**Problem Practical 1 Problem Statement**: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.

**Code:**

#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

using namespace std;

class Graph

{

public:

    int V;

    vector<vector<int>> adj;

    Graph(int vertices)

    {

        V = vertices;

        adj.resize(V);

    }

    void addEdge(int u, int v)

    {

        // Undirected Graph

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    void parallelBFS(int start)

    {

        vector<bool> visited(V, false);

        queue<int> q;

        visited[start] = true;

        q.push(start);

        cout << "Parallel BFS starting from node " << start << ":\n";

        while (!q.empty())

        {

            int levelSize = q.size();

#pragma omp parallel for

            for (int i = 0; i < levelSize; ++i)

            {

                int current;

#pragma omp critical

                {

                    if (!q.empty())

                    {

                        current = q.front();

                        q.pop();

                        cout << "Thread " << omp\_get\_thread\_num() << " visiting " << current << endl;

                    }

                }

#pragma omp parallel for

                for (int j = 0; j < adj[current].size(); ++j)

                {

                    int neighbor = adj[current][j];

#pragma omp critical

                    {

                        if (!visited[neighbor])

                        {

                            visited[neighbor] = true;

                            q.push(neighbor);

                        }

                    }

                }

            }

        }

    }

    void parallelDFSUtil(int v, vector<bool> &visited)

    {

#pragma omp critical

        {

            visited[v] = true;

            cout << "Thread " << omp\_get\_thread\_num() << " visiting " << v << endl;

        }

        for (int u : adj[v])

        {

            if (!visited[u])

            {

#pragma omp task firstprivate(u)

                {

                    parallelDFSUtil(u, visited);

                }

            }

        }

    }

    void parallelDFS(int start)

    {

        vector<bool> visited(V, false);

        cout << "Parallel DFS starting from node " << start << ":\n";

#pragma omp parallel

        {

#pragma omp single

            {

                parallelDFSUtil(start, visited);

            }

        }

    }

};

int main()

{

    int V;

    cout << "Enter number of nodes: ";

    cin >> V;

    Graph g(V);

    int E;

    cout << "Enter number of edges: ";

    cin >> E;

    cout << "Enter " << E << " edges (u v):\n";

    for (int i = 0; i < E; ++i)

    {

        int u, v;

        cin >> u >> v;

        g.addEdge(u, v);

    }

    int startNode;

    cout << "Enter start node for BFS/DFS: ";

    cin >> startNode;

    double start\_time = omp\_get\_wtime();

    g.parallelBFS(startNode);

    double end\_time = omp\_get\_wtime();

    cout << "BFS Execution time: " << (end\_time - start\_time) << " seconds\n\n";

    start\_time = omp\_get\_wtime();

    g.parallelDFS(startNode);

    end\_time = omp\_get\_wtime();

    cout << "DFS Execution time: " << (end\_time - start\_time) << " seconds\n";

    return 0;

}  
  
Steps to execute the code:

1. Open the cmd in the folder where the code is present
2. Run the command g++ -fopenmp file\_name.cpp -o file\_name
3. Run the command .\file\_name

Output:

C:\Users\Shravan\OneDrive\Desktop\Engineering Degree Stuff\4th Year Stuff\8th Sem Stuff\LP-5 Problem Statement & Programs\Programs\HPC Practical 1>**g++ -fopenmp bfs\_dfs.cpp -o bfs\_dfs**

C:\Users\Shravan\OneDrive\Desktop\Engineering Degree Stuff\4th Year Stuff\8th Sem Stuff\LP-5 Problem Statement & Programs\Programs\HPC Practical 1>**.\bfs\_dfs**

Enter number of nodes: 6

Enter number of edges: 7

Enter 7 edges (u v):

0 1

0 2

0 3

1 4

2 5

3 5

4 5

Enter start node for BFS/DFS: 0

Parallel BFS starting from node 0:

Thread 0 visiting 0

Thread 2 visiting 1

Thread 1 visiting 2

Thread 0 visiting 3

Thread 1 visiting 4

Thread 0 visiting 5

BFS Execution time: 0.0139999 seconds

Parallel DFS starting from node 0:

Thread 4 visiting 0

Thread 1 visiting 1

Thread 4 visiting 2

Thread 3 visiting 3

Thread 1 visiting 4

Thread 4 visiting 5

Thread 3 visiting 5

Thread 1 visiting 5

Thread 4 visiting 3

Thread 2 visiting 4

Thread 2 visiting 1

Thread 5 visiting 4

Thread 5 visiting 1

Thread 0 visiting 3

Thread 3 visiting 2

Thread 1 visiting 2

DFS Execution time: 0.0150001 seconds