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