CMT106: High Performance Computing

Lab 2: Programming with MPI

Where: C/2.08, Queen's Buildings (second floor, central building)
When: 1:30-3:15pm, 19 November

Description of problem

You will be given a partial MPI program, laplaceMPI_partial.c. You must insert the 4 sections of missing code to produce a code named laplaceMPI.c which solves the two-dimensional Laplace equation on a rectangle. The aim is to determine the electric field around a conducting object held at a fixed electrical potential inside a box at a fixed electrical potential of 1, with the electrical potential on the boundary being fixed at 0.

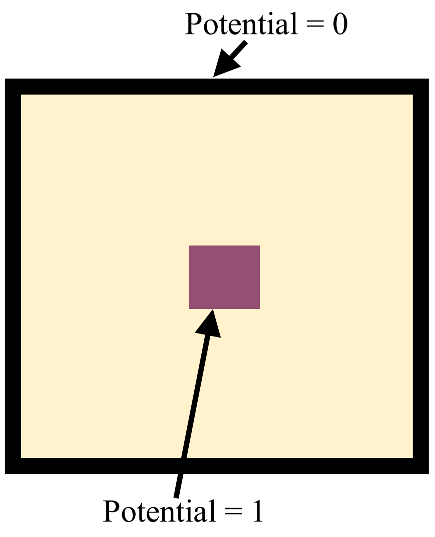

Figure 1: The conducting object is shown in purple. We want to find the electrical potential in the yellow shaded region.

The code uses an iterative algorithm to find the approximate solution at a rectangular grid of points within the yellow region in Fig. 1. The main computational work is done in the following lines of code:

```
for(k=1;k<=nsteps;k++){
    for(j=0;j<nptsy;j++)
      for(i=0;i<nptsx;i++)
        oldphi[j][i] = phi[j][i];
    for(j=0;j<nptsy;j++)
      for(i=0;i<nptsx;i++)
```

The outer loop over k is the iterative loop. This contains two doubly-nested for loops. The first copies the solution from the previous iteration (phi) to the oldphi array. The second doubly-nested loop updates the solution by replacing phi at each point by the average of the neighbouring values from the previous iteration. The mask array is set to 0 on the boundary and at the 4 central points (we don't want to change these values of phi), and is 1 at all other points.

What you must do

1. Follow the instructions on how to initialise the environment for running MPI given in the document *How To Run MPI Programs Using The Linux Lab Machines*, and check that you can successfully compile and run the code *hello.c*.
2. Insert MPI calls at 4 locations in the code, laplaceMPI_partial.c, to create a working MPI code named laplaceMPI.c. The 4 locations are clearly marked in the partial code, and deal with setting up a 2D communicator, and performing communication between neighbouring processes.

Optional Tasks

You probably won't have time for the following tasks in the lab:

1. After the code is working correctly you should run it for following numbers of processes: 1, 2, 4, 6, 8, 12, and 16 (see instructions below). You should use a problem size of 480x480 and 10000 iterations.
2. Plot a histogram of the time taken against the number of processes. Include a straight line to represent the time taken by the sequential code, laplaceSEQ.c. Include the problem size and number of iterations on the plot (these are the values nx, ny, and ns mentioned below).

What to do if you need help

If you need help, ask me or the lab assistant. If I'm not in the lab you can email me at WalkerDW@cardiff.ac.uk. You can find manual pages for MPI functions at https://www.mpich.org/static/docs/v3.2/www3/

How to compile and run the code

To compile your MPI code:

```
mpicc -o laplaceMPI laplaceMPI.c -lm
```

To run your MPI code on 8 processes:

```
mpiexec -n 8 laplaceMPI -nx 1000 -ny 1000 -ns 2000
```

This will run the parallel code on a 1000x1000 grid for 2000 iterations on the machine you are using. You can change the number 8 to any other valid integer to change the number of processes. To run across multiple machines in the Linux Lab do:

```
mpiexec -n 8 -hostfile machines.txt laplaceMPI -nx 1000 -ny 1000 -ns 2000
```

where the file machines.txt contains the names of the machines, as discussed in the lectures.

The sequential and MPI codes should give exactly the same output files. You can check this by doing:

```
diff outSEQ.ps outMPI.ps
```

If this produces no output then outSEQ.ps and outOMP.ps are the same.

You can visualise the output using any PostScript viewing tool. For example:

```
gs outMPI.ps
```