Project Report: Multi-threaded Implementation of Floyd-Warshall Algorithm

Introduction:

The Floyd-Warshall algorithm is a classic algorithm used for finding the shortest paths in a weighted graph. In this project, we implemented both single-threaded and multi-threaded versions of the Floyd-Warshall algorithm to observe the performance improvements achieved through parallelization. The primary objective was to compare the runtimes of the single-threaded and multi-threaded versions for graphs of different sizes and analyze the speed-up achieved by parallelization.

Code Explanation:

The implemented Floyd-Warshall algorithm utilizes both single-threaded and multi-threaded approaches to compute the shortest paths in a weighted graph. Below is an explanation of the key components of the code:

The code reads the input graph from a file named "graph.txt". The first line of the file specifies the number of nodes (N) and the number of undirected edges (M) in the graph. It then dynamically allocates memory for a 2D array to represent the adjacency matrix of the graph. The matrix is initialized with a value of infinity (INF) for all non-diagonal elements, indicating that there is no direct edge between nodes initially. The weights of undirected edges are read from the file and used to update the corresponding entries in the adjacency matrix.

The worker function (worker) implements the core logic of the Floyd-Warshall algorithm. It is responsible for updating the distance matrix based on the shortest paths computed using intermediate nodes. In the multi-threaded version, multiple instances of the worker function are executed concurrently using pthreads. Each thread computes the shortest paths for a subset of nodes, thereby parallelizing the computation and potentially reducing execution time.

The code utilizes semaphores (readLock and writeLock) to control access to shared resources (i.e., the distance matrix) and ensure thread safety during updates.

The main loop iterates over all possible intermediate nodes (k) and invokes the worker function to update the distance matrix accordingly. This process is repeated until all nodes have been considered as intermediate nodes. By the end of the algorithm execution, the final distance matrix is printed to the console, displaying the shortest paths between all pairs of nodes.

A screen shot of a computer code

Description automatically generated

Fig 1: Algorithm outputs shortest path between all pairs of nodes

System Configuration:

The experiments were conducted on a system equipped with the following specifications:

- Processor: Intel(R) Core(TM) i3-10110U CPU @ 2.10GHz 2.59 GHz

- Number of cores: 2

- RAM: 32 GB

Experimental Setup:

The implemented Floyd-Warshall algorithm was run on graphs of varying sizes: 10, 100, 1000, and 10000 nodes. For each graph size, both single-threaded and multi-threaded versions were executed, and the runtimes were recorded.

The runtime results obtained for both the single-threaded and multi-threaded versions of the algorithm are summarized in the graph below:

A graph with a blue line

Description automatically generated

Fig 2 : Time Vs #nodes (for non MTFW and MTFW)

From the runtime results, it is evident that the multi-threaded version of the Floyd-Warshall algorithm introduces overhead for all graph sizes (N>=100). This is likely due to the overhead associated with thread creation and synchronization.

Speed-up Calculation:

The speed-up is calculated as the ratio of the time taken by the single-threaded version to the time taken by the multi-threaded version:

The calculated speed-up values for each graph size are as follows:

|  |  |
| --- | --- |
| Graph Size (N) | Speed-up |
| 10 | 0.475 |
| 100 | 0.0066 |
| 1000 | 0.0209 |
| 10000 | 0.1027 |

A graph with a green line

Description automatically generated

Fig 3: Speed up Vs #nodes

If the speed-up ratio is less than 1, it suggests that the single-threaded version is faster than the multi-threaded version. This situation may occur due to various factors, such as overhead associated with thread creation, synchronization, or inefficiencies in parallelization. As we can see, all our speed up values are less than 1 which implies that the parallelization has introduced overhead that outweighs the benefits of parallel execution. In such cases, parallelization may not be advantageous, especially for smaller problem sizes or on systems with limited resources.

Conclusion:

The experiments conducted demonstrate that parallelizing the Floyd-Warshall algorithm using multi-threading does not necessarily lead to significant speed-up for large graph sizes. Also, for smaller graph sizes, the overhead of thread creation and synchronization typically outweighs the benefits of parallelization. It is essential to consider the graph size and system architecture when deciding whether to parallelize algorithms for performance optimization.

References:

[1]. S. Abraham, G.Peter, G.Greg, “Operating Systems Concepts” , 2018

# [2]. “Readers-Writers Problem | Set 1 (Introduction and Readers Preference Solution)

," GeeksforGeeks, [Accessed on March 30, 2024], Available: https://www.geeksforgeeks.org/readers-writers-problem-set-1-introduction-and-readers-preference-solution/”

[3]. C. Ritchie, "The C Programming Language," Prentice Hall, 1978.