CS411 Project 1

Rules

- Do not copy from others. You will get huge penalty according to the university policy. Sharing of code between students is considered as cheating.
- 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

This project is designed to learn the basic understanding of ML framework using PyTorch, C and CUDA. Your task is to implement an inference code for object detection using the Yolo-v2-tiny algorithm within the provided mytorch framework. Additionally, you are required to implement the equivalent C and CUDA codes that replicates the functionality of the PyTorch functions. Test code to validate your implementation will be provided.

Setup

Each team will receive a virtual machine id and password. Before diving into the project, you will need to install **NVIDIA GPU Driver**, **CUDA Toolkit** and **other libraries** that are compatible with the project's setting. You will be provided with three shell scripts to help you with your VM setup.

Procedure

1. VPN Program Installation

All the connection to the VM should be done through KCLOUDVPN. Follow the manual at KAIST KCLOUD web page. **For MacOS users** the program specified at the manual is outdated, so that you should download the program from here. Also, mere double clicking does not open the .pkg file. You should right click and open the file with "Open with" option.

2. Initial Project Setting

Run `./scripts/init_project.sh`. It will download the package required to compile GPU Driver, python libraries and the model weight for the project. According to the server network conditions, you may fail to download the large size packages such as PyTorch or CUDA. Please double check whether the installation has been aborted or not.

3. GPU Driver Installation

The GPU Driver installation script is placed at your home directory. Run `~/NVIDIA-Linux-x86_64-510.73.08-grid.run`. You can simply select all the recommended options. After installation, run `./scripts/post_driver_installation.sh`. Use `nvidia-smi` to validate the installation.

4. CUDA Toolkit Installation

Run `cuda_11.6.0_510.39.01_linux.run`. **You must opt-out any installation options except CUDA toolkit**. Installing the others via this script will conflict with the GPU Driver installed in the previous step. After installation, run `./scripts/post_toolkit_installation.sh`. It will set a path variable for CUDA compiler, `nvcc`. You should add path to current terminal by using `source ~/.bashrc`. Use `nvcc --version` to validate the installation.

Running in Local Environment

The scripts provided for the setup is not general enough to be run at other machines. The things to setup on your own when using your local machine are the followings:

- Install the GPU driver and CUDA toolkit that matches your environment
- Install the PyTorch compatible to the CUDA runtime
- Modify the Makefile functions compatible to the OS of the machine

Note Whichever environment you are using, the project will be graded at the given virtual machine environment. Before submitting the code, double check whether your code is runnable at the given VM.

Goals

In this assignment you will practice constructing DNN with high-level frameworks and implementing the backend of it with C and CUDA. The goals of this assignment are as follows:

- Understand the basic **Object Detection pipeline** and implement the **main classification network** with given framework.
- Understand how **ML framework backend** works, and implement it with **C and CUDA**.
- Understand the **GPU Architecture** and apply the **CUDA optimization technique**.
- Write a report about the experiment results.

Write a Comment for `layer_preproc` at `./lib/util.py` (10 pt)

`layer_preproc` is a decorator function that processes numpy array into the appropriate datatype for current RunMode. Walk through the code and write description for the function at the report.

You will get a full point if you describe at `layer_proc` todo comment.

Construct Yolo-v2-tiny Network Using `mytorch.functional` (15 pt)

Yolo-v2-tiny network is consisted of three sequential processes as mentioned in the previous lecture: preprocessing, fine-tuned classification network and postprocessing. The preprocessing and postprocessing code will be given at `./lib/util.py`. Given the image preprocessing code and weight parser code, you will implement the classification

network using functions at `./lib/mytorch/functional.py`. The network will take preprocessed image and weights as inputs, and will return the result tensor. Network configuration is given at the `./references` directory. Provided onnx model can be visualized with the web application, Netron. By running `python3 -m main` you will get the prediction result as `./prediction.jpg`, finding a dog and a car. You can test the network by comparing the result with given `./sample_prediction.jpg`.

Implement C and CUDA Backend for `mytorch` (25 pt)

mytorch is initially set to run with PyTorch backends. Your job is to implement a C and CUDA shared library and to integrate it with mytorch. The typecasting before and after running C or CUDA backend is implemented at `layer_preproc` decorator inside the file `./lib/util.py`.

The naive implementation is to loop through all variables of the output tensor. In C, the outermost loop iterates through each variable of the output tensor, populating values with corresponding activations, kernels and biases. In naive CUDA code, each thread will calculate a single variable of the output tensor, reading corresponding inputs from the global memory. Note that your naive CUDA implementation should not use shared memory.

You will get a full score if you pass all the validation test at `lib/test/base.py`. Run `python3 -m lib.test.test {C,CUDA} validations` to validate your code.

Apply CUDA Optimization Technique (20 pt)

There are various optimization techniques in CUDA programming that consider memory access patterns and reusability. In CUDA, threads in the same block shares the same on-chip SRAM space. Apply the optimization technique considering this CUDA property. There is no requirement for a significant increase in performance, but the trends in performance changes should be thoroughly explained. There is a well-explained post that explains how CUDA works by optimizing general matrix multiplication step by step. You can check the execution times by running scripts. `python3 -m lib.test.test {CUDA,CUDAOptimized} ${FUNC}`. Candidates for functions are stress_conv2d, stress_batch_norm, stress_leaky_relu, stress_maxpool_2d and stress_pad.

Write a Report (30 pt)

Write a report of maximum two pages long. Your report must include the followings:

- Brief explanation of your code and `layer_preproc`
- Theoretical numbers of calculation, memory load and store of each layer for both C and CUDA for every stress test
- Real execution time of each layer with respect to the implementation
- Analysis of the experiment result considering the theoretical number and execution time

The purpose of the report is to show your understanding. Please write the answer concisely.

Handin

Your final outcome should be a tar file with your codes and a report in pdf format. Run `./scripts/handin.sh`. The code will contain:

- `./lib/model.py` : Yolo-v2-tiny implementation
- `./lib/mytorch/{activation,batchnorm,conv,pad,pooling}.py` : Five python files of mytorch layer implementation
- `./src/` : Directory with source codes for shared C and CUDA library
- `./Makefile` : Makefile for src compilation
- `./log/{c,cuda,cuda_optimized}.log` : Log file created after executing `./scripts/log.sh`

The name of the file should be `[team number].tar` and `[team number].pdf`. Please upload your files to KLMS.

Author

- Sangyeop Lee (sangyeop-lee@casys.kaist.ac.kr)