

```

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 arguments
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 unrecognize
34:     #pragma warning disable 161
35:     // Ignore warnings for #pragma omp unrecognize
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:

```

```

1: #include "check_env.h"
2: #include "mylib.h"
3: #include <cstdlib>           // atoi()
4: #include <cstring>           // strcmp()
5: #include <ctime>
6: #include <iostream>
7: #include <omp.h>             // OpenMP
8: #include <sstream>
9: #include <string>
10: using namespace std;
11:
12: int main(int argc, char const *argv[])
13: {
14:
15:     omp_set_num_threads(4);
16:
17:     // #####
18:     unsigned int N = 50000001;
19:     int const NLOOPS = 50;           // chose a value such that the benchmark runs at 1
east 10 sec.
20:
21:     // unsigned int N = 11;
22:     // int const NLOOPS = 1;
23:     //#####
24:     // Read Parameter from command line (C++ style)
25:     cout << "Checking command line parameters for: -n <number> " << endl;
26:     for (int i = 1; i < argc; i++)
27:     {
28:         cout << " arg[" << i << "] = " << argv[i] << endl;
29:         string ss(argv[i]);
30:         if ("-n"==ss && i + 1 < argc) // found "-n" followed by another parameter
31:         {
32:             N = static_cast<unsigned int>(atoi(argv[i + 1]));
33:         }
34:         else
35:         {
36:             cout << "Corect call: " << argv[0] << " -n <number>\n";
37:         }
38:     }
39:
40:     cout << "\nN = " << N << endl;
41:
42:     check_env(argc, argv);
43:     //#####
44:     int nthreads;                       // OpenMP
45:     #pragma omp parallel default(none) shared(cout,nthreads)
46:     {
47:         int const th_id = omp_get_thread_num(); // OpenMP
48:         int const nthrds = omp_get_num_threads(); // OpenMP
49:         stringstream ss;
50:         ss << "C++: Hello World from thread " << th_id << " / " << nthrds << endl;
51:         #pragma omp critical
52:         {
53:             cout << ss.str();           // output to a shared ressource
54:         }
55:         #pragma omp master
56:         nthreads = nthrds;           // transfer nn to to master threa
d
57:     }
58:     cout << " " << nthreads << " threads have been started." << endl;
59:
60:     //#####
61:     // Memory allocation
62:     cout << "Memory allocation\n";
63:
64:     vector<double> x(N), y(N);
65:
66:     cout.precision(2);

```

```

67:      cout << 2.0 * N * sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n
";
68:      cout.precision(6);
69:
70:  //#####
71:  // Data initialization
72:  // Special: x_i = i+1; y_i = 1/x_i ==> <x,y> == N
73:  for (unsigned int i = 0; i < N; ++i)
74:  {
75:      x[i] = i + 1.0;
76:      y[i] = 1.0 / x[i];
77:  }
78:
79:  //#####
80:  cout << "\nStart Benchmarking\n";
81:
82:  // Do calculation
83:  double tstart = omp_get_wtime(); // OpenMP
84:
85:  double sk(0.0);
86:  for (int i = 0; i < NLOOPS; ++i)
87:  {
88:      sk = scalar2(x, y);
89:  }
90:
91:  double t1 = omp_get_wtime() - tstart; // OpenMP
92:  t1 /= NLOOPS; // divide by number of function calls
93:
94:  //#####
95:  // Check the correct result
96:  cout << "\n <x,y> = " << sk << endl;
97:  if (static_cast<unsigned int>(sk) != N)
98:  {
99:      cout << "  !!  W R O N G  result  !!\n";
100:  }
101:  cout << endl;
102:
103:  //#####
104:  // Timings and Performance
105:
106:  cout.precision(2);
107:  cout << "Timing in sec. : " << t1 << endl;
108:  cout << "GFLOPS : " << 2.0 * N / t1 / 1024 / 1024 / 1024 << endl;
109:  cout << "GiByte/s : " << 2.0 * N / t1 / 1024 / 1024 / 1024 * sizeof(x[0])
<< endl;
110:
111:  //#####
112:
113:  // ---- Speedup for scalar ----
114:  // Threads | Time | Speedup
115:  // 1 | 0.032s | 1
116:  // 2 | 0.022s | 1.45
117:  // 4 | 0.021s | 1.52
118:  // 8 | 0.022s | 1.45
119:  // 16 | 0.022s | 1.45
120:
121:  //#####
122:
123:
124:  cout << "\n Try the reduction with an STL-vektor \n";
125:
126:  auto vr = reduction_vec_append(10); ✓
127:  cout << "done\n";
128:  cout << vr << endl;
129:
130:
131:  return 0;
132: } // memory for x and y will be deallocated their destructors

```

133:

```
1: #include "mylib.h"
2: #include <cassert>           // assert()
3: #include <cmath>
4: #include <iostream>
5: #include <functional>       // multiplies<>{}
6: #include <list>
7: #include <numeric>          // iota()
8: #ifdef _OPENMP
9: #include <omp.h>
10: #endif
11: #include <vector>
12: using namespace std;
13:
14: double scalar(vector<double> const &x, vector<double> const &y)
15: {
16:     assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
17:     size_t const N = x.size();
18:     double sum = 0.0;
19: #pragma omp parallel for default(none) shared(x,y,N) reduction(+:sum) schedule(stati
c)
20:     for (size_t i = 0; i < N; ++i)
21:     {
22:         sum += x[i] * y[i];
23:         //sum += exp(x[i])*log(y[i]);
24:     }
25:     return sum;
26: }
27:
28: double scalar2(vector<double> const &x, vector<double> const &y)
29: {
30:     assert(x.size() == y.size());
31:     size_t const N = x.size();
32:     double sum = 0.0;
33:
34:     #pragma omp parallel shared(x,y,N,sum)
35:     {
36:         double local_sum = 0.0;
37:
38:         int const th_id = omp_get_thread_num();
39:         int const nthrds = omp_get_num_threads();
40:
41:         for (size_t i = th_id; i < N; i += nthrds) {
42:             local_sum += x[i] * y[i];
43:         }
44:
45:         #pragma omp atomic
46:         sum += local_sum;
47:     }
48:     return sum;
49: }
50:
51:
52: vector<int> reduction_vec(int n)
53: {
54:     vector<int> vec(n);
55: #pragma omp parallel default(none) shared(cout) reduction(VecAdd:vec)
56:     {
57:         #pragma omp barrier
58:         #pragma omp critical
59:         cout << omp_get_thread_num() << " : " << vec.size() << endl;
60:         #pragma omp barrier
61:         iota( vec.begin(), vec.end(), omp_get_thread_num() );
62:         #pragma omp barrier
63:         #pragma omp critical
64:         cout << vec << endl;
65:     }
66:     return vec;
67: }
```

```
68:
69: vector<int> reduction_vec_append(int n)
70: {
71:     vector<int> vec;
72:     #pragma omp parallel default(none) shared(cout,n) reduction(VecAppend:vec)
73:     {
74:         vector<int> local(n);
75:         #pragma omp barrier
76:         #pragma omp critical
77:         cout << omp_get_thread_num() << " : " << local.size() << endl;
78:         #pragma omp barrier
79:         iota( local.begin(),local.end(), omp_get_thread_num() );
80:         #pragma omp critical
81:         vec.insert(vec.end(), local.begin(), local.end());
82:     }
83:     return vec;
84: }
```



```

1: #pragma once
2:
3: #include <cassert>
4: #include <iomanip>           // setw()
5: #include <iostream>
6: #include <omp.h>
7: #include <vector>
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 scalar2(std::vector<double> const &x, std::vector<double> const &y);
16:
17: /**      Vector @p b adds its elements to vector @p a .
18:      @param[in] a      vector
19:      @param[in] b      vector
20:      @return           a+=b componentwise
21: */
22: template<class T>
23: std::vector<T> &operator+=(std::vector<T> &a, std::vector<T> const &b)
24: {
25:     assert(a.size()==b.size());
26:     for (size_t k = 0; k < a.size(); ++k) {
27:         a[k] += b[k];
28:     }
29:     return a;
30: }
31:
32: // Declare the reduction operation in OpenMP for an STL-vector
33: // omp_out += omp_in requires operator+=(vector<int> &, vector<int> const &) from
above
34: // -----
35: // https://scc.ustc.edu.cn/zlsc/tc4600/intel/2016.0.109/compiler_c/common/core/GUID-
7312910C-D175-4544-99C5-29C12D980744.htm
36: // https://gist.github.com/eruffaldi/7180bdec4c8c9a11f019dd0ba9a2d68c
37: // https://stackoverflow.com/questions/29633531/user-defined-reduction-on-vector-of-
varying-size
38: // see also p.74ff in https://www.fz-juelich.de/ias/jsc/EN/AboutUs/Staff/Hagemeier
_A/docs-parallel-programming/OpenMP-Slides.pdf
39: #pragma omp declare reduction(VecAdd : std::vector<int> : omp_out += omp_in) \
40:     initializer (omp_priv=omp_orig)
41:
42: // Templates are n o t possible, i.e. the reduction has to be declared fore a sp
ecified type.
43: //template <class T>
44: //#pragma omp declare reduction(VecAdd : std::vector<T> : omp_out += omp_in) initia
lizer (omp_priv(omp_orig))
45: // MS: template nach #pragma !?
46:
47: // -----
48:
49:
50: /**      Test for vector reduction.
51:      *
52:      * The thread-private vectors of size @p n are initialized via @f$v_k^{tID}=tID+k@f$
.
53:      * Afterwards these vectors are accumulated, i.e.,
54:      * @f$v_k = \sum_{tID=0}^{\text{numThreads}} v_k^{tID}@f$.
55:      *
56:      * @param[in] n      size of global/private vector
57:      * @return           resulting global vector.
58: */
59: std::vector<int> reduction_vec(int n);
60:
61:

```

```
62:
63: /**      Output of a vector.
64:     @param[in,out] s      output stream
65:     @param[in]      x      vector
66:     @return          modified output stream
67: */
68: template <class T>
69: std::ostream &operator<<(std::ostream &s, std::vector<T> const &x)
70: {
71:     for (auto const &v : x) s << std::setw(4) << v << " ";
72:     return s;
73: }
74:
75: #pragma omp declare reduction(VecAppend : std::vector<int> : omp_out.insert(omp_out.
end(), omp_in.begin(), omp_in.end()) ) initializer( omp_priv = std::vector<int>() )
76:
77: std::vector<int> reduction_vec_append(int n);
```

```

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 timing 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 B  sm  ller, 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:     };
71:         // ';' is needed for a visible documentation of this lambda-function
```