Exercise Manual

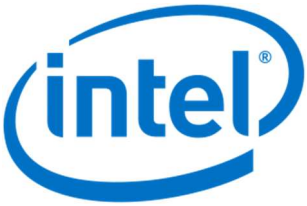
for

Introduction to OpenCL™ for Intel® FPGAs

Lab Exercise 2

Software Requirements

CentOS 7 Linux* OS
Eclipse IDE
Intel® FPGA SDK for OpenCL™ version 19.1
Intel® Quartus® Prime Pro software version 19.1 with Arria® 10 family
Intel® Code Builder for OpenCL™ (Needed to run fast emulator)



Exercise 2

Write a simple OpenCL™ kernel

In this exercise, you will write an OpenCL™ kernel and launch it from the host-code that you started writing in exercise 1.

Step 1. Writing and Compiling the Kernel

____ 1. Reopen the **SimpleOpenCL** project in Eclipse if it's not already open.

____ 2. Open **SimpleKernel.cl** by selecting **File -> Open File…** in the Eclipse menu and navigating to /home/student/fpga_trn/OpenCL/OCL_19_1/SimpleKernel.cl

____ 3. Write a kernel named **SimpleKernel** with **two** float input arguments, **one** float output argument, and **one** uint argument for the size of the arrays.

a. The 1st three arguments of the kernel should be pointers stored in global memory.

b. The 4th argument is passed in by value.

c. Run a loop iterating through all the elements of the arrays

d. Inside the loop, perform any math function you want on one element of the two inputs and store the result to one element of the output argument. Use the loop index to dereference the input and output arrays.

Refer to the slides to see the available math operations..

Your kernel can be something very simple, for example just a multiply or add operation or it can be something more involved.

____ 4. Using a terminal, go to the **/home/student/fpga_trn/OpenCL/OCL_19_1/** folder

____ 5. If you have not done it previously in this terminal, type "source opencl_init.sh"

This script sets the environment variables needed for kernel compilation. It points the compiler to the proper board support package and configures the path, among other things.

____ 6. Compile the kernel by typing the following command

```
aoc -march=emulator -fast-emulator -board=a10gx
SimpleKernel.cl
```

This compiles the kernel using the fast emulator offline compiler targeting the Intel® Arria® 10 FPGA reference board.

Regarding global pointers. When the AOC compiler performs dependency analysis to optimize the pipelined circuit created. Multiple pointers can technically point to the same area, which prevents the compiler from creating the optimal circuit. Using the "restrict" keyword lets the compiler know that in pointers won't point to the same area and that it can treat pointers as separate and make optimizations accordingly.

While for emulation it does not make a different, for later FPGA compilation you may want to add the ***restrict*** *keyword to each of the pointers in the kernel.*

e.g. ***__global const float * restrict in***

Remember if you change the kernel code, you will need to rerun the aoc compiler on the kernel file.

Step 2. Write the host code that launches the kernel

____ 1. Reopen **main.cpp** in Eclipse.

____ 2. Find "#define EXERCISE1" around line 13 in **main.cpp** and comment it out.

Commenting enables the portion of the code that launches the kernel verifies the results.

____ 3. Find the comment "Exercise 2 Step 2.3". Right beneath it, write the code that would create the cl::Program object from the aocx binary. Use the context and devices list created from the first lab.

If we were not running in emulation mode, the variable named "mybinaries" would store the information from the .aocx file used to program the FPGA. However, in the emulation mode, the aocx file is just a software library that can be dynamically linked with the host code.

____ 4. After the comment "Exercise 2 Step 2.4", call the function that **builds** the program from the program created in the previous two steps.

For Intel FPGA, this is only done as a formality.

____ 5. Call the OpenCL function to **Create** the cl::Kernel object from the **program** after the comment "Exercise 2 Step 2.5."

You can use the kernel_name variable in your argument which is set to "SimpleKernel" and should match your kernel name.

____ 6. Call the function to setup the Kernel arguments four times, once for each buffer: **Buffer_In**, **Buffer_In2**, and **Buffer_Out**, and another time to pass in the variable vectorSize. Do this after the comment "Exercise 2 Step 2.6."

____ 7. After the comment "Exercise 2 Step 2.7," **launch** the kernel using **enqueueTask**.

Use the command queue we created in the first exercise.

This command will execute the kernel on the OpenCL device when you are not in emulation mode.

____ 8. Following the comment "Exercise 2 Step 2.8," read back the results of the kernel execution by copying the contents of the Buffer_Out buffer to the array **Z**.

*The contents of Z will be verified later in **main**.*

Make sure the blocking argument of the call is set to CL_TRUE. This guarantees that after read function, Z will be ready to be used.

____ 9. After the comment "Exercise 2 Step 2.9," write the **same** math operation you have in the kernel. This time, however, it should operate on X[i] and Y[i] and store the result into CalcZ[i].

Here we're doing the same calculation the kernel is doing, except that we're using regular C++. The contents are then stored in CalcZ, which will be compared to the contents of Z.

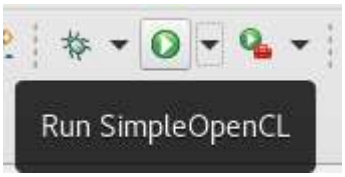
____ 10. Save **main.cpp.**

Step 3. Run and debug the kernel

____ 1. Compile/Build the project

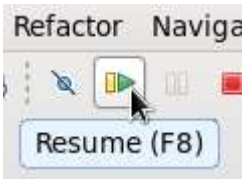
____ 2. Correct any errors if there are any and recompile.

____ 3. Run the program.



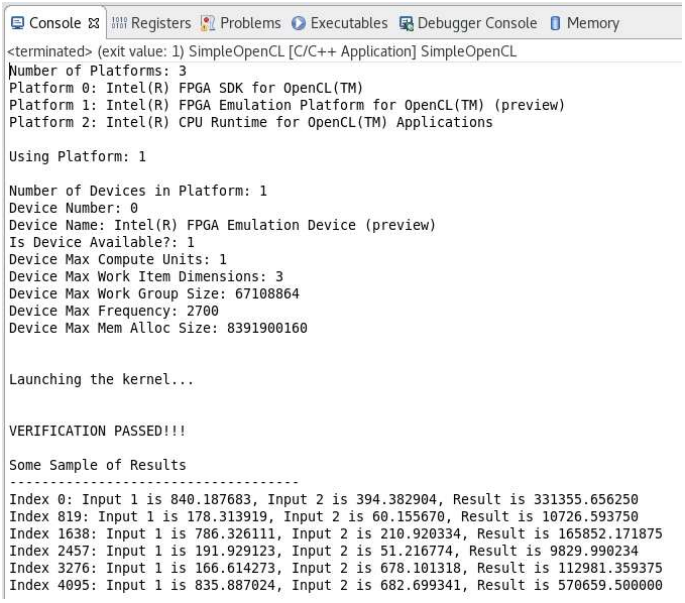
or

then



____ 4. You should see messages in the **Console** tab, as well as the "VERIFICATION PASSED!!!" message, along with some samples of results.

The Verification Passed message means the contents of Z and CalcZ are the same. You can also verify that the sample of results matches the math operation you've performed.



____ 5. If you have run-time errors or the verification fails, use breakpoints and stepping to debug your host program. Or use printf statments. Remember to compile your kernel in the terminal with aoc if you change it.

Exercise Summary

- Practiced writing a kernel
- Wrote host code that setup the arguments to the kernel
- Launched the kernel using enqueueTask

END OF EXERCISE 2
