Intro to HPC Assignment 1:

Serial optimisation

In this assignment, explore serial optimisations for the serial 5-point stencil code running on a single CPU core

Assignment description

- 5-point stencil starting code: https://github.com/UoB-HPC/intro-hpc-stencil
- Using this code as a starting point, apply some of the serial optimisations from the lectures to improve performance
- Your code should only use a single core (no MPI yet)
- In particular, consider the following optimisations:
  - Compiler choice and flags
  - Data layout and data type
  - Vectorisation
  - Anything else you can think of!
- Produce a short report discussing your findings
  - Describe the optimisations that you tried
  - Explain why your optimisations improved performance
  - Provide results to back up your statements and quantify performance gains

Ballpark running times after serial optimisation:

1024x1024: ~ 0.15s
4096x4096: ~ 4s
8000x8000: ~ 12s

Marking scheme – 50%+

- To achieve a mark of 50%+, you will need:
  - Optimised serial code that works and is faster than the original code
  - A reasonable 2 page report that demonstrates you have a some understanding of the main issues

- You should be able to get a simple serial optimised version of the code working in half a day (~3–4 hours)

Marking scheme – 60%+

- To achieve a good mark of 60%+, you will need:
  - Code that successfully uses most of the optimisations we describe
  - A well-written, 2 page report that clearly demonstrates you understand **what** you did, **why** the optimisations work, and **how much** performance you have gained
- You should be able to get this version working in about 1 day (~7–8 hours)
- Your time should be consistent with the ballpark timings in the table on page 2 of this assignment

Marking scheme – 70%+

- To aim for a first (70%+), you'll need:
  - An excellent 2 page report
    - Includes more detailed analysis, such as correctly applied roofline or equivalent techniques
  - Code that:
    - Applies further optimisations that improve performance above those we've described. These may include code transformations beyond those discussed in class. It should certainly include effective vectorisation.
    - Achieves single core performance appreciably faster than those given in the ballpark timings table on page 2 of this assignment
- With ~3 weeks allocated to the serial optimisation assignment, 10 hours allocated to the course each week for 10 weeks, and 4 hours per week spent in lectures and labs, don't spend more than 3 * (10 - 4) = ~18 hours on this assignment in total, including writing the report
  - It should only take half that time to do the simpler version which, along with some interesting experiments and a decent report, should be good enough to earn 60%+

Coursework submission

Your submission will be made via BlackBoard and should include:

1. A 1-to-2 page report in PDF form, which must include:
   - Your name and user ID
   - A description of your serial optimisations
   - Comparisons of your optimised performance vs unoptimised
   - Analysis of the effectiveness of different optimisations you tried
2. The working code you used to generate the results in your report
   - Your code must pass validation using the `check.py` script included in the repo

Submission specification

- Your **report** which must be in a file called “**report.pdf**”,
  - Lower case r: “**report.pdf**” NOT “**Report.pdf**”
- Your **source code**, i.e. “**stencil.c**”
- Your **makefile**, called “**Makefile**”
- An **env.sh** file containing any module commands you need to use to load specific compilers or anything required else to run your code properly. Our automaker script will just source your env.sh file then run ‘make’ before running your executable, so make sure this is all that’s required
- **Don’t** modify the timing code and text output in the starting code, as we’ll use these to automatically extract timing information from each submission
- We must be able to reproduce any runtimes you quote in your report by compiling and running the code that you submit
- Don’t zip these files up, instead submit them as separate files in SAFE

Things to avoid!

- Invalid commands in `env.sh`: -1
- Missing `env.sh`: -3
- Invalid Makefile syntax (line endings or spaces): -1
- Missing build files or otherwise severely broken build: -3
- Minor validation issue (~1 failure): -1
- Major validation issue (3+ failures): -3
- Catastrophic (almost complete) validation failure: -5
- No name on report: -1
- Altered program output: -1

How the automaker works

1. Starts with a clean environment
   - See: https://github.com/UoB-HPC/hpc-course-getting-started/blob/master/2_Modules.md#unloading-modules
2. Runs "**source env.sh**" to load your chosen modules
3. Runs "**make**" to build your application and check that stencil has been produced
4. Runs the application, e.g. "**./stencil 1024 1024 100**"
5. Runs the checking script

If any of the steps above results in an error, a failure is reported at that point. Otherwise, the time is recorded. We recommend you go through these steps manually with your submission, e.g. by redownloading your submitted zip file, to avoid any surprises with the automarker.

Plagiarism checking

- The HPC assignments are all for individuals, they are **not** group work
- We will check **all** submitted code for plagiarism using the MOSS online tool
  - MOSS ignores the example code we give you
  - MOSS will spot if any of you have worked together or shared code, so **please don't!**
- We'll also check **all** submitted reports using the TurnItIn tool, which will find any shared text
- So please don't copy code or text from each other! You **will** get caught, and then ***both*** the copier and original provider will get a **0** for the whole assignment.

Summary

Remember, you'll get marks for:

- A well written, comprehensive report
- A serial code that successfully explores most of the optimisations we suggest

Have fun writing your first optimised programs!