High-performance Computing

Coursework Assignment

Deadline: 22nd March 2023 - 23:00

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 assignment is to write a parallel numerical code to solve the 2D shallow-water equations, paying particular attention to computational performance. These partial differential equations (PDE) can be used to model, for example, the propagation of tsunamis.

The shallow-water equations in non-conservative form are given by

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial h}{\partial x} = 0 \tag{1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + g\frac{\partial h}{\partial y} = 0 \tag{2}$$

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial y} = 0 \tag{3}$$

where $u(x, y)$ and $v(x, y)$ are the x- and y-components of velocity, $h(x, y)$ is the surface height, and g is the acceleration due to gravity. For simplicity, we have neglected friction, Coriolis and viscous forces.

2 Algorithm

To numerically solve this problem, we will use the finite difference method. For spatial derivatives we will use a 6th-order central difference stencil; for example, the partial derivative of u with respect to x can therefore be numerically approximated by:

$$\frac{\partial u}{\partial x} \approx \frac{1}{\Delta x}\left(-\frac{1}{60}u_{i-3} + \frac{3}{20}u_{i-2} - \frac{3}{4}u_{i-1} + \frac{3}{4}u_{i+1} - \frac{3}{20}u_{i+2} + \frac{1}{60}u_{i+3}\right) \tag{4}$$

Similar expressions can be constructed for other derivatives. The PDE should be discretised on a periodic rectangular grid of size $N_x \times N_y$ with $dx = dy = 1$ and a domain of $[0, N_x] \times [0, N_y]$.

For the time-integration we will use 4th-order Runge-Kutta explicit scheme. This can be formulated as:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t$$

with

$$\begin{aligned} k_1 &= f(y_n) \\ k_2 &= f(y_n + \Delta t\, k_1/2) \\ k_3 &= f(y_n + \Delta t\, k_2/2) \\ k_4 &= f(y_n + \Delta t\, k_3) \end{aligned}$$

2.1 Evaluation of f

We can rearrange Equations (1)-(3) to put all but the time derivative terms on the right-hand side. The function f above is therefore used to evaluate the discrete approximation to these time derivatives by computing these right-hand side terms in discrete form. The bulk of the operations are accounted for by the numerical approximations to the derivatives. These could be represented as a *matrix-vector multiplication*, or could be implemented manually in a nested *loop-based* manner.

2.2 Initial and Boundary Conditions

Use the following initial condition:

$$\begin{aligned} u(x, y, 0) = v(x, y, 0) &= 0 \\ h(x, y, 0) &= h_0(x, y) \end{aligned}$$

where $h_0(x, y)$ is a function prescribing the initial surface height.

Periodic boundary conditions should be used on all boundaries, so that waves which propagate out of one of the boundaries re-enter on the opposite boundary. This means that, for example, with $dx = 1$ and $N_x = 100$, the last point is at $x = 99$, even though the domain is of length 100 in x. This is because the point at $x = 100$ is the same as the point at $x = 0$.

2.3 Test cases

Use the following initial conditions to test your code:

1. Plane waves propagating in x

$$h_0(x, y) = H(x, y) + \exp(-(x - 50)^2/25)$$

2. Plane waves propagating in y

$$h_0(x, y) = H(x, y) + \exp(-(y - 50)^2/25)$$

3. Single droplet

$$h_0(x, y) = H(x, y) + \exp(-((x - 50)^2 + (y - 50)^2)/25)$$

4. Double droplet

$$h_0(x, y) = H(x, y) + \exp(-((x - 25)^2 + (y - 25)^2)/25) + \exp(-((x - 75)^2 + (y - 75)^2)/25)$$

In each case, we assume a zero initial velocity, i.e. $u(x, y, 0) = v(x, y, 0) = 0$, a mean surface height of $H(x, y) = 10$, and $N_x = N_y = 100$. The time step can be set to $\Delta t = 0.1$.

2.4 Command-line input

Your code should accept the following command-line arguments (do **not** prompt the user to enter input during the execution of your program):

```
--dt arg            Time-step to use.
--T arg             Total integration time.
--Nx arg            Number of grid points in x
--Ny arg            Number of grid points in y
--ic arg            Index of the initial condition to use (1-4)
```

2.5 File output

At the end of the simulation, your code should generate a file called `output.txt` with the values of u, v and h at each grid-point x-y coordinate. The file should be structured as follows with 5 columns:

```
x0  y0 u v h
x1  y0 u v h
...
xNx y0 u v h

x0  y1 u v h
x1  y1 u v h
...
xNx y1 u v h

x0  y2 u v h
...
```

Note the empty line after each row of points has been written to the file.

You should be able to easily plot the output by running `gnuplot` from a terminal and entering the following commands at the prompt:

```
set pm3d
set hidden3d
splot 'output.dat' using 1:2:5
```

replacing `output.dat` with the appropriate filename.

Tasks

Code

Write a high-performance parallel, object-oriented, C++ code which will solve the 2D shallow-water equations as described above. In completing this assignment you should write a **single** code which satisfies the following requirements:

- Reads the parameters of the problem **from the command line** (do not prompt the user for input). Please use the **exact** syntax given for the parameters in section 2.4 as these will be used when testing your code. [5%]
- Implements a class called `ShallowWater` which encapsulates the solution fields u, v and h and provides functions `SetInitialConditions` and `TimeIntegrate`, with appropriate parameters and implementation, which implements the numerical scheme outlined above. The class should set the parameters of the model through the constructor or a `SetParameters` function. [20%]
- Implements both the matrix-based evaluation of f using BLAS and the loop-based evaluation of f, as described above.
- Is parallelised using MPI **or** OpenMP. It should be able to run on up to 40 cores on a single computer (as specified by the `-np` parameter to `mpiexec`, or `OMP_NUM_THREADS` environmental variable, depending on the choice of parallelisation). [25%]
- Uses a build system (Make or CMake) to compile your code and uses git for version control. There should be targets, named `test1` to `test4` which should run your compiled code with the parameters for the four test cases.

  Provide evidence of your use of git by running the command

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

  and including the file `repository.log` in your submission. [5%]
- Uses good coding practices (code layout, comments, etc) [5%]

Report (maximum 5 pages)

Write a short report (max 5 pages) which addresses the following:

1. Produce 2D contour plots of the solution to the four test cases given in Section 2.3 at $T = 80$. [8%]
2. Describe concisely using bullet points the optimisations you made to improve the performance of both your BLAS and loop-based versions of your code. Which changes had the greatest effect on performance and why? Provide evidence for your assessment using a code profiler. [10%]
3. Following your optimisation of both the BLAS and loop-based versions, which performed better? Discuss why this might have been the case. [6%]
4. Describe your approach to parallelisation and why you chose that approach? What design decisions did you make to enable this approach to parallelisation? [10%]
5. Plot the parallel scaling of your code for the fourth test case. This should show the speed up in execution time (relative to the execution in serial) as a function of the number of threads/processes used. How would you expect the parallel scaling plot to look in the ideal case? If your plot does not show ideal scaling, comment on why you think that may be the case. [6%]

Continues on next page ...

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 requirements. i.e. All `.cpp` and `.h` files necessary to compile and run the code.
  - The `Makefile` or `CMakeLists.txt` file (as preferred) used for compiling your code and running the test cases.
- Your report (in PDF format only).
- The git log (`repository.log`).

These files should be submitted in a **single `ZIP` archive file** to Blackboard Learn.

It is your responsibility to ensure all necessary files are submitted.

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

END OF ASSIGNMENT