CS/EE 217 Lab 1
Simple Matrix Multiplication
Due Wed. Oct 14 at 8pm

1) Unzip lab1-starter into your sdk projects directory

2) Edit the basicsgemm() and the mysgemm(...) function in kernel.cu as well as main in main.cu to complete the functionality of the matrix multiplication on the device. Do not change the source code elsewhere. The size of the matrix is defined such that one thread block will be sufficient to compute the entire solution matrix.

3) There are several modes of operation for the application. Check main() for a description of the modes (repeated below).

a) No arguments: The application will create two randomly initialized matrices to multiply size (1000x1000). After the device multiplication is invoked, it will compute the correct solution matrix using the CPU, and compare that solution with the device-computed solution. If it matches (within a certain tolerance), if will print out "Test PASSED" to the screen before exiting.

b) One argument: The application will use the random initialization to create the input matrices (size mxm, where m is the argument. Start your testing with small matrices.

c) Three arguments m, k, and n: The application will initialize the two input matrices with random values. A matrix will be of size m x k while the B matrix will be of size k x n, producing a C matrix of size m x n

Note that if you wish, you may add a mode to accept input matrices from files, or to dump input and output matrices to files to facilitate testing.

4) Answer the following questions:
1. How many times is each element of the input matrices loaded during the execution of the kernel?

2. What is the memory-access to floating-point computation ratio in each thread? Consider a multiply and addition as separate operations, and ignore the storing of the result. Only global memory loads should be counted towards your off-chip bandwidth

3. Describe two approaches to speedup the computation. You may assume large matrices.

Submission:

Upload a zip file with your updated code and report to iLearn. Please remove all executable files or other unnecessary files from the directory before creating the zip. Please name your report report.txt , report.pdf or report.doc

Grading:

Your submission will be graded on the following parameters.
Correctness: 35%

- Test passes in all three modes.

Functionality: 35%
- Correct usage of CUDA library calls and C extensions.
- Correct usage of thread id's in matrix computation.

Report: 30%
- Answer to question 1: 10%, answer to question 2: 12%, answer to quester 3: 8%

CS/EE 217 Lab 2
Tiled Matrix Multiplication
Due Mon. Oct 26 at 8pm

1) Download the lab2-starter zip file from the class website. Unzip lab2-starter into your sdk projects directory or working linux directory.

2) Edit the source files kernel.cu and main.cu to complete the functionality of the matrix multiplication on the device. The two matrices could be any size, but we will not test your code with an output matrix size exceeding 64,000 elements.

3) There are three modes of operation similar to those in the first lab. The difference in this lab. is that you will support these modes using a Tiled implementation.

4) Answer the following questions:

a-In your kernel implementation, how many threads can be simultaneously executing? Assume a GeForce GTX 280 GPU which has 30 streaming multiprocessors.

b-Use nvcc –ptxas-options="-v" to report the resource usage of your implementation. Note that the compilation will fail but you will still get a report of the relevant information. Experiment with the NviDia visual profiler, which is part of the CUDA toolkit, and use it to further understand the resource usage. In particular, report your branch divergence behavior and whether your memory accesses are coalesced.

c-Compare the performance of the The Tiled Matrix multiplication to the simple matrix multiplication as you increase the size of the matrices and for different tile sizes. Explain any trends that you see.

Grading:

Please upload your zipped directory (after cleaning up executables and any unnecessary files) to iLearn. Your submission will be graded on the following aspects.

Correctness and performance (50%)
1. Produces correct results
2. Shared memory is used correctly (tiling) to improve performance

Report (50%)
Answers to the questions above (10 points for a, and 20 points for each of b and c).

CS217 – LAB2

Shahriyar Valielahi Roshan – 861241935

1- We have 30 SM in GTX 280, and each SM can support up to 1024 active threads; as we used 16*16 block size, the number of threads per block is equal to 256.
   1-1- Number of Threads per block = 256
   1-2- Maximum number of Threads per SM = 1024
   1-3- Maximum number of Blocks per SM = 8

   So with simple calculation we can see that we can completely utilize 1 SM by using only 4 blocks. If we consider a calculation that has enough number of threads, we have:

   Total simultaneous threads: Number of SM * threads per SM = 30 * 1024 = 30720

   Note: We should consider two resource constraints:

   1- Registers: As I read, GTX280 has 64k registers per SM, so each threads in a fully utilized SM can use up to 64 register. PASSED
   2- Shared Memory: In matrix multiplication, each thread use two floating point memory load that means 2*4byte = 8 byte. If we consider 1024 simultaneous threads in each SM, we can reach to the 8KB memory in L1 cache (Shared Memory) which can fit into the memory. PASSED

2- I have read here that I can extract divergent_branch in the visual profiler, but I couldn't find it anywhere. The option to capture and report this feature was not available for the Tesla (k20c) GPU in the Configure Metric, and event menu. But for GS8400 it was available, so I guess that it is different from GPU to GPU.

I do not have access to the storm machine, and I use our lab machine called Sailor. (Embedded Systems Lab)

I choose different parameters for report. In this part I just execute the program without any argument as an input.

Duration	11.281ms
Registers Per thread	25
Shared memory per block	2 kiB
Global Load Efficiency	100%
Shared Efficiency	85.7%
Warp Execution Efficiency	100%
Shared Memory Requested	48kiB
Executed Control Flow Instructions	6128136
Issued Control Flow Instruction	6159888
Branch Efficiency	99%
gld_incoherent	0

From this statistics, I think my implementation has a little Branch Divergence (Only at the Boundaries that in a single warp, several threads may run else part instead of if part), and the memory accesses are also coalesced. Please check the images in the folder of my assignment for more details of visual profiler.

3- In the table below, you can see Launching kernel time for different tile size, and simple version. As you can see, tiled version is better than simple in all cases, because accessing to the shared memory is much faster than the global memory, and we should only pay the overhead of moving data from global memory, to the shared memory at the beginning of the kernel launch, especially in a case will 32 tile width, the ratio of Launching time of Simple to Tiled is 6.84 for 2000 * 2000 matrix. Actually we decrease the ration of calculation to memory access (global) by tile width in the tiled mode.

And about the increasing the size of block, there is a trade of here, for small matrixes, it works, but for the bigger matrixes, because of lack of resources, and other factors, the utilization factor will decrease, and you can see it does not help any more. (Consider 2000*2000 tiles version, and compare the result for tile width of 16 and 32), in the untiled version, increasing the block size makes the situation even worse than the tiled version.

Launching kernel			
	500	1000	2000
Tiled (8*8)	0.002666	0.019199	0.152043
Tiled (16*16)	0.084295	0.011400	**0.001604 ?!! ☺**
Tiled (32*32)	0.001174	0.008929	**0.067562**
Simple (8*8)	0.007597	0.056290	0.462036
Simple (16*16)	0.004901	0.035745	0.262300
Simple (32*32)	0.007595	0.056307	**0.462166**

I hope that this amount of information is enough for you. I wished that I had more time to analyze it more, but I run out of time ☺

You can also find more timing results in the txt file attached to this assignment files.

Thank you

CS/EE 217 Lab 3
Reduction and Parallel Scan
Due Wed. Nov. 11 at 8pm

1) Unzip lab3-starter into your SDK projects directory or working Linux directory

2) There are two parts to this assignment: reduction and scan (i.e., prefix sum). You should see two different directories with the starter code for each.

3) In each directory edit the source files kernel.cu and main.cu to complete the functionality of the reduction and scan respectively.

4) For both reduction and scan, the size of the array is a command line argument. If none is given, 1 million is the default size. Note that you only have to accumulate the partial sums into an array, which is copied back to the host and verified to check the final answer.

5) Answer the following questions:

a-Use visual profiler to report relevant statistics about the execution of your kernels. Did you find any surprising results?

b-For each of reduction and prefix scan, suggest one approach to speed up your implementation.

Grading:

Your submission will be graded on the following aspects.

Correctness (60%)
1. Reduction produces correct result (25%)
2. Parallel prefix produces correct results (35%)

Efficiency (20%)
- Efficient/Effective implementation is used in both cases.

Report (20%)
Answers to the questions above

CS/EE 217 Lab 4
Histogram
Due Monday Nov. 30, 2015
(Extra credit: 15% towards midterm)

1) Unzip lab6-starter into your SDK projects directory or working Linux directory

2) Examine the source files kernel.cu and main.cu and complete the functionality of the histogram kernel.

4) The program takes two arguments, m (number of elements) and n (number of bins). The default values are 1million and 4096.

5) Answer the following questions:

a-Use visual profiler to report relevant statistics about the execution of your kernels. Did you find any surprising results?

b- What, if any, limitations are there on m and n for your implementation? Explain these limitations and how you may overcome them with a different implementation.

Grading:

Your submission will be graded on the following aspects.

Correctness (60%)
1. Histogramming works for a range of m and n values.

Efficiency (20%)
- Efficient/Effective implementation is used in both cases.

Report (20%)
Answers to the questions above

CS217 – LAB4

Shahriyar Valielahi Roshan – 861241935

1- You can find related statistics here. I values are reported for execution of the code without any argument as an input (Default 1000000 4096). Please take a look at attached screen shots.
getHisto:

Metric	Value
Duration	205.9µs
Grid Size	977
Block Size	1024
Registers/Thread	11
Shared Memory/Block	16KiB
Occupancy	94.2%
Warp Execution Efficiency	93.2%
Global Memory Load Efficiency	100%
Global Memory Store Efficiency	**N/A**
Warp Non-predicted Execution Efficiency	87.9
Issued Control Flow Instructions	816491
Executed Control Flow Instructions	31392
Control Flow Ratio	**0.04%**

Convert_32_8:

Metric	Value
Duration	27.584µs
Grid Size	977
Block Size	1024
Registers/Thread	8
Shared Memory/Block	0KiB
Occupancy	**52.5%**
Warp Execution Efficiency	100%
Global Memory Load Efficiency	100%
Global Memory Store Efficiency	**50%**
Warp Non-predicted Execution Efficiency	99.8
Issued Control Flow Instructions	31392
Executed Control Flow Instructions	31392
Control Flow Ratio	100%

Low global memory store Efficiency is due to the poor alignment or access pattern.

2- We have limitations one size of m, and n. The problem on m is the limitation on a total number of blocks that a device can handle, we do not have infinitive number of blocks, and number of blocks is proportional to the block size, the size of input. We can make blocks bigger to overcome this problem but when we use the biggest possible blocks size, and the input size lead to a problem that number of blocks exceed the maximum number of blocks, we have only one solution to divide the problem into smaller problems, and have kind of hierarchy.
And the limitation on the bin size is that we have limited shared memory available. We should use dynamic shared memory, or we should divide each step into several steps that is going to be done by more threads rather than one. For example thread one calculate [0-1023] histo and so one, and we should somehow merge results together.