

Winter 2019

# CS 133 Lab 5

## FPGA w/ OpenCL: Convolutional Neural Network (CNN)

### <SDAccel Part>

## Description

Your task is to accelerate the computation of convolutional neural network (CNN) using OpenCL with a FPGA. The details of the algorithm can be found in the lecture slides and the Lab 3 description.

## Preparation

### FPGA development and Project Overview

FPGA development roughly involves 3 steps: high-level synthesis (HLS) run, bitstream generation, and on-board execution. In this class, we only target HLS step only using OpenCL as the entry point to be compiled by Xilinx's SDAccel tool. Also, you have *an option* of trying a flow with Falcon Computing's Merlin Compiler (using C + program approach similar to OpenMP).

If you successfully run the Merlin Compiler optional flow, you may earn up to 60 additional bonus points. Since you can earn up to 10 bonus points from the SDAccel flow, the total maximum bonus points for this lab is 70 points.

### Create an AWS Instance

Please create an AWS instance in US East (N. Virginia) for HLS step. You may choose any server with more than 32GB of memory (list available at <https://github.com/aws/aws-fpga>, FPGA Developer AMI section). TA used c5.4xlarge instance.

Please use your AWS account (not AWS Educate classroom account) for this lab. You will probably need to request increase of limit for the type of instance you want. To do this, go the AWS console -> Limits -> "Request limit increase" next to the instance type you want.

For SDAccel flow, when asked to choose AMI, please click AWS Marketplace and search for FPGA Developer AMI, as recommended in <https://github.com/aws/aws-fpga/tree/master/SDAccel> . The version should be 1.5.0.

When launching the instance, please click "storage" and increase the /dev/sdb disk size to about 15GB. FPGA projects may require a large disk space.

### Login to AWS

FPGA Developer AMI use CentOS. Thus, your login ID should be centos, not ubuntu.

## Xilinx Forum

You may visit Xilinx's AWS forum to see existing questions and answers:  
<https://forums.aws.amazon.com/forum.jspa?forumID=243>.

## Environment Setup

Xilinx has provided the SDAccel environment setup files. As mentioned in <https://github.com/aws/aws-fpga/blob/9a9deb5741bf251f4952c8fe93ac6c458e6f3da7/SDAccel/README.md>, please run the following commands:

```
git clone https://github.com/aws/aws-fpga.git $AWS_FPGA_REPO_DIR
cd $AWS_FPGA_REPO_DIR
source sdaccel_setup.sh
```

The git clone only needs to be done once, but the setup file would need to be sourced every time you login. The first run of the source command will install missing programs, so it will take longer than the following runs.

## Helloworld Example

Please follow the instructions in the following github page:

<https://github.com/aws/aws-fpga/blob/9a9deb5741bf251f4952c8fe93ac6c458e6f3da7/SDAccel/README.md>

You only need to follow the steps up to "SW emulation". The instructions are copied below:

```
cd $SDACCEL_DIR/examples/xilinx/getting_started/host/helloworld_ocl/
make clean
make check TARGETS=sw_emu DEVICES=$AWS_PLATFORM all
```

You may also choose to follow the instructions from our guest lecture on Monday:

<https://github.com/xupgit/awslabs>

### Tips

- To resume a session in case you lose your connection, you can run `screen` after login.
- You can recover your session with `screen -DRR` if you lost your ssh connection.
- You should **stop** your instance if you are going back and resume your work in a few hours or days. Your data will be preserved but you will be charged for the [EBS storage](#) for \$0.10 per GB per month (with default settings).
- The instance types used for FPGA development is **very expensive**. The cost table can be found in <https://aws.amazon.com/ec2/pricing/on-demand/>. Please pay careful attention to your spendings.
- You should **terminate** your instance if you are not going to back and resume your work in days or weeks. **Data on the instance will be lost.**
- You are recommended to use **private** repos provided by [GitHub](#). **Do not put your code in a public repo.**

## Run CNN in Software Emulation Mode

We have prepared the host code for you at [GitHub](#).

The files are mostly the same as Lab 3 and 4 except changes to the makefile that enables FPGA development in SDAccel environment. Also, the kernel file has renamed to `xilinx.cl`. Log in to your SDAccel instance and run the following commands:

```
git clone https://github.com/UCLA-VAST/cs-133-19w -o upstream
cd cs-133-19w/lab5
make swsim
```

The provided code will load test data and verify your results against a ground truth. It should run with large error and finish in a few seconds.

We will use this software emulation mode for correctness checking only. Since this is an emulated result, **the execution time provided after running the software emulation is incorrect**. The performance estimation we will use for grading can be obtained from the HLS step of SDAccel.

### Tips

- To check in your code to a **private** GitHub repo, [create a repo](#) first.

```
git branch -m upstream
git checkout -b master
git add xilinx.cl
git commit -m "lab5: first version" # change commit message accordingly
# please replace the URL with your own URL
git remote add origin git@github.com:YourGitHubUserName/your-repo-name.git
git push -u origin master
```

- You are recommended to `git add` and `git commit` often so that you can keep track of the history and revert whenever necessary.
- If you move to a new instance, just `git clone` your repo.
- Run `make test` to re-compile and test your code.
- If `make test` fails, it means your code produces wrong result.
- ***Make sure your code produces correct results!***

## HLS Step using SDAccel

SDAccel provides high-level synthesis (HLS) step that synthesizes OpenCL kernel code into a hardware description (in Verilog or VHDL language). Along with hardware description generation, HLS step also provides performance estimation which we will use for performance grading. Please run

```
make hls
```

and you will be able to see the result in the following file:

```
./_x/cnn.hw.xilinx_aws-vu9p-f1-04261818_dynamic_5_0/CnnKernel/CnnKernel/solution/syn/report/CnnKernel_csynth.rpt
```

### Warning

- Due to the limited time left for grading, we will put a timeout for SDAccel and Merlin HLS run. Your HLS step should be **completed within 90 minutes on c5.4xlarge** machine, or you will receive **no performance grade**.

In this file, please scroll down to Performance Estimates -> Latency -> Summary -> Latency max column, which provides the number of cycles.

Assuming a default clock cycle of 250MHz, the execution time can be estimated as: (clock cycles) / 250MHz. The performance in terms of GFLOPS would be :  $kNum * kNum * kImSize * kImSize * kKernel * kKernel * 2 * 250 / (clock\ cycles * 1e3)$ .

### Warning

- Please make sure that all of the loops in your kernel file have ***fixed loop bounds*** (either by using constant variables or macros). They should not have a variable value. If any of the loop bounds is a variable, Xilinx HLS tool will not be able to provide an accurate performance estimate. If your max and min latency is different, we will grade your performance **based on the max latency**. If your latency is '?', you will receive **no performance grade**.

In the same file, you can find FPGA resource (LUT/FF/DSP48E/BRAM\_18K) usage. You can find it at Utilization Estimates -> Summary -> Total. Note that single-precision floating point adder uses 2 DSP48Es, single-precision floating point multiplier uses 3 DSP48Es, and 1 BRAM\_18K is 18Kbit. You will probably observe that as you apply more optimizations, your resource usage will increase. Ideally, you should try to keep applying optimization until your kernel occupies about 80% of these resources. The remaining 20% is reserved for interface (DRAM/PCIE controller) and other uses.

### Warning

- Please make sure that resource utilization is **less than 80% for all FPGA resource (LUT/FF/DSP48E/BRAM\_18K)**. If any of the resource is over this limit, you will receive **no performance grade**.

Your task is to implement a fast, parallel version of CNN on FPGA. You can start with the sequential version provided in `cnn.cpp`. You should edit `xilinx.cl` for this task. Please keep in mind that, you are NOT allowed to adjust the global / local work size parameters in `params.sh` (they are fixed to 1).

**Instead, the parallelism should be exploited by using OpenCL FPGA directives.** The list of OpenCL directives supported in SDAccel can be found in Chapter 2 of the following document:

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2018\\_2/ug1253-sdx-pragma-reference.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1253-sdx-pragma-reference.pdf)

## Example Code Snippet

In `xilinx.cl` file, we have provided an example code that consists of high-level description of the task and some code snippets which achieve throughput of  $ll=1$  for a 5x5 tree reduction for the CNN convolution kernel (the p and q reduction loop). The code has been provided just for demonstration purpose, and it is up to you to decide whether to use whole or part of the code provided. Also, it is your task to attempt to convert it into a working version, and TAs will NOT be able to provide further assistance of the process.

## Submission

You need to report SDAccel-estimated performance results of your FPGA-based OpenCL implementation on Xilinx Ultrascale+ VU9P FPGA. Please express your performance in GFlops and the speedup compared with the sequential version in CPU (you can reuse the result from Lab 3 or Lab 4). In particular, you need to submit a brief report which summarizes:

- (20 points) Please explain the parallelization and optimization strategies you have applied. Include the directives (if any) or code segments you have added to achieve this. Evaluate the performance of each parallelization/optimization that you have incrementally applied and

explain why it improves the performance. Also, explain how your strategy differ from your Lab 3 CPU and Lab 4 GPU parallelization/optimization strategy and why you chose to apply different strategy.

- (5 points) Please report the FPGA resource (LUT/FF/DSP/BRAM) usage, in terms of resource count and percentage of total. Which resource has been used most, in terms of percentage? (*Optional, bonus +5*: Analyze your original code and check if the DSP/BRAM resource usage matches your expectation. You only need to account for the single-precision floating-point adders, multipliers, and size of arrays. Please attach related code segments to your report and show your work on how you computed the expected number. Provide reasonable discussion on possible reasons if they differ significantly.)
- *Optional*: The challenges you faced, and how you overcame them.

You will need to submit your optimized kernel codes. Please do not modify or submit the host code. Please submit to CCLE. Please verify the **correctness** of your code before submission.

Your final submission should be a tarball which contains and only contains the following files:

<Your UID>.tar.gz

└ <Your UID>

└─ xilinx.cl (SDAccel kernel, required)

└─ CnnKernel.cpp (Merlin kernel, optional)

└─ lab5-report.pdf (required)

File lab5-report.pdf must be in PDF format. You should make the tarball by copying your lab5-report.pdf to the lab5 directory and running

make tar UID=<Your UID>. The Merlin kernel, i.e. CnnKernel.cpp will be included by the tar command if there exists a merlin/src/CnnKernel.cpp. If you made the tarball in other ways, you **MUST** put it in the lab5 directory and check by running make check UID=<Your UID>. The check script will tell you whether the Merlin kernel is included properly. The make tar command and make check command should be invoked in the lab5 directory.

## Grading Policy

### Submission Format

**Your submission will only be graded if it complies with the requirement. In case of missing reports, missing codes, or compilation error, you will receive 0 for the corresponding category/categories.**

### Correctness (50%)

Please check the correctness of your implementation.

### Performance (25%+5%)

The performance point will be added only if you have the correct result, so please prioritize the correctness over performance. Your performance will be evaluated based on the ranges of throughput (GFlops). We will set five ranges after evaluating all submissions and assign the points as follows:

- Better than TA's performance: 25 points + 5 points (bonus)
- Range A GFlops: 25 points
- Range B GFlops: 20 points
- Range C GFlops: 15 points
- Range D GFlops: 10 points

- Speed up lower than range D: 5 points
- Slowdown: 0 points

## Report (25% + 5%)

Points may be deducted if your report misses any of the sections described above.

## <Optional Flow Using the Merlin Compiler>

### Create an AWS Instance

For Merlin flow, please click “My AMIs” first, and click “Shared with me”. Then search “merlin-compiler”. You should select “merlin-compiler-sdx18.02-2.3.2-a - ami-06c63b40e01d826c7”.

### Login to AWS

Merlin AMI use CentOS. Thus, your login ID should be centos, not ubuntu.

### Environment Setup

For Merlin flow, please follow instructions in Prerequisite and Compile on AWS with Merlin section of <https://github.com/falconcomputing/merlin-compiler/blob/master/On-Cloud/AWS/README.md>. Specifically, you should run

```
git clone https://github.com/falconcomputing/merlin-compiler.git
export XILINX_SDX=/opt/Xilinx/SDx/2018.2.op2258646
source $XILINX_SDX/settings64.sh
export FALCONLM_LICENSE_FILE=2300@ec2-34-227-110-85.compute-1.amazonaws.com
```

Again, the git clone needs to be done once, but the environment setup may need to be done every time you login.

### VectorAdd Example

Please follow the instructions in Compile on AWS with Merlin section of <https://github.com/falconcomputing/merlin-compiler/blob/master/On-Cloud/AWS/COMPILE.md>,

You only need to follow the instruction up to “Run your code on CPU”. In addition, try running the `mcc_esimate` flow to obtain cycle estimation. The instructions are copied below:

```
cd merlin-compiler/Examples/vectoradd
make cpu run
make xilinx mcc_estimate
```

Before running “`make xilinx mcc_estimate`”, please edit `build/Makefile`, and change “`XILINXSUBDIRS = xilinx_dse xilinx_mo`” to “`XILINXSUBDIRS = xilinx_mo`” in line 14. The entry “`xilinx_dse`” enables design space exploration step which is not required in our lab.

## Run CNN in Software Emulation Mode

We have added a Merlin subdirectory in your Lab 5 project folder. Log in to your Merlin instance and run the following commands:

```
git clone https://github.com/UCLA-VAST/cs-133-19w -o upstream
cd cs-133-19w/lab5/merlin
make cpu run
```

The kernel file `src/CnnKernel.cpp` already has a working code and should finish with no error.

## HLS Step

Merlin Compiler uses SDAccel's HLS flow to synthesize its C file into hardware description file. Merlin Compiler reads SDAccel's performance estimation file and generates its own performance estimate, which we will use for performance grading. Please run

```
make xilinx mcc_estimate
```

and you will be able to see the result in the following file:

```
./build/xilinx_mo/merlin.rpt
```

In this file, please scroll down to Source code hierarchy -> CnnKernel -> AC column, which provides the number of cycles.

Your task is to implement a fast, parallel version of CNN on FPGA. You can start with the baseline implementation provided in `src/CnnKernel.cpp`. The parallelism should be exploited by using Merlin directives. The list can be found in the Merlin manual provided in CCLE.

## Submission

You need to report Merlin-estimated performance results of your FPGA-based OpenCL implementation on Xilinx Ultrascale+ VU9P FPGA. Please express your performance in GFlops and the speedup compared with the sequential version in CPU (you can reuse the result from Lab 3 or Lab 4). In particular, you need to submit a brief report which summarizes:

- (bonus, +8 points) Please explain the parallelization and optimization strategies you have applied. Include the directives (if any) or code segments you have added to achieve this. Evaluate the performance of each parallelization/optimization that you have incrementally applied.  
Please concentrate on making comparison with the SDAccel version. That is, if a same strategy was applied as your SDAccel version, please mention this in your report and keep the description very brief. If the coding style (or directive) has changed from SDAccel version, you will need to show the difference. For the newly applied optimizations, please elaborate and explain why it improves the performance. If some optimizations were removed, please mention this in your report.
- (bonus, +2 points) Please report the FPGA resource (LUT/FF/DSP/BRAM) usage, in terms of resource count and percentage of total. Which resource has been used most, in terms of percentage? This information can be found in the SDAccel HLS report :  
`build/xilinx_mo/.merlin_prj/run/report/CnnKernel_csynth.rpt`

- (bonus, +10 points) Please analyze Merlin log file (build/xilinx\_mo/merlin.log), Merlin report file (build/xilinx\_mo/merlin.rpt), and SDAccel HLS report (build/xilinx\_mo/.merlin\_prj/run/report/CnnKernel\_csynth.rpt). Try to find the optimization Merlin has performed on your code. Explain why you reached this conclusion. Explain how such optimizations would increase the performance of your kernel.
- (bonus, +10 points) Compare the performance of SDAccel flow and Merlin flow. Which one is better? Why? Does the result meet your expectation? If not, please discuss possible reasons.

Note that you will receive bonus points for Merlin flow report, **only if you have received some points for your Merlin performance score.**

Please write your report in the same file as SDAccel report - lab5-report.pdf.

You will need to submit your optimized kernel codes. Please do not modify or submit the host code. Please submit to CCLE. Please verify the correctness of your code before submission.

For submitting Merlin kernel file, please read the submission instructions in the SDAccel section.

## Grading Policy

### Submission Format

**Your submission will only be graded if it complies with the requirement. In case of missing reports, missing codes, or compilation error, you will receive 0 for the corresponding category/categories.**

### Performance (30%)

The performance point will be added only if you have the correct result, so please prioritize the correctness over performance. Your performance will be evaluated based on the ranges of throughput (GFlops). We will set five ranges after evaluating all submissions and assign the points as follows:

- Range A GFlops: 30 points (bonus)
- Range B GFlops: 25 points (bonus)
- Range C GFlops: 20 points (bonus)
- Range D GFlops: 15 points (bonus)
- Speed up lower than range D: 10 points (bonus)
- Similar to the baseline code provided: 0 points

### Report (30%)

Points may be deducted if your report misses any of the sections described above.