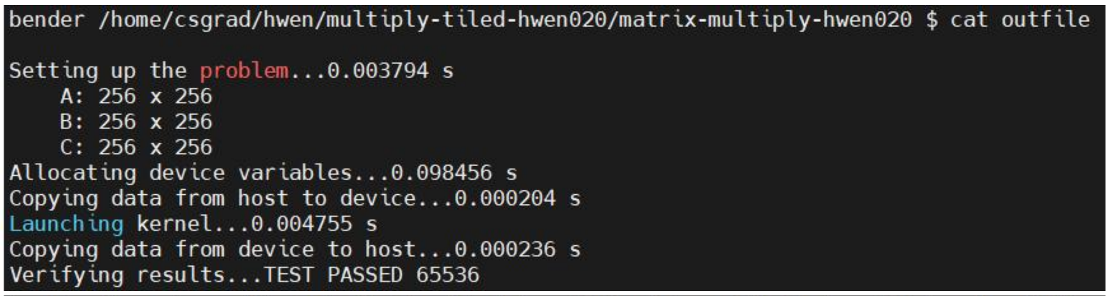
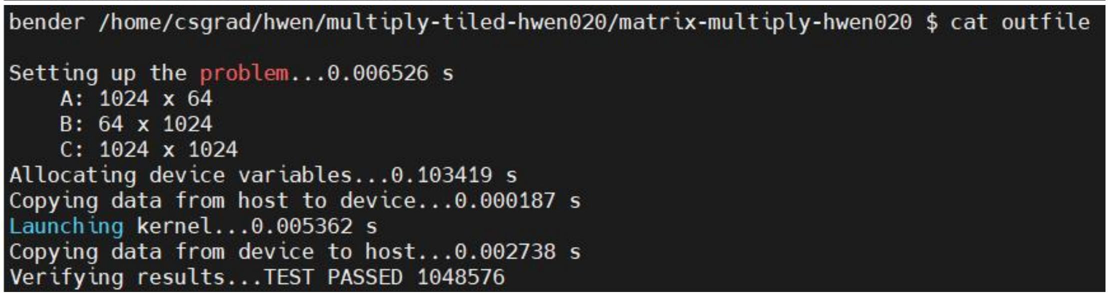
Hairu-Wen hwen020@ucr.edu

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.



```
bender /home/csgrad/hwen/multiply-tiled-hwen020/matrix-multiply-hwen020 $ cat outfile

Setting up the problem...0.003794 s
    A: 256 x 256
    B: 256 x 256
    C: 256 x 256
Allocating device variables...0.098456 s
Copying data from host to device...0.000204 s
Launching kernel...0.004755 s
Copying data from device to host...0.000236 s
Verifying results...TEST PASSED 65536
```



```
bender /home/csgrad/hwen/multiply-tiled-hwen020/matrix-multiply-hwen020 $ cat outfile

Setting up the problem...0.006526 s
    A: 1024 x 64
    B: 64 x 1024
    C: 1024 x 1024
Allocating device variables...0.103419 s
Copying data from host to device...0.000187 s
Launching kernel...0.005362 s
Copying data from device to host...0.002738 s
Verifying results...TEST PASSED 1048576
```

On Bender the 256 x 256 square matrix multiplication is significantly faster than the 1024 x 64 and 64 x 1024. All input matrix are 65536 in size. However, the 256 x 256 has significantly fewer data copy time and operations:

256 x 256: 16 x 16 blocks, each with 16 phases. Multiply-add operations=256^3 = 16,777,216

1024 x 64 or reverse: 64 x 64 blocks, each with 4 phases. Four times of 256 x 256's operations.

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. 64/16=4 blocks in the same row/column of tile grid (depends on which matrices does the element belong to) need to load this element from global memory.

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. 64 threads in the same row/column (depends on which matrices the element belongs to) need to load this element from global memory.

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:

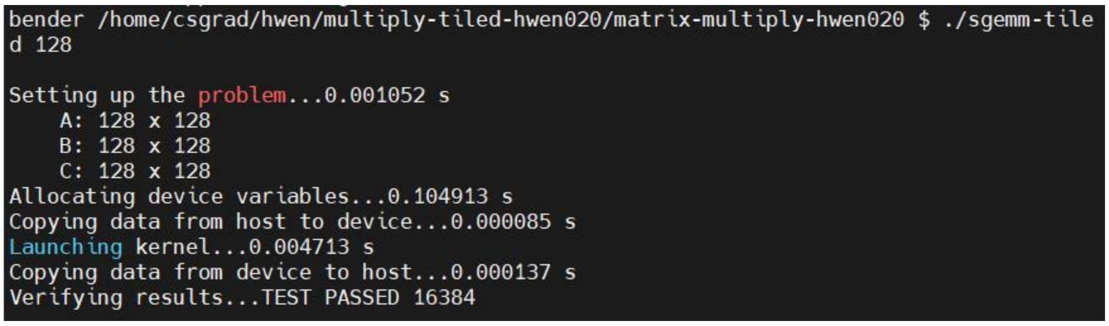
Tile size	8	16	32	Note
gpu_tot_sim_cycle	40089	26772	55590	Total cycles
gpu_tot_ipc	416.4558	447.3593	401.7160	Instruction per cycle
gpgpu_n_load_insn	524288	262144	131072	Total loads to global memory
gpgpu_n_store_insn	16384	16384	16384	Total stores to global memory
gpgpu_n_shmem_insn	4718592	4456448	4325376	Total accesses to shared memory

5、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?

Tile size of 32 results in least number of global memory access. Tile size of 8 results in greatest number of global memory access. The times each element of the input matrices loaded from global memory is decided by how many blocks is in one row/column. The larger tile size, the fewer loading, which also means more reuse in shared memory and less global memory access.

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

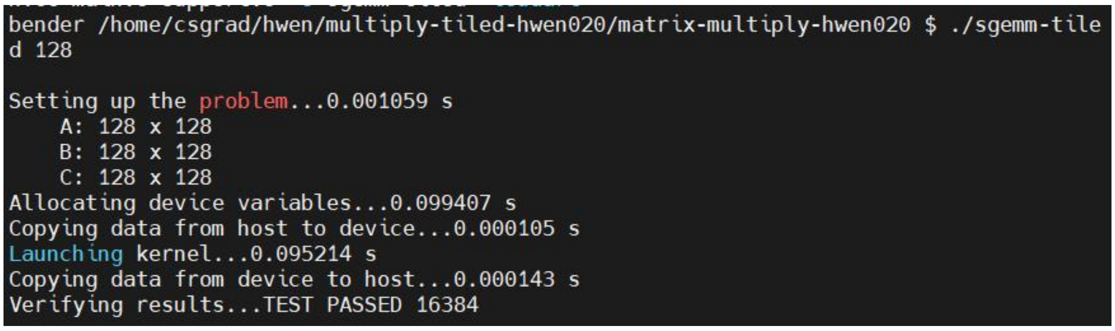
TILE_SIZE=16:



```
bender /home/csgrad/hwen/multiply-tiled-hwen020/matrix-multiply-hwen020 $ ./sgemm-tile
d 128

Setting up the problem...0.001052 s
    A: 128 x 128
    B: 128 x 128
    C: 128 x 128
Allocating device variables...0.104913 s
Copying data from host to device...0.000085 s
Launching kernel...0.004713 s
Copying data from device to host...0.000137 s
Verifying results...TEST PASSED 16384
```

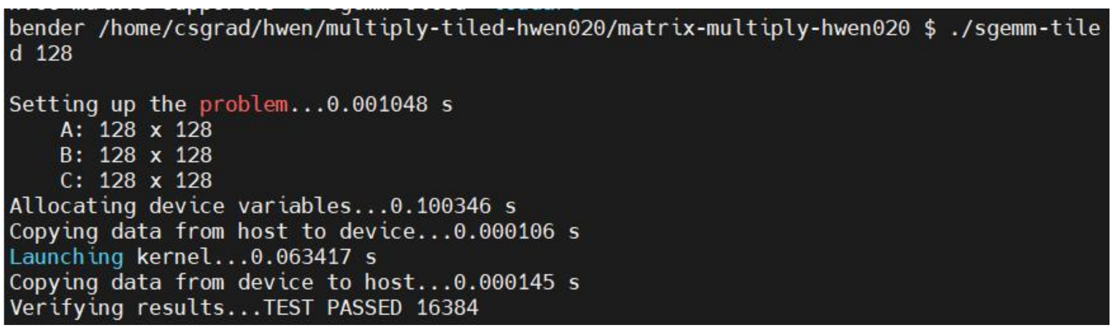
TILE_SIZE=8:



```
bender /home/csgrad/hwen/multiply-tiled-hwen020/matrix-multiply-hwen020 $ ./sgemm-tile
d 128

Setting up the problem...0.001059 s
    A: 128 x 128
    B: 128 x 128
    C: 128 x 128
Allocating device variables...0.099407 s
Copying data from host to device...0.000105 s
Launching kernel...0.095214 s
Copying data from device to host...0.000143 s
Verifying results...TEST PASSED 16384
```

TILE_SIZE=32:



```
bender /home/csgrad/hwen/multiply-tiled-hwen020/matrix-multiply-hwen020 $ ./sgemm-tile
d 128

Setting up the problem...0.001048 s
    A: 128 x 128
    B: 128 x 128
    C: 128 x 128
Allocating device variables...0.100346 s
Copying data from host to device...0.000106 s
Launching kernel...0.063417 s
Copying data from device to host...0.000145 s
Verifying results...TEST PASSED 16384
```

A tile size of 16 is the fastest and a tile size of 32 is the slowest.

The maximum threads per SM is 2048 in Maxwell GPUs. For tile size of 32, since there are too many threads in each of the blocks, threads are not made full use of due to the limitation of the number of threads per SM. And for tile size of 8, it performs too much loading from global memory.