```
    1: #pragma once
    2:
    3: #include <iostream>
    4: #ifdef _OPENMP
    5: #include <omp.h>
    6: #endif
    7: #include <unordered_map>
    8:
    9: //#####################################
   10: // G.Haase
   11: // See https://sourceforge.net/p/predef/wiki/Compilers/
   12: //     http://www.cplusplus.com/doc/tutorial/preprocessor/
   13: //  also:  export OMP_DISPLAY_ENV=VERBOSE
   14: //#####################################
   15: /**     Checks for compilers, its versions, threads etc.
   16:  *
   17:         @param[in] argc number of command line arguemnts
   18:         @param[in] argv command line arguments as array of C-strings
   19: */
   20: template <class T>
   21: void check_env(T argc, char const *argv[])
   22: {
   23:     std::cout << "\n#########################################################################
#######\n";
   24:     std::cout << "Code    :";
   25:     for (T k = 0; k < argc; ++k) std::cout << "  " << argv[k];
   26:     std::cout << std::endl;
   27:
   28:     // compiler:      https://sourceforge.net/p/predef/wiki/Compilers/
   29:     std::cout <<    "Compiler:  ";
   30: #if defined __INTEL_COMPILER
   31: #pragma message(" ##########  INTEL  ###############")
   32:     std::cout << "INTEL " << __INTEL_COMPILER;
   33:     // Ignore warnings for #pragma acc   unrecognice
   34: #pragma warning disable 161
   35:     // Ignore warnings for #pragma omp   unrecognice
   36: #pragma warning disable 3180
   37:
   38: #elif defined __PGI
   39: #pragma message(" ##########  PGI    ###############")
   40:     std::cout << "PGI " << __PGIC__ << "." << __PGIC_MINOR__ << "." << __PGIC_PATCHL
EVEL__;
   41: #elif defined  __clang__
   42: #pragma message(" ##########  CLANG    ###############")
   43:     std::cout << "CLANG " << __clang_major__ << "." << __clang_minor__ << "."; // <<
 __clang_patchlevel__;
   44: #elif defined __GNUC__
   45: #pragma message(" ##########  Gnu    ###############")
   46:     std::cout << "Gnu " <<  __GNUC__  << "." << __GNUC_MINOR__ << "." << __GNUC_PATC
HLEVEL__;
   47: #else
   48: #pragma message(" ##########  unknown Compiler   ###############")
   49:     std::cout << "unknown";
   50: #endif
   51:     std::cout << "  C++ standard: " << __cplusplus << std::endl;
   52:
   53:     // Parallel environments
   54:     std::cout <<    "Parallel:  ";
   55: #if defined MPI_VERSION
   56: #pragma message(" ##########  MPI    ###############")
   57: #ifdef OPEN_MPI
   58:     std::cout << "OpenMPI ";
   59: #else
   60:     std::cout << "MPI ";
   61: #endif
   62:     std::cout << MPI_VERSION << "." << MPI_SUBVERSION << "   ";
   63: #endif
   64:
```

```
   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>
#include <cmath>
using namespace std;

void benchmark(vector<double> &x, vector<double> &y, unsigned int NLOOPS)
{
    double sk = 0.0;
    for (int i = 0; i < NLOOPS; ++i)
    {
        sk += scalar(x, y);
        // or scalar_trans(x,y) / norm(x) if you want
    }
}

int main(int argc, char const *argv[])
{
    //int const NLOOPS = 5;        // chose a value such that the benchmark runs at least 10 sec.
    unsigned int N = 5000001;
    int const NLOOPS = 5;        // chose a value such that the benchmark runs at least 10 sec.
    //unsigned int N = 5000001;
//#########################################################################
//   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)
    {
        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
        {
            cout << ss.str();                       // output to a shared ressource
        }
        #pragma omp master
        nthreads = nthrds;                          // transfer nn to to master thread
    }
    cout << "    " << nthreads << "   threads have been started." << endl;

```

```
   66: //#############################################################################
   67: //  Memory allocation
   68:     cout << "Memory allocation\n";
   69:
   70:     vector<double> x(N), y(N);
   71:
   72:     cout.precision(2);
   73:     cout << 2.0 * N *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n
";
   74:     cout.precision(6);
   75:
   76: //#############################################################################
   77: //  Data initialization
   78: //  Special:  x_i = i+1;  y_i = 1/x_i  ==> <x,y> == N
   79:     for (unsigned int i = 0; i < N; ++i)
   80:     {
   81:         x[i] = i + 1;
   82:         y[i] = 1.0 / x[i];
   83:     }
   84:
   85: //#############################################################################
   86:     cout << "\nStart Benchmarking\n";
   87:
   88: // Do calculation
   89:     double tstart = omp_get_wtime();                    // OpenMP
   90:
   91:     double sk(0.0);
   92:     for (int i = 0; i < NLOOPS; ++i)
   93:     {
   94:         sk = scalar(x, y);
   95:         sk = norm(x);
   96:     }
   97:
   98:     double t1 = omp_get_wtime() - tstart;               // OpenMP
   99:     t1 /= NLOOPS;            // divide by number of function calls
  100:
  101: //#############################################################################
  102: // Check the correct result
  103:     cout << "\n <x,y> = " << sk << endl;
  104:     if (static_cast<unsigned int>(sk) != N)
  105:     {
  106:         cout << "  !!   W R O N G  result   !!\n";
  107:     }
  108:     cout << endl;
  109:
  110: //#############################################################################
  111: // Timings  and Performance
  112:     cout << endl;
  113:     cout.precision(2);
  114:     cout << "Timing in sec. : " << t1 << endl;
  115:     cout << "GFLOPS         : " << 2.0 * N / t1 / 1024 / 1024 / 1024 << endl;
  116:     cout << "GiByte/s        : " << 2.0 * N / t1 / 1024 / 1024 / 1024 * sizeof(x[0])
 << endl;
  117:
  118: //#############################################################################
  119:
  120:     cout << "\n  Try the reduction with an STL-vektor \n";
  121:
  122:     auto vr = reduction_vec(100);
  123:     cout << "done\n";
  124:     cout << vr << endl;
  125:
  126:     N=200;
  127:     //Data (re-)inizialiion
  128:     for (unsigned int i = 0; i < N; ++i)
  129:     {
  130:         x[i] = i + 1;
  131:         y[i] = 1.0 / x[i];
```

```
132:     }
133:
134:
135:
136:
137:
138:     int proc_count = omp_get_num_procs();
139:     cout << "Number of available processors: " << proc_count << endl;
140:
141:     for(int j=1; j<=proc_count; j++)
142:     {
143:         omp_set_num_threads(j);
144:         cout << "used threads: "<< j << endl;
145:
146:
147:          omp_set_schedule(omp_sched_static, 0);
148:         tstart = omp_get_wtime();
149:         benchmark(x, y, NLOOPS);
150:         t1 = (omp_get_wtime()-tstart)/NLOOPS;
151:         cout << "static (chunk 0) "<< t1 << endl;
152:         for(int i=0; i<= 5; i++)
153:         {
154:
155:             int chunk = 1 << i;
156:             cout << "chunk size: "<< chunk << endl;
157:
158:             // STATIC
159:             omp_set_schedule(omp_sched_static, chunk);
160:             tstart = omp_get_wtime();
161:             benchmark(x, y, NLOOPS);
162:             t1 = (omp_get_wtime()-tstart)/NLOOPS;
163:             std::cout << "static:  "<< t1 << " s\n";
164:
165:             // DYNAMIC
166:             omp_set_schedule(omp_sched_dynamic, chunk);
167:             tstart = omp_get_wtime();
168:             benchmark(x, y, NLOOPS);
169:             t1 = (omp_get_wtime()-tstart)/NLOOPS;
170:             std::cout << "dynamic: "<< t1 << " s\n";
171:
172:             // GUIDED
173:             omp_set_schedule(omp_sched_guided, chunk);
174:             tstart = omp_get_wtime();
175:             benchmark(x, y, NLOOPS);
176:             t1 = (omp_get_wtime()-tstart)/NLOOPS;
177:             std::cout << "guided:  "<< t1 << " s\n";
178:
179:             // AUTO
180:             omp_set_schedule(omp_sched_auto, chunk);
181:             tstart = omp_get_wtime();
182:             benchmark(x, y, NLOOPS);
183:             t1 = (omp_get_wtime()-tstart)/NLOOPS;
184:             std::cout << "auto:    "<< t1 << " s\n";
185:             cout << endl;
186:         }
187:         cout << endl;
188:     }
189:
190:
191:     cout << scalar_parrallel_env(x,y) << endl;
192:
193:
194:
195:     vector<int> vec = reduction_vec_append(N);
196:     unsigned int n = vec.size();
197:     cout << "vec.size() = " << n << "\n";
198:     for(int i=0; i< n; i++)
199:     {
```

```
200:         cout << vec[i] << ", ";
201:     }
202:     return 0;
203: }  // memory for x and y will be deallocated their destructors
204:
```

```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(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;
    if (!omp_in_parallel())
    {
        // Safe to start a parallel region
        #pragma omp parallel for default(none) shared(x,y,N) reduction(+:sum) schedule(runtime)
        for (size_t i = 0; i < N; ++i)
            sum += x[i] * y[i];
    }
    else
    {
        // Already inside parallel region: do it sequentially to avoid nested parallelism
        for (size_t i = 0; i < N; ++i)
            sum += x[i] * y[i];
    }
    return sum;
}


double scalar_parrallel_env(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;


    // Safe to start a parallel region
    #pragma omp parallel  default(none) shared(x,y,N,cout) reduction(+:sum)
    {
        int tid = omp_get_thread_num();
        int threadCount = omp_get_num_threads();
        cout << threadCount << endl;




        for (size_t i = tid*N/threadCount; i < tid*(N + 1)/threadCount; ++i)
        {
            sum += x[i] * y[i];
        }
    }

    return sum;
}

double norm(vector<double> const &x)
{
    size_t const N = x.size();
    double sum = 0.0;
#pragma omp parallel for default(none) shared(x,N) reduction(+:sum)
```

```
    for (size_t i = 0; i < N; ++i)
    {
        sum += x[i]*x[i];
    }
    return sum;
}



vector<int> reduction_vec(int n)
{
    vector<int> vec(n);
#pragma omp parallel default(none) shared(cout) reduction(VecAdd: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;
}

vector<int> reduction_vec_append(int n)
{
    vector<int> vec;
    #pragma omp parallel default(none) shared(cout,n) reduction(VecAppend:vec)
    {
        int tid = omp_get_thread_num();

        vector<int> local(n);
        iota(local.begin(), local.end(), tid);

        vec = local;
    }

    return vec;
}



double scalar_trans(vector<double> const &x, vector<double> const &y)
{
    assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
    vector<double> z(x.size());
    //list<double> z(x.size());  // parallel for-loop on iterators not possible (missing 'operator-')
                                 // c++-20 CLANG_, ONEAPI_:condition of OpenMP for loop must be a relational comparison

    transform(cbegin(x),cend(x),cbegin(y),begin(z),std::multiplies<>{});

    double sum = 0.0;
#pragma omp parallel for default(none) shared(z) reduction(+:sum)
    for (auto pi = cbegin(z); pi!=cend(z); ++pi)
    {
        sum += *pi;
    }
    //for (auto val: z)
    //{
        //sum += val;
    //}
    return sum;
}


```

133:

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

```
   61:
   62: /**     Test for vector reduction.
   63:  *
   64:  * The thread-private vectors of size @p n are initialized via @f$v_k^{tID}=tID+k@f$
.
   65:  * Afterwards these vectors are accumulated, i.e.,
   66:  * @f$v_k= \sum_{tID=0}^{numThreads} v_k^{tID}@f$.
   67:  *
   68:  *      @param[in] n  size of global/private vector
   69:  *      @return  resulting global vector.
   70: */
   71: std::vector<int> reduction_vec(int n);
   72: std::vector<int> reduction_vec_append(int n);
   73:
   74:
   75: /**     Output of a vector.
   76:         @param[in,out] s        output stream
   77:         @param[in]     x        vector
   78:         @return            modified output stream
   79: */
   80: template <class T>
   81: std::ostream &operator<<(std::ostream &s, std::vector<T> const &x)
   82: {
   83:     for (auto const &v : x)  s << std::setw(4) << v << "  ";
   84:     return s;
   85: }
   86:
```

```
    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:
```