MPI Programming Assignment– Solving the Wave Equation

Theory and discretisation

The governing equation - The aim of this assignment is to write a parallel solver for the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

Where u is the displacement and c is the speed of the wave. A simple way of discretising this problem is an explicit finite difference discretisation. For two spatial :

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

Note that the superscript n refers to the time step and is not a power!

The domain is a grid defined by:

$$t_n = t_0 + n\,\Delta t \qquad x_i = x_0 + i\,\Delta x \qquad y_j = y_0 + j\,\Delta y$$

Where n, i and j are integers.

The discretised equation is explicit and can be rearranged as follows:

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

Note that you don't need to store all of the time steps in memory (and you are likely to run out of memory quite rapidly if you try from all but the smallest problems). You will need to store 3 grids, one for the previous time step ($n-1$), one for the current time step (n) and the new time step ($n+1$). When moving to the next time step try and avoiding copying the data and rather just swap pointers (MUCH quicker).

As this is an explicit solution there are strict time step constraints:

$$\Delta t \ll \frac{min(\Delta x, \Delta y)}{c}$$

Boundary conditions - There are two types of boundary conditions that you can use if you want to put in edges or obstacles in your domain. The easiest is to have u fixed with a value of zero on boundaries (a Dirichlet boundary):

$$u = 0$$

This is equivalent to simulating a sheet attached at its edges (e.g. a drum skin). This type of boundary will reflect waves, but will cause them to become inverted.

The other type of boundary that you could impose is to have a zero gradient in u in the direction normal to the boundary. If $\boldsymbol{n}$ is a unit vector normal to the boundary:

$$\nabla u \cdot \boldsymbol{n} = 0$$

On a square grid this is equivalent to setting the value on the boundary equal to the neighbouring point just away from the boundary. This boundary condition will also reflect waves, but not cause

them to invert. This is more like the behaviour of small water waves hitting a vertical wall. Ideally you should implement both boundary types and give the user the option of which one to use.

Initial conditions – In order to generate a wave you need to have disturbance. The easiest disturbance will be to simply displace a single point, but such a disturbance will not give very pretty waves. Maybe have a radially sinusoidal disturbance over either one or half a period. What about more than one disturbance to see how they interact?

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. Far more efficient is 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.

This 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 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 may be 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.

In addition to documented source code you should also submit a short report which includes an analysis of the performance of the code. I wish to see how the Efficiency/Speedup changes with the size of the problem and the number of cores used. You will need to use the HPC system if you are to test this on a suitable number of cores to see good trends. How does the efficiency behaviour compare to you might have expected?

The due date for the assignment is 5pm on 23 April 2021

Submission

What is required in the GitHub repository

- Documented source code
    - Both simulation and post-processing code
- Short report including code documentation and results
    - The report has a 1000 word limit, but can be accompanied by figures and animations etc.
- Output log files from your HPC runs (*.o* files)

Mark Scheme

- Code documentation including commenting - 20%
- Code implementation, structure and efficiency - 40%
- Postprocessing and results - 10%
- Analysis and discussion of the program's performance - 30%

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 MPI techniques:

- ***Non-blocking point to point communications***
- ***Creating your own MPI variable types***

Optional

- ***Collective communications – E.g. if options are loaded from a file this could be done on one core and then distributed to the other cores***
- ***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***

- ***You can also create a SEPARATE/EXTRA version using one-sided communications – The main version associated with virtually all of the marks will be the version using Non-blocking point to point communications (there will be the possibility of some extra marks for this version, but very few). Therefore only try and do the one-sided communications version if the other version is complete, working well, profiled, documented etc and you have spare time. This would be more for your own interest than it would be for the extra marks!***