E5 - E8


```
    1: //              MPI code in C++.
    2: //              See [Gropp/Lusk/Skjellum, "Using MPI", p.33/41 etc.]
    3: //              and  /opt/mpich/include/mpi2c++/comm.h  for details
    4:
    5: #include "vec_func.h"
    6: #include <iostream>                                  // MPI
    7: #include <mpi.h>
    8: using namespace std;
    9:
   10: int main(int argc, char *argv[])
   11: {
   12:     MPI_Comm icomm = MPI_COMM_WORLD;
   13:     MPI_Init(&argc, &argv);
   14:
   15:         int myrank;
   16:     MPI_Comm_rank(icomm, &myrank);
   17:
   18:         // E5
   19:         vector<double> x(10,3);
   20:         if(myrank==0) {cout << endl << "E5" << endl;}
   21:         vecdebug(x, icomm);
   22:
   23:         // E6
   24:         vector<double> y(10,1.0/3);
   25:         double res = par_scalar(x,y,icomm);
   26:         if(myrank==0)
   27:         {
   28:                 cout << endl << endl << "E6" << endl;
   29:                 cout << "<x,y> = " << res << endl << endl;
   30:         }
   31:
   32:         // E7
   33:         vector<double> a{myrank*10, myrank*10-1, myrank*10-2};
   34:         if(myrank==0) {cout << "E7\n" << "original vector" << endl;}
   35:         vecdebug(a, icomm);
   36:         double min_val, max_val;
   37:         min_max_exch(a, min_val, max_val, icomm);
   38:         if(myrank==0) {cout << "min = " << min_val << "\tmax = " << max_val << endl
<< endl << "vector after changing min-max" << endl;}
   39:         vecdebug(a, icomm);
   40:
   41:         // E8
   42:         int n = 20;
   43:         vector<double> c(n);
   44:         for(size_t i=0; i<n; ++i)
   45:         {
   46:                 c[i] = myrank*100 +(i%5)*10+i;
   47:         }
   48:         vector<double> recv(n);
   49:  v      MPI_Alltoall(c.data(), 5, MPI_DOUBLE, recv.data(), 5, MPI_DOUBLE, icomm);
   50:     if(myrank==0) {cout << endl << "Ex 8\n" << "Alltoall" << endl;}
   51:     vecdebug(recv, icomm);
   52:
   53:  √  MPI_Alltoall(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, c.data(), 5, MPI_DOUBLE, icomm)
;
   54:     if(myrank==0) {cout << endl << "Ex 8\n" << "Alltoall - IN_PLACE" << endl;}
   55:     vecdebug(c, icomm);
   56:
   57:
   58:
   59:     MPI_Finalize();
   60:
   61:     return 0;
   62: }
   63:
   64:
```

```
    1: #include "vec_func.h"
    2: #include <cassert>
    3: #include <cstring>
    4: #include <iostream>
    5: #include <mpi.h>                                    // MPI
    6: #include <string>
    7: using namespace std;
    8:
    9: // see  http://www.open-mpi.org/doc/current
   10: // for details on MPI functions
   11:
   12: void vecdebug(vector<double> const & x, const MPI_Comm &icomm)
   13: {
   14:         int myrank, size;
   15:     MPI_Comm_rank(icomm, &myrank);
   16:     MPI_Comm_size(icomm, &size);
   17:     int ierr;
   18:
   19:     int n = x.size();
   20:
   21:     int take_process;                  ! 'size' iterations
   22:     for(int i=0; i<size; ++i)
   23:     {
   24:                 MPI_Barrier(icomm);
   25:                 if(myrank == 0) // only one process for input
   26:                 {
   27:                         cout << "\nChoose next process for printing vector (0-3): ";
   28:                         cout.flush();
   29:                         cin >> take_process;    // no check for wrong input
   30:                 }
   31:
   32:                 ierr = MPI_Bcast(&take_process, 1, MPI_INT, 0, icomm); // broadcast
value of "take_process" to all processes -> all know what's going on
   33:                 // information comes from process 0
   34:                 assert(ierr == 0);
   35:
   36:                 MPI_Barrier(icomm);
   37:
   38:                 if(take_process == myrank)
   39:                 {
   40:                         for(int k=0; k<n; ++k)
   41:                         {
   42:                                 cout << "x_" << k << " = " << x[k] << " (" << myrank
 << ")\t";
   43:                                 cout.flush();
   44:                                 // in brackets we have the actual process
   45:                         }
   46:                         cout << endl;
   47:                         cout.flush();
   48:                 }
   49:
   50:                 MPI_Barrier(icomm);
   51:         }
   52:         // to avoid some output chaos with cout you could also do it with MPI_Send a
nd MPI_Recv
   53:
   54:         return;
   55: }
   56:
   57:
   58: double par_scalar(vector<double> const & x, vector<double> const & y, const MPI_Comm
 &icomm)
   59: {
   60:         assert(x.size()==y.size());
   61:
   62:         double sum_local = 0;
   63:         double sum_global = 0;
   64:
```

```
   65:         for(size_t k=0; k<x.size(); ++k)
   66:         {
   67:                 sum_local += x[k]*y[k];
   68:         }
   69:
   70:         auto ierr = MPI_Allreduce(&sum_local, &sum_global, 1, MPI_DOUBLE, MPI_SUM, i
comm);  // reduce local sums to global sum
   71:     assert(ierr == 0);
   72:
   73:         return sum_global;
   74: }
   75:
   76:
   77: void min_max_exch(vector<double> &x, double &min_val, double &max_val, const MPI_Com
m &icomm)
   78: {
   79:         int myrank;
   80:     MPI_Comm_rank(icomm, &myrank);
   81:         int local_n = x.size();
   82:         int global_offset = myrank * local_n;   // for interchanging
   83:
   84:         struct {double value; int index;} local_min, local_max, global_min, global_m
ax;     // global index
   85:
   86:         local_min.value = x[0];
   87:         local_max.value = x[0];
   88:         local_min.index = global_offset;
   89:         local_max.index = global_offset;
   90:
   91:         // finding local min/max with the corresponding global index
   92:         for (int i = 1; i < local_n; ++i)
   93:         {
   94:             if (x[i] < local_min.value)
   95:             {
   96:                 local_min.value = x[i];
   97:                 local_min.index = global_offset + i;
   98:             }
   99:             if (x[i] > local_max.value)
  100:             {
  101:                 local_max.value = x[i];
  102:                 local_max.index = global_offset + i;
  103:             }
  104:         }
  105:
  106:         // reduction to the global one including the global index (need it later for
 interchanging)
  107:         MPI_Allreduce(&local_min, &global_min, 1, MPI_DOUBLE_INT, MPI_MINLOC, icomm)
;
  108:         MPI_Allreduce(&local_max, &global_max, 1, MPI_DOUBLE_INT, MPI_MAXLOC, icomm)
;
  109:         min_val = global_min.value;
  110:         max_val = global_max.value;
  111:
  112:         // calculating the process and the local index for interchanging the min and
 max value
  113:         int rank_min = global_min.index / local_n;
  114:         int rank_max = global_max.index / local_n;
  115:         int local_min_idx = global_min.index % local_n;
  116:         int local_max_idx = global_max.index % local_n;
  117:
  118:         // interchanging
  119:         if (rank_min != rank_max)
  120:         {
  121:             double recv_value;
  122:             if (myrank == rank_min)
  123:             {
  124:                 MPI_Sendrecv(&x[local_min_idx], 1, MPI_DOUBLE, rank_max, 0, &recv_va
lue, 1, MPI_DOUBLE, rank_max, 0, icomm, MPI_STATUS_IGNORE);     // from last it gets receiv
```

```
ed, first send it to
  125:                 x[local_min_idx] = recv_value;
  126:             }
  127:
  128:             if (myrank == rank_max)
  129:             {
  130:                 MPI_Sendrecv(&x[local_max_idx], 1, MPI_DOUBLE, rank_min, 0, &recv_va
lue, 1, MPI_DOUBLE, rank_min, 0, icomm, MPI_STATUS_IGNORE);
  131:                 x[local_max_idx] = recv_value;
  132:             }
  133:
  134:         }
  135:         else // min and max value are on same process
  136:         {
  137:                 swap(x[local_min_idx], x[local_max_idx]);
  138:         }
  139:
  140:         return;
  141: }
```

```
    1: //      general header for all functions in directory
    2:
    3: #pragma once
    4:
    5: #include <mpi.h>
    6: #include <vector>
    7:
    8: /**     Debug and print the vector with MPI, you can choose the process which should
 present its values
    9:          @param[in]      x       vector to print
   10:          @param[in]      icomm   the MPI process group that is used.
   11: */
   12: void vecdebug(std::vector<double> const & x, const MPI_Comm &icomm);
   13:
   14: /**     Calculate the scalar product of two vectors (MPI)
   15:          @param[in]      x       vector
   16:          @param[in]      y       vector
   17:          @param[in]      icomm   the MPI process group that is used.
   18:          @return sum             euclidean scalar product
   19: */
   20: double par_scalar(std::vector<double> const & x, std::vector<double> const & y, cons
t MPI_Comm &icomm);
   21:
   22:
   23: /**     Determine min and max value of the vector and exchanging those two values
   24:          @param[in,out]  x       vector
   25:          @param[in,out]  min_val
   26:          @param[in,out]  max_val
   27:          @param[in]      icomm   the MPI process group that is used.
   28: */
   29: void min_max_exch(std::vector<double> &x, double &min_val, double &max_val, const MP
I_Comm &icomm);
   30:
   31:
```