#include <iostream>

#include <omp.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

using namespace std;

void bubbleSortSequential(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]);

}

void bubbleSortParallel(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]);

}

}

}

}

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

vector<int> temp(right - left + 1);

int i = left, j = mid + 1, k = 0;

while(i <= mid && j <= right) {

if(arr[i] <= arr[j]) temp[k++] = arr[i++];

else temp[k++] = arr[j++];

}

while(i <= mid) temp[k++] = arr[i++];

while(j <= right) temp[k++] = arr[j++];

for(i = left; i <= right; ++i) arr[i] = temp[i - left];

}

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

if(left < right) {

int mid = (left + right) / 2;

mergeSortSequential(arr, left, mid);

mergeSortSequential(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void mergeSortParallel(vector<int>& arr, int left, int right, int depth = 0) {

if(left < right) {

int mid = (left + right) / 2;

if (depth < 4) {

#pragma omp parallel sections

{

#pragma omp section

mergeSortParallel(arr, left, mid, depth + 1);

#pragma omp section

mergeSortParallel(arr, mid + 1, right, depth + 1);

}

} else {

mergeSortSequential(arr, left, mid);

mergeSortSequential(arr, mid + 1, right);

}

merge(arr, left, mid, right);

}

}

void printArray(const vector<int>& arr) {

for (int val : arr) cout << val << " ";

cout << "\n";

}

int main() {

int size;

cout << "Enter the number of elements: ";

cin >> size;

vector<int> original(size);

cout << "Enter " << size << " integers:\n";

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

cin >> original[i];

vector<int> arr1 = original;

double start = omp\_get\_wtime();

bubbleSortSequential(arr1);

double end = omp\_get\_wtime();

cout << "\nSorted (Sequential Bubble Sort): ";

printArray(arr1);

cout << "Time: " << end - start << " seconds\n";

vector<int> arr2 = original;

start = omp\_get\_wtime();

bubbleSortParallel(arr2);

end = omp\_get\_wtime();

cout << "\nSorted (Parallel Bubble Sort): ";

printArray(arr2);

cout << "Time: " << end - start << " seconds\n";

vector<int> arr3 = original;

start = omp\_get\_wtime();

mergeSortSequential(arr3, 0, arr3.size() - 1);

end = omp\_get\_wtime();

cout << "\nSorted (Sequential Merge Sort): ";

printArray(arr3);

cout << "Time: " << end - start << " seconds\n";

vector<int> arr4 = original;

start = omp\_get\_wtime();

mergeSortParallel(arr4, 0, arr4.size() - 1);

end = omp\_get\_wtime();

cout << "\nSorted (Parallel Merge Sort): ";

printArray(arr4);

cout << "Time: " << end - start << " seconds\n";

return 0;

}

#include <iostream>

#include <vector>

#include <algorithm>

#include <cstdlib>

#include <ctime>

#include <omp.h>

void sequentialBubbleSort(std::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])

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

}

void parallelBubbleSort(std::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]) {

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

}

}

}

}

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

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

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

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

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

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 sequentialMergeSort(std::vector<int>& arr, int l, int r) {

if (l < r) {

int m = (l + r) / 2;

sequentialMergeSort(arr, l, m);

sequentialMergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

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

if (l < r) {

int m = (l + r) / 2;

if (depth < 4) { // limit depth to avoid thread explosion

#pragma omp parallel sections

{

#pragma omp section

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

#pragma omp section

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

}

} else {

sequentialMergeSort(arr, l, m);

sequentialMergeSort(arr, m + 1, r);

}

merge(arr, l, m, r);

}

}

void printArray(const std::vector<int>& arr) {

for (int val : arr)

{

std::cout << val << " ";

}

std::cout << "\n";

}

int main() {

int size;

std::cout << "Enter the number of elements: ";

std::cin >> size;

std::vector<int> original(size);

std::cout << "Enter " << size << " integers:\n";

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

std::cin >> original[i];

std::vector<int> arr1 = original;

double t1 = omp\_get\_wtime();

sequentialBubbleSort(arr1);

double t2 = omp\_get\_wtime();

printArray(arr1);

std::cout << "Time: " << t2 -t1 << " seconds\n";

std::vector<int> arr2 = original;

t1 = omp\_get\_wtime();

parallelBubbleSort(arr2);

t2 = omp\_get\_wtime();

printArray(arr2);

std::cout << "Time: " << t2 -t1 << " seconds\n";

std::vector<int> arr3 = original;

t1 = omp\_get\_wtime();

sequentialMergeSort(arr3, 0, size - 1);

t2 = omp\_get\_wtime();

printArray(arr3);

std::cout << "Time: " << t2 -t1 << " seconds\n";

std::vector<int> arr4 = original;

t1 = omp\_get\_wtime();

parallelMergeSort(arr4, 0, size - 1);

t2 = omp\_get\_wtime();

printArray(arr4);

std::cout << "Time: " << t2 -t1 << " seconds\n";

return 0;

}