```cpp
#pragma once

#include <iostream>
#ifdef _OPENMP
#include <omp.h>
#endif
#include <unordered_map>

//#####################################
// G.Haase
// See https://sourceforge.net/p/predef/wiki/Compilers/
//     http://www.cplusplus.com/doc/tutorial/preprocessor/
//  also:  export OMP_DISPLAY_ENV=VERBOSE
//#####################################
/**     Checks for compilers, its versions, threads etc.
 *
        @param[in] argc number of command line arguemnts
        @param[in] argv command line arguments as array of C-strings
*/
template <class T>
void check_env(T argc, char const *argv[])
{
    std::cout << "\n#########################################################################
#######\n";
    std::cout << "Code    :";
    for (T k = 0; k < argc; ++k) std::cout << "  " << argv[k];
    std::cout << std::endl;

    // compiler:      https://sourceforge.net/p/predef/wiki/Compilers/
    std::cout <<    "Compiler:  ";
#if defined __INTEL_COMPILER
#pragma message(" ##########  INTEL  ###############")
    std::cout << "INTEL " << __INTEL_COMPILER;
    // Ignore warnings for #pragma acc   unrecognice
#pragma warning disable 161
    // Ignore warnings for #pragma omp   unrecognice
#pragma warning disable 3180

#elif defined __PGI
#pragma message(" ##########  PGI    ###############")
    std::cout << "PGI " << __PGIC__ << "." << __PGIC_MINOR__ << "." << __PGIC_PATCHL
EVEL__;
#elif defined  __clang__
#pragma message(" ##########  CLANG    ###############")
    std::cout << "CLANG " << __clang_major__ << "." << __clang_minor__ << "."; // <<
 __clang_patchlevel__;
#elif defined __GNUC__
#pragma message(" ##########  Gnu    ###############")
    std::cout << "Gnu " <<  __GNUC__  << "." << __GNUC_MINOR__ << "." << __GNUC_PATC
HLEVEL__;
#else
#pragma message(" ##########  unknown Compiler   ###############")
    std::cout << "unknown";
#endif
    std::cout << "  C++ standard: " << __cplusplus << std::endl;

    // Parallel environments
    std::cout <<    "Parallel:  ";
#if defined MPI_VERSION
#pragma message(" ##########  MPI    ###############")
#ifdef OPEN_MPI
    std::cout << "OpenMPI ";
#else
    std::cout << "MPI ";
#endif
    std::cout << MPI_VERSION << "." << MPI_SUBVERSION << "   ";
#endif

```

```
   65: #ifdef _OPENMP
   66: //https://www.openmp.org/specifications/
   67: //https://stackoverflow.com/questions/1304363/how-to-check-the-version-of-openmp-on-
linux
   68:     std::unordered_map<unsigned, std::string> const map{
   69:         {200505, "2.5"}, {200805, "3.0"}, {201107, "3.1"}, {201307, "4.0"}, {201511,
 "4.5"}, {201611, "5.0"}, {201811, "5.0"}};
   70: #pragma message(" ##########  OPENMP    ###############")
   71:     //std::cout << _OPENMP;
   72:     std::cout << "OpenMP ";
   73:     try {
   74:         std::cout << map.at(_OPENMP);
   75:     }
   76:     catch (...) {
   77:         std::cout << _OPENMP;
   78:     }
   79:     #pragma omp parallel
   80:     {
   81:         #pragma omp master
   82:         {
   83:             const int nn = omp_get_num_threads();           // OpenMP
   84:             std::cout << " ---> " <<  nn << " Threads   ";
   85:         }
   86:         #pragma omp barrier
   87:     }
   88:
   89: #endif
   90: #ifdef _OPENACC
   91: #pragma message(" ##########  OPENACC    ###############")
   92:     std::cout << "OpenACC   ";
   93: #endif
   94:     std::cout << std::endl;
   95:     std::cout << "Date    :  " << __DATE__ << "  " << __TIME__;
   96:     std::cout << "\n#############################################################
#######\n";
   97: }
   98: // HG
   99:
```

```cpp
#include "check_env.h"
#include "mylib.h"
#include <cstdlib>          // atoi()
#include <cstring>          // strncmp()
#include <ctime>
#include <iostream>
#include <omp.h>            // OpenMP
#include <sstream>
#include <string>
using namespace std;

int main(int argc, char const *argv[])
{
    omp_set_schedule(omp_sched_static, 2000000);
    //omp_set_schedule(omp_sched_dynamic, 1000000);
    //omp_set_schedule(omp_sched_guided, 1000000);
    //omp_set_schedule(omp_sched_auto, 1); // chunk size does not matter for auto


    // Speedup for different number of cores (incl. hyperthreading)
    omp_set_num_threads(8);

    // Print number of available processors
    cout << "Number of available processors: " << omp_get_num_procs() << endl;

    // Currently executing parallel code? -> no
    cout << "Currently in parallel? " << omp_in_parallel() << endl;


    int const NLOOPS = 10;        // chose a value such that the benchmark runs at l
east 10 sec.
    unsigned int N = 500000001;


//#########################################################################
//   Read Parameter from command line  (C++ style)
    cout << "Checking command line parameters for: -n <number> " << endl;
    for (int i = 1; i < argc; i++)
    {
        cout << " arg[" << i << "] = " << argv[i] << endl;
        string ss(argv[i]);
        if ("-n"==ss && i + 1 < argc) // found "-n" followed by another parameter
        {
            N = static_cast<unsigned int>(atoi(argv[i + 1]));
        }
        else
        {
            cout << "Corect call: " << argv[0] << " -n  <number>\n";
        }
    }

    cout << "\nN = " << N << endl;

    check_env(argc, argv);
//#########################################################################
    int nthreads;                                       // OpenMP
    #pragma omp parallel default(none) shared(cout,nthreads)
    {
        stringstream inparallel;
        inparallel << "Currently in parallel? " << omp_in_parallel() << endl;


        int const th_id  = omp_get_thread_num();    // OpenMP
        int const nthrds = omp_get_num_threads();   // OpenMP
        stringstream ss;
        ss << "C++: Hello World from thread " << th_id << " / " << nthrds << endl;
        #pragma omp critical
        {
```

```
   68:             cout << ss.str();                       // output to a shared ressource
   69:             cout << inparallel.str() << endl;
   70:         }
   71:         #pragma omp master
   72:         nthreads = nthrds;                          // transfer nn to to master threa
d
   73:     }
   74:     cout << "    " << nthreads << "   threads have been started." << endl;
   75:
   76: //#########################################################################
   77: //  Memory allocation
   78:     cout << "Memory allocation\n";
   79:
   80:     vector<double> x(N), y(N);
   81:
   82:     cout.precision(2);
   83:     cout << 2.0 * N *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n
";
   84:     cout.precision(6);
   85:
   86: //#########################################################################
   87: //  Data initialization
   88: //  Special:  x_i = i+1;  y_i = 1/x_i  ==> <x,y> == N
   89:     for (unsigned int i = 0; i < N; ++i)
   90:     {
   91:         x[i] = i + 1;
   92:         y[i] = 1.0 / x[i];
   93:     }
   94:
   95: //#########################################################################
   96:     cout << "\nStart Benchmarking\n";
   97:
   98: // Do calculation
   99:     double tstart = omp_get_wtime();                    // OpenMP
  100:
  101:     double sk(0.0);
  102:     for (int i = 0; i < NLOOPS; ++i)
  103:     {
  104:         //sk = scalar(x, y);
  105:         sk = scalar_parallel(x, y);
  106:         //sk = scalar_trans(x, y);
  107:         //sk = norm(x);
  108:     }
  109:
  110:     double t1 = omp_get_wtime() - tstart;               // OpenMP
  111:
  112:     t1 /= NLOOPS;           // divide by number of function calls
  113:
  114: //#########################################################################
  115: // Check the correct result
  116:     cout << "\n <x,y> = " << sk << endl;
  117:     if (static_cast<unsigned int>(sk) != N)
  118:     {
  119:         cout << "  !!   W R O N G  result   !!\n";
  120:     }
  121:     cout << endl;
  122:
  123: //#########################################################################
  124: // Timings  and Performance
  125:     cout << endl;
  126:     cout.precision(2);
  127:     cout << "Total benchmarking time: " << t1*NLOOPS << endl;
  128:     cout << "Timing in sec. : " << t1 << endl;
  129:     cout << "GFLOPS         : " << 2.0 * N / t1 / 1024 / 1024 / 1024 << endl;
  130:     cout << "GiByte/s        : " << 2.0 * N / t1 / 1024 / 1024 / 1024 * sizeof(x[0])
 << endl;
  131:
  132: //#########################################################################
```

```
133:
134:     cout << "\n  Try the reduction with an STL-vektor \n";
135:
136:     auto vr = reduction_vec_append(5);
137:     cout << "done\n";
138:     cout << vr << endl;
139:
140:
141:     return 0;
142: }  // memory for x and y will be deallocated their destructors
143:
```

```cpp
#include "mylib.h"
#include <cassert>        // assert()
#include <cmath>
#include <iostream>
#include <functional>     // multiplies<>{}
#include <list>
#include <numeric>        // iota()
#ifdef _OPENMP
#include <omp.h>
#endif
#include <vector>
using namespace std;


double scalar_parallel(vector<double> const &x, vector<double> const &y)
{
    assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
    size_t const N = x.size();
    double sum = 0.0;
#pragma omp parallel default(none) shared(x,y,N, cout) reduction(+:sum)
    {
        const size_t nthreads = omp_get_num_threads();
        const size_t threadnum = omp_get_thread_num();
        const size_t chunksize = N/nthreads;


        size_t start = threadnum*chunksize;
        size_t end = start + chunksize;
        if (threadnum == nthreads - 1)
            end = N;


        for (size_t i = start; i < end; ++i)
        {
            sum += x[i] * y[i];
        }

    }
    return sum;
}


vector<int> reduction_vec_append(int n)
{
    vector<int> vec(n);
#pragma omp parallel default(none) shared(cout) reduction(VecAppend:vec)
    {
        #pragma omp barrier
        #pragma omp critical
        cout << omp_get_thread_num() << " : " << vec.size() << endl;
        #pragma omp barrier
        iota( vec.begin(),vec.end(), omp_get_thread_num() );
        #pragma omp barrier

    }
    return vec;
}
```

```
   69: double scalar(vector<double> const &x, vector<double> const &y)
   70: {
   71:     assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
   72:     size_t const N = x.size();
   73:     double sum = 0.0;
   74: #pragma omp parallel for default(none) shared(x,y,N) reduction(+:sum) schedule(runti
me) // added schedule(runtime)
   75:     for (size_t i = 0; i < N; ++i)
   76:     {
   77:         sum += x[i] * y[i];
   78:         //sum += exp(x[i])*log(y[i]);
   79:     }
   80:     return sum;
   81: }
   82:
   83:
   84: double norm(vector<double> const &x)
   85: {
   86:     size_t const N = x.size();
   87:     double sum = 0.0;
   88: #pragma omp parallel for default(none) shared(x,N) reduction(+:sum) schedule(runtime
) // added schedule(runtime)
   89:     for (size_t i = 0; i < N; ++i)
   90:     {
   91:         sum += x[i]*x[i];
   92:     }
   93:     return sum;
   94: }
   95:
   96:
   97: vector<int> reduction_vec(int n)
   98: {
   99:     vector<int> vec(n);
  100: #pragma omp parallel default(none) shared(cout) reduction(VecAdd:vec)
  101:     {
  102:         #pragma omp barrier
  103:         #pragma omp critical
  104:         cout << omp_get_thread_num() << " : " << vec.size() << endl;
  105:         #pragma omp barrier
  106:         iota( vec.begin(),vec.end(), omp_get_thread_num() );
  107:         #pragma omp barrier
  108:
  109:     }
  110:     return vec;
  111: }
  112:
  113:
  114: double scalar_trans(vector<double> const &x, vector<double> const &y)
  115: {
  116:     assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
  117:     vector<double> z(x.size());
  118:     //list<double> z(x.size());  // parallel for-loop on iterators not possible (mis
sing 'operator-')
  119:                                  // c++-20 CLANG_, ONEAPI_:condition of OpenMP for l
oop must be a relational comparison
  120:
  121:     transform(cbegin(x),cend(x),cbegin(y),begin(z),std::multiplies<>{});
  122:
  123:     double sum = 0.0;
  124: #pragma omp parallel for default(none) shared(z) reduction(+:sum) schedule(runtime)
// added schedule(runtime)
  125:     for (auto pi = cbegin(z); pi!=cend(z); ++pi)
  126:     {
  127:         sum += *pi;
  128:     }
  129:     //for (auto val: z)
  130:     //{
  131:         //sum += val;
```

```
  132:     //}
  133:     return sum;
  134: }
  135:
  136:
  137:
```

```
    1: #pragma once
    2: #include <cassert>
    3: #include <iomanip>              // setw()
    4: #include <iostream>
    5: #include <omp.h>
    6: #include <vector>
    7:
    8: /**     Inner product
    9:         @param[in] x    vector
   10:         @param[in] y    vector
   11:         @return             resulting Euclidian inner product <x,y>
   12: */
   13: double scalar_parallel(std::vector<double> const &x, std::vector<double> const &y);
   14: double scalar(std::vector<double> const &x, std::vector<double> const &y);
   15: double scalar_trans(std::vector<double> const &x, std::vector<double> const &y);
   16:
   17:
   18:
   19: // Declare additional reduction operation in OpenMP for STL-vector
   20: #pragma omp declare reduction(VecAppend : std::vector<int>  : omp_out.insert(omp_out
.end(), omp_in.begin(), omp_in.end())) \
   21:  initializer (omp_priv=omp_orig)
   22:
   23: std::vector<int> reduction_vec_append(int n);
   24:
   25:
   26: /**     l2-norm
   27:         @param[in] x    vector
   28:         @return             resulting Euclidian norm
   29: */
   30: double norm(std::vector<double> const &x);
   31:
   32: /**      Vector @p b adds its elements to vector @p a .
   33:         @param[in] a    vector
   34:         @param[in] b    vector
   35:         @return              a+=b componentwise
   36: */
   37: template<class T>
   38: std::vector<T> &operator+=(std::vector<T> &a, std::vector<T> const &b)
   39: {
   40:     assert(a.size()==b.size());
   41:     for (size_t k = 0; k < a.size(); ++k) {
   42:         a[k] += b[k];
   43:     }
   44:     return a;
   45: }
   46:
   47: // Declare the reduction operation in OpenMP for an STL-vector
   48: //   omp_out += omp_in  requires operator+=(vector<int> &, vector<int> const &) from
 above
   49: // ------------------------------------------------------------
   50: // https://scc.ustc.edu.cn/zlsc/tc4600/intel/2016.0.109/compiler_c/common/core/GUID-
7312910C-D175-4544-99C5-29C12D980744.htm
   51: // https://gist.github.com/eruffaldi/7180bdec4c8c9a11f019dd0ba9a2d68c
   52: // https://stackoverflow.com/questions/29633531/user-defined-reduction-on-vector-of-
varying-size
   53: //  see also p.74ff in  https://www.fz-juelich.de/ias/jsc/EN/AboutUs/Staff/Hagemeier
_A/docs-parallel-programming/OpenMP-Slides.pdf
   54: #pragma omp declare reduction(VecAdd : std::vector<int>  : omp_out += omp_in) \
   55:  initializer (omp_priv=omp_orig)
   56:
   57: //   Templates are  n o t  possible, i.e. the reduction has to be declared fore a sp
ecified type.
   58: //template <class T>
   59: //#pragma omp declare reduction(VecAdd : std::vector<T>  : omp_out += omp_in) initia
lizer (omp_priv(omp_orig))
   60: // MS: template nach #pragma  !?
   61:
```

```
   62: // ------------------------------------------------------------
   63:
   64: /**     Test for vector reduction.
   65:  *
   66:  * The thread-private vectors of size @p n are initialized via @f$v_k^{tID}=tID+k@f$
.
   67:  * Afterwards these vectors are accumulated, i.e.,
   68:  * @f$v_k= \sum_{tID=0}^{numThreads} v_k^{tID}@f$.
   69:  *
   70:  *      @param[in] n  size of global/private vector
   71:  *      @return  resulting global vector.
   72: */
   73: std::vector<int> reduction_vec(int n);
   74:
   75:
   76:
   77: /**     Output of a vector.
   78:         @param[in,out] s        output stream
   79:         @param[in]     x        vector
   80:         @return            modified output stream
   81: */
   82: template <class T>
   83: std::ostream &operator<<(std::ostream &s, std::vector<T> const &x)
   84: {
   85:     for (auto const &v : x)  s << std::setw(4) << v << "  ";
   86:     return s;
   87: }
   88:
```

```
    1: #pragma once
    2: #include <chrono>                    // timing
    3: #include <stack>
    4:
    5: using Clock = std::chrono::system_clock;   //!< The wall clock timer chosen
    6: //using Clock = std::chrono::high_resolution_clock;
    7: using TPoint= std::chrono::time_point<Clock>;
    8:
    9: // [Galowicz, C++17 STL Cookbook, p. 29]
   10: inline
   11: std::stack<TPoint> MyStopWatch; //!< starting time of stopwatch
   12:
   13: /** Starts stopwatch timer.
   14:  *  Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   15:  *
   16:  *  The timining is allowed to be nested and the recent time is stored on top of the
 stack.
   17:  *
   18:  *  @return recent time
   19:  *  @see toc
   20:  */
   21: inline auto tic()
   22: {
   23:     MyStopWatch.push(Clock::now());
   24:     return MyStopWatch.top();
   25: }
   26:
   27: /** Returns the elapsed time from stopwatch.
   28:  *
   29:  * The time from top of the stack is used
   30:  * if time point @p t_b is not passed as input parameter.
   31:  * Use as @code tic(); myfunction(...) ; double tsec = toc();  @endcode
   32:  * or as @code auto t_b = tic(); myfunction(...) ; double tsec = toc(t_b);  @endcode
   33:  * The last option is to be used in the case of
   34:  * non-nested but overlapping time measurements.
   35:  *
   36:  * @param[in]  t_b start time of some stop watch
   37:  * @return elapsed time in seconds.
   38:  *
   39: */
   40: inline double toc(TPoint const &t_b = MyStopWatch.top())
   41: {
   42:     // https://en.cppreference.com/w/cpp/chrono/treat_as_floating_point
   43:     using Unit       = std::chrono::seconds;
   44:     using FpSeconds = std::chrono::duration<double, Unit::period>;
   45:     auto t_e = Clock::now();
   46:     MyStopWatch.pop();
   47:     return FpSeconds(t_e-t_b).count();
   48: }
   49:
   50: #include <iostream>
   51: #include <string>
   52: /** Executes function @p f and measures/prints elapsed wall clock time in seconds
   53:  *
   54:  * Call as
   55:  * @code measure("Time for (b = b + 1)", [&]() {
   56:         thrust::transform(b.begin(), b.end(), b.begin(), increment());
   57:     }); @endcode
   58:  *
   59:  * @param[in]  label  additional string to be printed with the measurement.
   60:  * @param[in]  f      function to execute.
   61:  * @author Therese BÃ¶smÃ¼ller, 2025
   62:  *
   63: */
   64: auto measure = [](const std::string& label, auto&& f) {
   65:         auto start    = std::chrono::high_resolution_clock::now();
   66:         f();
   67:         auto stop     = std::chrono::high_resolution_clock::now();
```

```
   68:         auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop -
 start).count();
   69:         std::cout << label << ": " << duration << " microseconds" << std::endl;
   70: };          // ';' is needed for a visible documentation of this lambda-function
   71:
```