Lab 2 Report

Quan Fan
862099688
qfan005@ucr.edu

Questions

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.

Answer:

Kernel Execution Time of 256 x 256 is 0.000172s
Kernel Execution Time of 1024 x 64 and 64 x 1024 is 0.000545s
The square matrix multiplication is faster. In my opinion, there are two possible reasons:
(1) In later case there are more context switches between thread blocks. For 256x256, there are totally 16x16 tiles and each tile of C run 16 phases. For the other case, there are totally 64x64 tiles and each tile of C run 4 phases.
(2) The contest of readers is more severe in the rectangular matrix case. In this case, there are up to 1024 threads who will read a single element from global memory in parallel, while in the square matrix case, the number of threads is turned down to 256.

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.

Answer:

4 times.

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?

Answer:

64 times.

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	43433	27727	57904	Total cycles
gpu_tot_ipc	420.6055	460.3144	390.1895	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?*

Answer:

(Assume that this question is related to question 4)

The tile size of 32 resulted in the least number of accesses to global memory.

The tile size of 8 resulted in the most number of accesses to global memory.

The number of times that each element loaded from global memory to shared memory equals to the number of tiles of the row/column. Therefore, larger tile size leads to a smaller number of tiles and smaller number of accesses of global memory.

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

Answer:

(Assume that this question is related to question 4)

The tile size of 16 performed the fastest, while the tile size of 32 performed the slowest.

Due to the limits of shared memory size and maximum threads per SM, the tile size of 16 have the higher hardware utilization.