MPI Programming Assignment– Solving coupled PDEs

Theory and discretisation

The governing equation – The aim of this assignment is to write a parallel solver for a pair of coupled PDEs. These PDEs represent the reaction and diffusive transport of two chemical species, with concentrations C_1 and C_2:

$$\frac{\partial C_1}{\partial t} = \left(C_1(1 - C_1) - fC_2\frac{C_1 - q}{C_1 + q}\right)/\epsilon + D_1\nabla^2 C_1$$

$$\frac{\partial C_2}{\partial t} = C_1 - C_2 + D_1\nabla^2 C_1$$

Where f, q and ϵ are reaction parameters and D_1 and D_2 are the diffusivities of the two chemical species. More information on these equations can be found in papers by Janke *et al.*, 1988 (https://pubs.acs.org/doi/epdf/10.1021/j100339a047) and Mahanta *et al.*, 2018 (https://journals.aps.org/pre/pdf/10.1103/PhysRevE.97.022206).

Using a FTCS scheme, we can discretise these equations as follows:

$$C_{1_{i,j,n+1}} = C_{1_{i,j,n}} + \Delta t\left(\left(C_{1_{i,j,n}}\left(1 - C_{1_{i,j,n}}\right) - fC_{2_{i,j,n}}\frac{C_{1_{i,j,n}} - q}{C_{1_{i,j,n}} + q}\right)/\epsilon + D_1\left(\frac{C_{1_{i+1,j,n}} + C_{1_{i-1,j,n}} - 2C_{1_{i,j,n}}}{\Delta x^2} + \frac{C_{1_{i,j+1,n}} + C_{1_{i,j+1,n}} - 2C_{1_{i,j,n}}}{\Delta y^2}\right)\right)$$

$$C_{2_{i,j,n+1}} = C_{2_{i,j,n}} + \Delta t\left(C_{1_{i,j,n}} - C_{2_{i,j,n}} + D_2\left(\frac{C_{2_{i+1,j,n}} + C_{2_{i-1,j,n}} - 2C_{2_{i,j,n}}}{\Delta x^2} + \frac{C_{2_{i,j+1,n}} + C_{2_{i,j+1,n}} - 2C_{2_{i,j,n}}}{\Delta y^2}\right)\right)$$

Where:

$$x = x_0 + i\Delta x$$

$$y = y_0 + i\Delta y$$

$$t = t_0 + n\Delta t$$

We will use parameters from the second of these papers. The parameters that we will be using are:

$$f = 2.0$$

$$q = 0.002$$

$$\epsilon = 0.03$$

$$D_1 = 1.0$$

$$D_2 = 0.6$$

The initial conditions are those from the first paper, with the solution being implemented on a 2D domain of dimensions 30 by 30. The initial condition is given in polar coordinates with the origin in the centre of the domain:

$$C_1 = \begin{cases} 0.8 & \text{if } 0 < \theta < 0.5 \\ q\dfrac{f+1}{f-1} & \text{elsewhere} \end{cases}$$

$$C_2 = q\frac{f+1}{f-1} + \frac{\theta}{8\pi f}$$

Where θ is the polar angle in radians.

Boundary Conditions

We will try two different boundary conditions.

For the first boundary conditions we will use no flux at the domain edges, which implies a zero concentration gradient in the normal direction:

$$\nabla C_1 \cdot \hat{\mathbf{n}} = 0$$

$$\nabla C_2 \cdot \hat{\mathbf{n}} = 0$$

Where $\hat{\mathbf{n}}$ is the unit normal to the boundary.

For the second boundary condition, we will assume that the domain is periodic, which means that values wrap off one side and onto the other. As we will be assuming that the system is periodic in both directions, this is equivalent to solving the problem on a toroidal surface.

MPI based implementation

You must use domain decomposition in which each processor only allocates enough memory to store its portion of the domain (together with any padding required to receive the data from neighbouring processes). The easiest way to divide the domain is into vertical or horizontal strips, but this can become inefficient quite rapidly as the number of processors grows as it will result in very long thin domains with a large number of boundary elements relative to their area. It is far more efficient to divide the simulation into rectangular domains that are as close to square as possible. The communication pattern involved is virtually identical to that implemented in Exercise 3 of worksheet 2 and will involve transferring a layer of grid cells just inside the edge of the domain from the neighbouring processors and receiving a layer of grid cells situated just outside the domain back from the neighbours. Note that you do not need to transfer data into the corners of the ghost layer around the edge of the domain as the finite difference stencil is a cross.

Your communications will take the form of peer-to-peer communications with no master process, with each process communicating only with its vertical and horizontal neighbours.

I want you to be able to run the simulation with either fixed or periodic edges to the domain. With periodic edges you need to maintain communications across the periodic boundary. Watch out for the situation where the matching periodic boundary is on the same processes (this can easily occur when using only a few processes). Note that even with periodic edges you can still have internal obstacle with boundaries.

The code must be able to use an overall domain of arbitrary size so that you can test your simulator's efficiency as the domain size is changed. You should also ideally be able to change the

height and width of the domain independently of one another (how might this impact the best decomposition?).

You probably don't want to do an output every time step as the time steps are quite small, but rather at a user specified interval (ideally in terms of time rather than time steps). At these times you should write output to file. The most efficient will be for each process to write their own files. You should therefore write a separate program or script which collates these files. This can be done in Python in order to be able to easily generate graphical outputs.

Extensions and Additions

There is 20% of the marks associated with extensions, additional implementations and optimisations of the code. Note that you will get more credit for original solutions and additions compared to optimisations that a large number of people have done.

In addition to documented source code and instructions for compiling and running the code, you should also submit documentation that describes any optimisations that you have carried out and, potentially, how you investigated the code efficiency to optimise it.

Examples of additional implementations and extensions

- ***Doing calculations while waiting for communications to complete – You will need to watch out to make sure that you are not doing calculations that require the data from the neighbouring processors while waiting***
- ***Internal boundaries and obstacles***
- ***You can also create a SEPARATE/EXTRA version using one-sided communications – The main version associated with most of the marks will be the version using Non-blocking point to point communications. Therefore only try and do the one-sided communications version if the other version is complete, working well, documented etc and you have spare time.***

Management of code development

With this coursework there is also a specific mark for the management of the code development. At the very least we would expect regular commits to GitHub (once per day at the very minimum). We would also expect comments in the commits to explain what had been added to the code, bugs fixed, etc. since the previous commit.

What we definitely don't want to see is a single commit at the end of the coursework or where there is a sudden wholesale change in the code with no evidence of code development and progression.

The due date for the assignment is 5pm on 17th May 2024

Submission

What is required in the GitHub repository

- Documented source code
    - Both simulation and post-processing code
    - A GitHub history that demonstrates the development path for the code
- Documentation for how to compile and run the code
- Documentation describing code improvements and optimisations
- Videos/animations of the simulation output

Mark Scheme

- Code commenting and documentation - 10%
- Code implementation, structure and efficiency - 40%
- Postprocessing code - 10%
- Results - 10%
- Extensions, additional implementations and code optimisations - 20%
- Management of code development - 10%

Note that this is an individual coursework and you MUST NOT copy code from anyone else. This is a complex enough problem that we will be able to identify copied code (and changing a few variable names won't disguise the copying). In the case of collusion both the person doing the copying and the person allowing their code to be copied is liable to lose all or most of their marks for the assignment.

What is required in the code:

In this assignment I want you to demonstrate your MPI programming skills. The code MUST therefore make use of the following techniques:
Required:

- ***Non-blocking point to point communications***
- ***Calculate your own decomposition – there are functions available in MPI to do this for you for simple grids (e.g. MPI_Dims_create and MPI_Cart_create), but DON'T use these***
- ***Creating your own MPI variable types***
    - ***You MUST use MPI_Type_create_struct - There are other easier ways to make MPI data types for very structured data like this (e.g. MPI_Type_vector), but I want you to demonstrate that you know how to do generic MPI datatype creation***