```cpp
// STATISTICS.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 << "\nAverage = " << sum / long(n) << "\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 << "\nAverage = " << sum / long(n) << "\n";
}

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

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 << "\nSum = " << sum << "\n";
}

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 << "\nMax value = " << max_val << "\n";
}

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++) {
        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }
    // cout << "\nMax value = " << max_val << "\n";
}

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 << "\nMin value = " << min_val << "\n";
}

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 << "\nMin value = " << min_val << "\n";
}

std::string bench_traverse(std::function<void()> traverse_fn) {
    auto start = high_resolution_clock::now();
    traverse_fn();
    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 < 2) {
        std::cout << "Specify array length.\n";
        return 1;
    }
    int *a, n, i;

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

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

    cout << "Generated random array of length " << n << "\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";

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

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

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

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

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

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

    // cout << "\nSorted array is =>";
    // for (i = 0; i < n; i++) {
    //     cout << "\n" << a[i];
    // }
    return 0;
}

/*

OUTPUT:

Generated random array of length 100000000

Sequential Min: 567ms
Parallel (16) Min: 49ms

Sequential Max: 568ms
Parallel (16) Max: 46ms

Sequential Sum: 579ms
Parallel (16) Sum: 46ms

Sequential Average: 579ms
Parallel (16) Average: 45ms

*/
```