ECE3270 Digital System Design
Lab 5: Matrix Operations in OpenCL

Lab Overview: The purpose of this lab is to learn how to modify the host code of an OpenCL project and write multiple kernels. You will also write the same code in C for comparison against the OpenCL timings. You will complete a full report using the LaTeX template and include discussion about comparison between the two implementations. You will submit all edited files (c, cpp, cl) to the assign server as Assignment 5.

Part I

You need to write a C program that can complete Matrix – Matrix Addition and Matrix – Vector Multiplication. You will need to time both of these functions as well as overall execution time for the program, but do not include I/O in the timings (see http://stackoverflow.com/questions/5248915/execution-time-of-c-program for examples). The Matrix – Vector multiplication should take the result of the addition and then multiply that by a vector. This means the multiplication is ***dependent*** on the addition. Your program should be executed by typing:

'./matrix M1 M2 V ADDOUT MULTOUT'

M1 and M2 are the files containing matrices for addition, V is the file containing the vector, ADDOUT is the output of the addition, and MULTOUT is the output for the multiplication.

You will be responsible for writing the functions to create the matrices and vectors. The first 4 bytes of the file should be an integer that details the length of the vector or matrix (assume matrices are square). The remainder of the file should be the binary data for the matrix or vector. You should write the results using the same format.

Part II

You will modify an existing OpenCL Tarball to complete the same Matrix Operations. This will require editing the Makefiles, main.cpp, and the .cl file in the device folder. You can delete any .vhd files and xml files. Each Makefile should have a variable defined that you will change to an appropriate name. You can remove the ***lib*** target from the device Makefile. Your code should adhere to the same execution parameters as the c program.

When editing the host code, main.cpp, you will find the main() function calls init_problem(), init_opencl(), and run(). **init_problem** contains the initialization for program data. You should include reading the inputs here. **init_opencl** contains the platform layer initialization and buffer creation for device memory. **run** contains the runtime code which includes setting up arguments, calling the kernels, and handling all post-processing of data. Inside of run, you can find (possibly commented depending on which project you start with) code that determines execution time for the kernels and prints this information. Make sure any timings you use **DO NOT** time I/O.

You will write two *SEPARATE* kernels for the addition and the multiplication. The addition should occur first, followed by the multiplication. Be aware of data dependencies here. When setting up *NDRange* for these two kernels, carefully think about what a work item is and what it needs to do for these two algorithms. Addition is very simple, but the vector multiplication is a little trickier. Make sure to verify operation is correct for both kernels (it may be beneficial to write your C code here to check the kernels).

For the report, you will need to include speedup analysis of OpenCL vs C versions of these algorithms. For reference, speedup is shown below.

$$Speedup = \frac{Time_C}{Time_{OpenCL}}$$

You will need to test on many different sized matrices and vectors to see if speedup is greater at larger sizes. Include a graph of speedup vs size in your report and discuss the graph. Your smallest matrix should be at least 64x64, and you should include at least 5 other sized matrices that increase by at least a power of 2. I suggest you push the limits of the OpenCL memory and use this as your largest matrix (you will know when this happens). You should compare both addition algorithms, both multiplication algorithms, and the time it takes to execute the pipeline of both. You should compare the algorithms using the FPGA and the ARM processing core. Do not use a

ULLAB machine or a personal computer for the C timing. This means you should receive timings for both the C and OpenCL execution when you run the OpenCL code on the board. The FPGA is not very powerful and therefore is not very comparable to most modern processors.

Extra Credit

Implement an optimization from the AOCL Best Practices Guide and discuss how this improved the speedup over the naïve and non-optimized version of the OpenCL kernel.

Rubric	
Report - Proper format - All sections included - Valid images where applicable - Proper grammar, punctuation, and spelling	**50%** - 10% - 30% - 5% - 5%
Demo - Live Demonstration o Includes working code and answering questions from the TA - Comments o Thoughtful comments, not English translations of code	**40%** - 35% - 5%
Proper Assign Server Code Submission	**10%**