PX425 Assignment 2

Introduction

The purpose of this assignment is to apply some of the techniques discussed in lectures for optimising serial code.

First, visit the assignments section of the course web pages and download `burgers.zip` to your space on the SCRTP file-system. This zip file contains a program to implement simulation of a two-dimensional function u evolving according to the equation

$$\frac{\partial u}{\partial t} + u\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = +\nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right). \tag{1}$$

This is a simplified form of the 2D Burgers' equation: a model from fluid dynamics which itself is closely-related to the Navier-Stokes equation. It can be used to model the time evolution of a velocity field in a fluid. We have simplified the problem by setting the two components of the velocity to be equal ($u(x, y) = v(x, y)$) which allows us to study the evolution of a random initial velocity field into larger domains — an example of 'pattern forming' behaviour which is exhibited by many related differential equations of this form. More pragmatically, it is an example of integrating a differential equation to propagatetime, the result of which is a reasonably memorable pattern that you can use to check your code is still functioning properly as you optimise it.

We can discretise the equation onto a grid of dimension $N_x \times N_y$ points. $u_n^{i,j}$ represents the value of the function after time $n\Delta t$ at the grid point i, j. Δt is the time step over which the function u will be evolved at each iteration. The grid spacings are Δx and Δy in the two Cartesian directions, with *periodic boundary conditions*.

A finite difference approximation to equation 1 is

$$\begin{aligned}\frac{u_{n+1}^{i,j} - u_n^{i,j}}{\delta t} = &- u_n^{i,j}\left(\frac{u_n^{i+1,j} + u_n^{i-1,j}}{2\Delta x} + \frac{u_n^{i,j+1} + u_n^{i,j-1}}{2\Delta y}\right) \\ &+ \nu\left(\frac{u_n^{i+1,j} + u_n^{i-1,j} - 2u_n^{i,j}}{\Delta x^2} + \frac{u_n^{i,j+1} + u_n^{i,j-1} - 2u_n^{i,j}}{\Delta y^2}\right),\end{aligned}$$

which can be rearranged to give an explicit expression for $u_{n+1}^{i,j}$ involving values on nearby grid points at the previous time step. We use periodic boundary conditions such that the final grid point in each direction is taken to be a neighbour of the first, and vice-versa.

Running the code

The program will take a non-trivial amount of CPU time to execute. You therefore *must not* run it on `godzilla`. Instead, you should connect (using ssh or X2Go) to a compute server which we have set aside for this assignment. There are two such machines, named `stan1` and `stan2`. For `stan1` you could use:

```
ssh stan1.csc.warwick.ac.uk
```

from the terminal, which will log you into `stan1`, or edit your X2Go settings to point at `stan1`. Once there, you should be able to access all the same files/directories as on `godzilla`, since all the SCRTP machines share a common filesystem. Please do not run anything other than the code for this assignment on the `stan` nodes. First uncompress the code with

```
unzip burgers.zip
```

and then build the code using the provided `Makefile`. This file contains a set of instructions which define how the program is to be compiled. With this provided, one can compile the code using the simple command:

```
make
```

If you examine the `Makefile` in an editor, you will see the variable `CFLAGS` which defines the compiler options to be used. Note that we have initially disabled compiler optimisations with `-O0`. Now run the program with `./burgers` and note the output. The program reports how long it spent setting up the calculation, i.e. applying an initial function to the grid, and the time taken to evolve the equation through 10000 time steps. Run the code a few times (3 is probably enough) to obtain average timings. You should note these in your report (see below). Each run should take no more than a few minutes. If you see significant variability, please check if there other jobs running at the same time as this can slow down the processor somewhat. You will also notice that the program generates `.png` image files showing the state of the simulation grid at various times. Examine the pattern in the final image. Your final image should stay constant if you have implemented changes in a way that does not break the logic of the code.

Optimisation

The code has been written with a number of deliberately inefficient features. Your next task is to identify these, and to re-write the relevant sections of `burgers.c` to improve efficiency, both directly and by helping the compiler to make intelligent decisions. You should review the first two lectures and identify possible ways to optimise the code, considering the flow of both instructions and data into the CPU. Possible areas for attention include, but are not restricted to:

- Avoiding unnecessary repeated computation.
- Expensive arithmetic that could be replaced with simple multiplication.
- Factorising expressions to reduce the number of floating point operations required.
- Elimination of branches.

- Minimising the number of passes through data.
- Optimising memory access patterns for efficient use of cache.

After each change you make to the code, you should document that change. Explain in your report (briefly - we do not want essays) why you made each change. You should then obtain new average timings with your modifications in place, at whatever level of optimisation (`-O1`,`-O2` and `-O3`) gives you best performance, and assess whether the change was effective. You may also experiment with more advanced compiler optimisation settings (`man gcc`) if you wish. All timings should be presented in your report, and should be produced for 10,000 steps, on the grid size used in the original program, $N_x = 256, N_y = 256$.

Your timings must use the `gcc` compiler. If you wish to try the intel compilers please do so, and if you can improve upon the times possible with gcc then please note this in your report, but I still want to see a timing using gcc as well.

Any additional memory allocation or initialisation must be performed between the calls which time the region of code to be optimised. You do *not* need to make any changes to the random number generator or the code which generates simulation snapshots in `makePNG.c`.

The unmodified code, at -O0, compiled with gcc10.2 and run on stan1, takes around 47s to run (somewhat weather-dependent).

Validation

The program produces an output file `final_grid.dat`, which prints the final function u on the grid. Whatever optimisations you perform must not significantly change the computed solution, i.e. you should compare the contents of this file as produced by your optimised code, with that produced by the original (included as `final_grid_orig.dat`). You might find the command `diff` to be useful here. Comparison of the `png` files produced along the way may also be helpful. For debugging, you might (temporarily) modify the code to generate these more frequently by changing the block of code around line 160, and for ultimate performance you might want to disable it.

Optimisation will likely alter the order in which floating point operations are performed, which will inevitably lead to slight differences in the final few decimal places of the final grid. However, to the precision printed (5 significant figures) the `final_grid.dat` produced by your optimised code should match the original exactly.

Report

Your report should describe what changes you have made to the code and why. It needn't be any more substantial than a list of bullet points, each of which contains a two or three

line description of a change to the code, followed by average timings, and a statement of whether or not the change was effective You may prepare the report however you like (MS Word, LaTeX, OpenOffice) etc. Submission in Adobe PDF format is preferred, and it is preferable that the file should be called `report.pdf`. Failing that a MS Word format `report.docx` will suffice.

Deadline and Marking Criteria

You have one week in which to complete this assignment. Many people find it fun to squeeze extra performance out of the code, and get quite competitive about it, but I am most certainly *not* expecting you to spend all your waking hours pushing the efficiency of your program to its absolute limit. You should be looking to spend no more than four to eight hours on this assignment, and it should be possible to gain significant reductions in run time (and get respectable marks) with much less effort.

As well as the usual marking criteria (discussed in the 'general info' handout), marks will be lost if your modified program no longer produces the correct solution, or is slower than the original (on `stan1`). The modified program should be your own effort. Working together is not acceptable and plagiarism is taken very seriously.

You should submit only the version of your program which you have identified to be the fastest, along with the resulting output `final_grid.dat` and your report. Compress these into a zip file with the same name as your SCRTP user code, for example like this

```
zip phuxyz.zip burgers.c final_grid.dat report.pdf
```

where you use your username instead of `phuxyz`. Then upload the zip file to the Moodle page for Assignment 2.