HPC Lab Exp No.2

#include <omp.h>

#include <iostream>

#include <string>

#include <chrono>

#include <cstdlib>

using namespace std;

using namespace chrono;

void displayArray(string message, int nums[], int length) {

    cout << "\t" << message << ": [";

    for (int i = 0; i < length; i++)

        cout << nums[i] << (i != length - 1 ? ", " : "");

    cout << "]\n";

}

void copyArray(int src[], int dest[], int length) {

    for (int i = 0; i < length; i++)

        dest[i] = src[i];

}

void merge(int nums[], int l1, int r1, int l2, int r2) {

    int n = r2 - l1 + 1, temp[n], t = 0, i = l1, j = l2;

    while (i <= r1 && j <= r2)

        temp[t++] = (nums[i] <= nums[j]) ? nums[i++] : nums[j++];

    while (i <= r1) temp[t++] = nums[i++];

    while (j <= r2) temp[t++] = nums[j++];

    for (int k = 0; k < n; k++)

        nums[l1 + k] = temp[k];

}

void mergeSort\_seq(int nums[], int start, int end) {

    if (start < end) {

        int mid = (start + end) / 2;

        mergeSort\_seq(nums, start, mid);

        mergeSort\_seq(nums, mid + 1, end);

        merge(nums, start, mid, mid + 1, end);

    }

}

void mergeSort\_parallel(int nums[], int start, int end) {

    if (start < end) {

        int mid = (start + end) / 2;

#pragma omp parallel sections

        {

#pragma omp section

            mergeSort\_parallel(nums, start, mid);

#pragma omp section

            mergeSort\_parallel(nums, mid + 1, end);

        }

        merge(nums, start, mid, mid + 1, end);

    }

}

void bubbleSort\_seq(int nums[], int length) {

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

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

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

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

}

void bubbleSort\_parallel(int nums[], int length) {

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

        int start = i % 2;

#pragma omp parallel for

        for (int j = start; j < length - 1; j += 2)

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

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

    }

}

int main() {

    const int length = 1000;

    int original[length], nums1[length], nums2[length];

    srand(time(0));

    for (int i = 0; i < length; i++)

        original[i] = rand() % 100;

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

    copyArray(original, nums1, length);

    displayArray("Before", nums1, length);

    auto start1 = high\_resolution\_clock::now();

    bubbleSort\_seq(nums1, length);

    auto end1 = high\_resolution\_clock::now();

    displayArray("After", nums1, length);

    cout << "Execution time: " << duration\_cast<milliseconds>(end1 - start1).count() << " ms\n\n";

    cout << "Bubble Sort (Parallel OpenMP):\n";

    copyArray(original, nums2, length);

    displayArray("Before", nums2, length);

    auto start2 = high\_resolution\_clock::now();

    bubbleSort\_parallel(nums2, length);

    auto end2 = high\_resolution\_clock::now();

    displayArray("After", nums2, length);

    cout << "Execution time: " << duration\_cast<milliseconds>(end2 - start2).count() << " ms\n\n";

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

    copyArray(original, nums1, length);

    displayArray("Before", nums1, length);

    auto start3 = high\_resolution\_clock::now();

    mergeSort\_seq(nums1, 0, length - 1);

    auto end3 = high\_resolution\_clock::now();

    displayArray("After", nums1, length);

    cout << "Execution time: " << duration\_cast<milliseconds>(end3 - start3).count() << " ms\n\n";

    cout << "Merge Sort (Parallel OpenMP):\n";

    copyArray(original, nums2, length);

    displayArray("Before", nums2, length);

    auto start4 = high\_resolution\_clock::now();

    mergeSort\_parallel(nums2, 0, length - 1);

    auto end4 = high\_resolution\_clock::now();

    displayArray("After", nums2, length);

    cout << "Execution time: " << duration\_cast<milliseconds>(end4 - start4).count() << " ms\n";

    return 0;

}

OUTPUT: Just take large inputs like n>=1000 for accurate results

