Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

MSc. in HPC

HPC Software II (55612)

Programming Exercises 1

Darach Golden

March 3, 2025

1 Notes

- **To complete these exercises, submit a document containing results for each question and a separate tarball of the all code used to obtain the results. Details of how to compile and run the code must be provided in a "README" file. You *must* use a makefile to compile the code. The code should compile and run on `seagull`**
- **Use C to implement all exercises**

2 Exercises

These exercises use as a starting point the parallel implementation of a finite difference 2D Poisson solver that was partly written in class. A 1D **processor** decomposition of the finite difference grid was used in this parallelisation. The implementation developed in class will be referred to below as the "1D Poisson"

1. [**7%**] Change the 1D Poisson implementation to use `MPI_Cart_create()` and `MPI_Cart_shift` to obtain the neighbours of each process instead of explicitly using `myid-1` and `myid+1`
2. [**10%**] Use the finite difference approximation along with a 1D processor decomposition to find an approximate solution to the following Poisson equation on a grid size of 15 and on a grid of size 31.

   Let Ω be the region $[0,1] \times [0,1]$.

   $$\nabla^2 u(x,y) = \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y)\,, \quad 0 \leq x, y \leq 1\,.$$

   with $u(x,y) = \phi(x,y)$ on $\partial\Omega$ (Dirichlet boundary conditions).

   Let $f \equiv 0$. Let the Dirichlet boundary conditions be: $u(x,0) \equiv 0$, $u(x,1) = \frac{1}{(1+x)^2+1}$ $(0 \leq x \leq 1)$, and $u(0,y) = \frac{y}{1+y^2}$, $u(1,y) = \frac{y}{4+y^2}$ $(0 \leq y \leq 1)$.

   The solution is:

   $$u(x,y) = \frac{y}{(1+x)^2 + y^2}\,, \quad 0 \leq x, y \leq 1\,.$$

Run the code on at least 4 processors.

For both grid sizes, demonstrate how closely the finite difference solution matches the analytic solution after at most 2000 iterations of the Jacobi method.

3. [**18%**] Update the 1D Poisson implementation to include

   - A `write_grid()` function which allows each processor to write its copy of the grid to file or `stdout`. The output should be in "mesh/grid" format
   - A `GatherGrid()` function which gathers the parallel solution which is distributed across multiple processors onto one processor (rank 0)
   - Demonstrate the `GatherGrid()` function by running the code in parallel on 4 processors for a square grid of size 31 for the poisson equation example given in Q2 above. Use `GatherGrid()` to gather the full solution onto rank 0 and write this gathered global solution to file from rank 0 using `write_grid()`. Then show that the solution you have just written to file closely matches the solution of the poisson equation in Q2.
   - Create a "heatmap" plot of the global solution written to file from one processor

4. [**65%**] Write of version of the Poisson Solver which uses a *2D* processor decomposition of the grid rather than a 1D decomposition. Show examples of this code running on at least 4 and 16 processors for a square grid of size 31.

   - Use MPI Cartesian topology functions such as `MPI_Cart_create()` and `MPI_Cart_shift` to calculate the up, down, left and right neighbours of each process
   - implement two versions of the 2D ghost exchange routine:
     - a version which uses `sendrecv()`
     - a version which uses non-blocking sends and receives
   - When using a 2D processor decomposition, you will need to exchange ghost data in two directions. In one of those directions,

ghost data will be contiguous as in the 1D case. However in the other direction the data will not be contiguous. In this case use a `MPI_Type_vector` call to create a *strided* MPI datatype which matches the non-contiguous data to be exchanged

- Write a `GatherGrid2D()` function which gathers the whole grid onto rank 0. Demonstrate this function using at least 4 processors.

  For grid sizes of 15 and 31, show that the gathered global solution obtained using a 2D process decomposition closely matches the solution for the example poisson equation given in Q2 above.

Some useful utilities you may wish to use are:

- `MPI_Dims_create()`
- `MPI_Cart_get()`
- `MPI_Cart_coords()`