Project Report

Project Title: Graph traversing algorithms via multithreading (Parallelization using OpenMP)

PDC (CS3006)

Course Instructor: **Sir Kashif**

Group Members:

1) Ashar Usmani 22K-4581 (Group Leader)
2) Abdul Ahad 22K-4353
3) Abrar Malik 22K-4429

Key Features:

BFS (BREADTH FIRST SEARCHING)

GBFS (GREEDY BEST FIRST SEARCHING)

A* (A-STAR ALGORITHM)

INTRODUCTION

Graph algorithms are quite popular when it comes to data structures and algorithms in C++. We decided to test the efficiency of these algorithms when run in serial programs & parallel programs. As we know, multithreading in c++ allows us to perform task parallelism and data parallelism. Keeping this in mind we executed numerous algorithms of graph searching and path finding to see the action of threads in more depth. We wanted to find the efficiency and speedup between the serial and parallel (with different amount of cores) versions.

Code Repository

We used our own code, created with the help of Chatgpt. Our code will basically be implementing popular graph algorithms like A* star, BFS (Breadth First Search), GBFS (Greedy Best First Search) and then comparing between their serial and parallelized versions, in terms of how speedup and efficiency we can get on different number of cores. Our code is quite complex as for A* star algorithm first we need to check all the adjacent nodes, then their heuristic calculations till the end/destination node. Now if we did this for 100,000 nodes it would at least take at minimum 5-10, and if we would increase the number of nodes to a million it would increase to more than 20 minutes. This is time for a serial version of the program. We want to parallelize and test how much difference would it make, and if increasing the number of cores is really worth it or not.

Parallelization Strategy

We will be using OpenMP for creating our parallel version of the program. In A* star algorithm, we will be parallelizing the heuristic calculation part of our code, so we can at once find the best option for the adjacent node. We will be performing similar working for GBFS. And for BFS, we will parallelize the successor generation of each node on the same level as they're independent of each other.

Execution Plan

We will be testing using different number of cores ranging from 1-8. In terms of memory we will be working over 10^5-10^6 nodes in the 3 algorithms and finding our readings in terms of speedup & efficiency. According to our estimations for 1x10^5 nodes, our serial code runs for around 3-5 minutes.

Data & Performance Metrics

We will be testing over a range 10-1 million nodes. We will be checking our performance metrics in terms of execution time, speedup & efficiency. We will compare how much difference our serial and parallelized programs give, and then also check the efficiency of increased cores.

Numerical Results and Visualization

Our expected speedup for each algorithm is 2-4 times. And we expect our execution time to decrease by 2-4 times. Firstly we will be using videos to first record our findings. Then we'll store them to a table with different variables like sample size of the no. of nodes & no. of cores. We'll represent our findings on a graph (bar-graph or a line graph) using python mathplotlib or excel.

Testing and Validation

The correctness of parallel method will be tested by comparing the answer it gives with the corrected serial version. The serial and parallel outputs will be stress tested by generating thousands of test cases and matching the similarity between them. The benefits of parallelization will be analyzed by comparing the run time for each type of code. Along with this, we will test the performance on different types of graph like disconnected, strongly connected, tree and many more. This ensures that each algorithm is run on multiple test cases and use cases and gives the run time for each.

Project breakdown

Ashar Usmani worked on the A* star algorithm, while Abrar worked on the Breadth First Searching and Abdul Ahad worked on Best First Searching. The codes for the parallelization parts were written with mutual understanding.

Plagiarism & Originality Declaration

Our codes for the 3 algorithms were written by us, but we used ChatGPT & other AI tools to help us parallelize some parts of our code. ChatGPT was quite helpful in finding parts of our code where we could parallelize.