Name: Daniel A. Abreu Fernandez

Due Date: Sunday, June 4th, 2023

**Lab 1:** Basics of Intel Developer Cloud & OpenCL Programming

**Lab Description**:

The purpose of this lab was to get used to the Intel Developer Cloud and the tools required to get to the compute nodes to execute the OpenCL examples provided by the instructor. This involved setting up a fork on GitHub of the original repository of code. The examples included getting a simple hello world script to run and to also get a more complex matrix multiplication kernel running.

**Lab Summary**:

1. All the related lab tasks were completed successfully.
2. All the related lab questions were completed successfully.
3. The GitHub environment was linked to the Git environment in the DevCloud.

As presented in the short summary list above the presented tasks for this lab were completed. The GitHub environment was setup appropriately, and the SSH key required to get the DevCloud environment was also setup. The example programs provided, Hello World and the matrix multiplication, were ran successfully and were analyzed in order to understand the underlying code. Lab 1 was an amazing introduction and much needed setup for priming up for future Labs.

**Lab Questions**:

1. What are the differences between the head node and compute nodes on Intel Developer Cloud? Can you see the same files in your home directory, on both head node and compute nodes?
   1. The head node is simply a landing point that allows SSH login and access to the compute node. Once on the head node getting to the compute node is a matter of putting a command in the terminal. In the compute node, development tasks should take place and should not be used for low level tasks. The compute node should ideally only be used for compilation, testing, and debugging. The same files should be visible both to the compute and head nodes, the only difference will be their capabilities to run tasks.
2. On what node should you execute the executable? What outputs do you see? Should you stay logged on a Compute Node after your task is completed?
   1. The executable should be executed in the compute node as it is the node that has access to the heterogenous accelerators and for the reason already described in Question 1.
   2. This is the output seen:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

* 1. After the task is completed the compute node should be freed so that it can be used for other computationally intensive tasks. The compute node should never be held up for simple tasks such as reading a file, pushing updates to GitHub, surfing the internet etc.

1. Can you commit your code changes in the Head Node, or do you have to do it on a Compute Node?
   1. Code changes can be committed either on the head node or the compute node. However, compute node resources should not be wasted simply to carry out commit commands. The proper development cycle should involve testing code in the compute node iteratively until a solution is working as desired. Once the desired solution is solved, code changes can be committed in the head node.
2. What does this example do? What are the main operations on the host side? What are the main operations on the device side?
   1. The overall goal of the example is to multiply two matrices by using the computational resources of the “device”.
   2. The main operation of the host side is to set up the two matrices that will be multiplied. Allocate memory to both matrices on the device and utilize the command queue to copy the matrices to the device. It also sets up the kernel that will execute the task. Finally, the host utilizes the readbuffer command to copy the output of the device back to the host, this data is then verified by the host by computing the dimensions of the new matrix.
   3. To put it simply the main function of the device is to run the simpleMultiply kernel. The simplyMultiply kernel can compute the multiplication of the matrix one element of a time by locating the row and columns of the corresponding elements from the original two matrices.
3. What new folders or files are created within your lab1 folder? What do you think these files are for?

**Lab Hurdles**:

1. Had difficulties understanding how to clone/fork repo into devcloud environment.
   1. I have used git commands for a variety of reasons, mainly for code control across personal and work-related projects. However, everything I have used GitHub the environment has been setup in direct authentication with the online GH server. This lab was different, I had to create an authentication token so that the offline repo inside of the devcloud could communicate with the online GH server. I had never done this before, so it was definitely tricky.
2. When I ran the instructions in Task 5 to generate the FPGA binary it took a lot longer than 1 hour and even then, the whole operation crashed my connection to the devcloud.
   1. Error Image:

A screenshot of a computer program

Description automatically generated with medium confidence

1. There was a mismatch in instructions from the actual pdf file and the GitHub environment, so I tried using the wrong compute node.
   1. Error Image:

A screenshot of a computer

Description automatically generated

**Lab Takeaways**:

The overall lab was very basic and limited in scope (in terms of learning new concepts about OpenCL programming). This lab seemed to be geared towards making sure the user can compile simple examples and set up their code control environment. The inclusion of the matrix multiplication example was very important to have in this lab as it involved OpenCL code that was more involved but still simple in the grand scheme of things. Having to look through the matrix multiplication example gave an overview on how a real heterogenous system is running useful code.

**Link to GitHub Repository**:

<https://github.com/DanielAbreuFernandez/eece-6540-labs>