Andrea Pozzetti acse-ap2920

ACSE6-2 Assignment

Parallelising a wave equation solver with MPI

Introduction

The aim of this assignment was to parallelize a solver for the wave equation:

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

The equation was discretised using the following scheme:

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

After rearranging, we get the following stencil for updating our u value:

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

To compute our new u we need its value from the two previous time levels, which means we need to store 3 separate grids for our discretization.

MPI was used to run the process on separate cores and communicate between them.

Andrea Pozzetti acse-ap2920

Approach

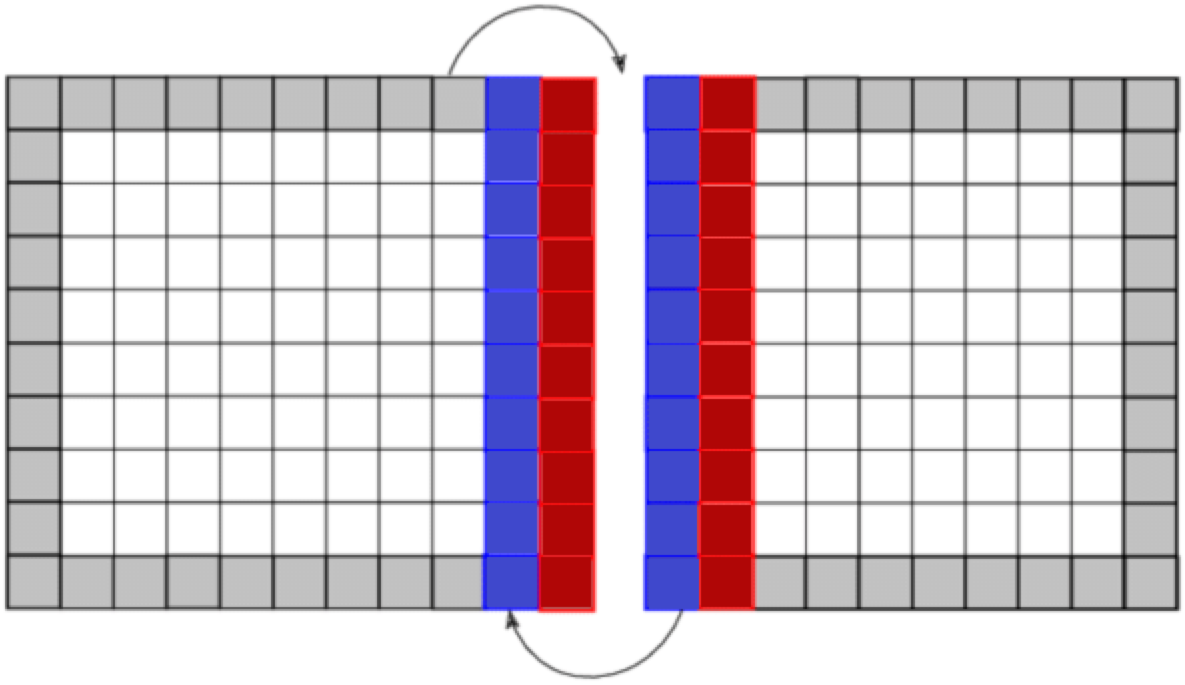
The whole domain is split into different rectangles based on the number of cores used.
For 6 cores for example, the whole domain is split into 6 rectangles.

All cores are used, which means that for prime numbers the subdivision is quite weird (for 7, 7 “rows” or 7 “columns”).

Each rectangle communicates with its four neighbours (unless it is on the boundary with non-parallel BCs)

Each process on each core stores an extra layer of “ghost” cells on each side which represents its neighbour’s boundary.

Each iteration the ghost cells are updated with the current (updated) values of the neighbour’s boundaries.



Communications are expensive, and from a purely theoretical perspective the more calculations happen per communication, the closer we get to achieving “perfect” speedup (proportional to number of cores used).

Much like the shape that maximises the area (calculations) for perimeter (communications) is the circle, the “optimal” shape for our subdomains is a square.

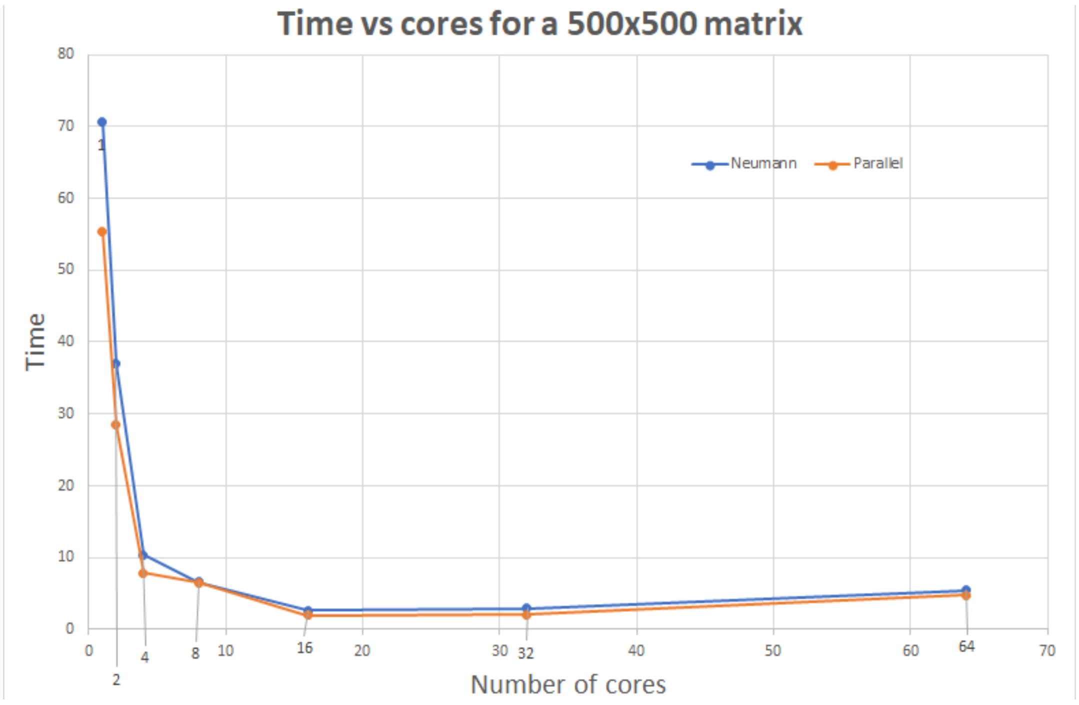
To subdivide our domain, I have therefore seeked to minimise the difference between the subdomain rows and columns to obtain the most “square” rectangles possible.

Andrea Pozzetti acse-ap2920

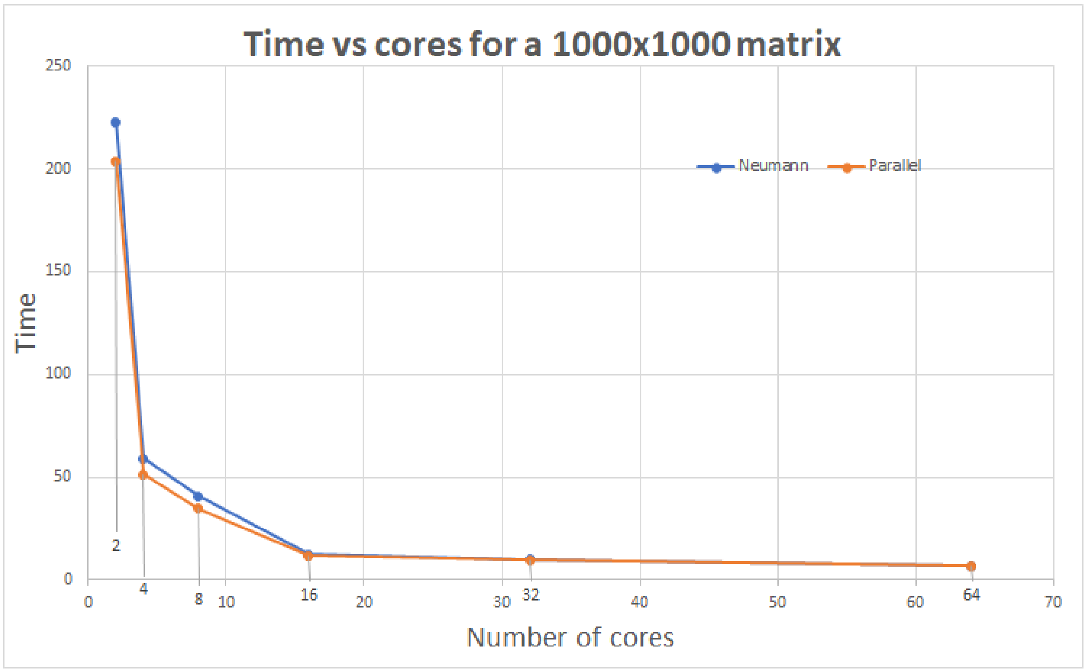
I have also introduced the possibility of introducing sinusoidal sources of a definite period and intensity for a certain amount of time.

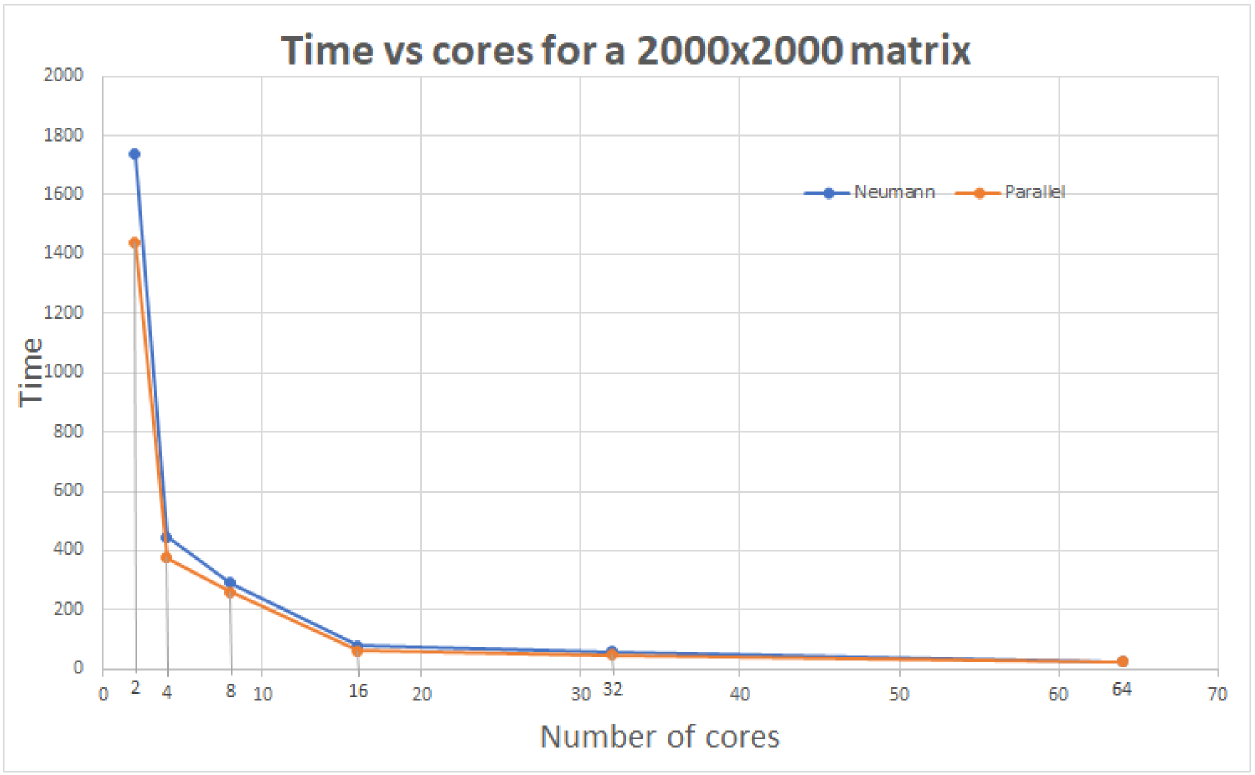
Dirichlet and Neumann boundary conditions can also be specified instead of parallel.

Performance analysis
These are graphs with my timings for 500x200, 5000x1000 and 2000x2000 grids.
They all had the same input source.



Andrea Pozzetti acse-ap2920





I have chosen to exclude the 1 core data point from the 1000x1000 and 2000x2000 matrices to help see the tail.

We can see that we are nowhere the perfect efficiency that we were hoping for and that drops in time taken are less than linearly correlated with the increase in cores for the most part.

That was to be expected since communications between cores take a significant amount of time.

We can also notice a few trends in our data.

- **Parallel BCs are slightly faster to compute for than Neumann BCs, at least for low core numbers**
  Applying boundaries conditions might be slower than just communicating them. Which is interesting. I wonder about the "inner" communications.

- **Speedup is better for bigger matrices.**
  This is possibly the easiest to explain: for smaller matrices the amount of time taken up by communications is bigger in comparison to the amount of time taken by calculations. Adding cores increases the number of communications needed and communication time while decreasing a proportionally smaller calculation time.
  We can actually see the performance worsening for the 500x500 matrix once we go over 16 cores.

- **Speedup is better for perfect squares**.
  As I mentioned in the Approach section, the optimal shape (less communications per computation) of a subdomain is a square. Using a non-square number of cores for a square grid leads to rectangular subdomains.
  There is less change proportionally from 1 to 2 than there is from 2 to 4, and there is very little change from 16 to 32, which astonished me the first time I saw it.
  Local runs for a 300x100 grid showed 3 cores (leading to 3 square 100x100 subdomains) perform better than 4 (leading to 4 150x50 subdomains).
  I wonder how easy it would be to implement a tool that searches for possible configurations with a little less cores (example: using a 5x6 grid and leaving 2 processors idle might actually be even better than using a 4x8 grid)

Room for future improvement
Maybe operations could be done while subdomain boundaries are sending but I find it hard to think of how with such a small amount of ghost cells. Perhaps I could do "inner communications" (when a core is its own neighbour) while everything else is communicating, but otherwise I find it hard to visualize.

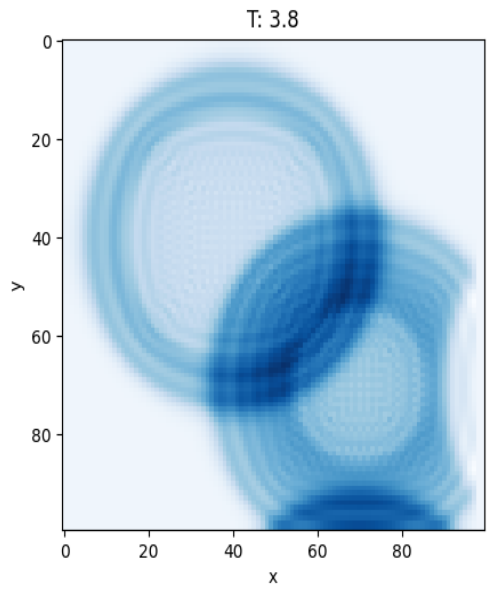
Perhaps we could have a 2 cell thick layer of ghost cells and do 2 iterations before communicating them all. In that case we would need to communicate multiple grids and not only the new one.

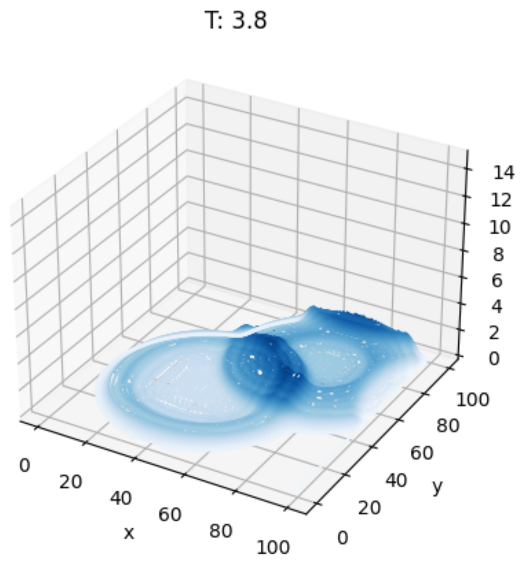
Another thing I would like to improve is the post processing because it is painfully slow.
I had started to try and parallelise the printing process but it's much harder on Python and I gave up quite soon.

Andrea Pozzetti acse-ap2920

Post processing

Post processing is done with python scripts. For animations, please check github repo.


T: 3.8
y
x
0
20
40
60
80


T: 3.8
x
y
0
20
40
60
80
100
14
12
10
8
6
4
2