Mahbod Afarin 862186340 – GPU Lab 3:

1- The execution time for 256 × 256 matrix multiplication is 0.00083 seconds, and the execution time for multiplication of 1024 × 64 and 64 × 1024 is 0.000187 seconds. We observed the execution time for 256 × 256 is bigger than the execution time of 1024 × 64 and 64 × 1024 matrix multiplication. 256 × 256 is faster because the output of the 256 × 256 matrix multiplication will be 256 × 256 matrix, but the output of the 1024 × 64 and 64 × 1024 matrix multiplication is 1024 × 1024 matrix and for the 256 × 256 we need 65536 elements and for 1024 × 1024 output we need 1048576 elements. It means that for 256 × 256 output we need lower threads compared to the 1024 × 1024 output. That is why 256 × 256 matrix multiplication is faster than the 1024 × 64 and 64 × 1024 matrix multiplication.

2- For the titled matrix multiplication it read each element 64/16 = **4** time. Because we have 16 × 16 tiles and we load from the global memory tile by tile.

3- For the non-tile matrix multiplication it read each element **64** times. Because for calculating each output elements we need to load n (entire row) elements from one matrix and n (entire column) elements from the other matrix. Therefore, we load each element n times.

4- The below table shows the results of the GPGPU-Sim simulator.

Tile Size	8	16	32	Note
gpu_tot_sim_cycle	40398	26881	57956	Total cycles
gpu_tot_ipc	413.6759	446.1548	385.5990	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

5- 32 tile size is resulted in least number of accesses to global memory (131072). 8 tile size is resulted in the most number of accesses to global memory (524288). The reason is that with 32 tile size, we load each element 128/32 = 4 while with 8 we have 128/8 = 16 loads. Therefore, 4 loads take less time compared to the 16 loads.

6- 16 tile size performs the fastest (26881). 32 tile size performs the slowest (57956). The reason is that for 32 tile size we have (32 × 32)/32 = 32 warps for each SM, but for 16 tile size we have (16 × 16)/32 = 8 warps for each SM. There is a tradeoff between memory access time and distribution of warps between SMs. For the 32 tile size we have 32 warps for each SM and the access time to memory is better, but the distribution of warps between SMs and parallelism is not good. For the 8 tile size the memory access time is not good, but the distribution of the warps between SMs is the best. For the 16 tile size we will reach the tradeoff and the total cycles is the smallest.