AE3-422 High-performance Computing

Coursework Assignment

Deadline: 20th March 2019 - 17:00

Instructions

Please take note of the following when completing this assignment:

- Read all the tasks carefully and plan ahead before you start designing and implementing your code.
- You may use any of the tools and libraries available on the provided Linux environment.
- Your submitted code **must** compile and run correctly on the provided Linux environment.

Make regular backups of your code onto a separate computer system; no allowance will be made for data loss resulting from human or computer error.

1 Introduction

The objective of this coursework is to write a parallel numerical code for finding the solution to the generalised Burgers' Equation in 2D using the finite difference method.

The strong form for Burgers' Equation is given by:

$$\frac{\partial \mathbf{u}}{\partial t} + ((\mathbf{a} + b\mathbf{u}) \cdot \nabla)\mathbf{u} = c\nabla^2\mathbf{u}, \tag{1}$$

where $\mathbf{u} = (u, v)$ and $\mathbf{a} = (a_x, a_y)$, b and c are constants. This is subject to initial and boundary conditions,

$$\mathbf{u}(x, y, 0) = \mathbf{f}(x, y), \qquad \text{on} \quad \Omega \tag{2}$$

$$\mathbf{u}(x, y, t) = \mathbf{g}(x, y), \qquad \text{on} \quad \partial\Omega \tag{3}$$

Equation 1 can be expanded as

$$\begin{aligned} \frac{\partial u}{\partial t} + (a_x + bu)\frac{\partial u}{\partial x} + (a_y + bv)\frac{\partial u}{\partial y} &= c\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\ \frac{\partial v}{\partial t} + (a_x + bu)\frac{\partial v}{\partial x} + (a_y + bv)\frac{\partial v}{\partial y} &= c\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \end{aligned} \tag{4}$$

Burgers' equation is obtained as a result of combining non-linear wave motion with linear diffusion and is the simplest model for analysing the combined effect of non-linear advection and diffusion. This equation has been found to describe various phenomena such as turbulence and approximate theory of flow through a shock wave travelling in a viscous fluid.

2 Finite Difference Method Discretisation

In order to solve Equation 1, space and time need to be discretized. For this assignment we will use an explicit (forward) time-integration scheme when discretising the time derivative.

Explicit integration scheme:

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

$$\frac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} + (a_x + bu_{i,j}^{n})\frac{v_{i,j}^{n} - v_{i-1,j}^{n}}{\Delta x} + (a_y + bv_{i,j}^{n})\frac{v_{i,j}^{n} - v_{i,j-1}^{n}}{\Delta y} = c\left(\frac{v_{i+1,j}^{n} - 2v_{i,j}^{n} + v_{i-1,j}^{n}}{\Delta x^2} + \frac{v_{i,j+1}^{n} - 2v_{i,j}^{n} + v_{i,j-1}^{n}}{\Delta y^2}\right) \quad (5)$$

In this assignment the domain is defined as:

$$\Omega = \{(x, y) : -L/2 \leq x \leq L/2, -L/2 \leq y \leq L/2\} \quad (6)$$

The initial conditions at $t = 0$ are:

$$f_u(x, y) = f_v(x, y) = f(r) := \begin{cases} 2(1 - r)^4(4r + 1) & r \leq 1, \\ 0 & r > 1, \end{cases} \quad (7)$$

where $r = \sqrt{x^2 + y^2}$, and the boundary conditions are:

$$\mathbf{g}(x, y, t) = \mathbf{0}, \qquad \text{on} \quad \partial\Omega, \quad t > 0 \quad (8)$$

3 Test cases

Your code should be tested using the following four test cases:

- An unsteady diffusion problem ($a_x = 0$, $a_y = 0$, $b = 0$, $c = 1$);
- A linear advection problem in x ($a_x = 1$, $a_y = 0$, $b = 0$, $c = 0$);
- A linear advection problem in y ($a_x = 0$, $a_y = 1$, $b = 0$, $c = 0$);
- A generalised Burgers problem ($a_x = 1.0$, $a_y = 0.5$, $b = 1.0$, $c = 0.02$).

Choose $L = 10$ and a final time of $T = 1$ for the four test cases above.

Other parameters should be chosen appropriately.

Continues on next page ...

Tasks

The objective of this coursework is to write a high-performance parallel C++ code which will solve the generalised Burgers' equation described above. A template code is provided to get you started and help you structure your code. Download this code and edit it to complete the following tasks.

1. Extend the template code provided to solve the generalised Burgers' equation, initially in serial:
   (a) Implement the `Model` class, declared in `Model.h`, to parse and store all the parameters as required. Parameters provided by the user should be read from the command-line, and validated where appropriate. [5%]
   (b) Write a new class called `Burgers` which has public member functions which: [20%]
   - Set the initial velocity field at time $t = 0$;
   - Time integrate the velocity field from $t = 0$ to $t = T$ (the final time);
   - Write the velocity field to a file.
   - Calculate the energy of the velocity field, given by

$$E = \frac{1}{2} \int_{\Omega} |\mathbf{u}|^2 \, d\Omega$$

   (c) Complete the `main` function provided to use the classes and verify using the first three test cases. [5%]
2. Create a Makefile to **build and run** your C++ code. [5%]
   (a) Create a target `compile` which compiles your code.
   (b) Create targets `diff`, `advx`, `advy` and `burg` which execute your compiled code for the four *Test Cases*.
   (c) Update your makefile to add a `clean` target which removes files generated during compilation.
   (d) Define appropriate `default` and `all` target rules in your makefile.
3. Parallelise your C++ program with MPI using two processes. [15%]
   (a) Partition the domain into two halves to solve Equation 1 in parallel.
   (b) Verify your parallel code using the first three test cases.
   (c) Update your makefile and add the targets `diffp`, `advxp`, `advyp` and `burgp` to run the code in parallel on two processes using the parameters for each of the test cases.
4. Parallelise your C++ program with MPI using a larger number of processes $P = P_x \times P_y$, where $P_x \geq 1$ and $P_y \geq 1$ are provided by the user on the command-line. Partition the domain such that the x- and y-directions are split into P_x and P_y parts, respectively. Verify your code. [30%]
5. Write a brief report (maximum 3 pages) which includes [15%]
   (a) Quantitative evidence of verification of your C++ code, in serial and parallel.
   (b) A plot of your solution to the fourth test case, on a grid of size 10×10 with 2001 grid points in each direction with a final time of $T = 1$ and 4000 time steps.
   (c) An analysis of the performance of your code and any optimisations you performed during its development.
6. Demonstrate use of good programming practices [5%]
   (a) Generate a log of your use of Git version control using the command:

```
git log --name-status > repository.log
```

   (b) Document your source code appropriately.

Submission and Assessment

When submitting your assignment, make sure you include the following:

- All the files needed to compile and run your C++ code:
  - Source files for a single C++ program which performs all the tasks. i.e. All `.cpp` and `.h` files necessary to compile and run the code.
  - The `Makefile` used for both compiling and running the code, including all the targets as specified in the tasks.
- Your three-page report (in PDF format only).
- The git log (`repository.log`) generated in Task 6.

These files should be submitted in a **single `tar.gz` archive file** to Blackboard Learn. To generate your `tar.gz` archive, put all files to be submitted in a directory (e.g. `ae3-422-assessment`) and run the following command from the directory above it:

```
tar -cvzf submission.tar.gz ae3-422-assessment
```

You may make unlimited submissions and the last submission before the deadline will be assessed.

END OF ASSIGNMENT

Competition! (entirely optional)

Maximising the efficiency and execution speed of a code is critical for High-performance Computing applications. Many aspects of a code's design affect performance, such as the ordering of operations, memory layout and loop ordering. As an additional challenge, you can **opt-in** to your code being included in the AE3-422 performance challenge! We will run your code both in serial and in parallel on a large parallel computer system and measure its performance.

Note: Submissions must successfully solve the fourth validation case and you need to have successfully completed Task 4 in order to be included in the parallel competition.

Choosing to enter the competition is entirely optional and participation (or lack of) will have no bearing on your mark for this module.

Entries for the competition will be assessed and ranked after your assignments are marked and feedback has been returned.