**Lab 6: Shared Memory Optimization**

ECE 455: GPU Algorithm and System Design

Due: Submit completed PDF to Canvas by 11:59 PM on 10/24

**Overview**

This lab focuses on using **shared memory** to optimize CUDA kernels. You will first write a warm-up kernel that loads data into shared memory for basic computation, then apply shared-memory tiling to matrix multiplication, and finally compare global vs. shared memory performance.

**Learning Objectives**

* Understand the role of CUDA shared memory.
* Implement a tiled shared-memory matrix multiplication.
* Compare runtime and scalability with a global-memory kernel.
* Summarize performance improvements and debugging challenges.

**Euler Instruction**

* $ ssh y o u r \_ C A E \_ a c c o u n t @ e u l e r . engr . wisc . edu
* $ sbatch y o u r \_ s l u r m \_ s c r i p t . slurm

Do not run on the login node. Work locally, push to GitHub, and run on Euler using Slurm.

**Submission Instruction**

Specify your GitHub link here:

https://github.com/YourGitHubName/ECE455/HW06

1

**Problem 1: Shared Memory Warm-Up**

**Task:** Write a CUDA kernel that demonstrates basic use of shared memory:

1. Load a block of elements from global memory into shared memory.
2. Square each element inside shared memory.
3. Write the results back to global memory.

**Kernel**

**Filename:** shared\_warmup.cu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **template** < **typename** T > | | | |  |
| \_ \_ g l o b a l \_ \_ | **void** |  | s q u a r e \_ s h a r e d \_ k e r n e l ( **const** | T \* in , T \* out , size\_t N ) { |
| \_ \_ s h a r e d \_ \_ | | T | tile [ B LO CK \_D IM ]; |  |
|  |  |  |  |  |
| size\_t | idx | = | blockIdx . x \* blockDim . x + | th re ad Id x . x ; |
| **if** ( idx | >= | N ) **return** ; | |  |

*// 1. Load* *from global to shared memory* tile [ th re ad Id x . x ] = in [ idx ]; \_ \_ s y n c t h r e a d s () ;

*// 2. Compute in shared* *memory*

tile [ th re ad Id x . x ] = tile [ th re ad Id x . x ] \* tile [ th re ad Id x . x ]; \_ \_ s y n c t h r e a d s () ;

*// 3. Write back to global* *memory*

out [ idx ] = tile [ th re ad Id x . x ];

}

Full source and main function: [GitHub Gist](https://gist.github.com/Randy1005/1020a2d264a6940c0ac4973f8fdd3d87)

**Slurm Script**

**Filename:** shared\_warmup.slurm

*#!/ usr/bin/env zsh*

*# SBATCH -- p ar ti ti on = i n s t r u c t i o n*

* *SBATCH -- time = 00 :0 1: 00*
* *SBATCH -- ntasks =1*
* *SBATCH -- cpus - per - task =1*
* *SBATCH -- gpus - per - task =1*

*# SBATCH* *-- output = s h a r e d \_ w a r m u p . output*

**cd** $ S L U R M \_ S U B M I T \_ D I R

module load nvidia / cuda

nvcc s h a r e d \_ w a r m u p . cu -o s h a r e d \_ w a r m u p ./ s h a r e d \_ w a r m u p

2

**Problem 2: Tiled Matrix Multiplication with Shared Memory**

**Task:** Implement a matrix multiplication kernel using shared-memory tiling. Each thread block should load a TILE\_SIZE × TILE\_SIZE tile of matrices A and B into shared memory, synchronize threads, and compute the corresponding tile of C.

**Kernel**

**Filename:** mm\_tiled.cu

**template** < **typename** T >

\_ \_ g l o b a l \_ \_ **void** mm\_tiled ( **const** T \* A , **const** T \* B , T \* C , **int** N ) {

*// Allocate shared - memory tiles for A and B*

\_ \_ s h a r e d \_ \_ T tile\_A [ T IL E\_ SI ZE ][ T IL E\_ SI ZE ];

\_ \_ s h a r e d \_ \_ T tile\_B [ T IL E\_ SI ZE ][ T IL E\_ SI ZE ];

* *Compute the row and column index this thread is r e s p o n s i b l e for* **int** row = blockIdx . y \* TI LE \_S IZ E + t hr ea dI dx . y ;

**int** col = blockIdx . x \* TI LE \_S IZ E + t hr ea dI dx . x ;

* val = 0 ;
  + *Loop over all tiles required to compute one C - tile*

**for** ( **int** t = 0 ; t < ( N + T IL E\_ SI ZE - 1) / T IL E\_ SI ZE ; ++ t ) {

* *Load one tile of A and one tile of B from global to shared memory*

**if** ( row < N && ( t \* T IL E\_ SI ZE + t hr ea dI dx . x ) < N )

tile\_A [ th re ad Id x . y ][ th re ad Id x . x ] =

A [ row \* N + t \* T IL E\_ SI ZE + t hr ea dI dx . x ];

**else**

tile\_A [ th re ad Id x . y ][ th re ad Id x . x ] = 0 ;

**if** ( col < N && ( t \* T IL E\_ SI ZE + t hr ea dI dx . y ) < N )

tile\_B [ th re ad Id x . y ][ th re ad Id x . x ] =

B [( t \* T IL E\_ SI ZE + t hr ea dI dx . y ) \* N + col ];

**else**

tile\_B [ th re ad Id x . y ][ th re ad Id x . x ] = 0 ;

* \_ s y n c t h r e a d s () ; *// Wait until all threads load their tile*

|  |  |  |
| --- | --- | --- |
| *// Multiply the two* | | *tiles* |
| **for** ( **int** k | = 0 ; k < T IL E\_ SI ZE ; ++ k ) | |
| val += | tile\_A [ th re ad Id x . y ][ k ] \* tile\_B [ k ][ t hr ea dI dx . x ]; | |
| \_ \_ s y n c t h r e a d s () ; *//* | | *Wait before loading the next tile* |

}

*// Write result to global* *memory*

**if** ( row < N && col < N )

C [ row \* N + col ] = val ;

}

Full source (with validation and timing): [GitHub Gist](https://gist.github.com/Randy1005/1c3633a3e95478f0b11443d2434c7028)

3

**Slurm Script**

**Filename:** mm\_tiled.slurm

*#!/ usr/bin/env zsh*

*# SBATCH -- p ar ti ti on = i n s t r u c t i o n*

* *SBATCH -- time = 00 :0 1: 00*
* *SBATCH -- ntasks =1*
* *SBATCH -- cpus - per - task =1*
* *SBATCH -- gpus - per - task =1*
* *SBATCH -- output = mm\_tiled . output*

**cd** $ S L U R M \_ S U B M I T \_ D I R

module load nvidia / cuda

nvcc mm\_tiled . cu -o mm\_tiled

./ mm\_tiled

4

**Problem 3: Global vs. Shared Memory Performance**

**Task:** Compare the runtime of a naive global-memory matrix multiplication and a tiled shared-memory version. Measure both runtimes using CUDA events and report the observed speedup.

**Naive Global-Memory Kernel**

**Filename:** mm\_compare\_tiled\_vs\_naive.cu

**template** < **typename** T >

* \_ g l o b a l \_ \_ **void** mm\_naive ( **const** T \* A , **const** T \* B , T \* C , **int** N ) { **int** tid = blockIdx . x \* blockDim . x + t hr ea dI dx . x ;

**int** t o t a l \_ e l e m s = N \* N ;

**if** ( tid >= t o t a l \_ e l e m s ) **return** ;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **int** | row | = | tid | / | N ; |  |  |  |  |
| **int** | col | = | tid | % | N ; |  |  |  |  |
| T val = | | 0 ; |  |  |  |  |  |  |  |
| **for** | ( **int** k | | = | 0 ; | k | < | N ; | ++ k ) |  |
|  | val | += | A [ row | | \* | N | + | k ] \* | B [ k \* N + col ]; |
|  |  |  |  |  |  |  |  |  |  |

C [ tid ] = val ;

}

**Tiled Shared-Memory Kernel**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **template** < **typename** T > | | |  |  |  |  |  |
| \_ \_ g l o b a l \_ \_ **void** | mm\_tiled ( **const** T \* A , **const** T \* B , T \* C , **int** N ) { | | | | | | |
| *// Declare* | *shared - memory* | | | | *tiles* |  |  |
| \_ \_ s h a r e d \_ \_ | T tile\_A [ T IL E\_ SI ZE ][ T IL E\_ SI ZE ]; | | | | | |  |
| \_ \_ s h a r e d \_ \_ | T tile\_B [ T IL E\_ SI ZE ][ T IL E\_ SI ZE ]; | | | | | |  |
| *// Compute* | *this thread ’s* | | | | *global row /col index* | |  |
| **int** row = blockIdx . y \* TI LE \_S IZ E + t hr ea dI dx . y ; | | | | | | |  |
| **int** col = blockIdx . x \* | | | | TILE\_SIZE + | | t hr ea dI dx . x ; |  |
| T val = 0 ; | *// A c c u m u l a t o r for the result* | | | | | |  |
| *// Loop through all tiles of A and* | | | | | | *B needed for this* | *output block* |
| **for** ( **int** t | = | 0 ; t | < ( N | + | TILE\_SIZE | - 1) / T IL E\_ SI ZE ; ++ t ) { | |
| *// Load* | *a* | *tile* | *of* | *A* | *and a tile* | *of B into shared* | *memory* |

**if** ( row < N && ( t \* T IL E\_ SI ZE + t hr ea dI dx . x ) < N )

tile\_A [ th re ad Id x . y ][ th re ad Id x . x ] =

A [ row \* N + t \* T IL E\_ SI ZE + t hr ea dI dx . x ];

**else**

tile\_A [ th re ad Id x . y ][ th re ad Id x . x ] = 0 ;

**if** ( col < N && ( t \* T IL E\_ SI ZE + t hr ea dI dx . y ) < N )

|  |  |  |  |
| --- | --- | --- | --- |
| tile\_B [ th re ad Id x . y ][ th re ad Id x . x ] | = |  |  |
| B [( t \* T IL E\_ SI ZE + t hr ea dI dx . y ) | | \* | N + col ]; |
| **else** |  |  |  |
| tile\_B [ th re ad Id x . y ][ th re ad Id x . x ] | = | 0 ; |  |

5

* \_ s y n c t h r e a d s () ; *// S y n c h r o n i z e all threads before co mp ut in g*

*//* *Compute partial products for this tile* **for** ( **int** k = 0 ; k < T IL E\_ SI ZE ; ++ k )

val += tile\_A [ th re ad Id x . y ][ k ] \* tile\_B [ k ][ t hr ea dI dx . x ];

\_ \_ s y n c t h r e a d s () ; *// Wait* *before loading next tile*

}

*// Write result to global* *memory*

**if** ( row < N && col < N )

C [ row \* N + col ] = val ;

}

Full source (with validation and timing): [GitHub Gist](https://gist.github.com/Randy1005/485e912e423a5f82446a6ddc62d7bed9)

**Slurm Script**

**Filename:** mm\_compare\_tiled\_vs\_naive.slurm

*#!/ usr/bin/env zsh*

*# SBATCH -- p ar ti ti on = i n s t r u c t i o n*

* *SBATCH -- time = 00 :0 1: 00*
* *SBATCH -- ntasks =1*
* *SBATCH -- cpus - per - task =1*
* *SBATCH -- gpus - per - task =1*

*# SBATCH* *-- output = m m \_ c o m p a r e \_ t i l e d \_ v s \_ n a i v e . output*

**cd** $ S L U R M \_ S U B M I T \_ D I R

module load nvidia / cuda

nvcc m m \_ c o m p a r e \_ t i l e d \_ v s \_ n a i v e . cu -o m m \_ c o m p a r e \_ t i l e d \_ v s \_ n a i v e ./ m m \_ c o m p a r e \_ t i l e d \_ v s \_ n a i v e

6

**Problem 4: Reflection**

**Task:** Summarize the challenges you faced in this lab.

7