Winter 2021

CS 133 Lab 3

CPU w/ OpenCL: Convolutional Neural Network (CNN)

Due Date : 2/18/21 11:00PM

Description

Your task is to accelerate the computation of two layers of a convolutional neural network (CNN) using OpenCL on a multi-core processor based on a provided sequential implementation. Specifically, for an input image with 228×228 pixels of 256 channels, you are going to calculate the tensors after going through a 2D convolutional layer with 256 filters of shape $256 \times 5 \times 5$ using the ReLU activation `relu(x) = max{x, 0}` with a bias value on each output channel. The output tensors are pooled using a max-pooling layer of 2×2. If you are familiar with Karas, they are basically `Conv2D(256, 5, activation='relu', input_shape=(256, 228, 228), data_format='channels_first')` with a given bias and `MaxPooling2D(pool_size=(2, 2))`. You will need to implement this function using OpenCL:

```
void CnnKernel(__constant float* input, __constant float* weight,
               __constant float* bias, __global float* output)
```

where `input` is the input image of size `[256][228][228]`, `weight` is the weights of the convolutional filters of size `[256][5][5]`, `bias` is the offset value added to the output of the convolutional per filter of size `[256]`, and `output` should be written to by you as defined above to store the result of `maxpool(relu(conv2d(input, weight) + bias))`. The output size is `[256][112][112]`.

If you are not familiar with those machine learning concepts and cannot understand this description, you can **refer to the provided sequential implementation `lib/cnn.cpp`**. The details will also be presented in the discussion session. Figure 1 shows the CNN layers that we will work on in this lab:

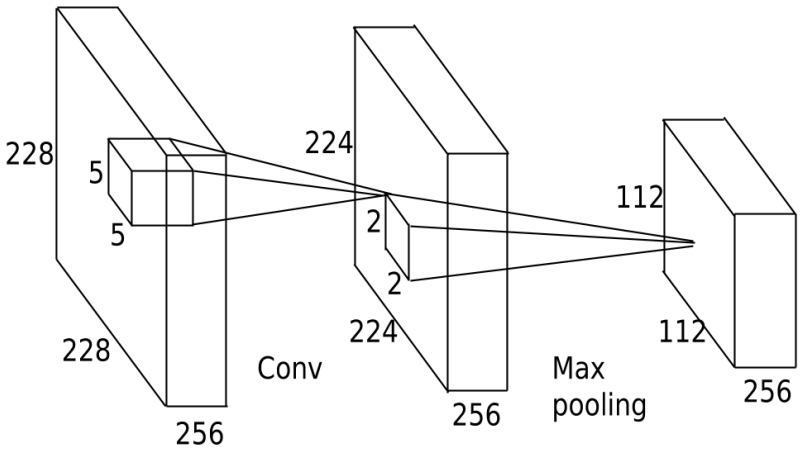

Figure 1: The CNN layers to be accelerated

How-To

Build and Test OpenCL Example: Vector Add

We have prepared the starter kit for you at GitHub. Log in to your AWS instance (recommended) or use the terminal of your own computer to run the following commands:

```
git clone https://github.com/UCLA-VAST/cs-133-21w -o upstream  # replace with
    `(cd cs-133-21w && git pull upstream main)` if you are reusing the folder
cd cs-133-21w/lab3
./setup.sh
make test-vadd
```

Important: Before running `./setup.sh` on an operating system other than macOS, please read and make sure you agree to the End User License Agreement for the Intel(R) Software Development Products and consent to the Intel(R) Software Improvement Program. Otherwise, please manually install an OpenCL runtime yourself of your choice.

It should run and show "PASS". We only provide support for Intel-based Ubuntu or any macOS. If you are running another environment, you can still try "`./setup.sh`", install **Intel® CPU Runtime for OpenCL™ Applications** on your operating system manually, or install the corresponding OpenCL runtime for your processors from the vendor. **You are recommended to use AWS instances for development in this lab if you are not using any of the supported environments. If you developed on macOS, you may want to fine-tune your parameters due to the different size limit of the stack.**

Run CNN with OpenCL

If you have successfully built and run the vector-add OpenCL example, you can now start to create your CNN kernel. The provided starter kit will load the test data from binary files and verify your results with the ground truth data. If you want to test your own test data, you can modify `LoadData` in `lib/cnn.cpp` by writing to the `input`, `weight`, and `bias` variables, and `ground_truth` in `Verify`.

Your task is to implement a fast, parallel version of CNN kernel we defined. You can start with the sequential version provided in `lib/cnn.cpp (CnnSequential)`. You should edit `cnn.cl` for this task.

To adjust the workgroup parameters, edit `params.sh`. For example, if you would like to set the global work size to be (1, 1, 1), you should change `params.sh` to `export OPENCL_WORKGROUP_GLOBAL='1 1 1'`. The workgroup size doesn't have to be 3-dimensional. You cannot put spaces around the equal sign (=) in `params.sh`. If your workgroup size is multi-dimensional, you cannot omit the quote marks (').

To test your implementation of CNN kernel:

```
make
```

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

```
Found 2120190 errors
FAIL
```

Your **performance** will be shown after `Perf`. You can run **the sequential kernel** by `make test-seq`.

Create an AWS Instance for Performance Experiments

Please refer to the discussion section slides and create an `m5.2xlarge` instance with Ubuntu Server 18.04 AMI. Please use your AWS educate classroom account for this lab.

Tips

- **We will use `c5.2xlarge` instances for grading, which should be faster than `m5.2xlarge`.**
- If 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 in days or weeks. ***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.cl
git commit -m "lab3: 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 performance results of your OpenCL implementation with a multi-core CPU on an `m5.2xlarge` instance. Please express your performance in GFlops and the speedup compared with the sequential version and **summarize your results in a table.** Your report should summarize:

- Please explain the parallelization strategies you applied for each step (convolution, max pooling, etc) in this lab. Why did you choose such a strategy?
- Please describe any optimization you have applied.
- In terms of the execution time, please make a comparison with the given sequential version, and discuss the scalability of your parallel implementation using 1, 2, 4, 8, 16, 32 work-items (or any different set of work-items depending on your implementation, with at least 6 tests). What is the global/local work size that gives you the best performance?
- *Optional*: Please discuss the challenges you faced, and how you overcame them.
- ***(Bonus +5pts)***: Please evaluate the performance of at least three (3) different optimization techniques that you have incrementally applied and explain why such optimization improves the performance. Changing parameters does not count as one optimization. Significant code

changes are needed between versions to be counted. In your report, please include the most important changes in your code for each optimization.

You also need to submit your optimized kernel code and the best-performing parameter settings. Do not modify the host code in the `lib` directory. Please submit on Gradescope. You will be prompted to enter a Leaderboard name when submitting, please enter the name you want to show to your classmates. You can submit as many times as you want before the deadline to improve your performance. Your final submission should contain and only contain these files individually:

```
├ cnn.cl
├ params.sh
└ lab3-report.pdf
```

File `lab3-report.pdf` must be in PDF format exported by any software.

Grading Policy

Your submission will be automatically sent to a `c5.2xlarge` instance for evaluation. **Your last Correctness and Performance marks will be final** if there is no cheating. You can see the performance of other students on the Leaderboard. You could request a manual regrade if there is a significant discrepancy between the performance on your instance and Gradescope.

Your submission will only be graded if it complies with the formatting requirements. In the case of missing reports, missing codes, or compilation errors, you will receive 0 for the corresponding category/categories. Please fix your files and resubmit if the AutoGrader returns you an error.

Correctness (50%)

Please check the correctness. You can see this score on Gradescope soon after your submission.

Performance (25%)

Your performance will be evaluated based on the workgroup settings you set in `params.sh`. 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 (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 [144, 204]: 25 points
- Range B GFlops [64, 144): 20 points
- Range C GFlops [24, 64): 15 points
- Range D GFlops [4, 24): 10 points
- Lower than range D [0, 4): 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. 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, 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**. 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 **17 hours** with a 10% penalty. No late submission will be accepted after that (you lost all points after the late submission time).