```
#include <iostream>
#include <vector>
#include <queue>
#include <omp.h>

using namespace std;

// Graph class representing the adjacency list
class Graph {
    int v; // number of vertices
    vector<vector<int>> adj; // adjacency list

public:
    Graph(int v) : v(v), adj(v) {}

    // Add an edge to the graph
    void addEdge(int v, int w) {
        adj[v].push_back(w);
    }

    // Parallel depth-first search
    void parallel(int startvertex) {
        vector<bool> visited(v, false);
        paralleldfs(startvertex, visited);
    }

    // Parallel DFS utility function
    void paralleldfs(int v, vector<bool>& visited) {
        visited[v] = true;
        cout << v << " ";

        #pragma omp parallel for
        for (int i = 0; i < adj[v].size(); ++i) {
            int n = adj[v][i];
            if (!visited[n])
                paralleldfs(n, visited);
        }
    }

    // Parallel breadth-first search
    void parallelBFS(int startvertex) {
        vector<bool> visited(v, false);
        queue<int> q;
        visited[startvertex] = true;
        q.push(startvertex);

        while (!q.empty()) {
            int v = q.front();
            q.pop();
            cout << v << " ";

            #pragma omp parallel for
            for (int i = 0; i < adj[v].size(); ++i) {
                int n = adj[v][i];
                if (!visited[n]) {
                    visited[n] = true;
                    q.push(n);
                }
            }
        }
    }
};

int main() {
    // Create a graph
    Graph g(7);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 0);

    cout << "Depth-first search (DFS): ";
    g.parallel(0);
    cout << endl;

    cout << "Breadth-first search (BFS): ";
    g.parallelBFS(0);
    cout << endl;

    return 0;
}
```