September 11, 2020

CMU 18-643: Reconfigurable Logic: Technology, Architecture and Applications

Handout #5: Intel DevCloud Tutorial (0 points)

Due Friday, 9/25/2020 noon

This lab can be completed individually or in a team of 2. Please complete the tutorial by the due date although there is nothing to hand-in. This tutorial will test your environment setup and introduce you to the Intel DevCloud and the Intel FPGA SDK for OpenCL. We will be using this environment in Lab 4.

Please feel free to post questions and answers on 18643's Piazza page to help each other out with tools and board related issues. The tutorial steps go quickly except for the FPGA synthesis step; plan to have something else to do for 1 hour.

We want to thank Intel Mindshare Curriculum Program for supporting the development of this lab exercise and the corresponding lecture materials[1]. We want to thank Intel DevCloud: (https://software.intel.com/en-us/devcloud/FPGA) for providing access and support to 18-643 students.

Warming Up:

You will be receiving an email from Intel about your DevCloud account. Log into an ECE Linux server in the *number* cluster: [ece000-ece031].ece.local.cmu.edu[2]. Follow the DevCloud instruction to configure ssh from your ECE Linux account. Afterwards, manually edit the file ~/.ssh/config, changing

```
ProxyCommand nc -x PROXY_HOSTNAME:PORT %h %p    to
ProxyCommand nc --proxy-type http --proxy proxy.its.ece.cmu.edu:3128 %h %p
```

and

```
ProxyCommand ssh -T devcloud-via-proxy          to
ProxyCommand ssh -qT devcloud-via-proxy
```

Execute the command ssh devcloud.proxy to verify you can access the DevCloud from a *number* cluster machine.

We will be using the Arria 10 PAC 1.2.1 systems for this lab. After logging into your DevCloud account, work through the emulation portion of the *hello_world* example in https://github.com/intel/FPGA-Devcloud/tree/master/main/QuickStartGuides/OpenCL_Program_PAC_Quickstart/Arria%2010. Read but do not actually do the synthesis portion of the example.

The Real Deal:

We will be compiling on ECE's servers and transferring the finished FPGA bitstream and host executable to the DevCloud for execution. We have created scripts to make this mostly transparent to you.

1. From an ECE Linux server in the *number* cluster, set up the ECE643 shell variable by executing the following (even better, add it to your .bashrc),

   ```
   export ECE643="/afs/ece.cmu.edu/class/ece643/software"
   ```

   Quartus and accompanying Intel software you need are in the directory /afs/ece.cmu.edu/class/ece643/software. If your environment is set up correctly, you can use

[1] http://users.ece.cmu.edu/~jhoe/course/ece643/F20handouts/L11.pdf;
http://users.ece.cmu.edu/~jhoe/course/ece643/F20handouts/L12.pdf;
http://users.ece.cmu.edu/~jhoe/course/ece643/F20handouts/L13.pdf.

[2] https://userguide.its.cit.cmu.edu/research-computing/computer-clusters/#RemoteAccessClusters

$ECE643 in place of typing out the full path. (Try executing the command cd $ECE643/scripts.) The directory $ECE643/scripts contains the scripts to help you compile OpenCL for Intel FPGAs.

- We will use the ECE Condor Cluster for long-running FPGA compilations. The sample job script condor.job is a template for submitting jobs to the Condor cluster.
- The script opencl_compile.sh contains the commands to compile using Intel FPGA SDK for OpenCL. Execute $ECE643/scripts/opencl_compile.sh -h see how to use this script.
- The script emulate.sh contains the commands to compile and run emulation on the DevCloud. It takes one argument which is the directory name to use on the DevCloud.
- The script run.sh contains the commands to run the compiled kernel on DevCloud FPGA nodes. It takes one argument which is the directory name to use on the DevCloud.

2. Download lab1.zip from Canvas.[3] Unpack the OpenCL project directory under your ECE AFS space.[4] Skim lab1/cnn/device/cnn.cl (what will be executed on the FPGA).[5] The host program for the CPU is lab1/cnn/host/src/main.cpp.

3. Before compiling for FPGA, you will first want to test/debug the functional correctness of your program in software emulation mode. Emulation compiles in 10s of seconds while FPGA synthesis will take hours. From the unpacked project directory lab1/cnn/, execute

   $ECE643/scripts/emulate.sh lab1

   This will (1) compile the kernel for software emulation; (2) compile the host program; (3) execute the program; and (4) return the program's stdio output in the file run_emulate.sh.o<job-id>.[6] In the Lab 1 example, the host program should display the status of the (emulated) kernel execution and finally report the execution performance (GFLOP).[7] If any errors occurred, messages might also be present in run_emulate.sh.e<job-id>.

4. To execute the OpenCL kernel on FPGA for real, you need to compile the kernel to produce an FPGA configuration bitstream file (lab1/cnn/bin/cnn_fpga.aocx). For the Lab 1 example, this process will take about 1 hour. From the project directory lab1/cnn/, execute[8]

   $ECE643/scripts/opencl_compile.sh -f cnn ./ -m "-fast-compile"

   Alternatively, you can use Condor to manage the job for you. To use Condor, copy $ECE643/scripts/condor.job to your own project directory. Follow the instructions in the file's header to use this condor script to launch the compilation.[9]

5. When compilation is completed, transfer lab1/cnn/bin/host and lab1/cnn/bin/cnn_fpga.aocx to the DevCloud to run them on an Arria10 PAC node. This can be all done by executing from the lab1/cnn/ directory

   $ECE643/scripts/run.sh lab1

[3] http://users.ece.cmu.edu/~jhoe/course/ece643/F20handouts/lab1.zip
[4] This project when built will need ~1.2GByte of quota. You don't want to do this from Andrew AFS.
[5] While not necessary at this point, for background, you can read C. Zhang, et al., "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," Proceedings of ISFPGA, 2015 (http://dl.acm.org/citation.cfm?id=2689060).
[6] We are actually compiling and executing the emulation on the DevCloud because, for some inexplicable reason, you cannot compile or run emulation on a machine without an PAC card.
[7] Since the kernel execution is only being emulated in software, the reported performance here is irrelevant.
[8] If you omit -m "-fast-compile", the compile will take longer, but the results will be faster and/or cheaper.
[9] Understanding how to use Condor will help you explore the design space faster in later labs.

After the script is finished, look for the captured output in the file run.sh.o<job-id>. Make a note of the reported performance from executing on FPGA. Do not be dismayed. We will be returning in Lab 4 to *fix* the performance of the program.