**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.**

**1) Parallel Bubble Sort**

**Code:**

#include <iostream>

#include <vector>

#include <omp.h>

#include <chrono>

using namespace std;

using namespace chrono;

void parallel\_bubble\_sort(vector<int>& arr) {

int n = arr.size();

bool swapped = true;

omp\_set\_num\_threads(2); // set number of threads to 2

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

swapped = false;

#pragma omp parallel for shared(arr)

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

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

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

swapped = true;

}

}

}

}

int main() {

vector<int> arr = {5, 1, 4, 2, 8, 9, 7, 6, 34, 11, 3, 50};

srand(time(nullptr)); // Seed the random number generator with the current time

//vector<int> arr(1000); // Create a vector of size 1000

//for (int i = 0; i < 1000; i++) {

// arr[i] = rand() % 1000;

//}

cout << "Original array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

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

parallel\_bubble\_sort(arr);

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(end - start);

cout << "Sorted array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

//cout << "Array sorted";

cout << endl;

cout << "Time taken by bubble sort: " << duration.count() << " microseconds" << endl;

int num\_threads = omp\_get\_max\_threads();

cout << "Number of threads used by OpenMP: " << num\_threads << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp bubble\_sort\_parallel.cpp -o bubble\_sort\_parallel***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ **.*/bubble\_sort***

***\_parallel***

***Original array: 5 1 4 2 8 9 7 6 34 11 3 50***

***Sorted array: 1 2 3 4 5 6 7 8 9 11 34 50***

***Time taken by bubble sort: 330 microseconds***

***Number of threads used by OpenMP: 2***

**Case 2 :Sorting random 1000 elements.**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp bubble\_sort\_parallel.cpp -o bubble\_sort\_parallel***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ **.*/bubble\_sort***

***\_parallel***

***Array sorted***

***Time taken by bubble sort: 51384 microseconds***

***Number of threads used by OpenMP:2***

**2) Sequential Bubble Sort**

**Code:**

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using namespace chrono;

void bubble\_sort(vector<int>& arr) {

int n = arr.size();

bool swapped;

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

swapped = false;

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

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

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

swapped = true;

}}

if (!swapped) {

break;

}}}

int main() {

vector<int> arr = {5, 1, 4, 2, 8};

//vector<int> arr(1000); // Create a vector of size 1000

//for (int i = 0; i < 1000; i++) {

//arr[i] = rand() % 1000;

//}

cout << "Original array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

// Time the sort function

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

bubble\_sort(arr);

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(end - start);

//cout << "Array sorted";

cout << "Sorted array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

cout << "Time taken by bubble sort: " << duration.count() << " microseconds" << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp bubble\_sort\_sequential.cpp -o bubble\_sort\_sequential***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./bubble\_sort\_sequential***

***Original array: 5 1 4 2 8***

***Sorted array: 1 2 4 5 8***

***Time taken by bubble sort: 2 microseconds***

**Case 2 :Sorting random 1000 elements.**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp bubble\_sort\_sequential.cpp -o bubble\_sort\_sequential***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./bubble\_sort\_sequential***

***Array sorted***

***Time taken by bubble sort: 28696 microseconds***

**3) Parallel Merge Sort**

**Code:**

#include <iostream>

#include <vector>

#include <omp.h>

#include <cstdlib>

#include <ctime>

#include <chrono>

using namespace std;

using namespace chrono;

void merge(vector<int>& arr, int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

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

for (i = 0; i < n1; i++)

L[i] = arr[left + i];

for (j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void merge\_sort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

#pragma omp parallel sections num\_threads(2)

// parallelize the two recursive calls to merge\_sort using two threads

{

#pragma omp section

{

merge\_sort(arr, left, mid);

}

#pragma omp section

{

merge\_sort(arr, mid + 1, right);

}

}

merge(arr, left, mid, right);

}

}

int main() {

vector<int> arr = { 38, 27, 43, 3, 9, 82, 10 };

int n = arr.size();

cout << "Original array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

//srand(time(nullptr)); // Seed the random number generator with the current time

//vector<int> arr(1000); // Create a vector of size 1000

// Generate random numbers between 0 and 999 and insert them into the vector

//for (int i = 0; i < 1000; i++) {

// arr[i] = rand() % 1000;

//}

//int n = arr.size();

// Time the sort function

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

merge\_sort(arr, 0, n - 1);

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(end - start);

cout << "Sorted array: ";

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

cout << arr[i] << " ";

}

cout << endl;

//cout << "Array sorted";

cout << endl;

cout << "Time taken by merge sort: " << duration.count() << " microseconds" << endl;

// Get the number of threads used by OpenMP

int num\_threads = omp\_get\_max\_threads();

cout << "Number of threads used by OpenMP: " << num\_threads << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp merge\_sort\_parallel.cpp -o merge\_sort\_parallel***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./merge\_sort\_parallel***

***Number of threads used by OpenMP: 20***

***Original array: 38 27 43 3 9 82 10***

***Sorted array: 3 9 10 27 38 43 82***

***Time taken by merge sort: 446 microseconds***

***Number of threads used by OpenMP: 20***

**Case 2 :Sorting random 1000 elements.**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp merge\_sort\_parallel.cpp -o merge\_sort\_parallel***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./merge\_sort\_parallel***

***Array sorted***

***Time taken by bubble sort: 3795 microseconds***

***Number of threads used by OpenMP: 20***

**4) Sequential Merge Sort**

**Code:**

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using namespace chrono;

void merge(vector<int>& arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

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

for (int i = 0; i < n1; i++) {

L[i] = arr[left + i];

}

for (int j = 0; j < n2; j++) {

R[j] = arr[mid + 1 + j];

}

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

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void merge\_sort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

merge\_sort(arr, left, mid);

merge\_sort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

vector<int> arr = {5, 1, 4, 2, 8};

//vector<int> arr(1000); // Create a vector of size 1000

//for (int i = 0; i < 1000; i++) {

// arr[i] = rand() % 1000;

//}

cout << "Original array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

// Time the sort function

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

merge\_sort(arr, 0, arr.size() - 1);

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(end - start);

//cout << "Array sorted";

cout << "Sorted array: ";

for (int x : arr) {

cout << x << " ";

}

cout << endl;

cout << "Time taken by merge sort: " << duration.count() << " microseconds" << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp merge\_sort\_sequential.cpp -o merge\_sort\_sequential***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./merge\_sort\_sequential***

***Original array: 5 1 4 2 8***

***Sorted array: 1 2 4 5 8***

***Time taken by merge sort: 20 microseconds***

**Case 2 :Sorting random 1000 elements.**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***g++ -fopenmp merge\_sort\_sequential.cpp -o merge\_sort\_sequential***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_2$ ***./merge\_sort\_sequential***

***Array sorted***

***Time taken by merge sort: 2159 microseconds***