Assignment – 4

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 aim is to create and execute a Message Passing Interface (MPI) code that parallelizes an algorithm that follows the Floyd-Warshall methodology. The goal of the procedure is to determine the shortest path between each pair of nodes in a matrix-representation of the graph. The inner loops (for-i and for-j) should be parallelized in the code while making sure that each process allots memory for the $n/\sqrt{P} \times n/\sqrt{P}$ sub-matrices of D and D0, where P is the number of processes.

Key Requirements and Constraints:

- The total number of processes is represented as P, and the code assumes that P is a perfect square ($P = \sqrt{P} \times \sqrt{P}$).
- The matrix size 'n' is assumed to be divisible by $\sqrt{P}$.
- The communication cost of the parallel algorithm should be bounded by $O(n\sqrt{P} \log 2\sqrt{P})$ for each k-loop iteration.

Technical Approach:

The technical approach to parallelizing the Floyd-Warshall algorithm with MPI involves several critical steps. In order to establish the communication infrastructure, MPI must first be initialized. The code then verifies that the number of processes, P, makes a perfect square. This guarantees that each process has an equal share of the workload. The sub-matrix dimensions are computed as 'n / sqrt(P)', and the size of the original matrix 'n' is set by the user. A crucial stage in the process is memory allocation. Local sub-matrices, like D0_local and D_local, which stand for the sections of the matrix that each process will work on, are allocated memory. The root process initializes D0_local, and here is where the user enters values for this sub-matrix.

The k-loop, an iteration from 0 to n-1, is the central component of the algorithm. D0_local is broadcast to all processes at the start of every k-loop in order to synchronize their computations. For every sub-matrix, the inner loops for i and j are parallelized, and D_local[i][j] is computed using the floyd_warshall function based on D0_local, D_local, and D0 from all processes. Maintaining coordination throughout the processes depends on communication. For the following k-loop iteration, D0_local is updated depending on D_local, and D_local is broadcast to update D0_local.

To produce the final D matrix, the D_local matrices from each step are collected. The code prints the final D matrix if the current process has a rank of 0. Ultimately, memory leaks are avoided by appropriately managing memory, and MPI is completed to release MPI resources and end parallel processing. By following the limitations outlined in the problem statement, this method guarantees effective Floyd-Warshall algorithm parallelization, making it appropriate for large matrices and high-performance computing settings.

Conclusion:

The provided MPI code parallelizes the Floyd-Warshall algorithm efficiently while respecting the constraints of the problem statement. By splitting the matrix into smaller sub-matrices, it reduces communication costs by enabling each operation to focus on a focused area. Throughout the k-loop iterations, the algorithm keeps synchronization and computes the final D matrix successfully. It can be used in high-performance computing environments and with huge matrices since it complies with the requirements of the problem, which guarantees effective distributed computing and parallelization.

Steps to Execute:

- Open the Group_29_assignment_4.c file's location in terminal.
- Compile the file Group_29_assignment_4.c using the below command:

  gcc -fopenmp -o Group_29_assignment_4 Group_29_assignment_4.c

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

  ./Group_29_assignment_4

Output: