MPI LAB 2

Project 1

A Token Ring

A "token ring" is a circular messaging system. Process 0 sends a "token" to Process 1, which sends it on to Process 2, and so forth. Write a program that implements this with MPI. Hint: separate the ranks into root and everybody else.

Project 2

A Relay

Write a program similar to the token ring, but in which process 0 sends a "baton". Use the special sender MPI_ANY_SOURCE (MPI.ANY_SOURCE in mpi4py) and a tag in the receiver of MPI_ANY_TAG (MPI.ANY_TAG).

Project 3

A Boundary-Value Problem

We will solve the heated-plate problem in parallel. In this case you will be provided with the serial code. Copy heatedplate.c, heatedplate.cxx, heatedplate.f90, or heatedplate.py as appropriate from /share/resources/tutorials/mpi on Rivanna . Be sure you are able to run it in serial before you attempt to parallelize it. In serial use a small value of epsilon such as 0.01 to obtain convergence in a reasonable time. Use a global grid of 500x500.

In parallel, use an epsilon of 1.e-6 and a local grid of size 500x500, excluding ghost zones. Do not use a global grid in any of your processes.

Run your program for 5 cores, 10 cores, and 20 cores on the parallel partition. Use the version of MPI_Wtime for your language to get the timing information (not the built-in clock or cpu_time). Plot the time for the run verses the number of cores. What kind of scaling is this?

C/C++

```
double starttime, endtime;

starttime=MPI_Wtime();

  Stuff

endtime=MPI_Wtime();

printf("Elapsed time %f\n",endtime-starttime);
```

Fortran:

```
double precision :: starttime, endtime;

starttime=MPI_Wtime()

  Stuff

endtime=MPI_Wtime()

print *, “Elapsed time”, endtime-starttime);
```

Python:

```
t_start=MPI.Wtime()

t_end=MPI.Wtime()

print “Elapsed time”,t_end-t_start
```

Plotting the results: You will need to devise a way to plot your solution. The serial code dumps out rows. You can make a contour plot of these data by any means you have available, including Excel. I have provided a sample contour.py in /share/resources/tutorials/mpi for those who know a little Python.

To extend this to parallel, I recommend generating a file name that includes the processor ID (converted to a string) concatenated on to the base name. Fortran and Python students have seen something similar to this already. C students, use strcat(str1,str2). C++ use string concatenation (+). Note that the serial code writes one or more dimensions at the top of the output file. You may or may not need these values depending on your plotting program, and if you do need them, only process 0 should write them and they should be for the *total* size of the grid. You can use cat or a similar command alone or in a short script to stitch your files together. Be sure to use >> to append rather than to overwrite.