School of Computer Science: assessment brief

Module title	Parallel Computation
Module code	XJCO3221
Assignment title	Coursework 3
Assignment type and description	OpenCL Programming Assignment
Rationale	Implementation of a general purpose GPU program in OpenCL. Correctly implement a GPU kernel to perform a common numerical task.
Guidance	See overpage
Weighting	15%
Submission deadline	10am, Friday 28th March
Submission method	Gradescope
Feedback provision	Marks and comments returned *via* Gradescope
Learning outcomes assessed	Apply parallel design paradigms to serial algorithms. Evaluate and select appropriate parallel solutions for real world problems.
Module lead	Peter Jimack

1. **Assignment guidance**

   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 shown for clarity),

```
0.000   0.000   0.000   0.000   ···
0.000   0.900   0.710   0.843   ···
0.000   0.211   0.942   0.120   ···
0.000   0.348   0.179   0.572   ···
  ⋮       ⋮       ⋮       ⋮
```

   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 (note this is the same as the von Neumann neighbourhood from Coursework 1). For the above example, this produces

```
0.000   0.000   0.000   0.000   ···
0.000   0.230   0.672   0.330   ···
0.000   0.548   0.305   0.759   ···
0.000   0.120   0.682   0.525   ···
  ⋮       ⋮       ⋮       ⋮
```

   Normally this algorithm would be applied iteratively, but for this assignment you need only sweep through the grid once.

   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

```c
int i, j;
for( i=0; i<N; i++ )
  for( j=0; j<N; j++ )
  {
    // Grid cells on the boundary remain zero.
    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]);
  }
```

2. **Assessment tasks**

   You will need to material up to and including Lecture 16 for this assignment.

   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 full list of files is as follows.

`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 examples, plus some specific routines for this coursework. **Do not modify**, as this will be replaced with a different version for assessment.
`cwk3-submit.sh`	:	A simple submission script to use with `sbatch` on `cloud-hpc1.leeds.ac.uk`[1].

   After compiling the code as per the instructions in lectures (using `nvcc -lOpenCL -o cwk3`), execute on the batch queue using the script provided (by typing `sbatch cwk3-submit.sh`). Note that the content of `cwk3-submit.sh` is initially as follows:

```
#!/bin/bash

#SBATCH --partition=gpu --gres=gpu:t4:1

./cwk3 8
```

   The C–code `cwk3.c` initialises a grid `hostGrid` of size `N*N` on the CPU, where the grid size `N` is specified by a command line argument (so `N=8` in the example above). After `hostGrid` has been initialised with some random values, `cwk3.c` 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 operation described above. The kernel should be included in the file `cwk3.cl` and the rest of your code should be in `cwk.c`.

   For the purposes of this assignment you must set the work group size yourself based on the device capacity, *i.e.*, you lose marks if you use `NULL` as the work group size argument when enqueueing the kernel. How to do this is explained near the end of Lecture 15.

   Note that, as evident from the serial code provided above, which has two arrays `grid` and `newGrid`, you will need to allocate memory for two grids on the device; one to read from, and a second one to write to. There is no need to use the red-black pattern that was covered in Lecture 5, and was assessed in Coursework 1.

   Your solution does not need to work for very large grids. It will only be tested on grid sizes ranging from $N = 4$ up to and including $N = 16$.

[1]On `cloud-hpc1.leeds.ac.uk` you must run via the batch queue – see Lecture 14 for details.

3. **General guidance and study support**

   You only need to use the material in Lectures 14, 15 and 16 for this coursework. If you have any queries specific to this coursework you may email Peter Jimack directly up to 48 hours before the deadline.

4. **Assessment criteria and marking process**

   Your code will be checked on a School machine that matches the GPU-node on `cloud-hpc1.leeds.ac.uk`.

   Staff will then inspect your code and allocate the marks as per the mark scheme (see below). Note that although submission of your assignment and the return of feedback will use Gradescope, your solution will not be autograded as the Gradescope autograder does not support OpenCL. Therefore you will not receive any information about the functionality of your submission until it has been manually assessed by staff.

5. **Submission requirements**

   You should only modify the files `cwk3.c` and `cwk3.cl`, consequently, only these should be uploaded.

   As noted above, submissions will be compiled and executed on a Linux system that is equivalent to `cloud-hpc1.leeds.ac.uk`. It is essential that you **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.

6. **Academic misconduct and plagiarism**

   Academic integrity means engaging in good academic practice. This involves essential academic skills, such as keeping track of where you find ideas and information and referencing these accurately in your work.

   By submitting this assignment, you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

   If you use material from outside the module material available on Minerva, include reference(s) in your comments. Generative AI is allowed in an assistive capacity only. Use comments in your code to declare any use of generative AI, making clear to what extent it was used.

   Code similarity tools will also be used to check for collusion.

7. **Assessment/marking criteria**

   There are 15 marks in total.

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