```cpp
#include<bits/stdc++.h>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

void bfs(TreeNode* root) {
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        cout << node->val << " ";
        if (node->left) {
            q.push(node->left);
        }
        if (node->right) {
            q.push(node->right);
        }
    }
}

void parallel_bfs(TreeNode* root) {
    queue<TreeNode*> q;
    q.push(root);
    #pragma omp parallel
    {
        while (!q.empty()) {
            #pragma omp for
            for (int i = 0; i < q.size(); i++) {
                TreeNode* node = q.front();
                q.pop();
                #pragma omp critical
                {
                    cout << node->val << " ";
                }
```

```
                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }
        }
    }
}

void dfs(TreeNode* root) {
    stack<TreeNode*> s;
    s.push(root);
    while (!s.empty()) {
        TreeNode* node = s.top();
        s.pop();
        cout << node->val << " ";
        if (node->right) {
            s.push(node->right);
        }
        if (node->left) {
            s.push(node->left);
        }
    }
}

void parallel_dfs(TreeNode* root) {
    stack<TreeNode*> s;
    s.push(root);
    #pragma omp parallel
    {
        while (!s.empty()) {
            #pragma omp for
            for (int i = 0; i < s.size(); i++) {
                TreeNode* node = s.top();
                s.pop();
                #pragma omp critical
                {
                    cout << node->val << " ";
```

```
                }
                if (node->right) {
                    s.push(node->right);
                }
                if (node->left) {
                    s.push(node->left);
                }
            }
        }
    }
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);
    root->left->left->left = new TreeNode(8);
    root->left->right->left = new TreeNode(9);
    root->right->right->left = new TreeNode(10);
    root->right->left->left = new TreeNode(11);
    root->left->right->right = new TreeNode(12);
    root->right->right->right = new TreeNode(9);

    cout << "BFS traversal: ";
    auto start = chrono::high_resolution_clock::now();
    bfs(root);
    auto end = chrono::high_resolution_clock::now();
    cout << "\nBFS took " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << "
microseconds." << endl;
    auto serb=chrono::duration_cast<chrono::microseconds>(end -
start).count();
    cout << endl;

    cout << "Parallel BFS traversal: ";
    start = chrono::high_resolution_clock::now();
```

```
    parallel_bfs(root);
    end = chrono::high_resolution_clock::now();
    auto perb=chrono::duration_cast<chrono::microseconds>(end -
start).count();
    cout << "\nParallel BFS took " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << "
microseconds." << endl;
    cout<<"Speed up : "<<(float)serb/perb<<"\n";

    cout <<
"-----------------------------------------------------"<<endl;

    cout << "DFS traversal: ";
    start = chrono::high_resolution_clock::now();
    dfs(root);
    end = chrono::high_resolution_clock::now();
    auto ser=chrono::duration_cast<chrono::microseconds>(end -
start).count();
    cout << "\nDFS took " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << "
microseconds." << endl;
    cout << endl;

    cout << "Parallel DFS traversal: ";
    start = chrono::high_resolution_clock::now();
    parallel_dfs(root);
    end = chrono::high_resolution_clock::now();
    auto per=chrono::duration_cast<chrono::microseconds>(end -
start).count();
    cout << "\nParallel DFS took " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << "
microseconds." << endl;

    cout<<"Speed up : "<<(float)ser/per<<"\n";



    return 0;
}
```

Output :

```
BFS traversal: 1 2 3 4 5 6 7 8 9 12 11 10 9
BFS took 102 microseconds.

Parallel BFS traversal: 1 2 3 4 5 6 7 8 9 12 11 10 9
Parallel BFS took 2 microseconds.
Speed up : 51
------------------------------------------------------------
DFS traversal: 1 2 4 8 5 9 12 3 6 11 7 10 9
DFS took 3 microseconds.

Parallel DFS traversal: 1 2 4 8 5 9 12 3 6 11 7 10 9
Parallel DFS took 2 microseconds.
Speed up : 1.5
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-exkvjpy3.5y
i" 1>"/tmp/Microsoft-MIEngine-Out-em2aoq4w.l3s"
```