HPC Lab Exp No.1

#include <omp.h>

#include <iostream>

#include <queue>

#include <vector>

#include <chrono>

using namespace std;

using namespace chrono;

class Node

{

public:

    int value;

    Node \*left, \*right;

    Node(int v) : value(v), left(NULL), right(NULL) {}

};

Node \*generateTree(vector<int> &values)

{

    if (values.empty())

        return NULL;

    vector<Node \*> nodes(values.size());

    for (int i = 0; i < values.size(); i++)

        nodes[i] = new Node(values[i]);

    int parent = 0, child = 1;

    while (child < values.size())

    {

        if (nodes[parent])

        {

            nodes[parent]->left = (child < values.size()) ? nodes[child++] : NULL;

            nodes[parent]->right = (child < values.size()) ? nodes[child++] : NULL;

        }

        parent++;

    }

    return nodes[0];

}

void parallel\_bfs(Node \*root)

{

    if (!root)

        return;

    queue<Node \*> q;

    q.push(root);

    while (!q.empty())

    {

        int size = q.size();

        vector<Node \*> level;

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

        {

            level.push\_back(q.front());

            q.pop();

        }

#pragma omp parallel for

        for (int i = 0; i < level.size(); i++)

        {

#pragma omp critical

            cout << level[i]->value << " -> ";

        }

        for (auto node : level)

        {

            if (node->left)

                q.push(node->left);

            if (node->right)

                q.push(node->right);

        }

    }

}

void parallel\_dfs(Node \*root)

{

    if (!root)

        return;

#pragma omp critical

    cout << root->value << " -> ";

#pragma omp parallel sections

    {

#pragma omp section

        parallel\_dfs(root->left);

#pragma omp section

        parallel\_dfs(root->right);

    }

}

int main()

{

    int n;

    cout << "Enter number of nodes (in space separated form): ";

    cin >> n;

    vector<int> values(n);

    cout << "Enter " << n << " node values:\n";

    for (int i = 0; i < n; i++)

        cin >> values[i];

    Node \*root = generateTree(values);

    auto start\_bfs = high\_resolution\_clock::now();

    cout << "\nParallel BFS: ";

    parallel\_bfs(root);

    auto stop\_bfs = high\_resolution\_clock::now();

    cout << "\nExecution time for Parallel BFS: " << duration\_cast<milliseconds>(stop\_bfs - start\_bfs).count() << " ms\n\n";

    auto start\_dfs = high\_resolution\_clock::now();

    cout << "Parallel DFS: ";

    parallel\_dfs(root);

    auto stop\_dfs = high\_resolution\_clock::now();

    cout << "\nExecution time for Parallel DFS: " << duration\_cast<milliseconds>(stop\_dfs - start\_dfs).count() << " ms\n";

    return 0;

}

OUTPUT:

