# Multicore and GPU Programming

**Prof. Lee Jinho**

Due - 6/10 Friday 11:59 pm

## 0 Introduction

In this project, you will implement a CUDA application of DNN with GPU Programming. The specification has the following contents. The skeleton is almost identical to previous assignments. Note that you need to use CUDA.



### !!!Caution: Plagiarism!!!

You need to be careful to comply with plagiarism regulations. There is an ambiguity between discussion, idea, and solution. Therefore, be careful not to take away the opportunity of your colleague.

- No web code-searching
- No code sharing
- No idea sharing
- Ok discussion about the project

# 1 Environment

This homework will proceed on 'VESSL', cloud-based on-demand GPU server.
For more information, please refer to this link.
(https://vesslai.notion.site/Yonsei-Dept-of-AI-VESSL-50964c741e6a4f9d8ba33f1e63fa5539).
We expect that this will ease the too-long-wait problem on the condor queue.

On the other hand, we also provide our server in the same manner as in previous homeworks. In this homework, our server is provided only for debugging purposes, not grading.

You don't need to set up the environment. We prepared everything you need to do an experiment about GPU programming. Check GNU GCC/G++ 9.2.1, GNU make, CUDA 11.3 and HTCondor documentation for more information.

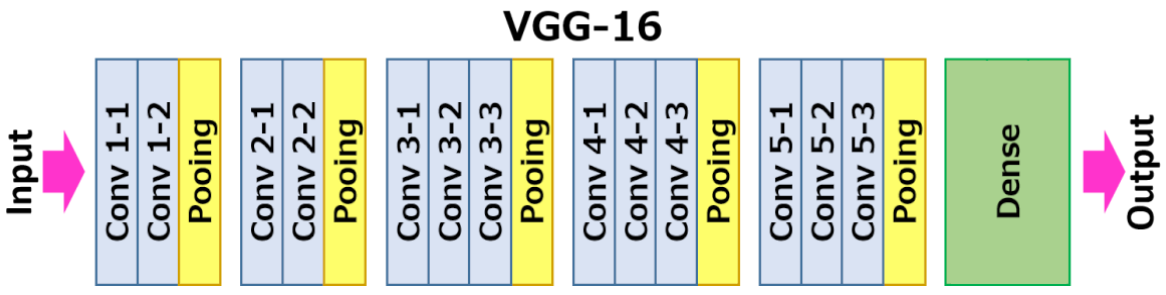
In our worker node (acsys02) and the login node (acsys01), there are RTX2060 GPUs that have compute capability 7.5. However, the gradining will be done on 'VESSL' environment, which has a **Tesla K80 GPU**. Try to search for its spec and tune accordingly.

# 2 CUDA-VGG

## 2-1 Related Things

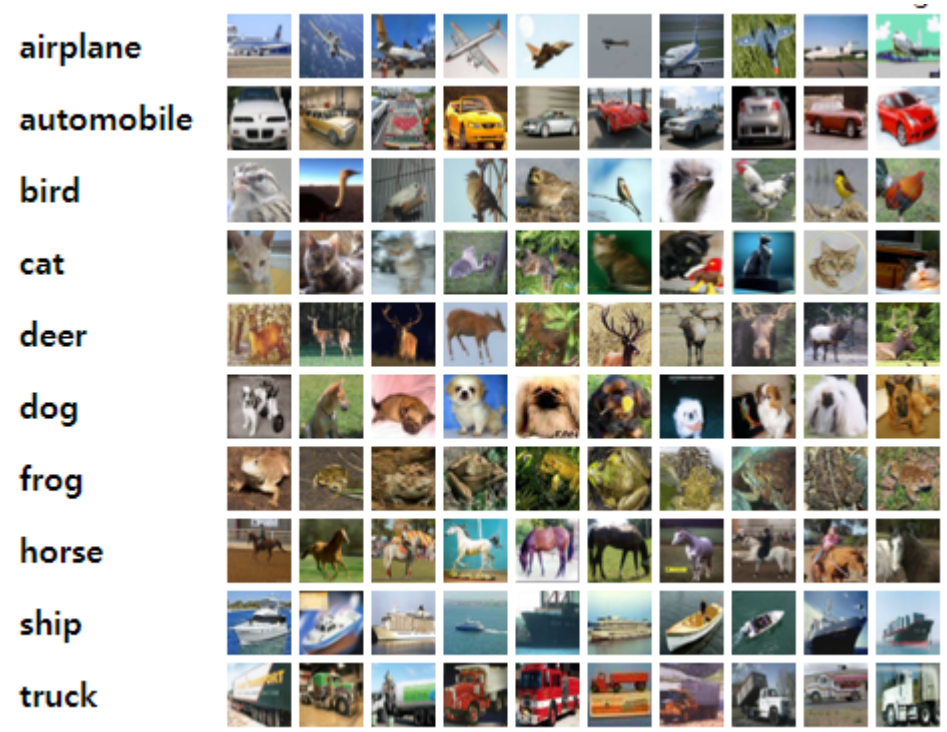
### 2-1-1 VGG16 Architectures

There are 13 convolution layers and 4 max-pooling layers. Then one fully connected layer are applied. You will implement these layers in CUDA version.




### 2-1-2 CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. You will use 10000 test images to inference the class of image.




## 2-2 Explanation about the structure of skeleton code

```
user@acsys:/HW7$ tree .
.
├── build                    # objective file will be there
├── hello_cuda.cmd               # Example for running cuda on
                           #     condor server
├── predict.cmd                  # Run predict on condor server
├── image                    # Images used by README.md
│   ├── cifar10_samples.PNG          # CIFAR-10 samples
│   ├── vgg16.PNG          # VGG16-architecture
│   └── LeNet5-architecture.jpg       # LeNet5-architecture
├── Makefile                  # Makefile
├── model                    # Provided pre-trained model
│   └── main.py                  # Used code to train
                           #     test accuracy 79%
├── README.md                   # What you are reading :3
├── run_on_vessl.sh        # Shell script for run your code on VESSL using CLI
├── result                   # Results from condor server
├── src                     # Source Code
│   ├── common.h                 # Common
│   ├── vgg16.h                 # Base VGG16
│   ├── vgg16.cpp
│   ├── vgg16_cpu.h              # CPU VGG16
│   ├── vgg16_cpu.cpp
│   ├── vgg16_cuda.h             # CUDA VGG16
│   ├── vgg16_cuda.cu
│   ├── main.cpp                # main program
```

```
│   ├── util.h                  # Util
│   └── util.cpp
└── tmp                         # program will save image and
                            #     prediction to here

6 directories, 16 files
```

***./predict help*** will print followings description about arguments.(You should compile first!)

```
user@acsys:/HW7$ ./predict help
[ERROR] Invalid arguments
Usage: ./predict INPUT_PATH DATA_OFFSET BATCH IMG_PATH_TEMPLATE
   INPUT_PATH: path to input data, e.g.
/nfs/home/mgp2022_data/hw7/cifar10/test_batch.bin
   DATA_OFFSET: data_offset for input data, e.g. 0
   BATCH: batch size to inference, e.g. 1
   IMG_PATH_TEMPLATE: path template to img, %d will data_offset and %s will be
label, e.g. tmp/cifar10_test_%d_%s.bmp
   PARAMETER_PATH: path to parameter, e.g.
/nfs/home/mgp2022_data/hw7/vgg_weight/values_vgg.txt
```

## 2-3 What you have to do

### 2-3-1: Check the data

!!!!!Don't write or update the data!!!!!

```
user@acsys:/HW7$ ls /nfs/home/mgp2022_data/hw7/cifar10/
batches.meta.txt  data_batch_2.bin  data_batch_4.bin  download_cifar10.sh
test_batch.bin
data_batch_1.bin  data_batch_3.bin  data_batch_5.bin  readme.html
```

### 2-3-2: Implement vgg16_cuda

Implement **normalize**, **conv**, **relu**, **pool**, **fc** with CUDA at **src/vgg16_cuda.cu**. There is a naive(1 thread, 1 block) implementation given already.

You should use 32 bit float or 64 bit double on CUDA. In other words, you can't use 8 bit, 16 bit quantization.
You should NOT modify any of main.cpp. We will give you a significant penalty if it is modified from the original code. If you think some modification is necessary, please contact us. (You may only need to modify **vgg16_cuda.cu, vgg16_cuda.h** )

In general, we have designed our skeleton code so that you only have to fill **vgg16_cuda.cu:predict()** with your own kernels. However, you can change the class structure at your will if you are not happy with what we've provided, as long as you don't touch main.cpp.

Check **model/main.py** to see original python code. Check **src/vgg16_cpu.cpp** to see converted c++ referenced code.

### 2-3-3: Compile and run on condor server

```
user@acsys:/HW7$ make run_on_server
mkdir -p result
condor_submit predict_b128.cmd
Submitting job(s).
1 job(s) submitted to cluster 14045.
```

### 2-3-4: Check the result

```
user@acsys:/HW7$ cat result/vgg16.out
[INFO] Arguments will be as following:
   INPUT_PATH: /nfs/home/mgp2022_data/hw7/cifar10/test_batch.bin
   DATA_OFFSET: 0
   BATCH: 128
   IMG_PATH_TEMPLATE: tmp/cifar10_test_%d_%s.bmp
   PARAMETER_PATH: /nfs/home/mgp2022_data/hw7/vgg_weight/values_vgg.txt
[INFO] Initialize variables
[INFO] Allocate memories
[INFO] Read image from data_offset 0 at
/nfs/home/mgp2021_data/cifar10/test_batch.bin
[INFO] Save image to tmp/cifar10_test_0_cat.bmp
[INFO] Save image to tmp/cifar10_test_1_ship.bmp
[INFO] Save image to tmp/cifar10_test_2_ship.bmp
[INFO] Save image to tmp/cifar10_test_3_airplane.bmp
[INFO] Save image to tmp/cifar10_test_4_frog.bmp
…
…
...
[INFO] Save image to tmp/cifar10_test_125_airplane.bmp
[INFO] Save image to tmp/cifar10_test_126_ship.bmp
[INFO] Save image to tmp/cifar10_test_127_cat.bmp
[INFO] CUDA elapsed time is 201.641 msec
[INFO] CUDA predict is as following:
CPU:CLASS(NUMBER,T/F),CUDA:CLASS(NUMBER,T/F),Label:CLASS(NUMBER)
CPU:       cat(3,1), CUDA:       cat(3,1), Label:       cat(3)
CPU:      ship(8,1), CUDA:      ship(8,1), Label:      ship(8)
```

```
CPU:       ship(8,1), CUDA:       ship(8,1), Label:       ship(8)
CPU:   airplane(0,1), CUDA:   airplane(0,1), Label:   airplane(0)
CPU:       frog(6,1), CUDA:       frog(6,1), Label:       frog(6)
….
….
….
CPU:       ship(8,1), CUDA:       ship(8,1), Label:       ship(8)
CPU:       ship(8,0), CUDA:       ship(8,0), Label:   airplane(0)
CPU:       ship(8,1), CUDA:       ship(8,1), Label:       ship(8)
CPU:        cat(3,1), CUDA:        cat(3,1), Label:        cat(3)

Correct

CPU error:21.0938% GPU error:21.0938%
```

### 2-3-5: Run on VESSL

First of all, each user need to sign up and configure VESSL environment. Refer to the 'VESSL Instruction.ppt'. Then, you can run your code on VESSL by 'run_on_vessl.sh'.

```
mgp100**@acsys01:~/HW7$ ./run_on_vessl.sh
/usr/lib/python3/dist-packages/requests/__init__.py:91:
RequestsDependencyWarning: urllib3 (1.26.9) or chardet (3.0.4) doesn't match a
supported version!
  RequestsDependencyWarning)
Organization: yonsei-acsys
Project: cuda-vgg
Uploading... [/nfs/home/MGP/mgp100**/HW7/src -> /home/vessl/local/]
Upload completed successfully.
Mount Info:
 /output/ -> (output)
 /home/vessl/local/ -> (Uploaded)
Created '@@'.
For more info: https://vessl.ai/yonsei-acsys/cuda-vgg/experiments/@@
```

### 2-3-6: Have fun speeding up!

- Matrix Multiplication Techniques
- Im2Col
- Relu, FC, etc
- Kernel fusion
- (We will check the performance with only batch=128)

## 2-4 Plus Alpha

### 2-4-1 Pretrained Model

There is a pre-trained model at **/nfs/home/mgp2022_data/hw7/vgg_weight/values_vgg.txt**. Loading pre-trained model from the txt file is already implemented in **src/vgg16.cpp**.

You can refer to code(**model/.main.py**).

Activations are outputs of each layer with index 0 image(cat) at test data.

# 3 Criteria

## 3-1 Requirements

### 3-1-0 General

- You should implement with CUDA.
- No external libraries. You can only use STL.
    - This of course means: No cuBLAS, No cuXXX
- CPU-only version is not acceptable. You have to use the GPU.

### 3-1-1 CUDA-VGG

- Problem: Inference of 128 images
- Performance requirements: 4.0 sec on VESSL
- You should implement entire VGG16 with cuda
- Your kernels should be functionally equivalent to the cpu version
- The trained model is supposed to give around 80% accuracy. We have set a huge margin of +-5% for the difference of the cuda version and the reference C++ version.
- If you fail to implement some part of the kernel in cuda, you can use the CPU version. However, it is your job to make sure to cudaMemcpy() so that the function still works correctly.
- No external libraries. Especially cuDNN. If you think you need something and it's not about cuda programming, contact us before doing so.
- We will measure the performance of batchsize=128

### 3-2 Measurements

We are measuring performance from VESSL environment, however it still involves a certain amount of luck. To reduce the luck effect as much as possible, we do the following:

- We will run your program at least five times and **use the minimum exec.time**.

## 4 Report

Your report should include

- What techniques you have implemented
- How to run your code
- How each technique affected your performance (+ comparison)
- Why you think your technique, or your combination of techniques produced the best result
- max 4 pages (firm)
- **PDF only**
- If 4 pages is too short to contain everything you want to say, use a double-column format (e.g., https://ieeecs-media.computer.org/assets/zip/Trans_final_submission.zip , https://ieeecs-media.computer.org/assets/zip/ieeetran-final_sub.zip )

## 5 Grading (will be scaled up to 200)

**Correct parallel implementation (30)** - finish faster than the reference code, produce **correct** result with a CUDA implementation. Plus, whether your code produce correct result or not. We have set a huge margin of +- 5% of the original model's accuracy.
**Performance bar (20)** - Based on last year's students and many criterion, we have set the bar as 4 seconds on VESSL, which is not too challenging. If your execution time is x seconds where x > bar, here's your score:

*score = (20*bar/x)*

We will not change performance bar. (we will not ask you to make a faster program in the middle)
Please keep in mind that, all the performance and ranking score will be graded based on the performance on VESSL. Local (acsys01,02) performance will not be considered.

**Ranking (30)** - 30 * (65 - rank)/64. The ranking is decided by the execution time of interest defined above.

- You need to pass the performance bar in order to get the ranking point
- We might choose to tie some rankings if the difference is small.

**Report (20)** - Refer to 4. Report

Please note that, correctness, performance, ranking score are affected by its former (correctness -> performance -> ranking). Which means that, if your code did not pass the performance bar, you can have correctness score but no ranking score. On the other hand, if your code did not produce correct result, you cannot get either performance or ranking score.

We will grade your code on VESSL, not our server. Please read VESSL instruction carefully and do your homework on VESSL. As we do grading on VESSL environment, if the code is not running correctly on VESSL environment, you cannot get performance and ranking score. In such case, you can get small amount of correctness score if your code only works on local machine (acsys01,02).

# 6 More materials

- GNU GCC/G++ 9.2.1
- GNU make
- CUDA
- HTCondor
- GAP Benchmark
- SNAP Dataset
- PageRank
- Very Deep Convolutional Networks for Large-Scale Image Recognition
- CIFAR-10 dataset
- Neural Network Tutorials - Pytorch