CS257 Advanced Computer Architecture Coursework Assignment

Matthew Leeke
matt@dcs.warwick.ac.uk

February 1, 2017

The purpose of this coursework is to give you some hands-on experience in code optimisation. By the time you read this you will have encountered a variety of code optimisation techniques, including loop unrolling and vectorisation. For many of you this coursework assignment will be your first opportunity to put these techniques into practice.

Your coursework submission will consist of two parts:

1. **Optimised Code**
   A piece of C code based on the initial implementation provided. This C code will be assessed with respect to your selection and understanding of optimisations, functional correctness, i.e., producing the right answer, and execution speed.

2. **Written Report**
   A 2-3 page report detailing your design and implementation decisions. Your report will be evaluated with respect to your understanding of code optimisation techniques and the optimisations you attempted. This means that your report should explain:

   (a) **which** optimisations you used/didn't use;

   (b) **why** your chosen optimisations improve performance; and

   (c) **how** your chosen optimisations affect floating-point correctness.

A sensible approach to the assignment is to build your solution incrementally, saving each partial solution and documenting the impact of each optimisation you make. Credit will be awarded for the correct identification of optimisations, as well as their suitable implementation and documentation. This means that it is in your interest to attempt as many different optimisations as you can.

You may discuss optimisation techniques with fellow students but you are **not** allowed to collaborate on solutions to this assignment. You are reminded that the University takes all forms of plagiarism seriously.

Using the Code

The code you are tasked with optimising carries out a simulation of N stars behaving under the influence of gravity, using a very simple scientific code. To begin the coursework assignment, you should download the code from http://go.warwick.ac.uk/cs257and save it to your home directory.

Next, extract the contents of the archive into a new directory. You should see three files: `Makefile`, `cs257.c` and `coursework.c`. Type `make` to compile them.

To run the code, type `./cs257`. You should see something like:

```
Usage: ./cs257 [\# stars] [\# timesteps] [visualisation (0 or 1)]
Eg   : ./cs257 10000 100 1
```

10000 stars is too much for the original, slow, code to handle, so you might like to start with something smaller. For example, if you trying running:

```
./cs257 1000 1000 1
```

You should see a window like the one in Figure 1. Note that, if you're X-forwarding or cannot run OpenGL, then you should turn off the visualisation by executing `./cs257 1000 1000 0` instead.

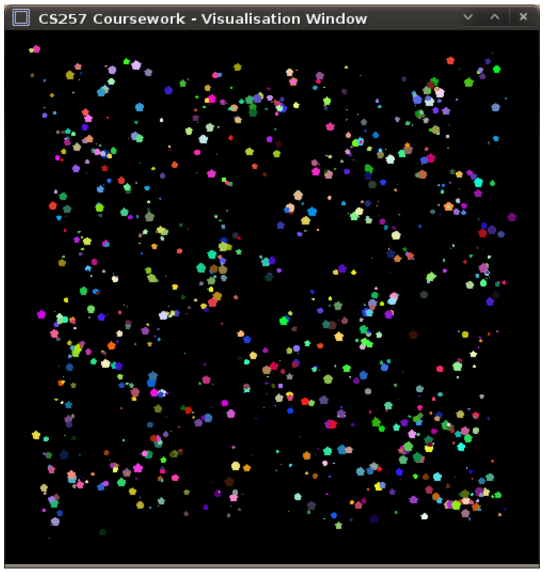

Figure 1: The visualisation window.

If everything has worked correctly, then the output you see in the terminal should match (or at least be close to) the example output below:

```
CS257 Optimisation Coursework
----------------------------------------
 Starting simulation of 1000 stars for 1000 timesteps.

 Loop 0 = 0.002765 seconds.
 Loop 1 = 21.720963 seconds.
 Loop 2 = 0.005372 seconds.
 Loop 3 = 0.006570 seconds.
 Total  = 21.735671 seconds.

 GFLOP/s = 0.919226
 GB/s = 0.001840

 Answer = 86672752.000000
```

Note that application performance is reported in three different ways:

1. Time in seconds;
2. Gigaflops per second (GFLOP/s); and
3. Gigabytes per second (GB/s).

Comparing the last two metrics to the theoretical peak performance of your machine will help you to judge how much additional room there is for code optimisation. Using joshua and GFLOP/s as an example:

$$\begin{aligned}\text{Peak GFLOP/s} =& 8 \text{ cores} \times 4 \text{ SSE units} \times 2 \text{ instructions per cycle} \times 2.4 \text{ GHz} \\ =& 153.6\end{aligned}$$

This means our baseline implementation is running around 170 times slower than it could *in theory*. But remember, there are all sorts of reasons that a theoretical peak performance number may be unattainable, so don't worry too much if you can't quite get there. There will be a special prize[1] to anybody who can demonstrate optimisations that achieve more than 70% of peak performance.

The performance-critical part of the code is contained inside `coursework.c`, in a function called `compute`. This is the only part of the program that you will need to change. You are free to make changes to the other functions if you believe your optimisations require it, though your program must compile and run as set out in this document.

[1]Or at least some hearty congratulations.

Rules

Your submitted solution must:

- Be compiled with gcc (We will test your solution using v4.4.5).
- Use optimisation level `-O2`.
- Produce an answer *justifiably* close to that produced by the original code. Any difference should be documented and explained in your report.

Your submitted solution must not:

- Use instruction sets unsupported by DCS machines, e.g., SSE4, AVX.
- Require additional hardware, e.g., GPUs.
- Add relaxed math options to the compile line, e.g., -ffast-math. Note: Manual use of approximate math functions is acceptable.

Instructions for Submission

Your solution should be submitted using Tabula. Please ensure that your code works on DCS machines prior to submission.

Submission Deadline: Noon on Wednesday 15th March 2016

Files Required:

- coursework.zip (an archive containing your optimised code and report)

Tutor Support

The module tutors is James Van Hinsbergh (j.van-hinsbergh@warwick.ac.uk) and Richard Kirk (r.kirk@warwick.ac.uk). They will be happy to respond to any of your questions by e-mail or to meet in person if you'd need assistance.

Your questions and feedback on this coursework are always welcome.

Good luck!