



GPUs II

Exam exercises



Agenda



Introduction
and slight
recall



Matrix
multiplication



Exercises!





Intro

The exam takeover from
the previous lesson



On the previous episode...

- Blocks, threads, grid



On the previous episode...

- Blocks, threads, grid
- Compute capability



Threads

In CUDA, a logic thread = a physical core



Threads

In CUDA, a logic thread = a physical core

You can picture a thread as the smallest unit being executed at a time



Threads

In CUDA, a logic thread = a physical core

You can picture a thread as the smallest unit being executed at a time

We have a huge number of threads/cores and thus execution needs to group them under a logical structure

This ensure the parallelism of execution up to a certain point, thus this leads us towards...



Blocks

A block is a logical (and physical) group of threads



Blocks

A block is a logical (and physical) group of threads

They group many cores running inside the same SM



Blocks

A block is a logical (and physical) group of threads

They group many cores running inside the same SM

Instruction pointer is shared amongst them



Grid

Computation gets so big that another logical grouping is needed, the Grid



Grid

Computation gets so big that another logical grouping is needed, the Grid

The Grid is yet another **logical grouping of Blocks**



Take away from this

A thread is just an element of a block



Take away from this

A thread is just an element of a block

A block is an element of the grid



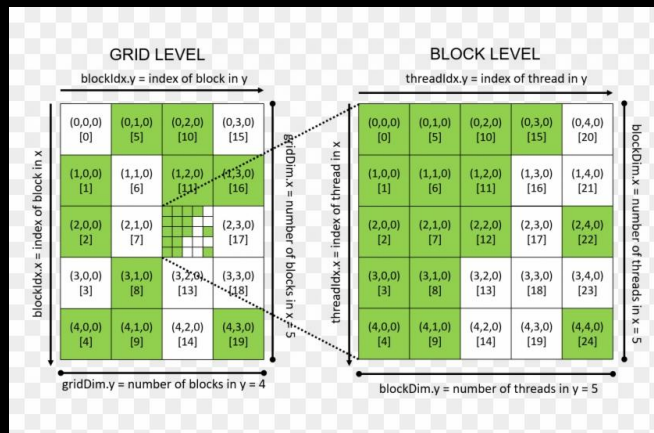
Take away from this

A thread is just an element of a block

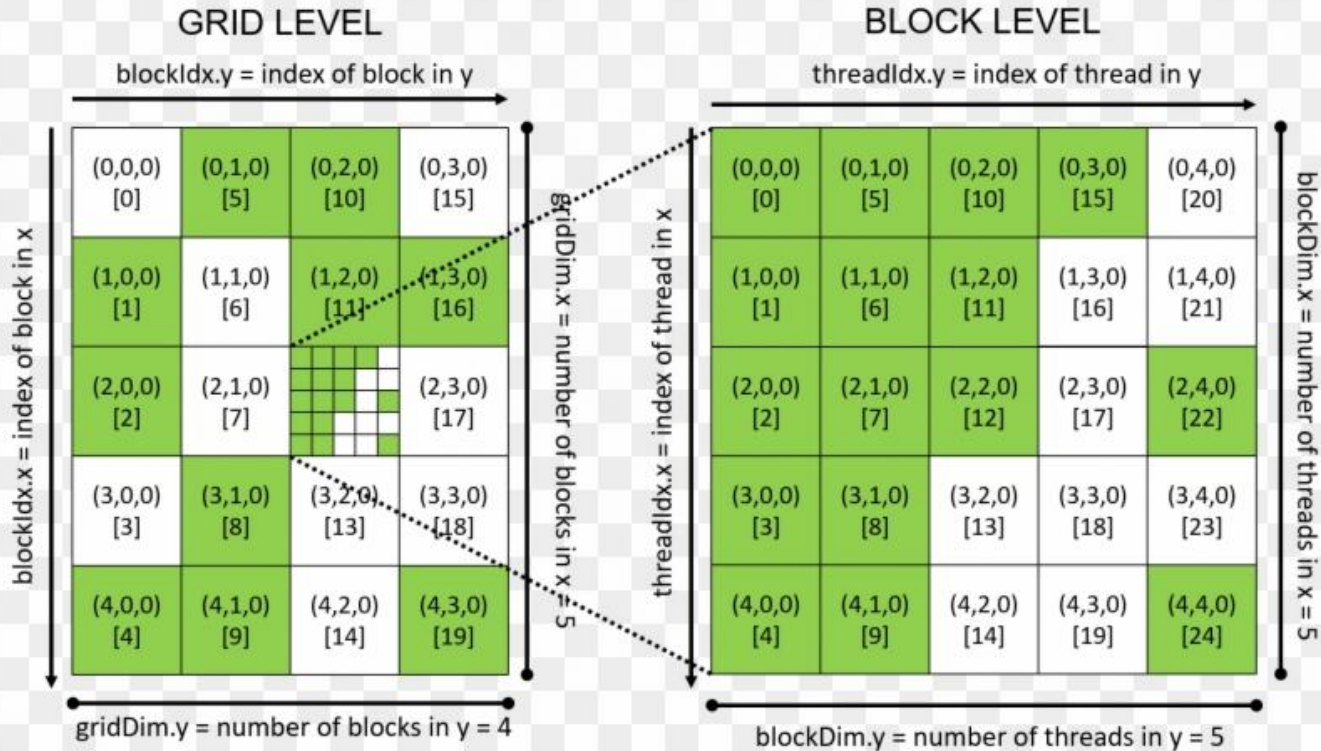
A block is an element of the grid

Each of them uses 3d indexing

(Next slide is
a full page
on this
image)



Indexing (z-axis misses for clarity)



Compute capability

Over the years new hardware was introduced and thus each GPU was differently capable than the previous one



Compute capability

Over the years new hardware was introduced and thus each GPU was differently capable than the previous one

This number represents the “power” of each device, the bigger, the better.



Matrix multiplication

The heart of 3d
graphics



Rows times columns

Not a math class, but genuine question...



Rows times columns

Not a math class, but genuine question...

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$
$$= \begin{bmatrix} 1 \times 10 + 2 \times 20 + 3 \times 30 & 1 \times 11 + 2 \times 21 + 3 \times 31 \\ 4 \times 10 + 5 \times 20 + 6 \times 30 & 4 \times 11 + 5 \times 21 + 6 \times 31 \end{bmatrix}$$
$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

Why?



(Extra) try it yourself!

Write down two matrices (2x2 for simplicity)



(Extra) try it yourself!

Write down two matrices (2x2 for simplicity)

Compute a linear operator like Ax



(Extra) try it yourself!

Write down two matrices (2x2 for simplicity)

Compute a linear operator like Ax

Now apply B to the result like $B(Ax)$



(Extra) try it yourself!

Write down two matrices (2×2 for simplicity)

Compute a linear operator like Ax

Now apply B to the result like $B(Ax)$

In principle the result should be equal to the operation $BA(x)$, you'll see the formula naturally appear!



(Extra) try it yourself!

Write down two matrices (2×2 for simplicity)

Compute a linear operator like Ax

Now apply B to the result like $B(Ax)$

In principle the result should be equal to the operation $BA(x)$, you'll see the formula naturally appear!



This but faster

Each dot product depends on the previous result...



This but faster

Each dot product depends on the previous result...

...But each row times the column is independent!



This but faster

Each dot product depends on the previous result...

...But each row times the column is independent!

Same operation with different data, HUGE GPU playground



The slow implementation

```
void matrix_mul(int*a, int * b, int width, int height){  
    for(int i=0;i<width;i++)  
        for(int j=0;j<height;j++){  
            int sum=0;  
            for(int k=0;k<height;k++){  
                int a = A[i * width + k];  
                int b = B[k * height + j];  
                sum += a*b;  
            }  
            B[i*width+j]=sum;  
        }  
    }
```



Why is it slow?

$$\sim O(n^3)$$



Why is it slow?

$$\sim O(n^3)$$

For large matrices (100.000x100.000) this is
already a problem

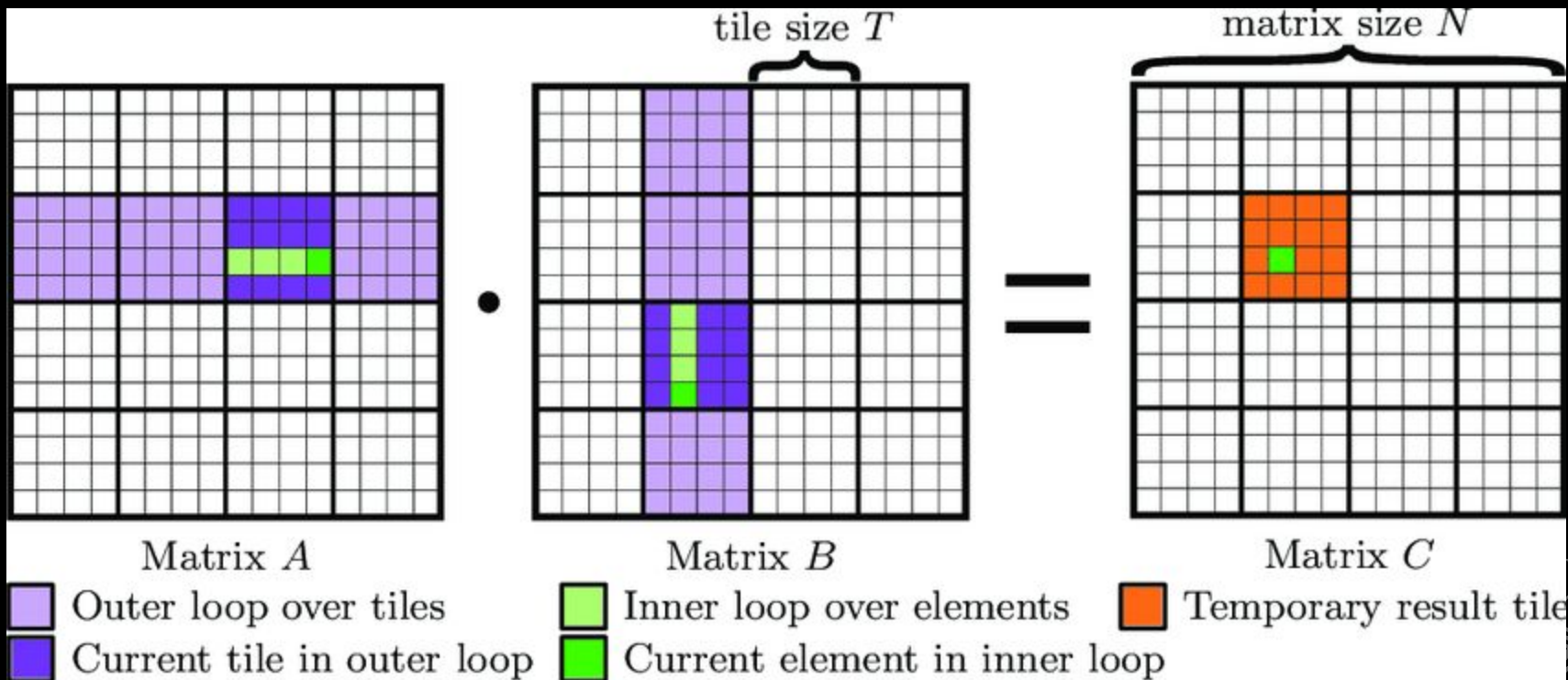


We can do better

The GPU implementation!
VSCODE incoming



Tiling





Exercises

Past years exam
exercises



Exercise 1, indexing

Describe how to obtain a unique ID for each thread by using the block ID and thread ID, in the case of a 2D grid and 3D blocks



Exercise 1, indexing

Describe how to obtain a unique ID for each thread by using the block ID and thread ID, in the case of a 2D grid and 3D blocks

```
ID = threadIdx.x +  
    blockDim.x * (threadIdx.y +  
    blockDim.y * (threadIdx.z +  
    blockDim.z * (blockIdx.x +  
    gridDim.x * blockIdx.y  
    )))
```



Exercise 2, thread scheduling

Assume a CUDA device allowing 8 blocks, 1024 threads per SM and 512 threads in each block

- For matrix multiplication, should we use 8x8, 16x16 or 32x32?
- Analyze the pros and cons of each choice



Exercise 2, thread scheduling

- 8x8

this means $8 \times 8 = 64$ threads per block
each SM have 1024 threads

$1024 / 64 = 16$ blocks, but we have 8 per SM



Exercise 2, thread scheduling

- 8x8

this means $8 \times 8 = 64$ threads per block
each SM have 1024 threads

$1024 / 64 = 16$ blocks, but we have 8 per SM

- 16x16

this means $16 \times 16 = 256$ threads per block

$1024 / 256 = 4$ blocks



Exercise 2, thread scheduling

- 8x8

this means $8 \times 8 = 64$ threads per block
each SM have 1024 threads

$1024 / 64 = 16$ blocks, but we have 8 per SM

- 16x16

this means $16 \times 16 = 256$ threads per block

$1024 / 256 = 4$ blocks

- 32x32

It's not even schedulable



Exercise 3, gpu capabilities

You need to write a kernel that operates on an image represented by a matrix of size $1440 \times 1280 \times 24$. You would like to assign one thread to each matrix element. You would like your thread blocks to use the maximum number of threads per block possible on your device.

- How would you select the dimensions of a 2D grid and 2D rectangular blocks for your kernel, minimizing the number of idle threads? Consider a device having compute capability 1.3.
- b) How would you select the dimensions of a 2D grid and 3D blocks with the three sides all equal for your kernel, minimizing the number of idle threads? Consider a device having compute capability 3.5



Exercise 3, gpu capabilities

Technical specifications	Compute capability (version)									
	1.0	1.1	1.2	1.3	2.x	3.0	3.5	3.7	5.0	5.2
Maximum dimensionality of grid of thread blocks	2				3					
Maximum x-dimension of a grid of thread blocks	65535					2 ³¹ -1				
Maximum y-, or z-dimension of a grid of thread blocks	65535									
Maximum dimensionality of thread block	3									
Maximum x- or y-dimension of a block	512				1024					
Maximum z-dimension of a block	64									
Maximum number of threads per block	512				1024					
Warp size	32									
Maximum number of resident blocks per multiprocessor	8					16			32	
Maximum number of resident warps per multiprocessor	24		32		48	64				
Maximum number of resident threads per multiprocessor	768		1024		1536	2048				
Technical specifications	1.0	1.1	1.2	1.3	2.x	3.0	3.5	3.7	5.0	5.2
	Compute capability (version)									

Exercise 3, gpu capabilities

max block size = 512 = 2^{**9}

we can split in half

- $x = 2^{**4}$

- $y = 2^{**5}$

$$\text{gridDim.x} = 1440 / x = 90$$

$$\text{gridDim.y} = (1280 * 24) / y = 960$$

We can do better with 2^{**3} and 2^{**6}

$$\text{gridDim.x} = 180$$

$$\text{gridDim.y} = 480$$



Exercise 3, gpu capabilities

max block size = 1024 = 2^{10}

we can split in three

- $x = 2^3$

- $y = 2^4$

- $z = 2^3$

$$\text{gridDim.x} = 1440 / x = 180$$

$$\text{gridDim.y} = (1280 * 24) / (y * z) = 240$$



The End

THE END

