CSCD 439/539 GPU Computing Lab4

Matrix Multiplication

No Late Submissions are accepted. **Rules:** Your code must use C and CUDA C Language. If your program shows a compilation error, you get a zero for this lab assignment.

**Submission:** Wrap up all your **source files and other data files** into a single zip file. Name your zip file as *FirstInitialYourLastName*Lab4.zip. For example, if your legal name is Will Smith, you should name your zip file as wsmithlab4.zip. A simple makefile has been provided in the zip file.

**Before you leave the laboratory, please show the TA or the instructor how your program works, they will give you a score for this Lab assignment.**

**For archive purpose, please also submit your single zip file on EWU Canvas by following CSCD439-01 Course AssignmentsLab4 Submit Assignment to upload your single zip file.**

**Problem Description:**

Based on the lecture about simple matrix multiplication on CUDA device, you are required to implement the following features and answer the questions.

In the provided lab package, you have three subfolders, **data**, **src** and **sampleCode**. The **data** folder contains all 2D matrices you will play with. The **src** folder has most of the source code you need to perform experiments. The sampleCode provides a single demo code to generate random integer numbers in C.

1, Read the provided code in **src** folder, specifically read the main function in the source file matrix\_multiplication.cu. Answer the questions below,

a) What tasks the program performs?

**The tasks that the program does is multiplying two matrices with the cpu and the gpu together then prints the output into two different files depending on where it was processed. Then it runs the kernel multiple times taking the average of time cost of all the runs. Also it runs the cpu matrix multiplication method and takes the average time cost of all the runs. Finally it reports the effectiveness of the throughput by measuring the floating point operations performed per second.**

b) What are the dependency C files and header files? How to call functions that is defined in another source file?

**ArrayUilts.c/h mul.c/h and timing.c/h. The way to call the functions that are defined is the method name and parameters ex: mul(h\_c, h\_a, h\_b, n)**

2, Based upon the lecture notes, please write the simple kernel function to perform matrix multiplication on GPU, on top of the source file **matrix\_multiplication.cu**.

3, Explore the Makefile, how shall we jointly compile .cu and .c files in a single project?

**The way we combine the .cu and .c files is by making two separate objects files for the cu and c then compiling them into a single executable file**

4, Check the APIs Docs and find out what cudaEventRecord() and cudaEventSynchronize() do?

**CudaEventRecord() - It captures the contents of a stream an event and recalling it on the same event will overwrite the previous capture.**

**CudaEventSynchronize() - It synchronizes work that is capture inside of an event, it synchronizes by waiting for completion of the work and blocks the threads until its done.**

5, In the main function, what is the equation that the program uses to compute the throughput (in unit of GLOPS)? Please interpret the equation.

**The equation that the program uses to compute the throughput is: (one mul + one addition) \* N^3 operations / (average\_simple\_time / 1000.0f) / 1000000000.0f**

**In matrix multiplication in order to find the ouptut at a specific index you need to complete the dot product**

**of the column and the row where the element is located.**

**To do this the addition and multiplication operations are used, which account for the (one mul + one addition) \* N**

**part of the equation.**

**However, to find a specific location of the output 2 more for loops are required.**

**This will account for the N^2 in the equation. Thus multiplying the 2 equations together gives us the entire equation;**

**(one mul + one addition) \* N^3 operations.**

**This part of the equation calculates the number of opperations performed. In order to find the time in milliseconds**

**we need to divide this entire operation by (average\_simple\_time / 1000.0f) / 1000000000.0f.**

6, Run your program after you finish your kernel on the dataset 1024.mat and 2048.mat, how much speedups do you obtain compared with CPU time cost? What are the GPU throughput and CPU throughput you observed in these cases?

**BlockWidth=32**

**The cpu time cost for 1024 is 4.95secs and the gpu is .021secs. The speedup you obtain is 235.77x.**

**Throughput of kernel = 102.22 GFLOPS. Throughput of Cpu = .43**

**The cpu time cost for 2048 is 110.45secs and the gpu is .172secs. The speedup you obtain is 49.59x.**

**Throughput of kernel = 99.65 GFLOPS. Throughput of Cpu = 0.16**

**The observation of the throughput is that depending on the amount of throughput that the gpu has compared to the cpu is the total amount of speedup that you obtain.**

7, Can you find some specification data about the peak performance (GFLOPS) for the GPU that we are using in the Lab (NVIDIA GTX 660 ti ) ? Is the GPU device throughput that you observed in step 6 close to their Peak performance? Guess the reason why they are close or why they are far away?

**The FP64(Double precision floating-point format) is 109.8. The throughput for the gpu is close to the GFLOPS peak performance, I believe the reason why this is close is because you are using multiple threads to complete the tasks.**

8, Refer to the sampleCode provided, you have to add another function in arrayUtils.c and arrayUtils.h, float \* fillArrayRandom( int row, int col), which populates a 2D matrix with random **float** number ranging 0 to 1. Redirect your **input** from data files to these randomly generated matrices, record only the GPU throughput and time cost for matrix multiplication with size 10000 \* 10000, 20000 \* 20000. ( FYI: please comment out the CPU sequential code test because they takes forever to finish. Also please make sure the thread block size is within limit.) If you can use **long long integers** for the size of the array, you can test arrays with much more bigger size.

**The first test of the 10000\*10000 matrix didn’t finish after 2.5 hours.**