Assignment 1:
Serial optimisations and OpenMP

In this assignment, optimise the serial lattice Boltzmann code, then use OpenMP running on all 16 cores in a Blue Crystal phase 3 node to make it go as fast as possible

PROGRESS: 42%

Serial and OpenMP

- 2D Lattice-Boltzmann: https://github.com/UoB-HPC/UoB-HPC-LBM-2016
- Your submission will be made via SAFE and should include:
  1. A *maximum* **four page** report in PDF form, which must include:
     a. Your name and user id
     b. A description of your Serial optimisations and OpenMP design;
     c. Comparisons of your parallel performance vs. serial performance;
     d. Analysis of effectiveness of different optimisations you tried;
     e. Make it clear what your best performance is for the "256x256" case;
  2. The working code you used to generate the results in your report.
- Results must be within acceptable tolerances.

Rules for “256x256” results

- In your written report, include your best performance for the “256x256” problem size (**input_256x256.params**)
- Your timings must be for the total time around the main loop, ignoring overhead for printfs etc, i.e.:

```
/* start timing here */
for (ii=0; ii < params.maxIters; ii++) {
   timestep(params, cells, tmp_cells, obstacles);
   av_vels[ii] = av_velocity(params, cells, obstacles);
}
/* stop timing here */
```

- Results files must be written out at the end (but don’t time this part!)
- Results must pass the results checking script

Submission requirements

- Your **report** which must be in a file called “**report.pdf**”,
  - Lower case r: “**report.pdf**” NOT “**Report.pdf**”
- Your **source code files**, e.g. “**d2q9-bgk.c**” etc
- Your **makefile**, called “**Makefile**”
- Your output filenames must remain unchanged from the example, i.e they must be **final_state.dat** and **av_vels.dat** (don’t submit these)
- Don’t modify the timing code in the example, as we’ll use this to automatically extract timing information from each submission
- We must be able to reproduce the best runtime 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

Testing your code

- We run all your submitted codes using an **auto testing script**
- To make this work you **must** to stick to the requirements for file names for the output
- Make sure you test your code against all three problems:
  1. **input_128x128.params**
  2. **input_128x256.params**
  3. **input_256x256.params**
- Use the test script to make sure your code produces correct results for each problem, e.g.:
  - **make check REF_FINAL_STATE_FILE=check/128x128.final_state.dat REF_AV_VELS_FILE=check/128x128.av_vels.dat**
- Example serial code timings (on one core of phase 3, compiled with -O3):
  - **(105s) input_128x128.params**
  - **(213s) input_128x256.params**
  - **(855s) input_256x256.params**

Plagiarism checking

- We will check **all** submitted code for plagiarism using the MOSS online tool
  - MOSS is clever enough to ignore the example code you’re all given
  - MOSS will spot if any of you have worked together or shared code, so **don’t!**
- We’ll also check all submitted reports using the TurnItIn tool, which will find if any of you have shared text
- So 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.

Getting good marks

- You’ll get marks for:
  - A well written, comprehensive, report
  - An OpenMP code that is fast and scales well
- Have fun writing your first shared memory parallel programs!