```cpp
// C++ Vorlesung  xxx
// C++-17: Threads, execution policies
//   Great web pages, great book by Filipek:   https://www.bfilipek.com/2018/06/parstl-tests.html

/*
 g++ -O3 -std=c++17 gh_main.cpp -ltbb
 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
 ---
 cppcheck --enable=all --inconclusive --std=c++11 --std=posix --suppress=missingIncludeSystem gh_main.cpp
 clang++ -O3 -std=c++17 -ltbb gh_main.cpp
 clang++ -std=c++17 -fsyntax-only -Wdocumentation -Wconversion -Wshadow -Wfloat-conversion -pedantic gh_main.cpp
 clang++ -std=c++17 -Weverything -Wno-c++98-compat -Wno-padded -ltbb gh_main.cpp
 clang++ -cc1 --help
 ---
 icpc -O3 -std=c++17 -ltbb -Wall -Wextra -pedantic gh_main.cpp
*/
#include <algorithm>
#include <chrono>
#include <execution>            // execution policy
#include <iostream>
#include <numeric>              //  accumulate
#include <random>
#include <vector>
using namespace std;
using namespace std::chrono;  // timing


/** \brief Prints the whole vector of base class pointers
 *  \param[in,out]  s   output stream
 *  \param[in]      v   vector
 *  \return        changed output stream
 */
template <class T>
ostream& operator<<(ostream &s, const vector<T>& v)
{
    for (const auto& it: v)             // Reference is required with unique_ptr. No copy constructor for unique_ptr available!
    {
        cout << *it << " ";
    }
    return s;
}


int main()
{
    cout << "Threads C++17" << endl;

    size_t const N = 1<<25;
    vector<double> v(N);
    iota(v.begin(), v.end(), 1);
    std::shuffle(v.begin(), v.end(), std::mt19937{std::random_device{}()});

    auto const v_bak(v);

    {
        v = v_bak;
        cout << "----   sort old   ----" << endl;
        auto t1 = system_clock::now();
        sort(v.begin(), v.end());
        auto t2 = system_clock::now();
        //auto duration = duration_cast<microseconds>(t2 - t1);
        auto duration = std::chrono::duration <double, std::micro>(t2 - t1);
        cout << "sort old   : " << duration.count() / 1e6 << " sec." << endl;
    }

    {
        v = v_bak;
        cout << "----   sort seq   ----" << endl;
        auto t1 = system_clock::now();
        sort(std::execution::seq, v.begin(), v.end());
        auto t2 = system_clock::now();
        auto duration = std::chrono::duration <double, std::micro>(t2 - t1);
        cout << "sort seq   : " << duration.count() / 1e6 << " sec." << endl;
    }

    {
        v = v_bak;
        cout << "----   sort par   ----" << endl;
        auto t1 = system_clock::now();
        sort(std::execution::par, v.begin(), v.end());
        //auto cnt = count(std::execution::par, v.begin(), v.end(), 17);
        auto t2 = system_clock::now();
        auto duration = std::chrono::duration <double, std::micro>(t2 - t1);
        cout << "sort par    : " << duration.count() / 1e6 << " sec." << endl;
    }

    {
        v = v_bak;
        cout << "----   sort par_unseq   ----" << endl;
        auto t1 = system_clock::now();
        sort(std::execution::par_unseq, v.begin(), v.end());
        auto t2 = system_clock::now();
        auto duration = std::chrono::duration <double, std::micro>(t2 - t1);
        cout << "sort par_unseq: " << duration.count() / 1e6 << " sec." << endl;
    }


    return 0;
}

```

```cpp
// C++ Vorlesung  xxx
// C++-17: Threads, execution policies
//   Great web pages, great book by Filipek:   https://www.bfilipek.com/2018/06/parstl-tests.html

/*
 g++ -O3 -std=c++17 main.cpp -ltbb
 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 main.cpp -ltbb
 ---
 cppcheck --enable=all --inconclusive --std=c++11 --std=posix --suppress=missingIncludeSystem main.cpp
 clang++ -O3 -std=c++17 -ltbb main.cpp
 clang++ -std=c++17 -fsyntax-only -Wdocumentation -Wconversion -Wshadow -Wfloat-conversion -pedantic main.cpp
 clang++ -std=c++17 -Weverything -Wno-c++98-compat -Wno-padded -ltbb main.cpp
 clang++ -cc1 --help
 ---
 icpc -O3 -std=c++17 -ltbb -Wall -Wextra -pedantic main.cpp
*/

#include <algorithm>
#include <chrono>
#include <execution>           // execution policy
#include <iostream>
#include <numeric>             //  accumulate
#include <random>
#include <vector>
using namespace std;
using namespace std::chrono;  // timing

// Great web pages, great book by Filipek
// https://www.bfilipek.com/2018/06/parstl-tests.html
template <typename TFunc> void RunAndMeasure(const char *title, TFunc func)
{
    const auto start = std::chrono::steady_clock::now();
    auto ret = func();
    const auto end = std::chrono::steady_clock::now();
    std::cout << title << ": " <<
              std::chrono::duration <double, std::milli>(end - start).count()
              << " ms, res " << ret << "\n";
```

```cpp
}

int main()
{
    //std::vector<double> v(6000000, 0.5);
    std::vector<double> v(1<<30, 0.5);

    RunAndMeasure("std::warm up", [&v]
    {
        return std::reduce(std::execution::seq, v.begin(), v.end(), 0.0);
    });

    RunAndMeasure("std::accumulate", [&v]
    {
        return std::accumulate(v.begin(), v.end(), 0.0);
    });

    RunAndMeasure("std::reduce, seq", [&v]
    {
        return std::reduce(std::execution::seq, v.begin(), v.end(), 0.0);
    });

    RunAndMeasure("std::reduce, par", [&v]
    {
        return std::reduce(std::execution::par, v.begin(), v.end(), 0.0);
    });

    RunAndMeasure("std::reduce, par_unseq", [&v]
    {
        return std::reduce(std::execution::par_unseq, v.begin(), v.end(), 0.0);
    });

    RunAndMeasure("std::find, seq", [&v]
    {
        auto res = std::find(std::execution::seq, std::begin(v), std::end(v), 0.6);
        return res == std::end(v) ? 0.0 : 1.0;
    });

    RunAndMeasure("std::find, par", [&v]
    {
        auto res = std::find(std::execution::par, std::begin(v), std::end(v), 0.6);
        return res == std::end(v) ? 0.0 : 1.0;
    });

    cout << "------------------------------------------------\n";
    const size_t VecSize=10*20000000;
    cout << "N = " << VecSize << endl;
    vector<double>  vec(VecSize);
    iota(begin(vec),end(vec),0.1);

    vector<double>  out(VecSize);

    auto heavy_fkt = [](double a){return std::sin(a)*std::cos(a);};
    auto light_fkt = [](double a){return 1.0/a;};
    //auto light_fkt = [](double a){return std::sqrt(1.0/a);};

    RunAndMeasure("heavy  std::transform seq", [&vec, &out, heavy_fkt]
    {
        auto res = std::transform(std::execution::seq, cbegin(vec), cend(vec), begin(out),
             //[](double a){return std::sin(a)*std::cos(a);}
             heavy_fkt
             );
        return res == std::end(vec) ? 0.0 : 1.0;
    });

    RunAndMeasure("heavy  std::transform par", [&vec, &out, heavy_fkt]
    {
        auto res = std::transform(std::execution::par, cbegin(vec), cend(vec), begin(out),
             heavy_fkt
             );
        return res == std::end(vec) ? 0.0 : 1.0;
    });


    RunAndMeasure("light  std::transform seq", [&vec, &out, light_fkt]
    {
        auto res = std::transform(std::execution::seq, cbegin(vec), cend(vec), begin(out),
             light_fkt
             );
        return res == std::end(vec) ? 0.0 : 1.0;
    });

```

```cpp
    RunAndMeasure("light  std::transform par", [&vec, &out, light_fkt]
    {
        auto res = std::transform(std::execution::par, cbegin(vec), cend(vec), begin(out),
            light_fkt
            );
        return res == std::end(vec) ? 0.0 : 1.0;
    });

    return 0;
}

```