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.

Name: Onasvee Banarse

Roll No: 09

BE COMPUTER SHIFT 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};
    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 << 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:

PS C:\Practical_2> g++ -fopenmp bubble_sort_parallel.cpp -o bubblesortp

PS C:\Practical_2> .\bubblesortp.exe

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: 1002 microseconds.

Number of threads used by OpenMP: 2

Case 2: Sorting random 1000 elements.

PS C:\Practical_2> g++ -fopenmp bubble_sort_parallel.cpp -o bubblesortp

PS C:\Practical_2> .\bubblesortp.exe

Array Sorted

Time taken by bubble sort: 25006 microseconds.

Number of threads used by OpenMP: 2

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;
```

```
    // }


    // 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 << endl;
    cout << "Time taken by bubble 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:

PS C:\Practical_2> g++ -fopenmp merge_sort_parallel.cpp -o mergesortp

PS C:\Practical_2> .\mergesortp.exe

Original array: 38 27 43 3 9 82 10

Sorted array: 3 9 10 27 38 43 82

Time taken by bubble sort: 0 microseconds

Number of threads used by OpenMP: 12

Case 2: Sorting random 1000 elements.

PS C:\Practical_2> g++ -fopenmp merge_sort_parallel.cpp -o mergesortp

PS C:\Practical_2> .\mergesortp.exe

Array Sorted

Time taken by bubble sort: 2998 microseconds

Number of threads used by OpenMP: 12