Spring 2025

CS 133 Lab 4

FPGA with HLS on Convolutional Neural Network (CNN)

Description

Your task is to accelerate the computation of a convolutional neural network (CNN) layer using a **high-level synthesis (HLS) tool** on an FPGA. Specifically, for an input image with 228×228 pixels and 256 channels, you are going to calculate the tensor after going through a 2D convolution layer. The convolution layer has 256 filters of shape $256 \times 5 \times 5$. You will need to implement this function using HLS:

```
void cnn(const float* input, const float* weight, float* output)
```

where `input` is the input image of size `[256][228][228]`, `weight` stores the weights of the convolution filters of size `[256][256][5][5]`, and `output`. The output size is `[256][112][112]`.

How-To

FPGA accelerator compilation typically involves three (3) stages: high-level synthesis (HLS), bitstream generation, and onboard execution. The last two stages can take days to complete. Therefore, in this lab, we only focus on the first stage: HLS. Your performance will only be assessed using the estimation in the HLS reports, which is usually accurate. However, you are welcome to try out the last two steps if you are interested.

Create an AWS Instance for Development

Create an **`m5.2xlarge`** instance with **Vitis 2023.2 Developer AMI**. You can also try a bigger instance like m5.4xlarge. They may yield faster compilation, but also cost more.

Development Environment Setup

Log in to your AWS instance using username **`ubuntu`** (e.g. `ssh -i key.pem ubuntu@1.2.3.4`), clone the lab repository and run the three setup scripts provided. The first script will install Docker and pull the Merlin Docker. The second script will run a Merlin container and connect to it. The third script will set up the environment within the container.

```
unzip lab4.zip
source /tools/Xilinx/Vitis_HLS/2023.2/settings64.sh
cd lab4/
```

You can run `exit` to quit the container when you are done with it. You need to execute the last two scripts every time you want to work on a new container, such as after restarting the instance.

Build and Run Baseline with Software Simulation

We have prepared the starter kit for you. Please run:

```
ulimit -s unlimited
make csim
```

This command will perform a software simulation of the provided starter FPGA HLS kernel. It should show “PASS”. **You need to use FPGA Developer AMI in this lab unless you are using a computer with Xilinx Vitis HLS installation.** However, you are still suggested to develop code and run software simulation locally to test the correctness. You can move to AWS once you enter the tuning stage.

Modify the HLS CNN Kernel

If you have successfully built and run the baseline HLS CNN kernel, you can now optimize the code to design your CNN kernel. **Your task** is to implement a fast, parallel version of the CNN kernel on FPGA. You should start with the provided starter kit. You should edit `cnn.cpp` for this task.

Parallelism should be exploited by using AMD Vitis HLS pragmas. You are encouraged to focus on Vitis pragmas (`#pragma HLS unroll`, `#pragma HLS pipeline` and `#pragma HLS array_partition`).

In the starter kit, we provide a sequential CNN implementation that is wrapped with basic pragmas to enable data caching and memory coalescing. This baseline design achieves approximately 12 GFLOPs.

Test Your HLS CNN Kernel with Software Simulation

To perform software emulation of your FPGA implementation of CNN kernel:

```
ulimit -s unlimited
make csim
```

If you see something similar to the following message, your implementation is incorrect.

```
FAIL
```

Since the software simulation step uses the CPU to emulate the hardware behavior, it only serves as correctness test and its execution time doesn’t reflect that of actual hardware. Your **estimated execution time** should be retrieved using the command below:

```
make vitis
```

You can get the performance by “

0.25 * `kNum*kNum*kImSize*kImSize*kKernel*kKernel*2/latency (max cycles)`” to get the performance in GFLOPS. 0.25 is for the frequency.

IMPORTANT: Please make sure that all your loops have fixed loop bounds. **If any of the loop bounds are variable, a performance estimation will not be shown and you will receive no performance grade.**

IMPORTANT: The “`make vitis`” command should finish in 20 minutes, or in two hours with highly-complex optimizations. Our recommendation is to halt your estimation using `Ctrl-C` when the time exceeds 20 minutes, except for your last step (after you reach ~100 GOPS). **More than 12 hours in the estimation will result in zero for the performance score.** As your kernel design becomes more complex, the software simulation and the estimation will start to take a significantly longer time.

IMPORTANT: As you apply more optimizations, your resource usage will also increase. Please make sure that resource utilization is **less than 100% for all FPGA resources**. If any of the resources are over this limit, you will receive **no performance grade**.

IMPORTANT: You can check the HLS report by opening **`/home/ubuntu/lab4/cnn.prj/solution/syn/report/kernel_cnn_csynth.rpt`** with a text editor. This file will be generated with the command **`make vitis`**. **You must submit this file with your final submission. You should not modify this file in your submission, and it will be all verified after submission due. Any modification to this file in your submission constitutes academic misconduct and will be reported.**

General Tips

- When you develop on AWS, to resume a session in case you lose your connection, you can run `screen` after login. You can recover your session with `screen -DRR`. You should ***stop*** your AWS instance if you are going to come 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). You should ***terminate*** your instance if you are not going to come back and resume your work. ***Data on the instance will be lost***.
- You are recommended to use ***private*** repositories provided by GitHub to backup your code. ***Never put your code in a public repo to avoid potential plagiarism.*** To check in your code to a private GitHub repo, create a repo first.

```
git branch -m upstream
git checkout -b main  # skip these two lines if you are reusing the folder in Lab
1
... // your modifications
git add cnn.cpp
git commit -m "lab4: 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 main
```

- You are recommended to `git add` and `git commit` often so that you can keep track of the history and revert whenever necessary.
- ***Make sure your code produces correct results!***

Submission

You need to report the estimated performance results of your FPGA-based implementation on a Xilinx Ultrascale+ xcu200-fsgd2104-2-e. Please express your performance in GFLOPS and the speedup compared with the starter-kit version**.** Your report should also include:

- Please explain the **parallelization and optimization strategies** you have applied for each loop in the CNN program (convolution, max pooling, etc) in this lab. Include the pragmas (if any) or code segments you have added to achieve your strategy.
- Please incrementally **evaluate each parallelization/optimization** that you have applied and explain why it improves the performance.
- Please explain **how your strategy differs** from your Lab 3, and **why**.
- Please report the **FPGA resources (LUT/FF/DSP/BRAM) usages**, in terms of resource count and percentage of the total. Which resource has been used most, in terms of percentage?
- *Optional*: The challenges you faced, and how you overcame them.
- **(*Bonus +5pts)***: Analyze your code and check if the DSP/BRAM resource usage matches your expectation. Only the adders, multipliers, and size of arrays need to be considered. Please attach related code segments to your report and show how you computed the expected number. Provide a discussion on possible reasons if they differ significantly.

You also need to submit your optimized kernel code. Do not modify code in the `lib` directory. Please submit on Gradescope. Your final submission should contain and only contain these files individually:

```
├ cnn.cpp
├ kernel_cnn_csynth.rpt
└ lab4-report.pdf
```

File `lab4-report.pdf` must be in PDF format.

To create the zip, please run:

```
make zip
```

Grading Policy

Your submission will only be graded if it complies with the formatting requirements. Missing reports/code or compilation errors will result in 0 for the corresponding category(ies).

Correctness (50%)

Please check the correctness using the command “**`make csim`**”.

Performance (25%)

Your performance will be evaluated based on the estimation report generated using the command "`make estimate`". 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 (GOPS). Ranges A+ and A++ will be defined after all the submissions are graded:

- Range A++, better than Range A+ performance: 25 points + 5 points (bonus)
- Range A+, better than Range A performance: 25 points + 3 points (bonus)
- Range A GFlops [92, 121): 25 points
- Range B GFlops [63, 92): 20 points
- Range C GFlops [34, 63): 15 points
- Range D GFlops [5, 34): 10 points
- Slowdown: 0 points

Report (25%)

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

Academic Integrity

All work is to be done individually, and any sources of help are to be explicitly cited. You must not modify the HLS report `merlin.rpt` in your submission. Any instance of academic dishonesty will be promptly reported to the Office of the Dean of Students. Academic dishonesty includes, but is not limited to, cheating, fabrication, plagiarism, copying code from other students or from the internet, modifying the software-generated report, or facilitating academic misconduct. We'll use automated software to identify similar sections between **different student programming assignments**, against **previous students' code**, or against **Internet sources**. We'll run **HLS on all submissions and compare the reproduced HLS report with the submitted report**. Students are not allowed to post the lab solutions on public websites (including GitHub). Please note that any version of your submission must be your own work and will be compared with sources for plagiarism detection.

Late policy: Late submission will be accepted for **24 hours** with a 10% penalty. No late submission will be accepted after that (you lost all points after the late submission time).