**ECE569**

**Homework 2**

**Individual Submission**

1. **Objective**

The purpose of this lab is to introduce you to the CUDA API by implementing vector operations in the context of addition and color space conversion. You will first implement vector addition by writing the GPU kernel code as well as the associated host code. You will then convert an RGB image into a gray scale image.

1. **Logistics**

In hw1, you created “labs” and “build”dir” directories.

In this Assignment2 folder you will find the “hw2” directory along with the “CmakeLists.txt”.

Place them into your “labs” folder.

Place the “run\_hw2.slurm” script into you “build\_dir” directory.

1. **Task 1: Instructions for Vector Addition**

You will find the temple source code named “solution.cu” in hw2/VectorAdd

Edit this file to perform the following:

* Implement vector addition kernel
* Allocate device memory
* Copy host memory to device
* Initialize thread block and kernel grid dimensions
* Invoke CUDA kernel
* Copy results from device to host
* Free device memory

The code handles the import and export as well as the checking of the solution. Instructions about where to place each part of the code is demarcated by the //@@ comment lines. You should keep all other parts of the template code unchanged.

**Compiling and Execution Instructions**

You will need to compile your code from the “build\_dir” folder similar to the process described in hw1.

From your “build\_dir” run the following two commands:

**$module load cuda11/11.0**

**$make**

The makefile compiles and generates the executable “VectorAdd\_Solution in the “build\_dir” directory. The executable generated as a result of compiling the lab can be run using the following command:

./VectorAdd\_Solution -e <expected.raw> -i <intput1.raw>,<input2.raw> \  
 -o <output.raw> -t vector

where <expected.raw> is the expected output, <input0.raw>,<input1.raw> is the input dataset, and <output.raw> is an optional path to store the results.

You need to generate datasets using the “VectorAdd\_DatasetGenerator” built as part of the compilation process. Run the following command from your “build\_dir” directory:

**$./VectorAdd\_DataSetGenerator**

then check the “build\_dir/VectorAdd/DataSet” folder to access the ten test directories.

input0.raw, input1.raw and output.raw files in each dataset directory are the two input vectors and the expected output respectively.

If you want to experiment with nvprof, the full command is:

nvprof ./VectorAdd\_Solution -e ./VectorAdd/Dataset/0/output.raw -i ./VectorAdd/Dataset/0/input0.raw,./VectorAdd/Dataset/0/input1.raw -o vaddoutput.raw -t vector

Running multiple test cases:

**Review the slurm script provided in hw2 and modify path used in this script (line number: 34). This file should be in your “build\_dir” directory. Make sure that you are still in the “build\_dir” directory.**

**Create the following two directories in your “build\_dir” directory: VectorAdd\_output,** **ImageColorToGrayScale\_output**

**$makdir VectorAdd\_output**

**$mkdir ImageColorToGrayScale\_output**

**You should comment out ImageColorToGrayScale related runs from the slurm script at this point.**

**$srun run\_hw2.slurm**

**This script will run all test cases and place the outputs into your newly generated “VectorAdd\_output” folder.**

The outputs generated after running each test case will be placed in the corresponding output folders you created. The output files will have execution time profile information and will indicate whether you code passed the test case or not.

Suggested exercise: Profile your implementation for a very large vector size. You can generate new test vectors by modifying the “dataset\_generator.cpp” in the hw2/VectorAdd directory.

For running multiple tests you may want to read about running jobs with slurm at

<https://public.confluence.arizona.edu/display/UAHPC/Running+Jobs+with+SLURM>

1. **Task 2: Instructions for Color Space Conversion**

The input is an RGB triple of float values and you will convert that triple to a single float grayscale intensity value. A pseudo-code version of the algorithm is shown below:

for ii from 0 to height do  
 for jj from 0 to width do  
 idx = ii \* width + jj  
 # here channels is 3  
 r = input[3\*idx]  
 g = input[3\*idx + 1]  
 b = input[3\*idx + 2]  
 grayImage[idx] = (0.21\*r + 0.71\*g + 0.07\*b)  
 end  
end

For those who are developing on their own system, the input image is stored in PPM P6 format while the output grayscale image is stored in PPM P5 format. You can create your own input images by exporting your image into PPM images. The easiest way to create image is via external tools. On Unix, bmptoppm converts BMP images to PPM images.

**Instructions for Color Space Conversion**

You will find the temple source code named “solution.cu” in hw2/ ImageColorToGrayscale

Edit the code to perform the following:

* Implement the kernel
* allocate device memory
* copy host memory to device
* initialize thread block and kernel grid dimensions
* invoke CUDA kernel
* copy results from device to host
* deallocate device memory

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

Compiling Your Code: Follow the steps given for the VectorAdd exercise above.

The executable generated as a result of compiling the lab can be run using the following command for a single test case:

./ImageColorToGrayscale\_Solution -e <expected.pbm> -i <input.ppm> -t image > output.txt

where <expected.pbm> is the expected output, <input.ppm> is the input dataset, and <output.txt> stores the results.

The datasets can be generated using the dataset generator built as part of the compilation process. You can locate the dataset generator in your “build\_dir” directory.

**ImageColorToGrayscale\_DatasetGenerator** in the build-dir generates the data set.

**$./ ImageColorToGrayscale\_DatasetGenerator**

then go into the **ImageColorToGrayscale** folder to access the test files.

**You should comment out VectorAdd related runs from the slurm script at this point.**

**$srun run\_hw2.slurm**

**Common Mistakes:**

**Are you using the correct number of channels in your kernel code?**

**You may want to print the number of channels from your kernel code to check**

**If this is fine then in your kernel code, have only thread 0 in block 0 print the values it is receiving and producing.**

**Always have error check for data transfers**

**Are you launching legal thread block configuration. Consider the limitation on number of threads per block.**

**Are you using the “ceil” operator? If so are you typecasting the thread block size related integer variable to float?**

1. **Submission Instructions**

**In the “hw2” folder you will find the file “questions.docx”. You will need to answer the questions.**

**Rename your solution.cu files as VectorAddition.cu and ImageColorToGrayScale.cu. Make sure to use exact naming convention. Refer to penalty conditions below. Your source files should have sufficient comments for the reader to be able to follow your implementation.**

Create a folder and name it as your “net id”. Include the following files:

* **VectorAddition.cu**
* **ImageColorToGrayScale.cu**
* **questions.docx**
* **VectorAdd\_output (folder that contains all output files)**
* **ImageColorToGrayScale\_output (folder that contains all output files)**

**Zip the main folder and submit to the designated D2L folder.**

**Your submission will have total of 23 files**

**questions.docx (35 pts)**

VectorAddition.cu (with proper comments and coding style) (5 pts)

ImageColorToGrayScale.cu (with proper comments and coding style) (10 pts)

Vector addition output files (output0-output9) (5 pts)

Gray scale output files (output0-output9) (5 pts)

Vector Addition 5 pts per private test case (5\*10) (50 pts)

Gray Scale 9 pts per private test case (9\*10) (90 pts)

Total 200 points

Penalty Conditions:

* 50 points penalty for not following submission instructions.
* 20 points penalty per day for late submission.