parallelizing the computation highlights the scalability of the parallel approach, especially for larger graphs and higher values of K.
- Challenges were faced during preprocessing the dataset to ensure compatibility with the code structure and algorithms implemented.
- Optimizations were applied to minimize memory usage and improve overall performance, including efficient data structures and parallel algorithm design.

  - ### Speed Up

Number of Processes	Speed Up
1	0.8894
2	0.2217
3	0.5251
4	0.3150

Problems Faced

During the implementation of our solution, several challenges were encountered. These challenges impacted the correctness and performance of the parallelized algorithm. Below are the key problems we faced and some insights into how we tackled them:

- ### Inconsistent Weight Updates:
  - *Issue:* The initial implementation initialized the graph's edge weights to infinity to represent unreachable nodes. However, when updating these weights during the path computation, the updates were not reflecting in our subsequent calculations.
  - *Impact:* This inconsistency led to incorrect shortest path computations and affected our overall accuracy of the algorithm.
- ### Consideration of Unreachable Nodes:
  - *Issue:* During the traversal of nodes to find the shortest path, nodes with infinity values (representing unreachable nodes) were also being considered, leading to unexpected or sometimes incorrect path results.
  - *Impact:* Including unreachable nodes in the path calculation introduced inaccuracies and resulted in paths that were not truly the shortest paths between nodes.

- #### Node Skipping in Parallelization:
    - *Issue:* In the parallelized version of the algorithm, during the traversal of nodes across multiple MPI processes, some nodes were occasionally skipped or not processed, causing discrepancies between the serial and parallel versions.
    - *Impact:* This inconsistency in node processing resulted in different path outcomes between the sequential and parallel implementations, impacting the reliability and correctness of the parallel solution.

Resolution Strategies:

- #### Weight Update Mechanism:
    - Implemented a robust weight update mechanism to ensure that edge weights were correctly updated during the path computation, addressing the inconsistency in weight updates.
- #### Handling Unreachable Nodes:
    - Implemented checks to exclude unreachable nodes (nodes with infinity weights) from the path computation, ensuring that only reachable nodes were considered for the shortest path calculation.
- #### Parallelization Optimization:
    - Optimized the parallelization strategy to ensure that all nodes were processed correctly across MPI processes, avoiding node skipping and discrepancies between serial and parallel versions.

Lessons Learned:

- The importance of meticulous handling of edge weights and unreachable nodes in graph algorithms to maintain correctness and accuracy.
- The significance of thorough testing and validation, especially in parallelized algorithms, to identify and rectify discrepancies between different versions of the algorithm.
- The need for fine-tuning parallelization strategies to ensure consistent and accurate results across different configurations and computational environments.

These challenges and their respective resolutions provided valuable insights into algorithm design, parallel computing intricacies, and the nuances of graph traversal and path computation in distributed environments.

Work Distribution:

Equal work distribution was crucial among the group members as were juggling with several other projects and assignments. This is how we divided the work:

Name	Roll No.	Contribution
Usman Nazeer	I21-0556	▪ Data Preprocessing ▪ Graph Representation ▪ Data Accuracy ▪ Implementation
Bilal Saleem	I21-0464	▪ Result Analysis

		▪ Sequential Version ▪ Performance Testing ▪ Implementation
Muhammad Kashif	I21-0851	▪ Parallelization Strategy ▪ Designing Solution ▪ Implementation ▪ Documentation