**1541 Take Home Final: GPU and CUDA programming**

**Name:\_\_Andrew Tran\_\_\_\_\_\_\_\_\_\_\_\_\_**

The following problems should be completed individually.

1. (8pts) Suppose a thread block has a shared memory array of 4KB. Each thread in this kernel needs 16 registers. Threads can execute only if there are enough registers to use. Assume a GPU has 4 SMs, each with a register file of 2048 registers, and a shared memory of 16KB. Answer the following questions:
2. What is the maximum number of threads that can execute simultaneously on the GPU?

2048 registers/SM / 16 registers/thread = 128 threads/SM \* 4 SM =

= **512 concurrent threads**

1. How many thread blocks can maximally co-run in one SM?

16KB/SM / 4KB/block =

= **4 thread blocks per SM**

1. If the kernel is launched via <<<nblocks, blksize>>>, to execute the maximum number of threads simultaneously, how should you choose nblocks and blksize? Enumerate all possibilities.

**Each SM can run a max 128 threads (blksize<=128) and 4 thread blocks (nblocks<=16). If less thread blocks are created, the minimum number of threads per block needs to be increased.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **nblocks** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** |
| **blksize(min)** | **32** | **35** | **37** | **40** | **43** | **47** | **52** | **57** | **64** | **74** | **86** | **103** | **128** |

1. (16pts) Read the following CUDA code, write the output of array A after execution of the following program:

\_global\_ F(int \*A)

{ int idx = blockIdx.x \* blockDim.x + threadIdx.x ;

A[idx] = idx ;

A[blockIdx.x] = blockIdx.x ;

} ;

void main()

{

Allocate a 16 element int array A  in the GPU global memory and initialize its elements to 0 ;

F<<<2,4>>> (A) ;

}

Output: **0 0 1 0 2 0 3 0 4 1 5 1 6 1 7 1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

1. (18pts) In this problem, you need to refer to page #41 of our lecture slides and refresh the coordinates of blocks and threads in 2D data structure. Note that the x-coordinates indicate the column number, and the y-coordinates indicate the row number.

Read the following CUDA code, show the output of the content of array A after the execution of the program:

global\_ F(int \*A)

{ int row = blockIdx.y \* blockDim.y + threadIdx.y ;

int col  = blockIdx.x \* blockDim.x + threadIdx.x ;

A[row][col] = blockIdx.x + blockIdx.y + threadIdx.x ;

} ;

void main() {  Allocate an 6x6 array A  in the GPU global memory ;  initialize A’s elements to 0 ; dim3 grid(2,2) ;     // a 2x2 array of blocks dim3 blocks(3,3); // each block is a 3x3 array of threads

F<<<grid,blocks>>>(A) ;

}

Output: **0 1 2 0 1 2 0 1 2 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 2 3 4 2 3 4 2 3 4 2 3 4**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Y,X** | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | 1 | 2 | 1 | 2 | 3 |
| **1** | 0 | 1 | 2 | 1 | 2 | 3 |
| **2** | 0 | 1 | 2 | 1 | 2 | 3 |
| **3** | 1 | 2 | 3 | 2 | 3 | 4 |
| **4** | 1 | 2 | 3 | 2 | 3 | 4 |
| **5** | 1 | 2 | 3 | 2 | 3 | 4 |

1. (18pts) Recall from what you learned in parallel programming, if two concurrent processes or threads want to update the same memory location, and if there are no proper locking mechanism or synchronization mechanism to guard such contending operations, the content of memory location will become non-deterministic. In problem III, answer the question again of line

A[row][col] = blockIdx.x + blockIdx.y + threadIdx.x ;

is replaced with

A[threadIdx.y][threadId.x] = blockIdx.x;

Output: **0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Y,X** | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 or 1 | 0 or 1 | 0 or 1 | 0 | 0 | 0 |
| **1** | 0 or 1 | 0 or 1 | 0 or 1 | 0 | 0 | 0 |
| **2** | 0 or 1 | 0 or 1 | 0 or 1 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 |