CSCI-GA.3033-022 – Lab3

This lab is intended to be performed **individually**, care will be taken in verifying that students are authors of their own submission. Coding exercises are identified by **C<number>**.

C1 – Tiled Convolution in CUDA

In this part of the lab we implement a convolution of an image using a set of filters. Consider the following:

- An input tensor *I* with dimensions: C, *H*, *W*. Each element of *I* is generated as follows:

  $$I[c,x,y] = c \cdot (x+y)$$

- A set of convolution filters with dimensions: *K*, *C*, *FH*, *FW*. Each element of the filter *F* is generated as follows:

  $$F[k,c,i,j] = (c+k) \cdot (i+j)$$

  Filter dimensions can be restricted to be odd.
- Dimensions are: H=1024, W=1024, C=3, FW=3, FH=3, K=64
- All tensors have *double* elements (double precision)
- The output tensor *O* with dimensions: *K*, *W*, *H*. Each pixel of the output tensor $O[k,x,y]$ is obtained as:

$$O[k,x,y] = \sum_{c=0}^{C-1} \sum_{j=0}^{FH-1} \sum_{i=0}^{FW-1} F[k,c,FW-1-i,FH-1-j] \cdot I_{\square}[c,x+i,y+j]$$

Note that we need to use the transpose of the filter in order to compute a convolution rather than a cross-correlation and that we need padding, i.e. accesses outside the dimensions of the image have to be filled in to be 0. Implement a convolution algorithm in CUDA with tiling and shared memory.

- Print the checksum as the total sum of the elements along all their dimensions for both I and O.
- Report the time to copy the image data to the gpu, the time to execute the CUDA kernel with the convolution and the time to copy the result back to the host. Note: You have to use a cuda synchronization primitive to obtain correct timing.
- Run 5 times. The execution times should be very stable (variation <10%) after the first.

Example output (replace 'kernel' with 'cudnn' for C2):

```
I = checksum: <double precision floating point number>
Copy host->dev kernel x.xxxxxx sec
time kernel x.xxxxxx sec
Copy dev->host kernel x.xxxxxx sec
CUDA O = checksum: <double precision floating point number>
```

C2 –Convolution with cuDNN

Implement the convolution of exercise C1 using cuDNN with the CUDNN_CONVOLUTION_FWD_PREFER_FASTEST algorithm selection.

- Print the checksum of I and O along all their dimensions. The checksum is expected to be the same as C1
- Report the 3 times as in C1
- Run 5 repetitions, time variation should be small (<10%) after the first

Grading

The grade will be a total of 50 points distributed as follows:

EXERCISE	DESCRIPTION	POINTS
C1	Tiled Convolution with CUDA using shared memory	35
C2	Convolution with cuDNN	15

Other grading rules:

- Late submission is -3 points for every day, maximum 3 days.
- Non-compiling code: 0 points

Program output

A single binary generated from a single .cu file (no headers) that compiles and runs with the command stated in submission instructions and executes 5 reps of C1 and 5 reps of C2, i.e. prints 10 measurement blocks (described above) as well as a final line generated by

```
printf("\n\n <Time>: Conv %lf sec  cuDNN %lf sec\n", timeConv, timecuDNN);
```

with the average times for both cases across the 5 repetitions.

Failing to respect this format is -3 points.

Running and Submission instructions

Submission through NYU Classes.

Submission format:

- Please submit a tar file with file name *your-netID*.tar with a folder named as your netID (example: uf5.tar containing uf5/)
- The folder should contain the following files: A file named lab3.cu with exercises C1, C2 and a text file 'output.txt' with the result of your run.
- Before compiling on Prince make sure you load the cuda and cudnn module:

```
module purge
module load cuda/9.0.176
module load cudnn/9.0v7.0.5
```

- Compile and run with “nvcc -o lab3 lab3.cu -lcublas -lcudnn ; ./lab3”
- Run experiments on Prince with flags –gres=gpu:p40:1 –mem 20GB

Failing to follow the right directory/file name specification is -1 point.