```
#include <iostream>
#include <ostream>
#include <sstream>  //for stringstream/ostringstream/istringstream

#include <stdio.h>
#include <string.h>
#include <mpi.h>

using std::cout;
using std::endl;
using std::ostringstream;

int main(int argc, char **argv) {
    int     comm_sz;    /* number of processes  */
    int     my_rank;    /* process rank         */

    //Initialize the command line arguments on every process
    MPI_Init(&argc, &argv);

    //Get the number of processes in MPI_COMM_WORLD, and put
    //it in the 'comm_sz" variable; ie., how many processes
    //are running this program (the same as what you put in the
    //-np argument).
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    //Get the rank of this particular process in MPI_COMM_WORLD,
    //and put it in the 'my_rank' variable -- ie., what number
    //is this process
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    int array_size = 20;
    double* array = new double[array_size];

    int slice_size = array_size / comm_sz;
    double* array_slice = new double[slice_size];

    for (int i = 0; i < slice_size; i++) {
        array_slice[i] = 0;
    }

    if (my_rank == 0) {
        srand48(time(NULL));
        for (int i = 0; i < array_size; i++) {
            array[i] = drand48();
        }
    } else {
        for (int i = 0; i < array_size; i++) {
            array[i] = 0;
        }
    }

    ostringstream oss;  /*Writing what we want to say to a string stream
                          and then writing it's contents to cout will
                          prevent interleaving of print statements to
                          the root process */
    if (my_rank == 0) {
        oss << "[process: " << my_rank << "]:";
        for (int i = 0; i < array_size; i++) {
            oss << " " << array[i];
        }
        cout << oss.str() << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Scatter(array /*the data we're scattering*/,
                slice_size /* the size of the data we're scattering
                              to each process */,
                MPI_DOUBLE /* the data type we're sending */,
                array_slice /*where we're receiving the data */,
                slice_size /*the amount of data we're
                                         receiving per process*/,
                MPI_DOUBLE /* the data type we're receiving */,
                0 /*the process we're sending from*/,
                MPI_COMM_WORLD);

    oss.str("");
    oss.clear();
    oss << "[process: " << my_rank << "]:";
    for (int i = 0; i < slice_size; i++) {
        oss << " " << array_slice[i];
    }
    cout << oss.str() << endl;

    //double all the data
    for (int i = 0; i < slice_size; i++) {
        array_slice[i] *= 2;
    }

    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Allgather(array_slice /*the data we're gathering */,
               slice_size /* the size of the data we're gathering */,
               MPI_DOUBLE /* the data type we're sending */,
               array /*where we're receiving the data */,
               slice_size /*the amount of data we're
                             receiving from each process*/,
               MPI_DOUBLE /* the data type we're receiving */,
// We don't need the next line because the data is gathered to all processes:
//               0 /*the process we're sending from*/,
               MPI_COMM_WORLD);

//    if (my_rank == 0) {
        oss.str("");
        oss.clear();
        oss << "[process: " << my_rank << "]:";
        for (int i = 0; i < array_size; i++) {
            oss << " " << array[i];
        }
        cout << oss.str() << endl;
    }

    MPI_Finalize();
}
```