ASSIGNMENT 3

Name - Srushti Hembade
Student Id - 862395839
Net id - shemb001

Question 1: On Bender, compare the execution time of a 256 x 256 square matrix multiplication compared to a 1024 x 64 and 64 x 1024 rectangular matrix multiply. All input matricies have 65k entries. What do you observe? Which is faster? Can you explain the observed behavior? Tip: You may want to comment out the verify() function in main.cu when timing this question.

→ Execution Time of **256 x 256** = 0.105563 s

```
bender /home/cegrad/shembade/matrix-multiply-srushtih1 $ ./sgemm-tiled 256

Setting up the problem...0.004185 s
    A: 256 x 256
    B: 256 x 256
    C: 256 x 256
Allocating device variables...0.096610 s
Copying data from host to device...0.000190 s
Launching kernel...0.004341 s
Copying data from device to host...0.000237 s
```

Execution Time of **1024 x 64** and **64 x 1024** = 0.105609 s

Screenshot for **1024 x 64**:

```
bender /home/cegrad/shembade/matrix-multiply-srushtih1 $ ./sgemm-tiled 1024 64 1024

Setting up the problem...0.004244 s
    A: 1024 x 64
    B: 64 x 1024
    C: 1024 x 1024
Allocating device variables...0.096266 s
Copying data from host to device...0.000192 s
Launching kernel...0.002574 s
Copying data from device to host...0.002333 s
```

Here as we can see Square matric 256 x 256 is faster. I believe the potential cause is: Compared to 1024*64 and 64*1024, there are fewer context shifts between thread blocks for 256 x 256 which makes it faster.

Question 2: Conceptual Question: For a 64 square tiled matrix multiplication, how many times is each element of the input matrices loaded from global memory? Assume 16x16 tiles.

→ 4 Times

Question 3: Conceptual Question: For a 64 square non-tiled matrix multiplication, how many times is each element of the input matrices loaded from global memory?

→ 64 Times

Question 4: GPGPU-Sim related question: In this part, we will compare the execution of a 128x128 square tiled matrix multiplication across different tile sizes. Run ./sgemm-tiled 128 in GPGPU-Sim with TILE_SIZE of 8, 16 (default), and 32. Fill the following table:

Tile size	8	16	32	Note
gpu_tot_sim_cycle	42633	27516	56422	Total cycles
gpu_tot_ipc	426.9609	461.4624	399.2769	Instruction per cycle
gpgpu_n_load_insn	524288	262144	131072	Total loads to global memory
gpgpu_n_store_insn	16384	16384	16384	Total stores to global memory
gpgpu_n_shmem_insn	4718592	4456448	4325376	Total accesses to shared memory

Question 5: Which tile size resulted in the least number of accesses to global memory? Which tile size resulted in the most number of accesses to global memory? What is the reasoning behind this observation?

→ With 32 sizes, the fewest global memory accesses were needed. There were the most global memory accesses from the 8-size tile. The number of tiles in a row or column is equal to the number of times each element loads from global memory to shared memory. Consequently, fewer tiles and fewer global memory accesses arise from greater tile sizes.

Question 6: Which tile size performed the fastest, which tile size performed the slowest? Why do you think that is?

→ In terms of performance, the 16-size tile was the fastest and the 32-size tile was the slowest. The size constraints of shared memory and the maximum number of threads per SM cause 16-size tiles to have a greater hardware consumption.