

```
1: // C++ Vorlesung xxx
2: // C++-17: Threads, execution policies
3: // Great web pages, great book by Filipek: https://www.bfilipek.com/2018/06/parstl-tests.html
4:
5: /*
6:  g++ -O3 -std=c++17 gh_main.cpp -ltbb
7:  g++ -O3 -std=c++17 -pedantic -Weffc++ -Wall -Wextra -pedantic -Wswitch-default -Wfloat-equal -Wundef -Wredundant-decls -Winit-self -Wshadow -Wparentheses -Wshadow -Wunreachable-code -Wuninitialized -Wmaybe-uninitialized gh_main.cpp -ltbb
8:  ---
9:  cppcheck --enable=all --inconclusive --std=c++11 --std=posix --suppress=missingIncludeSystem gh_main.cpp
10: clang++ -O3 -std=c++17 -ltbb gh_main.cpp
11: clang++ -std=c++17 -fsyntax-only -Wdocumentation -Wconversion -Wshadow -Wfloat-conversion -pedantic gh_main.cpp
12: clang++ -std=c++17 -Weverything -Wno-c++98-compat -Wno-padded -ltbb gh_main.cpp
13: clang++ -ccl --help
14: ---
15: icpc -O3 -std=c++17 -ltbb -Wall -Wextra -pedantic gh_main.cpp
16: */
17: #include <algorithm>
18: #include <chrono>
19: #include <execution>           // execution policy
20: #include <iostream>
21: #include <numeric>           // accumulate
22: #include <random>
23: #include <vector>
24: using namespace std;
25: using namespace std::chrono; // timing
26:
27:
28: /** \brief Prints the whole vector of base class pointers
29:  * \param[in,out] s output stream
30:  * \param[in] v vector
31:  * \return changed output stream
32:  */
33: template <class T>
34: ostream& operator<<(ostream &s, const vector<T>& v)
35: {
36:     for (const auto& it: v)           // Reference is required with unique_ptr. No copy constructor for unique_ptr available!
```

```
37: {
38:     cout << *it << " ";
39: }
40: return s;
41: }
42:
43:
44: int main()
45: {
46:     cout << "Threads C++17" << endl;
47:
48:     size_t const N = 1<<25;
49:     vector<double> v(N);
50:     iota(v.begin(), v.end(), 1);
51:     std::shuffle(v.begin(), v.end(), std::mt19937{std::random_device{}}());
52:
53:     auto const v_bak(v);
54:
55:     {
56:         v = v_bak;
57:         cout << "---- sort old ----" << endl;
58:         auto t1 = system_clock::now();
59:         sort(v.begin(), v.end());
60:         auto t2 = system_clock::now();
61:         //auto duration = duration_cast<microseconds>(t2 - t1);
62:         auto duration = std::chrono::duration<double, std::micro>(t2 - t1);
63:         cout << "sort old : " << duration.count() / 1e6 << " sec." << endl;
64:     }
65:
66:     {
67:         v = v_bak;
68:         cout << "---- sort seq ----" << endl;
69:         auto t1 = system_clock::now();
70:         sort(std::execution::seq, v.begin(), v.end());
71:         auto t2 = system_clock::now();
72:         auto duration = std::chrono::duration<double, std::micro>(t2 - t1);
73:         cout << "sort seq : " << duration.count() / 1e6 << " sec." << endl;
74:     }
75:
76:     {
77:         v = v_bak;
```

```
78:     cout << "----    sort par    ----" << endl;
79:     auto t1 = system_clock::now();
80:     sort(std::execution::par, v.begin(), v.end());
81:     //auto cnt = count(std::execution::par, v.begin(), v.end(), 17);
82:     auto t2 = system_clock::now();
83:     auto duration = std::chrono::duration<double, std::micro>(t2 - t1);
84:     cout << "sort par    : " << duration.count() / 1e6 << " sec." << endl;
85: }
86:
87: {
88:     v = v_bak;
89:     cout << "----    sort par_unseq    ----" << endl;
90:     auto t1 = system_clock::now();
91:     sort(std::execution::par_unseq, v.begin(), v.end());
92:     auto t2 = system_clock::now();
93:     auto duration = std::chrono::duration<double, std::micro>(t2 - t1);
94:     cout << "sort par_unseq: " << duration.count() / 1e6 << " sec." << endl;
95: }
96:
97:
98:     return 0;
99: }
100:
```

```

./main.cpp      Wed Jun 23 16:07:51 2021      1

1: // C++ Vorlesung xxx
2: // C++-17: Threads, execution policies
3: // Great web pages, great book by Filipek:  https://www.bfilipek.com/2018/06/parstl-tests.html
4:
5: /*
6:  g++ -O3 -std=c++17 main.cpp -ltbb
7:  g++ -O3 -std=c++17 -pedantic -Weffc++ -Wall -Wextra -pedantic -Wswitch-default -Wfloat-equal -Wundef
def -Wredundant-decls -Winit-self -Wshadow -Wparentheses -Wshadow -Wunreachable-code -Wuninitialized -Wm
ybe-uninitialized main.cpp -ltbb
8:  ---
9:  cppcheck --enable=all --inconclusive --std=c++11 --std=posix --suppress=missingIncludeSystem main
.cpp
10: clang++ -O3 -std=c++17 -ltbb main.cpp
11: clang++ -std=c++17 -fsyntax-only -Wdocumentation -Wconversion -Wshadow -Wfloat-conversion -pedant
ic main.cpp
12: clang++ -std=c++17 -Weverything -Wno-c++98-compat -Wno-padded -ltbb main.cpp
13: clang++ -ccl --help
14: ---
15: icpc -O3 -std=c++17 -ltbb -Wall -Wextra -pedantic main.cpp
16: */
17:
18: #include <algorithm>
19: #include <chrono>
20: #include <execution>           // execution policy
21: #include <iostream>
22: #include <numeric>           // accumulate
23: #include <random>
24: #include <vector>
25: using namespace std;
26: using namespace std::chrono; // timing
27:
28: // Great web pages, great book by Filipek
29: // https://www.bfilipek.com/2018/06/parstl-tests.html
30: template <typename TFunc> void RunAndMeasure(const char *title, TFunc func)
31: {
32:     const auto start = std::chrono::steady_clock::now();
33:     auto ret = func();
34:     const auto end = std::chrono::steady_clock::now();
35:     std::cout << title << ": " <<
36:         std::chrono::duration <double, std::milli>(end - start).count()
37:         << " ms, res " << ret << "\n";

```

```
38: }
39:
40: int main()
41: {
42:     //std::vector<double> v(6000000, 0.5);
43:     std::vector<double> v(1<<30, 0.5);
44:
45:     RunAndMeasure("std::warm up", [&v]
46:     {
47:         return std::reduce(std::execution::seq, v.begin(), v.end(), 0.0);
48:     });
49:
50:     RunAndMeasure("std::accumulate", [&v]
51:     {
52:         return std::accumulate(v.begin(), v.end(), 0.0);
53:     });
54:
55:     RunAndMeasure("std::reduce, seq", [&v]
56:     {
57:         return std::reduce(std::execution::seq, v.begin(), v.end(), 0.0);
58:     });
59:
60:     RunAndMeasure("std::reduce, par", [&v]
61:     {
62:         return std::reduce(std::execution::par, v.begin(), v.end(), 0.0);
63:     });
64:
65:     RunAndMeasure("std::reduce, par_unseq", [&v]
66:     {
67:         return std::reduce(std::execution::par_unseq, v.begin(), v.end(), 0.0);
68:     });
69:
70:     RunAndMeasure("std::find, seq", [&v]
71:     {
72:         auto res = std::find(std::execution::seq, std::begin(v), std::end(v), 0.6);
73:         return res == std::end(v) ? 0.0 : 1.0;
74:     });
75:
76:     RunAndMeasure("std::find, par", [&v]
77:     {
78:         auto res = std::find(std::execution::par, std::begin(v), std::end(v), 0.6);
```

```
79:         return res == std::end(v) ? 0.0 : 1.0;
80:     });
81:
82:     cout << "-----\n";
83:     const size_t VecSize=10*20000000;
84:     cout << "N = " << VecSize << endl;
85:     vector<double> vec(VecSize);
86:     iota(begin(vec), end(vec), 0.1);
87:
88:     vector<double> out(VecSize);
89:
90:     auto heavy_fkt = [](double a){return std::sin(a)*std::cos(a);};
91:     auto light_fkt = [](double a){return 1.0/a;};
92:     //auto light_fkt = [](double a){return std::sqrt(1.0/a);};
93:
94:     RunAndMeasure("heavy std::transform seq", [&vec, &out, heavy_fkt]
95:     {
96:         auto res = std::transform(std::execution::seq, cbegin(vec), cend(vec), begin(out),
97:             //[] (double a){return std::sin(a)*std::cos(a);}
98:             heavy_fkt
99:         );
100:         return res == std::end(vec) ? 0.0 : 1.0;
101:     });
102:
103:     RunAndMeasure("heavy std::transform par", [&vec, &out, heavy_fkt]
104:     {
105:         auto res = std::transform(std::execution::par, cbegin(vec), cend(vec), begin(out),
106:             heavy_fkt
107:         );
108:         return res == std::end(vec) ? 0.0 : 1.0;
109:     });
110:
111:
112:     RunAndMeasure("light std::transform seq", [&vec, &out, light_fkt]
113:     {
114:         auto res = std::transform(std::execution::seq, cbegin(vec), cend(vec), begin(out),
115:             light_fkt
116:         );
117:         return res == std::end(vec) ? 0.0 : 1.0;
118:     });
119:
```

Compute bound

Heavy bound

```
120: RunAndMeasure("light std::transform par", [&vec, &out, light_fkt]
121: {
122:     auto res = std::transform(std::execution::par, cbegin(vec), cend(vec), begin(out),
123:         light_fkt
124:     );
125:     return res == std::end(vec) ? 0.0 : 1.0;
126: });
127:
128: return 0;
129: }
130:
```