Problem set 4, OpenCL Intro

TDT4200, Fall 2015

Deadline: 08.10.2015 at 23:59. Contact course staff if you cannot meet the deadline.

Evaluation: Pass/Fail

Delivery: Use It's Learning. Deliver exactly two files:

- *yourNTNUusername_ps4.pdf*, with answers to the theory questions
- *yourNTNUusername_code_ps4.{zip |tar.gz |tar}* containing your modified versions of the files:
  - `pinkfloyd.c`
  - `kernel.cl` - A version that runs on an ITS015 machine, not CMB.
- Do *not* include `image.png`. Your handed in archive *must* have the same folder structure as the archives handed out (all files unpacked directly, no sub-directories whatsoever). Failing to do this can impact your passing or *failing*.

General notes: Code must compile and run on the following systems:

1. `its015-01.idi.ntnu.no`
2. `Problem_set_4` in the *TDT4200_H2015* group on `climb.idi.ntnu.no`.
3. (Note that the `kernel.cl` you upload to CMB might have to differ from the one that works on ITS015).

You should only make changes to the files indicated. Do not add additional files or third party code/libraries.

Part 1, Theory

Problem 1, OpenCL

a) Which of the below claims hold true for OpenCL? If assumptions are made in your answer, state these.

i) OpenCL is "compile once, run everywhere".

ii) OpenCL is a more verbose language (C-extension) than CUDA.

iii) OpenCL only works on AMD/CUDA/Mali architectures.

iv) OpenCL GPU kernels can only run on GPGPUs, such as CUDA kernels.

Problem 2, OpenCL vs. CUDA

a) What are the OpenCL equivalents for the following CUDA terms:

i) Thread

ii) Block

iii) Local Memory

iv) Shared Memory

v) Global Memory

b) For CUDA and OpenCL, answer the below questions for each technology:

i) Is the technology [CUDA/OpenCL] a single-platform SW, or a multi-platform SW?

ii) Describe how (if at all possible) the technology [CUDA/OpenCL] handles compilation to different target architectures.

Part 2, Code

Neither of the CMB/ITS015 versions you implement/hand in should print anything, nor save their output images to any other locations, or names, than as specified in the example code given!

The `Makefile` handed out in `pinkfloyd.zip` compiles the handed out `pinkfloyd.c`, together with `kernel.cl`, with the help of `lodepng.c`, creating an executable named "`program`".

You can test your output by running the `make image.png` command, which saves the output in `image.png`.

Your job is to implement the rest of the program, which should perform as described in Problem 1.

Problem 1, Pinkfloyd part 1/2

a) In the kernel file `kernel.cl`, implement a kernel containing the function which OpenCL should run on the GPGPU (Nvidia GPU on ITS015, Mali GPU on CMB). The kernel is supposed to receive the positions and sizes of every primitive given as input, and draw these on the specified canvas. The primitives are defined in `input_tdsotm.txt`, and explained below in the input specifications.

b) In the file `pinkfloyd.c`, the required setup of memory/variables, invocation of the kernel-function contained within `kernel.c`, the related transfers to/from CPU/GPGPU, and freeing of the same are required for the executable `program` to work correctly.

   i) Any code in the hand-out versions of `kernel.cl` and `pinkfloyd.c` can be built upon/expanded, or completely re-written, as you wish.

Pinkfloyd 1/2 input specifications

This program receives primitives as input for a canvas of certain quadratic size, and draws the different primitives on the canvas before saving the canvas-image to `stdout`.
The program receives input from command line redirection at execution. The input is structured as follows:

```
<canvaswidth>,<canvasheight>
<number of primitives in input>
<primitive one>
<primitive two>
...
```

Primitives can be defined as the following*:

```
circle centerx,centery radius hue,value
line startx,starty endx,endy thickness hue,value
```

*Mark the spaces on each line! Anything remaining on the same line, after the primitive's specifications are given, can be ignored.

- `hue` is a float given in degrees, from $[0, 360]$, inclusive.
- `value` is a float with value between $[0, 255]$, inclusive.
- The coordinates are given as float, with values from $[0, 1]$, inclusive.
  - The coordinate system is $(0, 0)$ in top left corner, and $(1, 1)$ in bottom right corner.

If two primitives overlap, their color `value`s are simply added together.

Additional details can be found in the recitation slides for this Problem set.