Parallel Quick Sort Code

#include <omp.h>

#include <iostream>

#include <chrono>

#include <random>

using namespace std;

int partition(int arr[], int s, int e) {

    int pivot = arr[e];

    int i = s - 1;

    for (int j = s; j <= e - 1; j++) {

        if (arr[j] < pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[e]);

    return i + 1;

}

void quicksort\_parallel(int arr[], int s, int e) {

    if (s < e) {

        int index = partition(arr, s, e);

        #pragma omp parallel sections

        {

            #pragma omp section

            quicksort\_parallel(arr, s, index - 1);

            #pragma omp section

            quicksort\_parallel(arr, index + 1, e);

        }

    }

}

void quicksort\_serial(int arr[], int s, int e) {

    if (s < e) {

        int index = partition(arr, s, e);

        quicksort\_serial(arr, s, index - 1);

        quicksort\_serial(arr, index + 1, e);

    }

}

int main() {

    int n = 100000;

    int arr[n];

    // Seed and generate random numbers

    random\_device rd;

    mt19937 gen(rd());

    uniform\_int\_distribution<> dist(1, 1000);

    cout << "Generating random elements for the array...\n";

    for (int i = 0; i < n; i++) {

        arr[i] = dist(gen);

    }

    // cout << "The generated array is: ";

    // for (int i = 0; i < n; i++) {

    //     cout << arr[i] << " ";

    // }

    // cout << endl;

    // Parallel quicksort

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

    quicksort\_parallel(arr, 0, n - 1);

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

    chrono::duration<double, milli> parallel\_time = end\_time - start\_time;

    // cout << "\nArray after sorting by parallel processing: ";

    // for (int i = 0; i < n; i++) {

    //     cout << arr[i] << " ";

    // }

    cout << "\nTime taken for parallel sorting: " << parallel\_time.count() << " ms" << endl;

    // Serial quicksort

    // Regenerate the random array to ensure a fair comparison

    for (int i = 0; i < n; i++) {

        arr[i] = dist(gen);

    }

    auto s\_time = chrono::high\_resolution\_clock::now();

    quicksort\_serial(arr, 0, n - 1);

    auto e\_time = chrono::high\_resolution\_clock::now();

    chrono::duration<double, milli> serial\_time = e\_time - s\_time;

    // cout << "\nArray after sorting by serial processing: ";

    // for (int i = 0; i < n; i++) {

    //     cout << arr[i] << " ";

    // }

    cout << "\nTime taken for serial sorting: " << serial\_time.count() << " ms" << endl;

    return 0;

}

// OUTPUT:

// Generating random elements for the array...

// Time taken for parallel sorting: 16.065 ms

// Time taken for serial sorting: 23.501 ms