**Problem Statement:** Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

**Code:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <omp.h>

using namespace std;

// Sequential Bubble Sort

void bubbleSort(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n-1; i++)

for (int j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

swap(arr[j], arr[j+1]);

}

// Parallel Bubble Sort using Odd-Even Transposition

void parallelBubbleSort(vector<int>& arr) {

int n = arr.size();

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

#pragma omp parallel for

for (int j = (i % 2); j < n - 1; j += 2) {

if (arr[j] > arr[j + 1])

swap(arr[j], arr[j + 1]);

}

}

}

// Sequential Merge Sort

void merge(vector<int>& arr, int l, int m, int r) {

int n1 = m - l + 1, n2 = r - m;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[l + i];

for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];

int i = 0, j = 0, k = l;

while (i < n1 && j < n2)

arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(vector<int>& arr, int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

// Parallel Merge Sort using OpenMP

void parallelMergeSort(vector<int>& arr, int l, int r, int depth = 0) {

if (l < r) {

int m = l + (r - l) / 2;

if (depth < 4) { // limit depth to avoid oversubscription

#pragma omp parallel sections

{

#pragma omp section

parallelMergeSort(arr, l, m, depth + 1);

#pragma omp section

parallelMergeSort(arr, m + 1, r, depth + 1);

}

} else {

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

}

merge(arr, l, m, r);

}

}

int main() {

int n;

cout << "Enter size of array: ";

cin >> n;

vector<int> arr(n);

srand(time(0));

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

arr[i] = rand() % 10000;

// Sequential Bubble Sort

vector<int> b1 = arr;

double start = omp\_get\_wtime();

bubbleSort(b1);

double end = omp\_get\_wtime();

cout << "Sequential Bubble Sort time: " << (end - start) << " seconds\n";

// Parallel Bubble Sort

vector<int> b2 = arr;

start = omp\_get\_wtime();

parallelBubbleSort(b2);

end = omp\_get\_wtime();

cout << "Parallel Bubble Sort time: " << (end - start) << " seconds\n";

// Sequential Merge Sort

vector<int> m1 = arr;

start = omp\_get\_wtime();

mergeSort(m1, 0, n - 1);

end = omp\_get\_wtime();

cout << "Sequential Merge Sort time: " << (end - start) << " seconds\n";

// Parallel Merge Sort

vector<int> m2 = arr;

start = omp\_get\_wtime();

parallelMergeSort(m2, 0, n - 1);

end = omp\_get\_wtime();

cout << "Parallel Merge Sort time: " << (end - start) << " seconds\n";

return 0;

}

**Steps to run the code:**

1. Enter Command “ g++ -fopenmp -O2 file\_name -o file\_name ”
2. Enter command “.\file\_name ”

**Output:**

C:\Users\Shravan\OneDrive\Desktop\Engineering Degree Stuff\4th Year Stuff\8th Sem Stuff\LP-5 Problem Statement & Programs\Programs\HPC Practical 2>g++ -fopenmp -O2 bubble\_merge\_sort.cpp -o bubble\_merge\_sort

C:\Users\Shravan\OneDrive\Desktop\Engineering Degree Stuff\4th Year Stuff\8th Sem Stuff\LP-5 Problem Statement & Programs\Programs\HPC Practical 2>.\bubble\_merge\_sort

Enter size of array: 50000

Sequential Bubble Sort time: 18.643 seconds

Parallel Bubble Sort time: 11.143 seconds

Sequential Merge Sort time: 0.029 seconds

Parallel Merge Sort time: 0.0480001 seconds