Programming Assignment: Parallelization of Dijkstra's Algorithm

Overview

The goal of this assignment is to implement a parallel version of Dijkstra’s algorithm for finding the shortest paths from a source node to all other nodes in a weighted, undirected graph. Students will choose either OpenMP (shared-memory parallelism) or MPI (distributed-memory parallelism) to optimize the algorithm's performance. This assignment is designed to enhance your understanding of parallel programming concepts applied to graph algorithms.

Learning Objectives

By completing this assignment, you will:

1. Learn how to implement Dijkstra’s algorithm to compute single-source shortest paths.
2. Explore the challenges and strategies of parallelizing graph algorithms.
3. Gain practical experience using OpenMP or MPI for parallel programming.
4. Understand the trade-offs between shared-memory and distributed-memory approaches.

Assignment Details

1. **Input Format**:
   Your program will take a graph in **edge list** format:

   ```
   num_nodes num_edges
   u1 v1 weight1
   u2 v2 weight2
   ...
   ```

   Example:

   ```
   5 6
   0 1 7
   0 2 3
   1 3 9
   2 4 4
   3 4 6
   1 4 2
   ```

2. **Output Format**:
   Your program should output the shortest distance from the source node to all other nodes. For unreachable nodes, display "INF".

   Example Output:

   ```
   Shortest distances from node 0:

   Node 0: 0

   Node 1: 7

   Node 2: 3

   Node 3: 10

   Node 4: 7
   ```

3. **Parallelization Requirements**:

   - Implement the parallel Dijkstra's algorithm using **either OpenMP or MPI**.
   - Use parallelism for the following:
   - Finding the node with the minimum distance.
   - Updating the distances of neighboring nodes.
   - Ensure that your implementation is scalable and handles larger graphs efficiently.

4. **Graph Generation**:

   - Use the provided graph generator (graph_generator.c) to create test cases.
   - The generator produces a weighted, undirected graph in edge list format with configurable node count, edge count, and weight range.
   - Compile the program: gcc -o graph_generator graph_generator.c
   - Run the program:
     ./graph_generator 1000 5000 10 weighted_graph.txt

5. **Performance Evaluation**:

   - Test your implementation on graphs of varying sizes (e.g., 1000 nodes, 10,000 edges).
   - Measure execution time for the following:
   - Sequential Dijkstra's algorithm.
   - Your parallel implementation.
   - Compare the speedup achieved by your parallel implementation.

Submission Guidelines

1. **Code**:
    - Submit your source code (dijkstra_openmp.c or dijkstra_mpi.c) along with any supporting files.
    - Your code should compile and run without errors.
2. **Report**:
    - Submit a short report (1–2 pages) containing:
        - Explanation of your parallelization strategy.
        - Performance analysis (tables/graphs showing speedup and runtime).
        - Challenges and lessons learned.
3. **Input Files**:
    - Include the graphs you used for testing.
4. **Execution Instructions**:
    - Provide a README file with compilation and execution instructions.