Homework #4 (HW4)

Autumn 2022 - GPU Computing

DUE: 16 November 2022, 5:59 PM

Learning Objectives

- Reinforce understanding of control divergence, coalesced memory access, and other basic CUDA performance concepts.
- Get more practice using Nsight Compute (NC) for profiling and performance analysis
- Implement the MMULT corner turning algorithm in a CUDA kernel.
- EXTRA CREDIT: implement a matrix transpose kernel to support the corner turning kernel.

Questions

- Complete the multiple-choice short answer written question set provided in the separate file (on class Canvas website)
    - /FIles/Homeworks/HW4/Aut22_GPUCompute_HW4 Questions.pdf

Programming and Analysis Exercises

NOTE: Always use RELEASE (x64) build when profiling your code with Nsight Compute!

Coalesced Memory Access

For this exercise we'll use square matrices of a simple size which is conducive to analysis.
- Use matrices of size 256 x 256 elements.

Using NC profile the memory workload analysis performance of the MMULT kernels
- all-global
- tiled/phased
- For each, profile several 2D TILE_WIDTH = thread-block dimensions
    - 8x32
    - 16x16
    - 32x8

- Review the profile report Shared Memory, L1/TEX, L2 cache, and Device Memory access statistics and analyze the transaction sizes and patterns between each memory level in terms of Bytes and Sectors transferred, and wavefronts.
- Describe how this relates to the warp memory access pattern and the use and efficiency of coalesced accesses.
    - Consider the total # bytes read from each memory level/unit for each kernel thread.
    - If you see any discrepancies or unexpected numbers in the profile report, identify those and consider if the "complicating factors" discussed in L5 might be contributing. Why or why not?

Corner Turning MMULT Kernel

Implement an extension to the basic tiled/phased shared memory MMULT kernel which supports the corner turning algorithm, as shown in Lecture 5 slide 41 and in the image below.

It is OK to use the non-general MMULT version (which requires square matrices of a size which is even multiple of TILE size).

In the below figure, note that matrix B must be stored in column-major layout, not row-major. In this case, as shown in the figure below, the case (A) on left side will not have coalesced access when warps load an input tile from global memory into the shared memory region. Case (B) on the right shows the correct coalesced access pattern for a matrix which is stored in column-major layout.

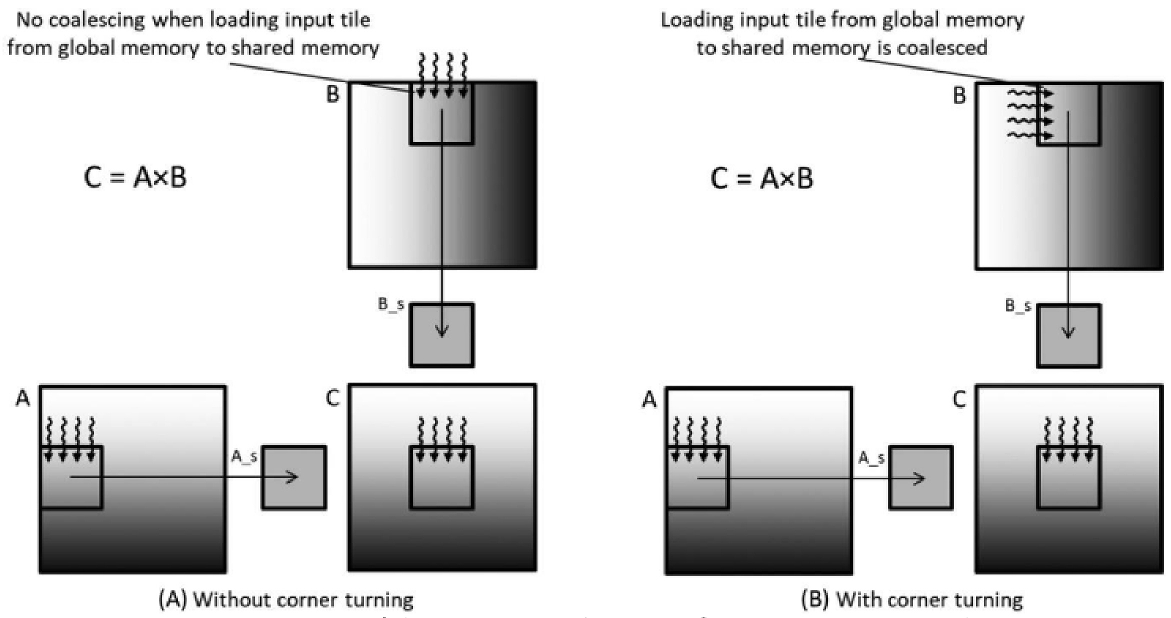
No coalescing when loading input tile from global memory to shared memory
Loading input tile from global memory to shared memory is coalesced
C = A×B
B
B_s
A
A_s
C
(A) Without corner turning
(B) With corner turning

FIGURE 6.4 Applying corner turning to coalesce accesses to matrix B, which is stored in column-major layout.

1. In order to setup the cornerTurning kernel input matrix data, which requires matrix B to be in column-major layout, the matrix can be transposed on the Host application prior to transferring over to the device memory.
    a. Recommended alternative is to do the EXTRA CREDIT option below.
2. EXTRA CREDIT (5pts): write a CUDA kernel which performs a matrix transpose on the GPU.
    a. Since global memory data has a lifetime scope of the application, the transposed matrix values will persist on the device between kernel calls, so you can first call the Transpose kernel, and then use the transposed matrix in your cornerTurning kernel, without moving the data back and forth between Host and GPU device memory in between kernel calls!
3. Note that the “trick” is in the writing of the per-thread data from global into the shared memory tile in the proper format during the load phase, so that the matrix element inner-product calculation is done correctly.
4. Provide verification results showing that the same input matrices A & B will produce the same (correct) results
    a. On the CPU sequential reference MMULT
    b. On the GPU non-corner-turning algorithm (either all-global or tiled/SMEM)
    c. On the GPU corner-turning algorithm

SUBMIT

1. Submit answers to the questions, including shown work and explained reasoning.
2. Submit your CUDA host application source code
    a. .cu, .cpp and .h files only - do not submit all project & solution files and other build by-products!
3. Submit your CUDA kernel functions
    a. may be in the same file as your host code if using a single *.cu file
4. Submit your NCU profiling reports along with a separate write-up of your analysis of the profiling results for both the Coalesced Memory Access and Corner Turning exercises.