E517/317 – Introduction to High Performance Computing
Fall 2019
Assignment 3

1. Modify the sendrecv.c example from Lecture 10 to send the following user-defined data type to the process on the left (rank-1) and receive from the process on the right (rank+1).

```
 4 typedef struct {
 5   int max_iter;
 6   double t0;
 7   double tf;
 8   double xmax[12];
 9   double xmin;
10 } Pars;
```

That is: modify the following code to send the Pars structure:

```
 1 #include "mpi.h"
 2 #include <stdio.h>
 3
 4 typedef struct {
 5   int max_iter;
 6   double t0;
 7   double tf;
 8   double xmax[12];
 9   double xmin;
10 } Pars;
11
12 int main(int argc, char *argv[])
13 {
14     int myid, numprocs, left, right;
15     Pars buffer, buffer2;
16     MPI_Request request;
17     MPI_Status status;
18
19     MPI_Init(&argc,&argv);
20     MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
21     MPI_Comm_rank(MPI_COMM_WORLD, &myid);
22
23     right = (myid + 1) % numprocs;
24     left = myid - 1;
25     if (left < 0)
26         left = numprocs - 1;
27
28     // initialize the send buffer
29     buffer.max_iter = myid ;
30     buffer.t0 = 3.14*myid ;
31     buffer.tf = 1.67*myid ;
32     buffer.xmin = 2.55*myid ;
33     for (int i=0;i<12;i++) {
34       buffer.xmax[i] = 2.7*myid;
35     }
36
37     // Modify this
38     // send myid to the left
39     //MPI_Sendrecv(&buffer, 1, FIXME, left, 123,
40     //   &buffer2, 1, FIXME, right, 123, MPI_COMM_WORLD, &status);
41
42     printf(" Process %d received %d\n",myid,buffer2.max_iter);
43
44     MPI_Finalize();
45     return 0;
46 }
```

2. Use MPI to parallelize the following serial code to compute the dot product of two vectors. The parallel code should preserve correctness regardless of the number of processes used. The correct answer is provided in the output. Some MPI commands have already been added for your convenience.

```c
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <math.h>

int main(int argc,char **argv) {
  MPI_Init(&argc,&argv);
  int rank,p,i, root = 0;
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&p);

  // Make the local vector size constant
  int global_vector_size = 10000;

  double pi = 4.0*atan(1.0);

  // initialize the vectors
  double *a, *b;
  a = (double *) malloc(
       global_vector_size*sizeof(double));
  b = (double *) malloc(
       global_vector_size*sizeof(double));
  for (i=0;i<global_vector_size;i++) {
    a[i] = sqrt(i);
    b[i] = sqrt(i);
  }

  // compute the dot product
  double sum = 0.0;
  for (i=0;i<global_vector_size;i++) {
    sum += a[i]*b[i];
  }

  if ( rank == root ) {
    printf("The dot product is %g.  Answer should be: %g\n",
           sum,0.5*global_vector_size*(global_vector_size-1));
  }

  free(a);
  free(b);
  MPI_Finalize();
  return 0;
}
```

3. Using the MPI Mandelbrot code from Lecture 10, produce a weak-scaling plot and strong scaling plot from 2^0 to 2^6 processes (staying in powers of two). For the strong scaling plot, use a problem size of 32768 x 32768 pixels (if it does not fit in memory then lower it to 16384 x 16384). For the weak-scaling plot, specify in your plot the global problem size for each point in the graph. You may pick the local problem size for the weak scaling problem. Run performance analysis on this code using the perf utility (see Lecture 3 demos) and explain the results. Use BigRed 2 or any other cluster.

4. Describe the three principal abstract components of the CSP model. You may wish to draw a diagram or describe an example application.