Assignment – 3

Group number-29

1. Manvitha Aathmuri – R11847781
2. Raviteja Sirisanagandla - R11851152
3. Sai Tejaswini Chirumamilla - R11840694
4. Preetham Alwala - R11846544
5. Girish Gandham - R11870388
6. Ganesh Reddy Mannem - R11849144

Problem Statement:

The problem at hand is how to parallelize a sequential function that uses Dijkstra's technique to determine the shortest distances in a graph. The function determines the separation between a source vertex and all other vertices by calculating the length of each edge in a given graph as input. By parallelizing two specific for-loops in the code, the performance of this approach is intended to be improved.

Technical Idea:

The 'found' array is initialized and memory is allocated for it at the beginning of the code. Additionally, it parallelizes the initialization of the 'distance' array with data particular to the source vertex using the #pragma omp parallel for. The core of the algorithm is within a 'while' loop that continues until 'count' equals 'n', indicating that all vertices have been processed. Using OpenMP directives, numerous tasks are parallelized within this loop.

The code parallelizes the task of finding the vertex with the least distance. Concurrently, several threads look for the minimal distance and corresponding vertex. Each thread's distance is stored in a private variable called private_tmp, and a crucial section is used to make sure that only one thread updates the shared ‘least’ and ‘leastPos’ variables at a time. The distances to other vertices are updated in another section of the code. Using OpenMP directives, this is parallelized with multiple threads updating the distances at once. When updating the distance array is required, a critical section is employed to make sure that only one thread is doing it.

Conclusion:

The Dijkstra's algorithm is successfully parallelized by the code, allowing it to make use of several processor cores and greatly enhancing its speed on multi-core platforms. OpenMP, a popular method for introducing parallelism to C/C++ applications, is utilized to enable parallelization, and critical sections are employed to assure thread safety when modifying shared variables. The method makes it possible to compute shortest distances in big graphs more quickly, which is useful for a number of applications, including network routing and path planning. By carefully managing shared data and employing OpenMP's parallelization techniques, it guarantees the accuracy of the results. To fully benefit from multi-threaded execution, the parallelized code should be built with OpenMP support enabled in the compiler settings.

Steps to Execute:

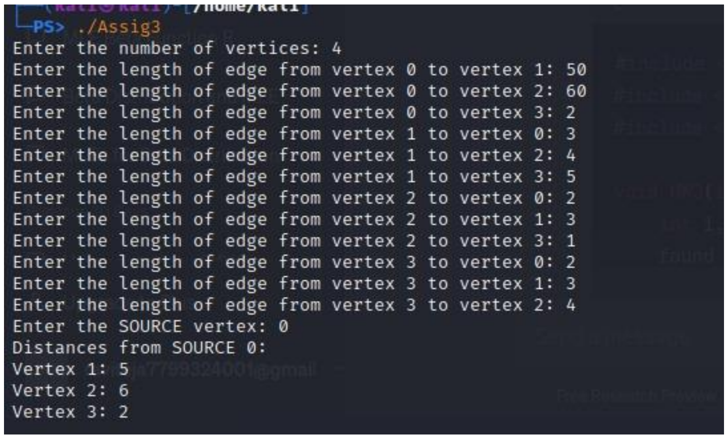
- Open the Group_29_assignment_3.c file's location in terminal.
- Compile the file Group_29_assignment_3.c using the below command:

  mpicc Group_29_assignment_3.c -o Group_29_assignment_3 -lm

- Execute the file using below command for the output to be visible in console log:

  mpirun -np 6 ./ Group_29_assignment_3

Output:



```
└─PS> ./Assig3
Enter the number of vertices: 4
Enter the length of edge from vertex 0 to vertex 1: 50
Enter the length of edge from vertex 0 to vertex 2: 60
Enter the length of edge from vertex 0 to vertex 3: 2
Enter the length of edge from vertex 1 to vertex 0: 3
Enter the length of edge from vertex 1 to vertex 2: 4
Enter the length of edge from vertex 1 to vertex 3: 5
Enter the length of edge from vertex 2 to vertex 0: 2
Enter the length of edge from vertex 2 to vertex 1: 3
Enter the length of edge from vertex 2 to vertex 3: 1
Enter the length of edge from vertex 3 to vertex 0: 2
Enter the length of edge from vertex 3 to vertex 1: 3
Enter the length of edge from vertex 3 to vertex 2: 4
Enter the SOURCE vertex: 0
Distances from SOURCE 0:
Vertex 1: 5
Vertex 2: 6
Vertex 3: 2
```