School of Computing: assessment brief

Module title	Parallel Computation
Module code	COMP3221
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	2pm, Tuesday 7$^{\text{th}}$ May.
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	David Head

1. **Assignment guidance**

   For this coursework you are required to write an OpenCL application to perform a parallel calculation on a GPU. The problem is loosely based on a machine learning algorithm[1], as this field is a major user of general purpose GPUs, but no prior knowledge is assumed. This coursework requires the material up to an including Lecture 16.

2. **Assessment tasks**

   There are three arrays in this problem:

`gradients`	:	A floating point array of size N.
`inputs`	:	A floating point array of size M.
`weights`	:	A floating point array of size $N \times M$.

   In serial, each element of `weights` would be updated as follows:

   ```
   for( i=0; i<N; i++ )
        for( j=0; j<M; j++)
             weights[i*M+j] += gradients[i] * input[j];
   ```

   You need to implement this in parallel on a GPU, with N and M specified as command line arguments. You can assume both N and M are powers of 2 – the provided starting code will check this for you.

   Note: For the purposes of this assignment you *must* set the work group size yourself based on the device capacity, *i.e.* you cannot use `NULL` as the work group size argument when enqueueing the kernel. How to do this is explained near the end of Lecture 15.

   A starting point has been provided that includes the setup of an OpenCL context and command queue, and the initialisation of the host arrays. The provided code on Minerva is as follows:

`cwk3.c`	:	The starting C-code.
`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 to this coursework.
`makefile`	:	A simple makefile that selects between `nvcc -lOpenCL` for Linux systems (*e.g.* school machines), and `gcc -framework OpenCL` for Macs. Usage optional. On School machines you still need to load the CUDA module first – see Lecture 14 for details.

   Once successfully compiled, you should launch your executable with the required values of N and M. For instance, for $N = M = 16$ you would launch as

   ```
   ./cwk3 16 16
   ```

   The provided code will check N and M for suitability in the function `getCmdLineArgs`.

[1] It is essentially the back-propagation algorithm applied to one layer of a multilayer perceptron.

3. **General guidance and study support**

   If you have any queries about this coursework, visit the Teams page for this module. If your query is not resolved by previous answers, post a new message. Support will also be available in labs during the timetabled sessions.

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

4. **Assessment criteria and marking process**

   Your code will be downloaded from Gradescope and tested for functionality on a School machine, similar to those in the computing laboratory. Staff will then inspect your code then allocate the marks as per the mark scheme (see below), and then upload the mark and feedback to Gradescope.

5. **Submission requirements**

   Submission is *via* Gradescope.

   It is expected that you will only modify the files `cwk3.c` and `cwk3.cl`, in which case these should be uploaded.

   If you add any files, update the makefile if necessary, but do not change the executable name. As before, do not use subdirectories - 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.

   Note that there is no **CHECK** submission portal for for this coursework as in the previous courseworks, as Gradescope's autograder system does not support OpenCL. Therefore you should submit your solution directly to the submission portal for 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. Also use comments to declare any use of generative AI, to avoid your submission being flagged as plagiarism. Code similarity tools will also be used to check for collusion.

7. **Assessment/marking criteria**

   There are 15 marks in total:

6 marks	:	Calculation is performed correctly.
5 marks	:	Kernel design, execution, and management.
4 marks	:	Device memory usage and management.