Assignment: OpenCL

In this assignment, use OpenCL to get the Lattice Boltzmann code running on the Nvidia Tesla K20 GPUs in BlueCrystal Phase 3

PROGRESS: 100%

OpenCL

- You can re-use your optimised serial code from OpenMP, but we're also giving you example code which includes almost all of the OpenCL host code:

```
git clone -b opencl https://github.com/UoB-HPC/UoB-HPC-LBM-2016 lbm-opencl
```

- You need to add a new module before you can compile any OpenCL code:

```
module load cuda/toolkit/7.5.18
```

- The example code has almost all the OpenCL host code you need, and kernel code for `accelerate_flow` and `propagate`
- You will need to port the remaining functions to OpenCL. For `av_velocity`, this will involve writing a reduction (see the 'Pi' Exercise09 in HandsOnOpenCL for a simple, non-optimal example that you can use)
- There are **NO GPUs in the head nodes** so you can **ONLY** run OpenCL codes on the GPU nodes via the queue! We've updated the job submission script to request GPUs from the queue.

Assignment guidance - I

- You can achieve a good mark (a 2:1, i.e. 60%+) for:
  - A well-written, 2-3 page report that clearly demonstrates you understand what you did
  - Code that:
    - Has all the main functions ported to OpenCL kernels (rebound, collision, av_velocity). We've already given you accelerate_flow and propagate as kernels
    - Does something sensible with the data movement to/from the GPU (move the clEnqueueWriteBuffer and clEnqueueReadBuffer outside of the main timestep loop
  - You should be able to get this version of the code working in about 1 day (~7-8 hours)
  - This version should have a ball-park time of ~16s for 128x128 (the example code we give you to start is very slow, at about 133s for 128x128). This version of your code should also take about ~100s for the 256x256 problem on one NVIDIA K20 GPU in Blue Crystal phase 3

Assignment guidance - II

- To aim for a first (70%+), you'll need an excellent 2-3 page report, along with code that exploits most of the following:
  - Optimises the data layout of the cells to better suit GPU-style access (e.g. Array of Structure to Structure of Array transformations, discussed in the OpenCL masterclass and in HandsOnOpenCL online: http://handsonopencl.github.io)
  - Implements a reasonably fast reduction in av_velocity
  - Fuses most (but not necessarily all) of the kernels
  - Experiments with work-group (tile) sizes to find performance sweet spots
  - Explores a few other optimisations from HandsOnOpenCL
  - Achieves a ballpark performance of ~10s or faster on the 256x256 problem on one K20 GPU in phase 3 (probably under 5s for 128x128)
- With ~2.5 weeks allocated to the OpenCL assignment, don't spend more than 2.5*(200/12 - 4*12)/12 = ~30 hours on this assignment in total
  - It should only take 7-8 hours to do the simple version which should be good enough to earn 60%+

How fast could OpenCL go?

- An NVIDIA Tesla K20m has a theoretical peak memory bandwidth of 208 GB/s
- GPU-STREAM achieves 152 GB/s on this device (83% of peak, taking ECC into account)
  - https://uob-hpc.github.io/GPU-STREAM/
- The CPUs in a single node achieve ~75 GB/s (102.4 GB/s peak) for STREAM, so your OpenCL implementation *could* be (152/75) i.e. about 2X faster than a good OpenMP version of your code…

Other advice

- Get your code working on the smaller problem sizes first
- Then run larger problem sizes only once that works
- Save running the largest sizes (256x256, 1024x1024) for when your code is running fast, so that you don't keep the GPUs busy for too long
- We'll only time the 256x256 case for OpenCL, so you don't need to run the 1024x1024 example very often

Submission requirements

- Your submission will be made via the website and should include:
  1. A **two or three page** report in PDF form, which must include:
     a. Your name and user id;
     b. A description of your OpenCL design;
     c. A description of your efforts to optimise the OpenCL performance;
     d. A comparison of your OpenCL performance on one GPU vs. your best serial times as well as OpenMP and MPI if you can;
     e. Include your best performance just for the 256x256 problem size (average over 5 runs);
  2. The working code you used to generate the results in your report.
- Results must be within acceptable tolerances.

Rules for performance results

- 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 */
```

- This should include the time taken to transfer data to/from the OpenCL device at the beginning and end
- Results must be written out at the end (but don't time this part!)
- Results must pass the results checking script

Submission components

- Your **report** which must be in a file called "**report.pdf**"
- Your **source code files**, e.g. "**d2q9-bgk.c**", "**kernels.cl**" etc
- Your **makefile**, called "**Makefile**"
- If you need to modify the default environment (e.g. you've used a different compiler, or you wish to set some environment variables for MPI), then you should create an **env.sh** file containing any **module load** or **export** commands that need to be run
- 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)
- 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 four problems:
    1. **input_128x128.params**
    2. **input_128x256.params**
    3. **input_256x256.params**
    4. **input_1024x1024.params (this one is slow so don't run it often!)**
- Use the test script to make sure your code produces correct results for each problem
- 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**
    - **(3477s) input_1024x1024.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.

Class prize

- Your overall score for the HPC unit will earn you a ranking:
  - 50-59%: **HPC Cadet**
  - 60-69%: **HPC Professional**
  - 70%+: **HPC Elite**

CRAY
THE SUPERCOMPUTER COMPANY

- The top student overall will earn the title "***HPC Prime***" and win a mystery prize donated by **Cray**…