Winter 2020

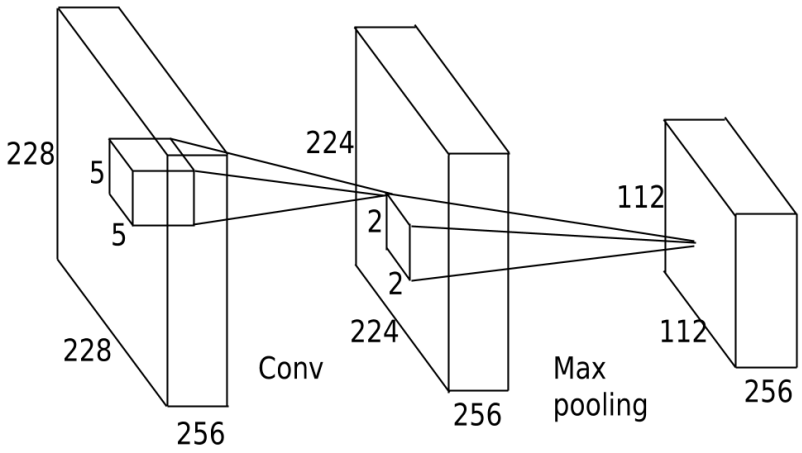
CS 133 Lab 3

CPU w/ OpenCL: Convolutional Neural Network (CNN)

Due: 2/20/20 10:00PM

Description

Your task is to accelerate the computation of a layer of convolutional neural network (CNN) using OpenCL with a multi-core CPU. The details of the algorithm will be discussed in the discussion session. Figure 1 shows the CNN layer that we will work on in this lab:



Tip

- We will use `m5.2xlarge` instances for grading.

Preparation

Create an AWS Instance

Please refer to the tutorial slides and create an `m5.2xlarge` instance with an Ubuntu 18.04 AMI.

Run OpenCL Example: Vector Add

We have prepared the host code for you at GitHub. Log in to your instance and run the following commands:

```
git clone https://github.com/UCLA-VAST/cs-133-20w -o upstream
cd cs-133-20w/lab3
./setup.sh
make test-vadd
```

It should run without error and finish in a few seconds.

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).
- 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 with OpenCL

If you have successfully launched an instance and run the vector add code, you can start to create your CNN kernel. The provided code will load test data and verify your results against a ground truth.

Your task is to implement a fast, parallel version of CNN. You can start with the sequential version provided in `cnn.cpp`. You should edit `intel.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 uncomment the second line of `params.sh` by deleting the leading pound sign (`#`). Note that 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 (`'`).

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 intel.cl params.sh
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 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.
- You can run the sequential CNN by `make test-seq`.
- If `make test` fails, it means your code produces a wrong result.
- ***Make sure your code produces correct results!***

Submission

You need to report the performance results of your CPU-based OpenCL implementation on an `m5.2xlarge` instance. Please express your performance in GFlops and the speedup compared with the sequential version. **Please summarize your results in a table**. In particular, you need to submit a brief report which summarizes:

- 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. (*Optional, bonus +5*: Evaluate the performance of at least 3 different optimization techniques that you have incrementally applied and explain why such optimization improves the performance. Simply changing parameters does not count and sufficient code change is needed between versions. In your report, please include the most important changes you have applied to your code for each optimization.)
- 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). What is the global/local work size that gives you the best performance?
- *Optional*: The challenges you faced, and how you overcame them.

You will need to submit your optimized kernel code and the parameter settings. Please do not modify or submit the host code. Please submit to CCLE. Please verify the ***correctness*** of your code before submitting it.

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

```
<Your UID>.tar.gz
└ <Your UID>
  ├ intel.cl
  ├ params.sh
  └ lab3-report.pdf
```

If your submission doesn't follow this, it will not be graded so please be careful. Note that the script that we're going to use is **case-sensitive** (so don't name your file Lab3-report.pdf). File `lab3-report.pdf` must be in PDF format. You should make the tarball by copying your `lab3-report.pdf` to the `lab3` directory and running `make tar UID=<Your UID>`. If you made the tarball in other ways, you ***MUST*** put it in the `lab3` directory and check by running `make check UID=<Your UID>`.

Grading Policy

Submission Format

Your submission will only be graded if it complies with the requirement. In the 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%)

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:

- 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
- Speed up lower than range D: 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 or against popular Internet sources.

Students are not allowed to post the lab solutions on public websites (including Github).

Late policy: Late submission will be accepted for **12 hours** with a 10% penalty. No late submission will be accepted after that (i.e., work that is more than two days late will lose all the points).