Computing Lab #9: Coordination between many processes with MPI

Due: Tuesday, November 28, 2017 (midnight)

Programming language: Write, compile, and run a Fortran, C or C++ program. Your choice! This lab assumes the C programming language.

Points: 10

In this lab you will continue working on the MPI optimization program you started on Tuesday. See Lab 8 for the code you should already have as well as the problem description.

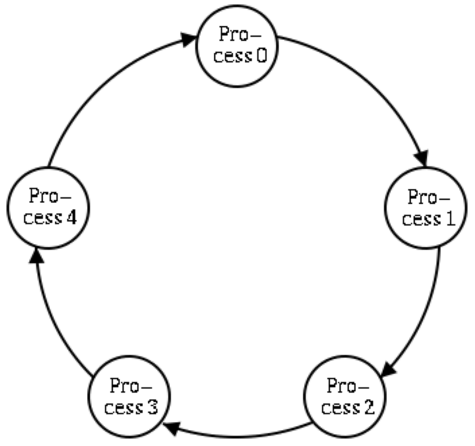
I. TASK #1: PASSING A MESSAGE AROUND THE RING (5 POINTS)

In this lab you will continue working on your MPI optimization code. Currently (from lab 8), each process computes its own maximum value of the function (along with the corresponding (x_1, x_2) value). Here, you will implement a strategy for all processes to share their own maximum. The strategy will be for the maximum to pass around information in a ring:

1. Process 0 pass its maximum value, lets call it M_0, to process 1.

2. Process 1 compares its own maximum value, lets call it M_1, to the value it has received from process 0. The larger of the two values (M_0 and M_1) is passed to process 2.

3. ...and this continues until the last process receives its value from the next-to-last process. A final comparison is made and the value is printed to screen. This way of passing information is known as a "ring topology" (see the figure).



Implement this ring-like message passing strategy using the functions MPI_Send and MPI_Recv.

1. Below is a snippet of code (this snippet shows the most important piece, but it doesn't yet form a working program) which is exactly the ring passing structure I described in class.

```
double send_number = 9;
double rec_number;
MPI_Status status;


// This next if-statement implements the ring topology
// the last process ID is size-1, so the ring topology is: 0->1, 1->2, ... size-1->0
// rank 0 starts the chain of events by passing to rank 1
if(rank==0) {

  // only the process with rank ID = 0 will be in this block of code.

  //  send data to process 1
  MPI_Send(&send_number, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);

  // receive data from process size-1
  MPI_Recv(&rec_number, 1, MPI_DOUBLE, size-1, 0, MPI_COMM_WORLD, &status);

}
else if( rank == size-1) {

  // receive data from process rank-1 (it "left" neighbor")
  MPI_Recv(&rec_number, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &status);

  // send data to its "right neighbor", rank 0
  MPI_Send(&send_number, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}
else {

  // receive data from process rank-1 (it "left" neighbor")
  MPI_Recv(&rec_number, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &status);

  // send data to its "right neighbor" (rank+1)
  MPI_Send(&send_number, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
}
printf("Process %i send %1.3e and recieved %1.3e\n", rank, send_junk, rec_junk );
```

2. Now integrate this ring structure code into the optimization code you wrote for last week's lab. Your code should perform the ring passing strategy described at the start of this lab.

3. Compile and run the code (use the same mpicc and mpirun commands you used last week) – and confirm that the final result printed to screen is the global maximum. Provide evidence that your code is working by having each process report its maximum. Run with 4 processes.

II. TASK #2: REDUCTION (5 POINTS)

Now lets use the MPI reduction function to find the maximum.

1. Comment (but do not delete) the entire block of code which implements the ring passing.

2. In its place, put the MPI reduction function we discussed in class

```
double x_process_max = // this is the maximum found by the process
double x_max=-100; // processor 0 will hold the global maximum
MPI_Reduce(&x,&x_max,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
printf("Process ID %i, x = %f, max x = %f\n",rank, x_process_max, x_max);
```

3. Compile and run the code. Confirm that the final result printed to screen is the global maximum. Provide evidence that your code is working by having each process report its maximum.

... the great news about reductions is that they are not only easier but more efficient!

III. UPLOAD YOUR WORK TO GIT

Congratulations! You've completed the lab. Now lets upload your work to git.

Follow the same series of steps as you've done for the last few labs. Place all work for this lab (Just upload the code from task 2 which includes both the reduction and the ring. Provide output from your program to show that it works) into a folder called "Lab9" and push it to bitbucket.