**Practical 1 : BFS ( Sequential and Parallel )**

#include <iostream>

#include <queue>

#include <omp.h>

#include <chrono>

#include <cstdlib>

using namespace std;

using namespace std::chrono;

class node

{

public:

node \*left, \*right;

int data;

};

node \*insert(node \*root, int data)

{

if (!root)

{

root = new node;

root->left = NULL;

root->right = NULL;

root->data = data;

return root;

}

queue<node \*> q;

q.push(root);

while (!q.empty())

{

node \*temp = q.front();

q.pop();

if (temp->left == NULL)

{

temp->left = new node;

temp->left->left = NULL;

temp->left->right = NULL;

temp->left->data = data;

return root;

}

else

{

q.push(temp->left);

}

if (temp->right == NULL)

{

temp->right = new node;

temp->right->left = NULL;

temp->right->right = NULL;

temp->right->data = data;

return root;

}

else

{

q.push(temp->right);

}

}

return root;

}

void bfs(node \*head)

{

queue<node \*> q;

q.push(head);

while (!q.empty())

{

node \*currNode = q.front();

q.pop();

cout<<currNode->data<<", ";

if (currNode->left)

q.push(currNode->left);

if (currNode->right)

q.push(currNode->right);

}

}

int main()

{

cout << "This is Atharva Pingale's Code";

cout << "\nPractical 1 : BFS ( Sequential and Parallel )";

node \*root = NULL;

node \*root2 = NULL;

int data;

long int n, i;

double start\_time, end\_time;

cout << "\n\nEnter number of nodes : ";

cin >> n;

for (i = 0; i < n; i++)

{

int random\_value = (rand() % (999999 - 999 + 1) + 999);

root = insert(root, random\_value);

root2 = insert(root2, random\_value);

}

// Sequential BFS timing

start\_time = omp\_get\_wtime();

bfs(root);

end\_time = omp\_get\_wtime();

double seq\_time = end\_time - start\_time;

// Parallel BFS timing

start\_time = omp\_get\_wtime();

queue<node \*> q;

q.push(root2);

bool empty\_flag = false;

#pragma omp parallel

{

while (true)

{

node \*currNode;

bool local\_empty\_flag = false;

#pragma omp critical

{

if (!q.empty())

{

currNode = q.front();

q.pop();

}

else

{

local\_empty\_flag = true;

}

}

#pragma omp critical

{

empty\_flag = empty\_flag || local\_empty\_flag;

}

if (empty\_flag)

break;

#pragma omp single nowait

{

cout << "\t" << currNode->data; // Print the node

}

#pragma omp critical

{

if (currNode->left)

q.push(currNode->left); // Push the left child

if (currNode->right)

q.push(currNode->right); // Push the right child

}

}

}

end\_time = omp\_get\_wtime();

double parallel\_time = end\_time - start\_time;

cout << "\n\nSequential BFS Time: " << seq\_time << " seconds";

cout << "\n\nParallel BFS Time: " << parallel\_time << " seconds\n";

delete root;

delete root2;

return 0;

}

**Output :**

****

Sequential BFS Time : 0.072 seconds

Parallel BFS Time : 0 seconds

****

Sequential BFS Time : 0.192 seconds

Parallel BFS Time : 0 seconds



Sequential BFS Time : 0.106 seconds

Parallel BFS Time : 0 seconds