BFS DFS PARALLEL

```
#include <iostream>

#include <vector>
#include <queue>
#include <stack>
#include <omp.h>

using namespace std;

void bfs_parallel(vector<vector<int>> &graph, int start, vector<int> &visited)
{
    queue<int> q;
    visited[start] = 1;
    q.push(start);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";
        #pragma omp parallel for
        for (int i = 0; i < graph[v].size(); i++) {
            int u = graph[v][i];
            if (!visited[u]) {
                visited[u] = 1;
                q.push(u);
            }
        }
    }
}

void dfs_parallel(vector<vector<int>> &graph, int start, vector<int> &visited)
{
    stack<int> s;
    s.push(start);
    while (!s.empty()) {
        int v = s.top();
        s.pop();
        if (!visited[v]) {
            visited[v] = 1;
            cout << v << " ";
            #pragma omp parallel for
            for (int i = graph[v].size() - 1; i >= 0; i--) {
                int u = graph[v][i];
                if (!visited[u]) {
                    s.push(u);
                }
            }
        }
```

```
    }
}

int main() {
    int n = 8;
    vector<vector<int>> graph(n);
    graph[0].push_back(1);
    graph[0].push_back(2);
    graph[1].push_back(3);
    graph[1].push_back(4);
    graph[2].push_back(5);
    graph[2].push_back(6);
    graph[3].push_back(7);
    vector<int> visited(n, 0);
    cout << "Parallel BFS Traversal:" ;
    bfs_parallel(graph, 0, visited);
    cout << endl;
    visited.assign(n, 0);
    cout << "Parallel DFS Traversal:" ;
    dfs_parallel(graph, 0, visited);
    cout << endl;
    return 0;
}
```

Output: