Name- Shubham Shrotriya
Student ID- 862464519

Answer the following questions:

Q1. 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 matrices have 65k entries. What do you observe? Which is faster? Can you explain the observed behaviour? Tip: You may want to comment out the verify() function in main.cu when timing this question.

Ans1. The runtime of the 256 x 256 matrix square multiplication is:

Setting up the problem...0.004171 s

A: 256 x 256

B: 256 x 256

C: 256 x 256

Allocating device variables...0.175956 s

Copying data from host to device...0.000209 s

Launching kernel...0.000187 s

Copying data from device to host...0.000208 s

The runtime of the 1024 x 64 and 64 x 1024 matrix multiplication is:

Setting up the problem...0.004161 s

A: 1024 x 64

B: 64 x 1024

C: 1024 x 1024

Allocating device variables...0.181899 s

Copying data from host to device...0.000211 s

Launching kernel...0.000198 s

Copying data from device to host...0.002151 s

Looking at the computational performance, we can see that the 256x256 square multiplication takes only 0.187 ms to complete, whereas the 1024x64*64x1204 multiplication takes 0.198 ms when only the "launching kernel" time is taken into account. The rectangular shapes of the 1024x64 and 64x1024 matrices, which have different aspect ratios, are the cause of this disparity. Because of this, these matrices require more context switches than two 256x256 matrices would.

In addition, the product of the 256x256 square multiplication yields a 256x256 matrix, but the product of the 1024x64 and 64x1024 multiplication yields a 1024 matrix with a somewhat larger element count. As a result, the procedure of transferring data back to the device takes longer.

Q2. 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.

Ans2. 4

Q3. Conceptual Question: For a 64 square *non-tiled* matrix multiplication, how many times is each element of the input matrices loaded from global memory?

Ans3. 64

Q4. 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:

Ans4.

	Tile size	**8**	**16**	**32**	**Note**
	gpu_tot_sim_cycle	44463	29836	68220	Total cycles

	gpu_tot_ipc	429.2864	468.4124	406.5980	Instruction per cycle
	gpgpu_n_load_insn	540672	278528	147456	Total loads to global memory
	gpgpu_n_store_insn	16384	16384	16384	Total stores to global memory
	gpgpu_n_shmem_insn	5013504	5013504	5406720	Total accesses to shared memory

Q5.
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?

Ans5. When the tile size is 32, there are the fewest global memory accesses, and when the tile size is 8, there are the most global memory accesses.

This is because the number of global memory accesses is determined by dividing the number of rows' tiles by the matrix's columns. As a result, as tile size grows, fewer accesses to global memory are made.

Q6. Which tile size performed the fastest, which tile size performed the slowest? Why do you think that is?

Ans6. 16-sized tiles functioned the fastest, while 32-sized tiles operated the slowest.

There are 256 threads for a 16x16 tile size. A float value is loaded into global memory 256*2 = 512 times each block, and add/multiply operations total 256*(2*16) = 8192 times per block. As a result, the 16x16 tile size shows fewer cycles and has a higher instruction execution capacity per cycle.

In contrast, 1024 threads load 1024*2 = 2048 float values from global memory for the 32x32 tile size, producing 65536 add/multiply operations (1024*2*32). Performance is thus slower than with the 16x16 tile size.