**Assignment No 1**

Name: Poonam Kisan Salbande

Roll No: 20121011

Title: Design and implement Parallel Breadth First Search based on existing algorithms using

open MP use a tree or an undirected graph for BFS.

Code:-

#include<iostream>

#include <queue>

#include <omp.h>

#define MAX\_NODES 1000

using namespace std;

int visited[MAX\_NODES];

int n\_threads;

struct TreeNode {

int value;

vector<TreeNode\*> children;

};

void bfs(TreeNode\* root) {

queue<TreeNode\*> q;

q.push(root);

visited[root->value] = 1;

while (!q.empty()) {

#pragma omp parallel num\_threads(n\_threads)

{

#pragma omp for

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

TreeNode\* node = q.front();

q.pop();

for (auto child : node->children) {

if (!visited[child->value]) {

visited[child->value] = 1;

q.push(child);

}

}

}

}

}

}

int main() {

// Construct a tree

TreeNode\* root = new TreeNode{0};

for (int i = 1; i < 10; i++) {

TreeNode\* child = new TreeNode{i};

root->children.push\_back(child);

for (int j = 0; j < i; j++) {

TreeNode\* grandchild = new TreeNode{j};

child->children.push\_back(grandchild);

}

}

// Initialize OpenMP

n\_threads = omp\_get\_max\_threads();

// Run BFS

bfs(root);

// Print visited nodes

for (int i = 0; i < 10; i++) {

if (visited[i]) {

cout << i << " ";

}

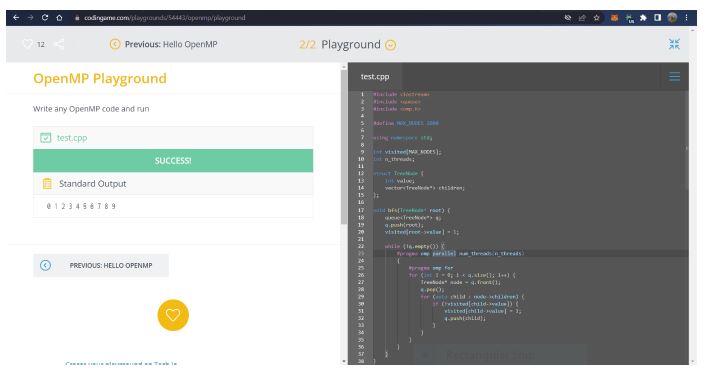
}

cout << endl;

return 0;

}

Output:



Title: Design and implement Parallel Depth First Search based on existing algorithms using

open MP use a tree or an undirected graph for DFS.

#include <iostream>

#include <vector>

#include <omp.h>

#define MAX\_NODES 1000

using namespace std;

int result[MAX\_NODES];

int n\_threads;

struct TreeNode {

int value;

vector<TreeNode\*> children;

};

void dfs(TreeNode\* node, int depth) {

// Base case: leaf node

if (node->children.empty()) {

#pragma omp critical

{

result[depth] += node->value;

}

} else {

// Recursive case: internal node

#pragma omp parallel num\_threads(n\_threads)

{

#pragma omp single nowait

{

for (auto child : node->children) {

#pragma omp task

{

dfs(child, depth + 1);

}

}

}

}

}

}

int main() {

// Construct a tree

TreeNode\* root = new TreeNode{1};

TreeNode\* child1 = new TreeNode{2};

TreeNode\* child2 = new TreeNode{3};

root->children.push\_back(child1);

root->children.push\_back(child2);

TreeNode\* grandchild1 = new TreeNode{4};

TreeNode\* grandchild2 = new TreeNode{5};

TreeNode\* grandchild3 = new TreeNode{6};

child1->children.push\_back(grandchild1);

child2->children.push\_back(grandchild2);

child2->children.push\_back(grandchild3);

// Initialize OpenMP

n\_threads = omp\_get\_max\_threads();

// Run DFS

dfs(root, 0);

// Print results

for (int i = 0; i < MAX\_NODES; i++) {

if (result[i] > 0) {

cout << "Depth " << i << ": " << result[i] << endl;

}

}

return 0;

}

Output:

