$s = \sum x_i y_i$

dpcpp .. -qopenmp

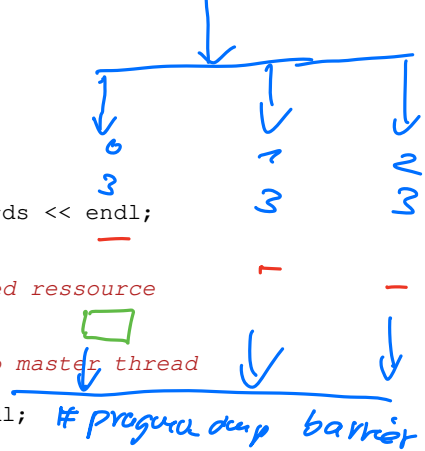
g++ ... -fopenmp main.cpp

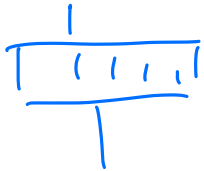
#pragma omp

```
1: #include "mylib.h"
2:
3: #include <cassert>
4: #include <cstdlib>            // atoi()
5: #include <cstring>            // strncmp()
6: #include <ctime>
7: #include <iostream>
8: #include <omp.h>              // OpenMP
9: #include <sstream>
10:
11: using namespace std;
12:
13: int main(int argc, char **argv)
14: {
15: // observe a good speedup because vectors fit into local caches
16:     int const NLOOPS = 5000;        // chose a value such that the benchmark runs at least 10 sec.
17:     unsigned int N = 500001;
18: // nearly no speedup because vectors too large for caches
19:     //int const NLOOPS = 50;         // chose a value such that the benchmark runs at least 10 sec
.
20:     //unsigned int N = 50000001;
21: //#####################################################################
22: //   Read Paramater from command line  (C++ style)
23:     cout << "Checking command line parameters for: -n <number> " << endl;
24:     for (int i = 1; i < argc; i++)
25:     {
26:         cout << " arg[" << i << "] = " << argv[i] << endl;
27:         if (std::strncmp(argv[i], "-n", 2) == 0 && i + 1 < argc) // found "-n" followed by another
 parameter
28:         {
29:             N = static_cast<unsigned int>(atoi(argv[i + 1]));
30:         }
31:         else
32:         {
33:             cout << "Corect call: " << argv[0] << " -n  <number>\n";
34:         }
35:     }
36:
37:     cout << "\nN = " << N << endl;
38: //#####################################################################
39:     int nthreads;                                   // OpenMP
```

shared ressource

```
40:     #pragma omp parallel default(none) shared(cout,nthreads)
41:     {
42:         int const th_id  = omp_get_thread_num();   // OpenMP
43:         int const nthrds = omp_get_num_threads();  // OpenMP
44:         stringstream ss;
45:         ss << "C++: Hello World from thread " << th_id << " / " << nthrds << endl;
46:         #pragma omp critical
47:         {
48:             cout << ss.str();                      // output to a shared ressource
49:         }
50:         #pragma omp master
51:         nthreads = nthrds;                         // transfer nn to to master thread
52:     }
53:     cout << "    " << nthreads << "   threads have been started." << endl;
54:
55: //#########################################################################
56: //  Memory allocation
57:     cout << "Memory allocation\n";
58:
59:     vector<double> x(N), y(N);
60:
61:     cout.precision(2);
62:     cout << 2.0 * N *sizeof(x[0]) / 1024 / 1024 / 1024 << " GByte Memory allocated\n";
63:     cout.precision(6);
64:
65: //#########################################################################
66: //  Data initialization
67: //  Special:  x_i = i+1;  y_i = 1/x_i  ==> <x,y> == N
68:     for (unsigned int i = 0; i < N; ++i)
69:     {
70:         x[i] = i + 1;
71:         y[i] = 1.0 / x[i];
72:     }
73:
74: //#########################################################################
75:     cout << "\nStart Benchmarking\n";
76:
77: // Do calculation
78:     double tstart = omp_get_wtime();                // OpenMP
79:
80:     double sk(0.0), sg(0.0);
```





```
   81:     for (int i = 0; i < NLOOPS; ++i)
   82:     {
   83:         sk = scalar(x, y);
   84:         //sk = scalar17(x, y);
   85: //      sk = norm(x);
   86:         sg += sk;
   87:     }
   88:
   89:     double t1 = omp_get_wtime() - tstart;              // OpenMP
   90:     t1 /= NLOOPS;            // divide by number of function calls
   91:     assert( std::abs(sg/NLOOPS-sk)<1e-5 );
   92:
   93: //#########################################################################
   94: // Check the correct result
   95:     cout << "\n <x,y> = " << sk << endl;
   96:     if (static_cast<unsigned int>(sk) != N)
   97:     {
   98:         cout << "  !!   W R O N G  result   !!\n";
   99:     }
  100:     cout << endl;
  101:
  102: //#########################################################################
  103: // Timings  and Performance
  104:     cout << endl;
  105:     cout.precision(2);
  106:     cout << "Timing in sec. : " << t1 << endl;
  107:     cout << "GFLOPS         : " << 2.0 * N / t1 / 1024 / 1024 / 1024 << endl;
  108:     cout << "GiByte/s       : " << 2.0 * N / t1 / 1024 / 1024 / 1024 * sizeof(x[0]) << endl;
  109:
  110: //#########################################################################
  111:     return 0;
  112: }  // memory for x and y will be deallocated their destructors
  113:
```

```
 1: #include "mylib.h"
 2: #include <cmath>
 3: #include <cassert>        // assert()
 4: #include <execution>      // execution policy
 5: #include <vector>
 6: using namespace std;
 7:
 8: double scalar(vector<double> const &x, vector<double> const &y)
 9: {
10:     assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
11:     size_t const N = x.size();
12:     double sum = 0.0;
13: #pragma omp parallel for default(none) firstprivate(N) shared(x,y) reduction(+:sum)
14:     for (size_t i = 0; i < N; ++i)
15:     {
16:         sum += x[i] * y[i];
17:         //sum += exp(x[i])*log(y[i]);
18:     }
19:     return sum;
20: }
21:
22: double scalar17(vector<double> const &x, vector<double> const &y)
23: {
24:     assert(x.size() == y.size()); // switch off via compile flag: -DNDEBUG
25:     return transform_reduce(std::execution::par_unseq,cbegin(x),cend(x),cbegin(y),0.0,
26:            std::plus<>(),std::multiplies<>());
27: }
28:
29:
30: double norm(vector<double> const &x)
31: {
32:     size_t const N = x.size();
33:     double sum = 0.0;
34:     for (size_t i = 0; i < N; ++i)
35:     {
36:         sum += x[i] * x[i];
37:     }
38:     return std::sqrt(sum);
39: }
40:
```

```
 1: #ifndef FILE_MYLIB
 2: #define FILE_MYLIB
 3: #include <vector>
 4:
 5: /**     Inner product
 6:         @param[in] x    vector
 7:         @param[in] y    vector
 8:         @return             resulting Euclidian inner product <x,y>
 9: */
10: double scalar(std::vector<double> const &x, std::vector<double> const &y);
11:
12: /**     Inner product
13:         @param[in] x    vector
14:         @param[in] y    vector
15:         @return             resulting Euclidian inner product <x,y>
16: */
17: double scalar17(std::vector<double> const &x, std::vector<double> const &y);
18:
19:
20: /**     L_2 Norm of a vector
21:         @param[in] x    vector
22:         @return             resulting Euclidian norm <x,y>
23: */
24: double norm(std::vector<double> const &x);
25:
26:
27:
28:
29: #endif
```