School of Computing, University of Leeds

COMP3221
Parallel computation
Coursework 3: General purpose GPU programming with OpenCL
Deadline: 10am Tuesday 3rd May

If you have any queries about this coursework, please visit the Microsoft Teams page for this module. If your query is not already resolved, post your query as a new conversation.

This piece of work is worth **15%** of the final module grade.

Learning objectives

- Implementation of a general purpose GPU program in OpenCL.
- Correctly implement a GPU kernel to perform a common numerical task.

Task

For this coursework you are required to design an OpenCL application that applies a numerical operation, similar to something known as the heat equation, to a two–dimensional grid of size $N \times N$. More precisely, suppose you are given the following grid of values, where each value might refer to *e.g.* a temperature (only top–left portion of grid shown for clarity),

$$\begin{matrix} 0.000 & 0.000 & 0.000 & 0.000 & \dots \\ 0.000 & 0.900 & 0.710 & 0.843 & \dots \\ 0.000 & 0.211 & 0.942 & 0.120 & \dots \\ 0.000 & 0.348 & 0.179 & 0.572 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{matrix}$$

Given this initial grid, each value, apart from the boundaries, needs to be replaced with the average of its 4 surrounding values, *i.e.* a quarter of the sum of the grid values above, below, to the left, and to the right. For the above example, this is

$$\begin{matrix} 0.000 & 0.000 & 0.000 & 0.000 & \dots \\ 0.000 & 0.230 & 0.672 & 0.330 & \dots \\ 0.000 & 0.548 & 0.305 & 0.759 & \dots \\ 0.000 & 0.120 & 0.682 & 0.525 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{matrix}$$

This 'smoothing' corresponds somewhat to (heat) diffusion, hence the name 'heat equation.'

In serial, and supposing the grid values are stored in a one–dimensional array `float *grid` with row `i` and column `j` indexed as `grid[i*N+j]`, the C-code for this operation is

```
int i, j;
for( i=0; i<N; i++ )
  for( j=0; j<N; j++ )
  {
    // Grid cells on the boundary remain zero, otherwise average.
    if( i==0 || j==0 || i==N-1 || j==N-1 )
      newGrid[i*N+j] = 0.0;
    else
      newGrid[i*N+j] = 0.25*( grid[i*N+(j-1)] + grid[i*N+(j+1)]
                            + grid[(i+1)*N+j] + grid[(i-1)*N+j] );
  }
```

Note that the new values are stored in a separate array `newGrid`, so you will need to allocate two grids on the GPU, one for 'before' and one for 'after'.

You should start by downloading the code from Minerva, and inspect the file `cwk3.c` until you are happy you understand how it works. The code initialises a grid `hostGrid` of size `N*N` on the CPU, where the grid size `N` is specified by a command line argument as *e.g.*

```
./cwk3 16
```

The code initialises `hostGrid` with some random values, and displays the initial grid. It then displays `hostGrid` again. Your task is to include code before the grid is displayed the second time that calls an OpenCL kernel to perform the above operation. The kernel should be included in the file `cwk3.cl` and the rest of your code should be in `cwk.c`.

You only need to use the material in Lectures 14, 15 and 16 for this coursework.

Provided files

`cwk3.c`	:	The starting point for your solution.
`ckw3.cl`	:	The kernel code. Currently this file only contains a comment.
`helper_cwk.h`	:	Includes the same helper routines as used in the lecture, plus some specific routines for this coursework. **Do not modify**, as this will be replaced with a different version for assessment.
`makefile`	:	A simple makefile that selects between `nvcc -lOpenCL` for Linux systems (*e.g.* school machines), and using the OpenCL framework on Macs. Usage optional. Note your submission will be tested on a School machine.

Marks

6 marks	:	Allocation and management of device memory.
6 marks	:	Parallel functionality of heat equation for grids of various sizes.
3 marks	:	Kernel code, management, and execution.

Total: 15 marks.

Submission

It is expected that you will only modify the files `cwk3.c` and `cwk3.cl`. If you add any files, update the makefile if necessary, but do not change the executable name. Do not use sub-directories - keep all files in a flat directory - and **do not alter the file `helper_cwk.h` or remove calls to the routines it contains**, as it will be replaced with a modified version for the assessment.

Archive your submission in either `.tar.gz` or `.zip` format only, and upload to Minerva.

All submissions will be compiled and executed on a School Linux machine after loading a CUDA module. **If you alter the makefile for any reason, ensure it still works on a School machine before submitting**.

Disclaimer

This is intended as an individual piece of work and, while discussion of the work is encouraged, what you submit should be entirely your own work. Code similarity tools will be used to check for collusion, and online source code sites will be checked.

Ensure you retain the receipt for your submission as you may be asked to produce it at a later date in the event of a disputed submission time.

The standard late penalty of 5% per day applies for work submitted after the deadline.