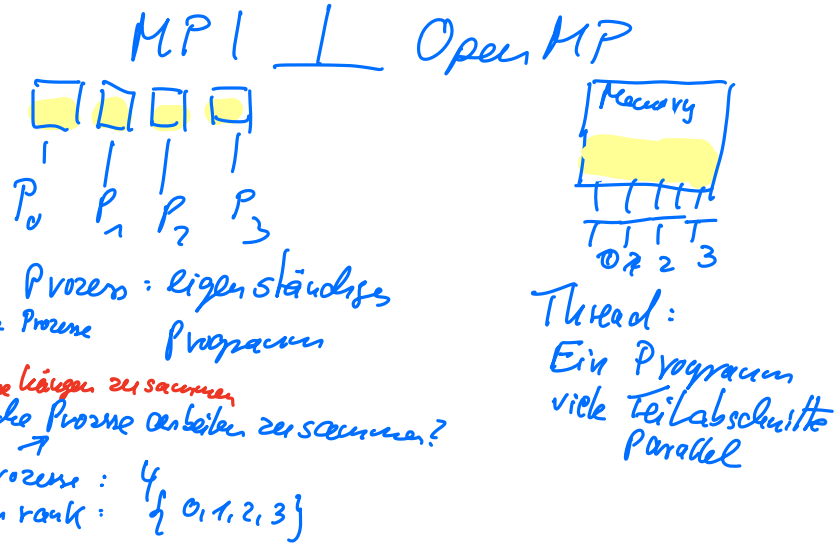
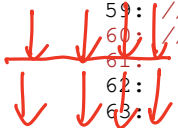
```
  1: #include "mylib.h"
  2: #include <cstdlib>            // atoi()
  3: #include <cstring>            // strncmp()
  4: #include <ctime>
  5: #include <iostream>
  6: #include <mpi.h>              // MPI
  7: #include <omp.h>              // OpenMP
  8: #include <sstream>
  9: using namespace std;
 10:
 11: int main(int argc, char **argv)
 12: {
 13:     MPI_Init(&argc,&argv);
 14:     MPI_Comm icomm(MPI_COMM_WORLD);
 15:     int numprocs;
 16:     MPI_Comm_size(icomm, &numprocs);
 17:     int myrank;
 18:     MPI_Comm_rank(icomm, &myrank);
 19:
 20:     int const NLOOPS = 50;        // chose a value such that the benchmark runs at least 10 sec.
 21:     unsigned int N = 50000001;
 22:
 23: //#############################################################################
 24: //   Read Paramater from command line  (C++ style)
 25:     if (myrank == 0) cout << "Checking command line parameters for: -n <number> " << endl;
 26:     MPI_Barrier(icomm);
 27:     for (int i = 1; i < argc; i++)
 28:     {
 29:         if (myrank == 0) cout << " arg[" << i << "] = " << argv[i] << endl;
 30:         if (std::strncmp(argv[i], "-n", 2) == 0 && 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: //#########################################################################
43:     //int nthreads;                                          // OpenMP
44:     //#pragma omp parallel default(none) shared(cout,nthreads)
45:     //{
46:     //int const th_id  = omp_get_thread_num();    // OpenMP
47:     //int const nthrds = omp_get_num_threads();   // OpenMP
48:     //stringstream ss;
49:     //ss << "C++: Hello World from thread " << th_id << " / " << nthrds << endl;
50:     //#pragma omp critical
51:     //{
52:     //cout << ss.str();                          // output to a shared ressource
53:     //}
54:     //#pragma omp master
55:     //nthreads = nthrds;                                 // transfer nn to to master thread
56:     //}
57:     //cout << "    " << nthreads << "   threads have been started." << endl;
58:
59: //#########################################################################
60: //  Memory allocation
61:     MPI_Barrier(icomm);
62:     if (myrank == 0) 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;
76:         y[i] = 1.0 / x[i];
77:     }
78:
79: //#########################################################################
80:     if (myrank == 0) cout << "\nStart Benchmarking\n";
81:
```



Empfehlung: nicht in Alternativen nur Root-process mit Ausgabe

$N_0\ N_1\ N_2\ N_3$

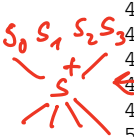
lokale Vektorlänge ⇒ globale Vektorlänge ← # processes

```
 82: // Do calculation
 83:     double tstart = MPI_Wtime();                    // Wall clock
 84:
 85:     double sk(0.0);
 86:     for (int i = 0; i < NLOOPS; ++i)
 87:     {
 88:         sk = par_scalar(x, y);
 89:         //sk = par_scalar(x, y, icomm);
 90:         //sk = scalar(x, y);
 91: //      sk = norm(x);
 92:     }
 93:
 94:     double t1 = MPI_Wtime() - tstart;               // Wall clock
 95:     //t1 /= CLOCKS_PER_SEC;    // now, t1 in seconds
 96:     t1 /= NLOOPS;            // divide by number of function calls
 97:
 98:     if (myrank == 0)
 99:     {
100: //############################################################################
101: // Check the correct result
102:         cout << "\n <x,y> = " << sk << endl;
103:         if (static_cast<unsigned int>(sk) != N * numprocs)
104:         {
105:             cout << "  !!   W R O N G  result   !!\n";
106:         }
107:         cout << endl;
108:
109: //############################################################################
110: // Timings  and Performance
111:         cout << endl;
112:         cout.precision(2);
113:         cout << "Timing in sec. : " << t1 << endl;
114:         cout << "GFLOPS         : " << 2.0 * N / t1 / 1024 / 1024 / 1024 << endl;
115:         cout << "GiByte/s       : " << 2.0 * N / t1 / 1024 / 1024 / 1024 * sizeof(x[0]) << endl;
116:
117: //############################################################################
118:     }
119:     MPI_Finalize();
120:     return 0;
121: }  // memory for x and y will be deallocated their destructors
122:
```

Unabh. Prozesse

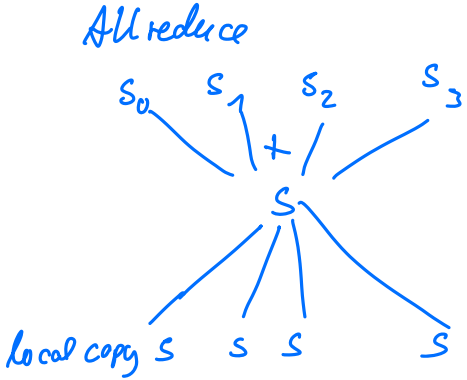
```cpp
#include "mylib.h"
#include <cmath>
#include <cassert>        // assert()
#include <mpi.h>             // MPI
#include <vector>
using namespace std;

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;
#pragma omp parallel for default(none) shared(x,y,N) reduction(+:sum)
    for (size_t i = 0; i < N; ++i)
    {
        sum += x[i] * y[i];
        //sum += exp(x[i])*log(y[i]);
    }
    return sum;
}


double norm(vector<double> const &x)
{
    size_t const N = x.size();
    double sum = 0.0;
    for (size_t i = 0; i < N; ++i)
    {
        sum += x[i] * x[i];
    }
    return std::sqrt(sum);
}

/*
double par_scalar_cpp(vector<double> const &x, vector<double> const &y, MPI::Intracomm const& icom
m)
{
  const double s = scalar(x,y);
        double sg;
  icomm.Allreduce(&s,&sg,1,MPI::DOUBLE,MPI::SUM);

```

```
41:   return(sg);
42: }
43: */
44:
45: double par_scalar(vector<double> const &x, vector<double> const &y, MPI_Comm const& icomm)
46: {
47:   const double s = scalar(x,y);
48:         double sg;
49: MPI_Allreduce(&s,&sg,1,MPI_DOUBLE,MPI_SUM,icomm);
50:
51:   return(sg);
52: }
```



$S_k := \sum_{i \in w_k} x_i \cdot y_i$

$S = \sum_{k=0}^{3} S_k$



Reduce

Broadcast

erweitbr. Kontext
(ohne Arithm.)

Gather

Scatter

Gather/Scatter

```
    1: #ifndef FILE_MYLIB
    2: #define FILE_MYLIB
    3: #include <mpi.h>            // MPI
    4: #include <vector>
    5:
    6: /**     Inner product
    7:         @param[in] x    vector
    8:         @param[in] y    vector
    9:         @return             resulting Euclidian inner product <x,y>
   10: */
   11: double scalar(std::vector<double> const &x, std::vector<double> const &y);
   12:
   13: /**     L_2 Norm of a vector
   14:         @param[in] x    vector
   15:         @return             resulting Euclidian norm <x,y>
   16: */
   17: double norm(std::vector<double> const &x);
   18:
   19: /**     Parallel inner product with C++ bindings.
   20:         @param[in] x    vector
   21:         @param[in] y    vector
   22:     @param[in] icomm  MPI communicator
   23:         @return             resulting Euclidian inner product <x,y>
   24:     @warning The C++ bindings were deprecated as of MPI-2.2, see <a href="https://www.mpi-forum.or
g/docs/mpi-3.1/mpi31-report/node405.htm#Node40">MPI 3.1</a>
   25: */
   26: /*
   27: double par_scalar_cpp(std::vector<double> const &x, std::vector<double> const &y,
   28:                   MPI::Intracomm const& icomm=MPI::COMM_WORLD);
   29: */
   30:
   31: /**     Parallel inner product
   32:         @param[in] x    vector
   33:         @param[in] y    vector
   34:     @param[in] icomm  MPI communicator
   35:         @return             resulting Euclidian inner product <x,y>
   36: */
   37: double par_scalar(std::vector<double> const &x, std::vector<double> const &y,
   38:                   MPI_Comm const& icomm=MPI_COMM_WORLD);
   39:
   40: #endif
```