Korea Advanced Institute of Science and Technology
CS492 Spring 2021
Project 4

Jinwoo Hwang
jwhwang@casys.kaist.ac.kr

Due: May. 21th 11:59 PM KST

Rules

- Do not copy from others. You will get huge penalty according to the University policy. Sharing of code between students is viewed as cheating.
- Every code was written and will be tested in the provided docker environment. You need to implement your code in the given environment.
- Read carefully not only this document but the comment in the skeleton code.
- This is the version 1.0 document which means not perfect. Any specifications or errors can be changed during the project period with announcement. Stay tuned to Piazza.

Introduction

In this project, you are going to optimize the DNN graphs from the previous project by exploiting the massive parallel compute power from the GPUs.

In this project you will learn (1) how to write GPU accelerated kernels and (2) how to use BLAS library provided from Nvidia.

Project Structure

This project contains the following files:

- `examples/`: Example files for the two strategies
- `src/`: Reference codes of project 2
- `sample.jpg, sample_out.jpg`: Sample input, output image file
- `Makefile`: Makefile to run the code

Read the content of the `Makefile` carefully and make sure your code works with the following four commands since it will be used when grading your codes.

```
$ make {run_cuda|run_cuda_debug|run_cublas|run_cublas_debug}
```

Strategy

CUDA

CUDA is a computing platform that enables parallel programing with GPU. You can set the behavior of one computation unit in GPU then annotate how the work should be parallelized onto each compute units in the GPU. Check `examples/CUDA` for example. See how the inputs are initialized before they are fed to the GPU and how the `add` function is implemented and called.

cuBLAS

Like the OpenBLAS we used in the previous project, cuBLAS is another library to accelerate Basic Linear Algebra Subprograms (BLAS). cuBLAS specifically aims optimizations on the Nvidia GPUs and are known to be hand optimized in assembly level by the Nvidia developers. Furthermore, cuBLAS could further be accelerated using the Tensor Cores deployed in the recent Nvidia GPUs, which is a custom execution units designed specifically for performing the tensor / matrix operations. Check `examples/cuBLAS/cublas_example_fp16.cu` for example.

Implementation

Overview

Same as before, start from the reference solution for the Project 2 in the `src/`. You may also start from yours if you want. Optimize the computation of `dnn.py` by using the aforementioned libraries one at a time. The name of the files must be `dnn_cuda.py`, `dnn_cuda.c`, `dnn_cublas.py`, `dnn_cublas.c`. You will get full points only if your implementation outperforms the baseline code. Also, implement a debug mode which will save the intermediate values from each layer into `intermediate/layer_#.npy`. You can assume that `intermediate` directory already exists. The code takes `--debug` flag and turning it on will set `DNNInferenceEngine.debug` to `True`.

- CUDA (15pt)
- cuBLAS (15pt)
- Debug mode(5pt)

Running Time Contest (10pt each, 20pt total)

In project 4, we are hosting the inference time contest again. We will measure the inference time of the two versions of your code; CUDA and cuBLAS. The implementation that takes the shortest time will get the highest score. The first-place gets 10 points, the second gets 9 points, the third-place gets 8 points, 4-9th get 6 points, 10-15th get 4 points, 10-22th get 2 points.

Report (10 pt)

Write a detailed report that describes your code. Your report must include

- Key functions for parallelization (e.g., cuBLAS - `cublasSgemm`) and how they work
- Parallelization strategies: which parts in `dnn.py` and how you parallelized them
- Performance gain from offloading the code and analysis for each parallelization scheme.

The running time should be measured in the given KCloud GPU server. The purpose of the report is to check your understanding. Please write the answer clear.

Using the GPU Server

For project 4, you don't have to use the docker image. The GPU server has been set up as the same environment as the docker image. Running your code on the server should not take up the entire GPU memory, but your code might fail if many students are running their code at the same time. In such case, you will have to wait until the GPU is idle. You can check who is using the GPU with `nvidia-smi`, and other tools like `htop` and `who` will be helpful as well.
DO NOT launch python in interpreter mode and import TensorFlow module since it would hold up the GPU resource and prevent your classmates from using it.

Handin

You will need to submit a single tarball containing 4 files(`dnn_*.py, dnn_*.c`) and a report in PDF format. The name of the file should be `[student ID].tar`. Upload your tar file to KLMS.

Reference

The APIs in CUDA and cuBLAS tend to change often as their version changes. Note that guides for different versions might contain misleading information. The CUDA & cuBLAS version in the given docker image is `10.0`.

- CUDA - https://docs.nvidia.com/cuda/archive/10.0/index.html
- CUDA - http://developer.download.nvidia.com/books/cuda-by-example/cuda-by-example-sample.pdf
- cuBLAS - https://docs.nvidia.com/cuda/archive/10.0/cublas/index.html
- cuBLAS - https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf