

HPC/3/statistics.cpp

```
1  #include <limits.h>
2  #include <omp.h>
3  #include <stdlib.h>
4
5  #include <array>
6  #include <chrono>
7  #include <functional>
8  #include <iostream>
9  #include <string>
10 #include <vector>
11
12 using std::chrono::duration_cast;
13 using std::chrono::high_resolution_clock;
14 using std::chrono::milliseconds;
15 using namespace std;
16
17 void s_avg(int arr[], int n) {
18     long sum = 0L;
19     int i;
20     for (i = 0; i < n; i++) {
21         sum = sum + arr[i];
22     }
23     cout << sum / long(n);
24 }
25
26 void p_avg(int arr[], int n) {
27     long sum = 0L;
28     int i;
29 #pragma omp parallel for reduction(+ : sum) num_threads(16)
30     for (i = 0; i < n; i++) {
31         sum = sum + arr[i];
32     }
33     cout << sum / long(n);
34 }
35
36 void s_sum(int arr[], int n) {
37     long sum = 0L;
38     int i;
39     for (i = 0; i < n; i++) {
40         sum = sum + arr[i];
41     }
42     cout << sum;
43 }
44
45 void p_sum(int arr[], int n) {
46     long sum = 0L;
47     int i;
48 #pragma omp parallel for reduction(+ : sum) num_threads(16)
49     for (i = 0; i < n; i++) {
50         sum = sum + arr[i];
51     }
52     cout << sum;
53 }
54
55 void s_max(int arr[], int n) {
56     int max_val = INT_MIN;
57     int i;
58     for (i = 0; i < n; i++) {
59         if (arr[i] > max_val) {
60             max_val = arr[i];
61         }
62     }
63     cout << max_val;
64 }
65
66 void p_max(int arr[], int n) {
67     int max_val = INT_MIN;
68     int i;
69 #pragma omp parallel for reduction(max : max_val) num_threads(16)
70     for (i = 0; i < n; i++) {
```

```

71         if (arr[i] > max_val) {
72             max_val = arr[i];
73         }
74     }
75     cout << max_val;
76 }
77
78 void s_min(int arr[], int n) {
79     int min_val = INT_MAX;
80     int i;
81     for (i = 0; i < n; i++) {
82         if (arr[i] < min_val) {
83             min_val = arr[i];
84         }
85     }
86     cout << min_val;
87 }
88
89 void p_min(int arr[], int n) {
90     int min_val = INT_MAX;
91     int i;
92     #pragma omp parallel for reduction(min : min_val) num_threads(16)
93     for (i = 0; i < n; i++) {
94         if (arr[i] < min_val) {
95             min_val = arr[i];
96         }
97     }
98     cout << min_val;
99 }
100
101 std::string bench_traverse(std::function<void()> traverse_fn) {
102     auto start = high_resolution_clock::now();
103     traverse_fn();
104     cout << " (";
105     auto stop = high_resolution_clock::now();
106
107     // Subtract stop and start timepoints and cast it to required unit.
108     // Predefined units are nanoseconds, microseconds, milliseconds, seconds,
109     // minutes, hours. Use duration_cast() function.
110     auto duration = duration_cast<milliseconds>(stop - start);
111
112     // To get the value of duration use the count() member function on the
113     // duration object
114     return std::to_string(duration.count());
115 }
116
117 int main(int argc, const char **argv) {
118     if (argc < 3) {
119         std::cout << "Specify array length and maximum random value\n";
120         return 1;
121     }
122     int *a, n, rand_max;
123
124     n = stoi(argv[1]);
125     rand_max = stoi(argv[2]);
126     a = new int[n];
127
128     for (int i = 0; i < n; i++) {
129         a[i] = rand() % rand_max;
130     }
131
132     cout << "Generated random array of length " << n << " with elements between 0 to " << rand_max
133         << "\n\n";
134     cout << "Given array is =>\n";
135     for (int i = 0; i < n; i++) {
136         cout << a[i] << ", ";
137     }
138     cout << "\n\n";
139
140     omp_set_num_threads(16);
141
142     std::cout << "Sequential Min: " << bench_traverse([&] { s_min(a, n); }) << "ms)\n";
143
144     std::cout << "Parallel (16) Min: " << bench_traverse([&] { p_min(a, n); }) << "ms)\n\n";

```

```

145     std::cout << "Sequential Max: " << bench_traverse([&] { s_max(a, n); }) << "ms)\n";
146
147     std::cout << "Parallel (16) Max: " << bench_traverse([&] { p_max(a, n); }) << "ms)\n\n";
148
149     std::cout << "Sequential Sum: " << bench_traverse([&] { s_sum(a, n); }) << "ms)\n";
150
151     std::cout << "Parallel (16) Sum: " << bench_traverse([&] { p_sum(a, n); }) << "ms)\n\n";
152
153     std::cout << "Sequential Average: " << bench_traverse([&] { s_avg(a, n); }) << "ms)\n";
154
155     std::cout << "Parallel (16) Average: " << bench_traverse([&] { p_avg(a, n); }) << "ms)\n";
156     return 0;
157 }
158
159 /*
160
161 OUTPUT:
162
163 Generated random array of length 100 with elements between 0 to 200
164
165 Given array is =>
166 183, 86, 177, 115, 193, 135, 186, 92, 49, 21, 162, 27, 90, 59, 163, 126, 140, 26, 172, 136, 11, 168,
167 167, 29, 182, 130, 62, 123, 67, 135, 129, 2, 22, 58, 69, 167, 193, 56, 11, 42, 29, 173, 21, 119,
168 184, 137, 198, 124, 115, 170, 13, 126, 91, 180, 156, 73, 62, 170, 196, 81, 105, 125, 84, 127, 136,
169 105, 46, 129, 113, 57, 124, 95, 182, 145, 14, 167, 34, 164, 43, 150, 87, 8, 76, 178, 188, 184, 3,
170 51, 154, 199, 132, 60, 76, 168, 139, 12, 26, 186, 94, 139,
171
172 Sequential Min: 2 (0ms)
173 Parallel (16) Min: 2 (0ms)
174
175 Sequential Max: 199 (0ms)
176 Parallel (16) Max: 199 (0ms)
177
178 Sequential Sum: 10884 (0ms)
179 Parallel (16) Sum: 10884 (1ms)
180
181 Sequential Average: 108 (0ms)
182 Parallel (16) Average: 108 (0ms)
183
184
185 OUTPUT:
186
187 Generated random array of length 100000000 with elements between 0 to 100000000
188
189 Sequential Min: 0 (185ms)
190 Parallel (16) Min: 0 (19ms)
191
192 Sequential Max: 99999999 (187ms)
193 Parallel (16) Max: 99999999 (18ms)
194
195 Sequential Sum: 4942469835882961 (191ms)
196 Parallel (16) Sum: 4942469835882961 (14ms)
197
198 Sequential Average: 49424698 (190ms)
199 Parallel (16) Average: 49424698 (14ms)
200
201 */
202
203

```