MA6252 Topics in Applied Mathematics II
Fourth homework (50%)

1. Write a simple program to determine the value of π. The algorithm suggested here is chosen for its simplicity: The method evaluates the integral of 4/(1+x*x) between 0 and 1.

The integral is approximated by a sum of n intervals which is defined in the beginning of the program; the approximation to the integral in each interval is (1/n)*4/(1+x*x). Each process then adds up every n'th interval (x = rank/n, rank/n+size/n,...). Finally, the sums computed by each process are added together using a reduction.

2. Implement a MPI parallel Jacobi algorithm for an one-dimensional domain. The mapping of the domain to the MPI processes is depict in figure 1. Assume that the domain is not periodic; that is, the top process (rank = size - 1) only sends and receives data from the one under it (rank = size - 2) and the bottom process (rank = 0) only sends and receives data from the one above it (rank = 1).

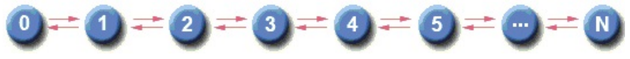

Figure 1: Decomposition of the domain to the ranks.

For the computation that we're going to perform on an array data structure, we'll need the adjacent values. That is, to compute a new x[i], we will need x[i+1] and x[i-1].

The outermost two of these could be a problem if they are not in the local x but are instead on the adjacent processors. To handle this difficulty, we define ghost points that we will contain the values of these adjacent points. These ghost points are updated at the boundaries of the partitions after each iteration (see figure 2).
Calculate and print out the residual after each Jacobi iteration. Verify the correctness of the solvers by comparing them to the serial case.

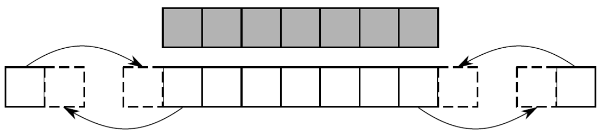

Figure 2: Ghostlayer exchange between the processes.

To avoid deadlocks, you can use the asynchronous MPI_Isend and MPI_Irecv in combination with the MPI_Waitall call. The following code snippet is a possible way to determine the required numbers of requests:

```
MPI_Request request[4];
MPI_Status status[4];
int reqcount = 0;

// check if neighbor exists then:
  MPI_Irecv(..., MPI_COMM_WORLD, &request[reqcount]);
  reqcount++;
...
// check if neighbor exists then:
  MPI_Isend(..., MPI_COMM_WORLD, &request[reqcount]);
  reqcount++;
...
MPI_Waitall(reqcount, request, status);
```

3. Run the program with 2 and 4 processes and different number of grid points 10^k, for k = 2, 3, 4, 5, and 6. Plot the ratio of your parallel run times to the serial code run time as a function of k.

4. Describe how you would implement a parallel Gauss-Seidel method in two dimensions (i.e. which ordering would you choose and why?).