HPC/3/statistics.cpp

```cpp
#include <limits.h>
#include <omp.h>
#include <stdlib.h>

#include <array>
#include <chrono>
#include <functional>
#include <iostream>
#include <string>
#include <vector>

using std::chrono::duration_cast;
using std::chrono::high_resolution_clock;
using std::chrono::milliseconds;
using namespace std;

void s_avg(int arr[], int n) {
    long sum = 0L;
    int i;
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
    }
    cout << sum / long(n);
}

void p_avg(int arr[], int n) {
    long sum = 0L;
    int i;
#pragma omp parallel for reduction(+ : sum) num_threads(16)
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
    }
    cout << sum / long(n);
}

void s_sum(int arr[], int n) {
    long sum = 0L;
    int i;
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
    }
    cout << sum;
}

void p_sum(int arr[], int n) {
    long sum = 0L;
    int i;
#pragma omp parallel for reduction(+ : sum) num_threads(16)
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
    }
    cout << sum;
}

void s_max(int arr[], int n) {
    int max_val = INT_MIN;
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    cout << max_val;
}

void p_max(int arr[], int n) {
    int max_val = INT_MIN;
    int i;
#pragma omp parallel for reduction(max : max_val) num_threads(16)
    for (i = 0; i < n; i++) {
```

```cpp
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    cout << max_val;
}

void s_min(int arr[], int n) {
    int min_val = INT_MAX;
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] < min_val) {
            min_val = arr[i];
        }
    }
    cout << min_val;
}

void p_min(int arr[], int n) {
    int min_val = INT_MAX;
    int i;
#pragma omp parallel for reduction(min : min_val) num_threads(16)
    for (i = 0; i < n; i++) {
        if (arr[i] < min_val) {
            min_val = arr[i];
        }
    }
    cout << min_val;
}

std::string bench_traverse(std::function<void()> traverse_fn) {
    auto start = high_resolution_clock::now();
    traverse_fn();
    cout << " (";
    auto stop = high_resolution_clock::now();

    // Subtract stop and start timepoints and cast it to required unit.
    // Predefined units are nanoseconds, microseconds, milliseconds, seconds,
    // minutes, hours. Use duration_cast() function.
    auto duration = duration_cast<milliseconds>(stop - start);

    // To get the value of duration use the count() member function on the
    // duration object
    return std::to_string(duration.count());
}

int main(int argc, const char **argv) {
    if (argc < 3) {
        std::cout << "Specify array length and maximum random value\n";
        return 1;
    }
    int *a, n, rand_max;

    n = stoi(argv[1]);
    rand_max = stoi(argv[2]);
    a = new int[n];

    for (int i = 0; i < n; i++) {
        a[i] = rand() % rand_max;
    }

    cout << "Generated random array of length " << n << " with elements between 0 to " << rand_max
         << "\n\n";
    cout << "Given array is ⇒\n";
    for (int i = 0; i < n; i++) {
        cout << a[i] << ", ";
    }
    cout << "\n\n";

    omp_set_num_threads(16);

    std::cout << "Sequential Min: " << bench_traverse([&] { s_min(a, n); }) << "ms)\n";

    std::cout << "Parallel (16) Min: " << bench_traverse([&] { p_min(a, n); }) << "ms)\n\n";
```

```cpp

    std::cout << "Sequential Max: " << bench_traverse([&] { s_max(a, n); }) << "ms)\n";

    std::cout << "Parallel (16) Max: " << bench_traverse([&] { p_max(a, n); }) << "ms)\n\n";

    std::cout << "Sequential Sum: " << bench_traverse([&] { s_sum(a, n); }) << "ms)\n";

    std::cout << "Parallel (16) Sum: " << bench_traverse([&] { p_sum(a, n); }) << "ms)\n\n";

    std::cout << "Sequential Average: " << bench_traverse([&] { s_avg(a, n); }) << "ms)\n";

    std::cout << "Parallel (16) Average: " << bench_traverse([&] { p_avg(a, n); }) << "ms)\n";
    return 0;
}

/*

OUTPUT:

Generated random array of length 100 with elements between 0 to 200

Given array is ⇒
183, 86, 177, 115, 193, 135, 186, 92, 49, 21, 162, 27, 90, 59, 163, 126, 140, 26, 172, 136, 11, 168,
167, 29, 182, 130, 62, 123, 67, 135, 129, 2, 22, 58, 69, 167, 193, 56, 11, 42, 29, 173, 21, 119,
184, 137, 198, 124, 115, 170, 13, 126, 91, 180, 156, 73, 62, 170, 196, 81, 105, 125, 84, 127, 136,
105, 46, 129, 113, 57, 124, 95, 182, 145, 14, 167, 34, 164, 43, 150, 87, 8, 76, 178, 188, 184, 3,
51, 154, 199, 132, 60, 76, 168, 139, 12, 26, 186, 94, 139,

Sequential Min: 2 (0ms)
Parallel (16) Min: 2 (0ms)

Sequential Max: 199 (0ms)
Parallel (16) Max: 199 (0ms)

Sequential Sum: 10884 (0ms)
Parallel (16) Sum: 10884 (1ms)

Sequential Average: 108 (0ms)
Parallel (16) Average: 108 (0ms)


OUTPUT:

Generated random array of length 100000000 with elements between 0 to 100000000

Sequential Min: 0 (185ms)
Parallel (16) Min: 0 (19ms)

Sequential Max: 99999999 (187ms)
Parallel (16) Max: 99999999 (18ms)

Sequential Sum: 4942469835882961 (191ms)
Parallel (16) Sum: 4942469835882961 (14ms)

Sequential Average: 49424698 (190ms)
Parallel (16) Average: 49424698 (14ms)

*/

```