VRIJE UNIVERSITEIT BRUSSEL

ULB UNIVERSITÉ LIBRE DE BRUXELLES

GPU Computing

Lesson 5: Thread blocks and shared memory

Gauthier Lafruit & Jan Lemeire

2022-2023

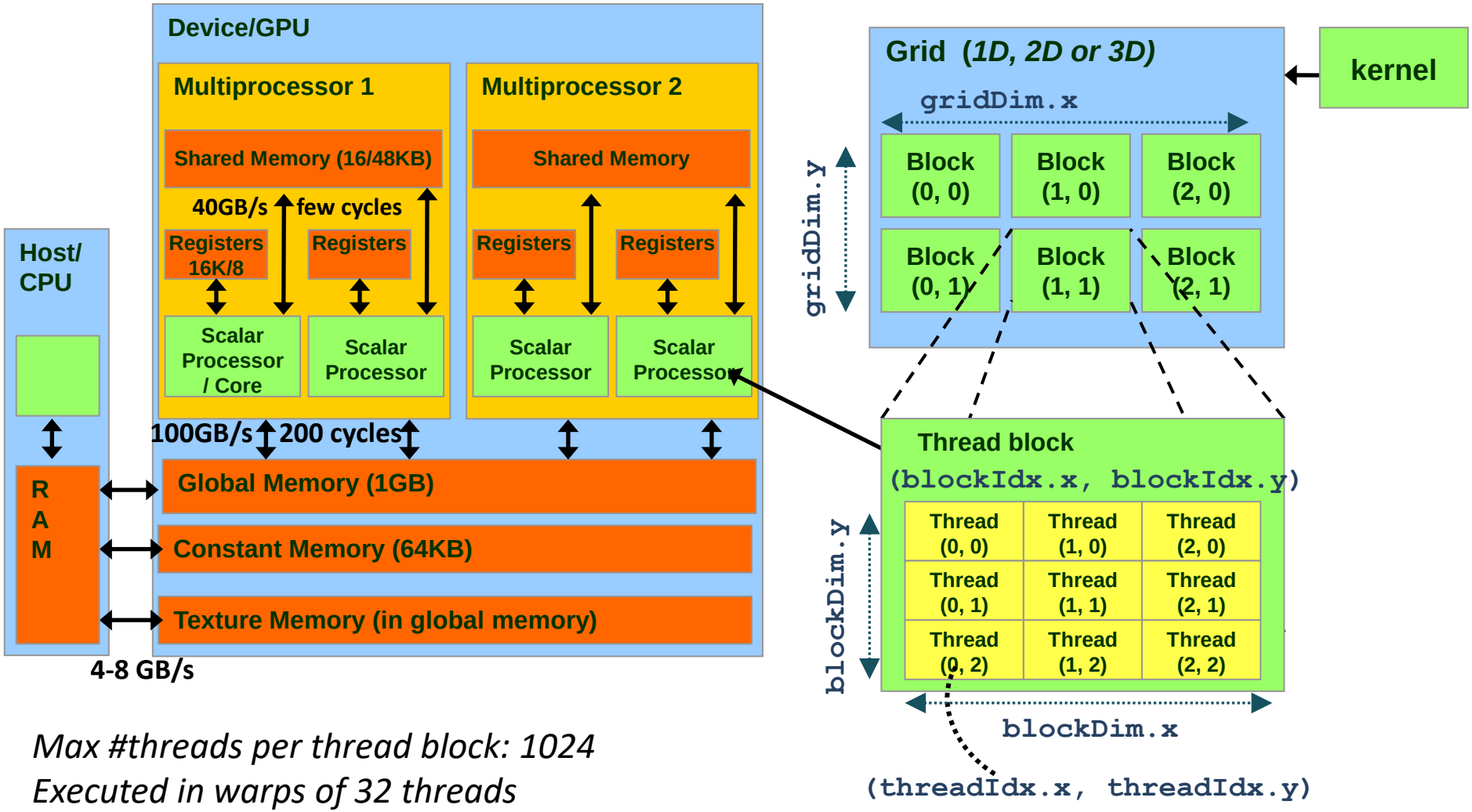
http://parallel.vub.ac.be/education/gpu

VRIJE UNIVERSITEIT BRUSSEL

Levels of Understanding

- Level 0
  - Host code
- Level 1
  - Parallel execution on the device
- Level 2 *=> explained here*
  - Device model and work groups
- Level 3 *=> explained later*
  - Hardware threads & SIMT

GPU Concepts



Max #threads per thread block: 1024
Executed in warps of 32 threads
Max thread blocks simultaneously on MP: 8
Max active warps on MP: 24/48

CUDA terminology

Level 2

GPU Model and Thread blocks

VRIJE UNIVERSITEIT BRUSSEL

Execution Model

- Kernel = smallest unit of execution, like a C function, executed by each thread
- Data parallelism: kernel is run by a grid of thread blocks
- A thread block consists of instances of the same kernel: thread
- Different data elements are fed into the threads of the thread blocks
  - ➡ We talk about *stream computing*

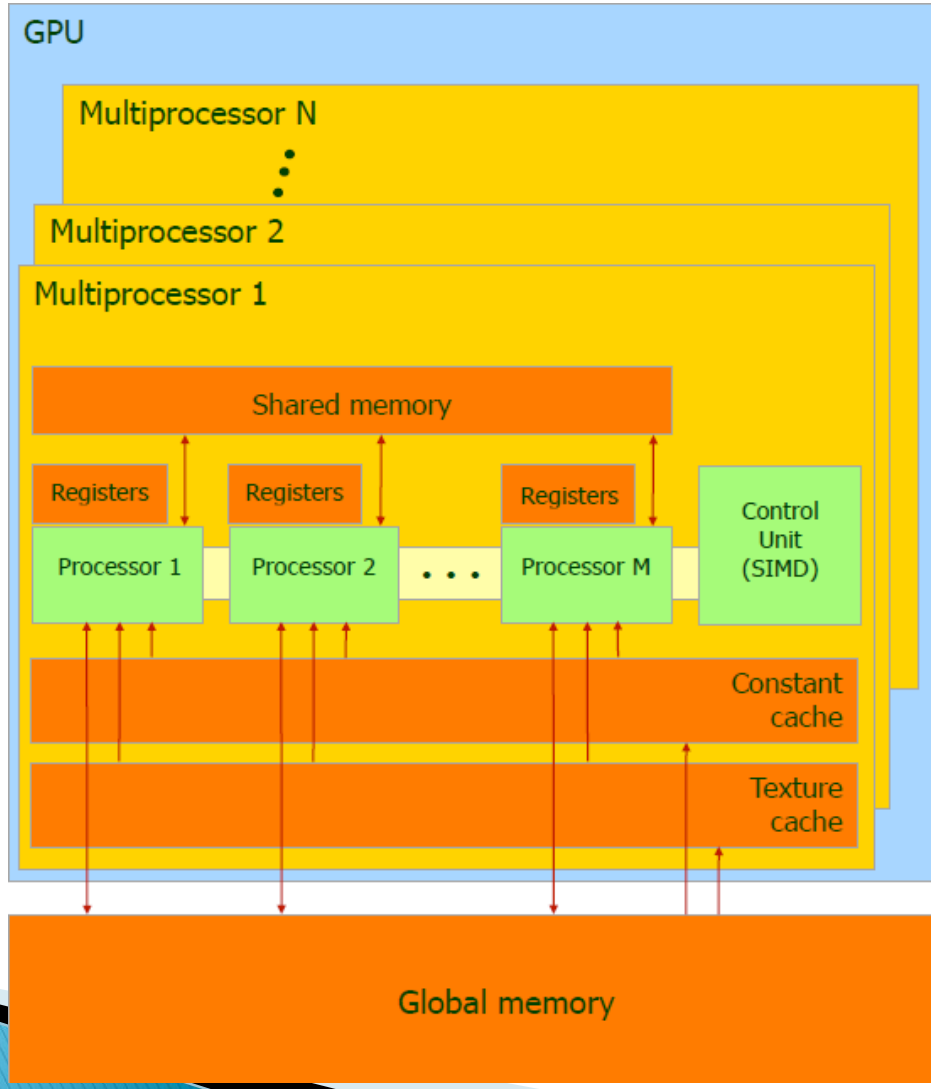
VRIJE UNIVERSITEIT BRUSSEL

Thread blocks

- Threads are grouped into thread blocks
  - Number of threads of a block is determined by the programmer (same for all blocks)
- **A thread block is executed on one multiprocessor** (MP)
  - From start to end
- Threads of the same thread block share **shared memory**
  - Kind of explicit cache
- Within a thread block, **synchronization** among threads is possible
  - With the barrier statement.
  - Synchronization between blocks is NOT possible

VRIJE UNIVERSITEIT BRUSSEL

GPU Model



VRIJE UNIVERSITEIT BRUSSEL

Memory & Data Locality

- We have multiple types of memory:
  - Global memory/Constant memory
  - Shared memory
  - Local memory
- We want to exploit multilevel caches by using
  - Spatial locality (address space)
  - Temporal locality (data accessed before will be likely accessed again)
  - Tasks repeated many times (last accessed)
- On GPU YOU are responsible for the content of the caches!
  - Advantage Control when the writes happen.

- Tip: It's a perfectly valid approach to develop a program, prove the concept, and then deal with locality issues

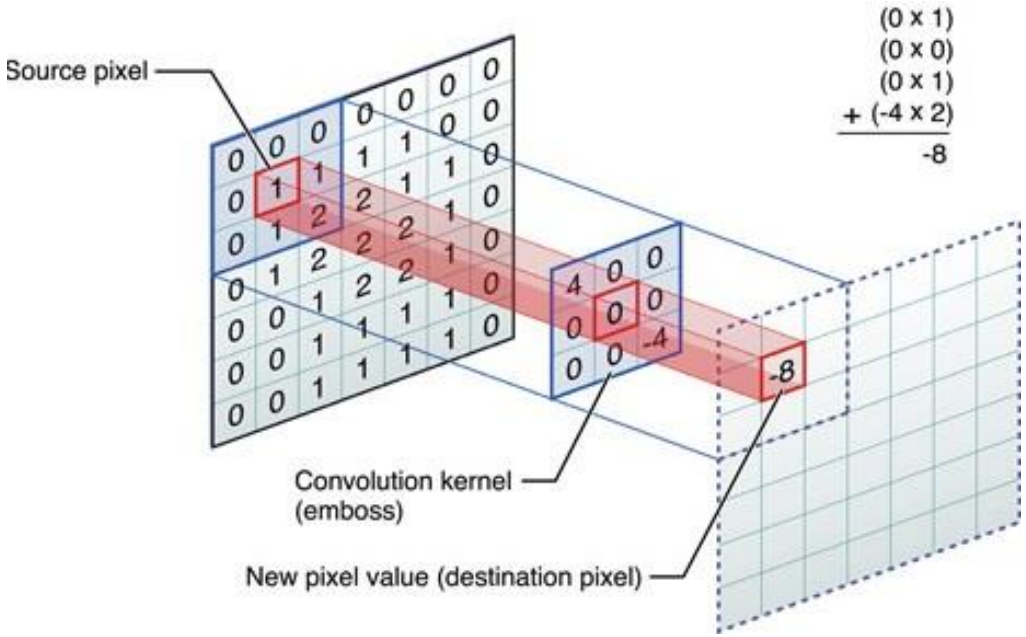
Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	One thread	Thread
Local	Off-chip	No	R/W	One thread	Thread
Shared	On-chip	N/A	R/W	All threads in a block	Block
Global	Off-chip	No	R/W	All threads + host	Application
Constant	Off-chip	Yes	R	All threads + host	Application
Texture	Off-chip	Yes	R	All threads + host	Application

VRIJE UNIVERSITEIT BRUSSEL

Using shared memory

- **Dynamic**
  - Kernel_function<<<num_blocks, num_threads, shared_memory_size * sizeof(int)>>>(params…)
  - __global__ void Kernel(params) {
    extern __shared__ int a[];
    }
- **Static**
  - #define CONSTANT_SIZE 100
  - __global__ void Kernel(params) {
    __shared__ int a[CONSTANT_SIZE];
    }
- See: https://stackoverflow.com/questions/5531247/allocating-shared-memory

Example: convolution



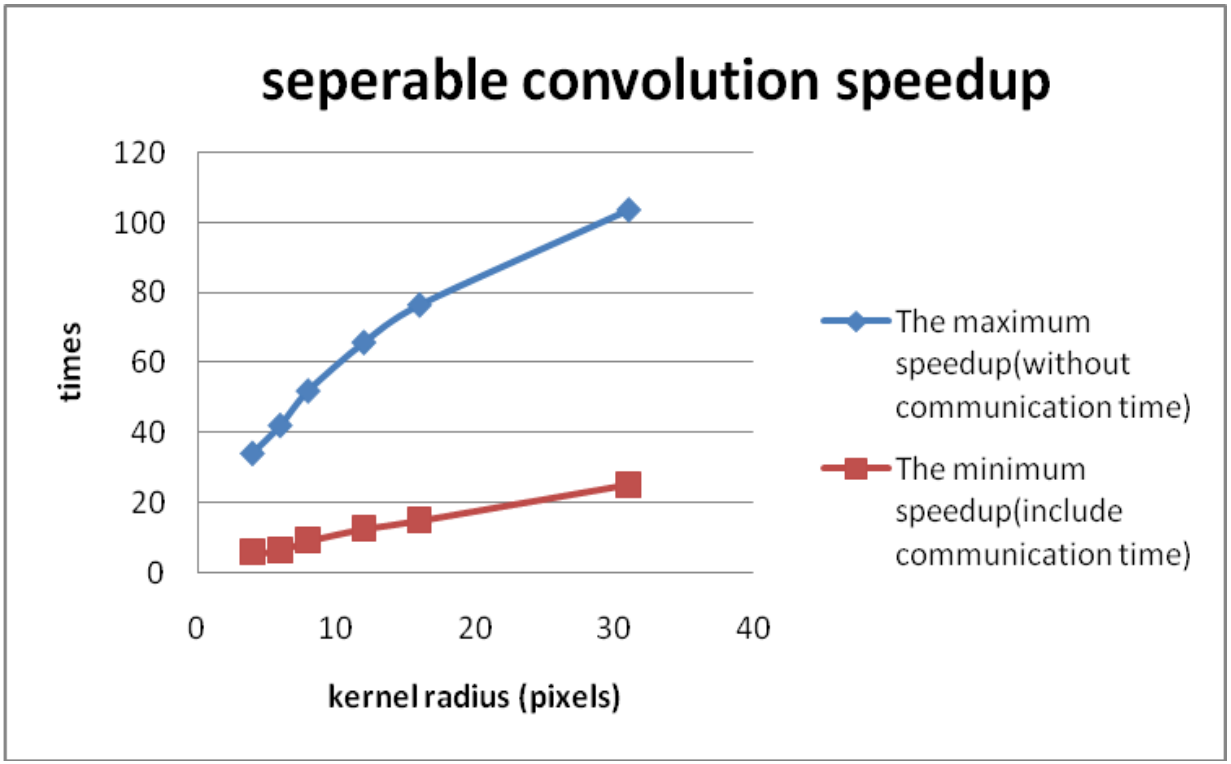
Parallelism: +++
Locality: ++
Work/pixel: ++

3x3 kernel (also called *filter* or *mask*) is applied to each pixel of the image

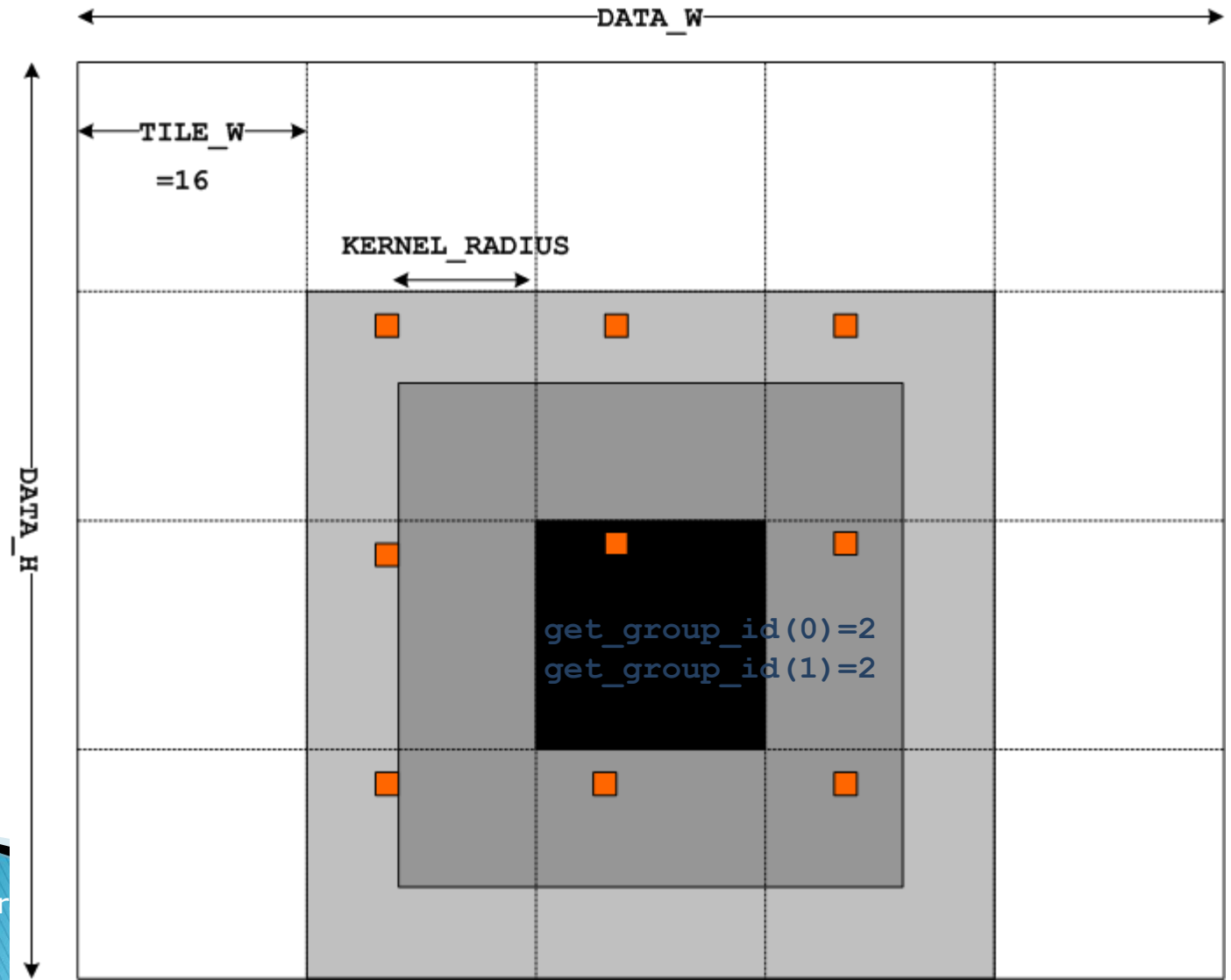
Examples of convolution

Edge detection with sobel filter

Speedup


seperable convolution speedup
120
100
80
60
40
20
0
times
0
10
20
30
40
kernel radius (pixels)
The maximum speedup(without communication time)
The minimum speedup(include communication time)

Convolution on GPU

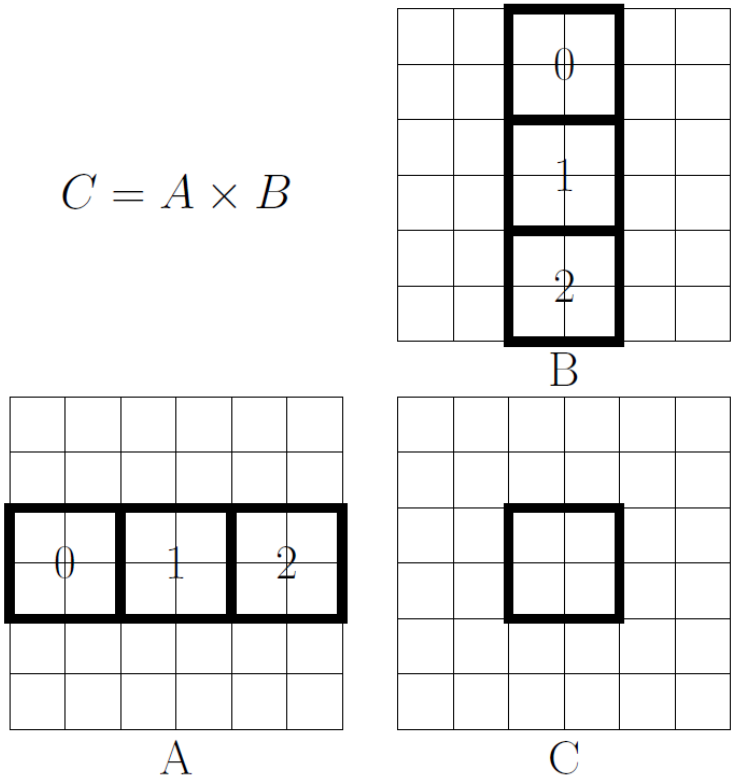


Convolution Kernel Code

```
__global__ void Convolution(float* A, float* B, float* C, int DATA_WIDTH, int
KERNEL_WIDTH)
{
    int col = blockIdx.x * BLOCK_SIZE + threadIdx.x;
    int row = blockIdx.y * BLOCK_SIZE + threadIdx.y;

     __shared__ float shm[BLOCK_SIZE][BLOCK_SIZE];
     shm[threadIdx.y][threadIdx.x] = A[col * DATA_WIDTH + row];
      … // copy 9 pixels to shared

      __syncthreads();

      float tmp = 0;
      for (int i = 0; i< KERNEL_WIDTH;i++)
            for (int j = 0;j<KERNEL_WIDTH;j++)
                  tmp += shm[threadIdx.y + i][threadIdx.x + j] * C[j*KERNEL_WIDTH + i];
      B[col*DATA_WIDTH + row] = tmp;
}
```

Matrix Multiplication

$C = A \times B$



Happens in iterations:

- *first blocks 0 are multiplied, then 1 are multiplied and added, and at last blocks 2*
- *In each iteration:*
  - *Thread copies element of A and B to shared*
  - *Barrier synchronization*
  - *Calculates sum of products of A row and B column*

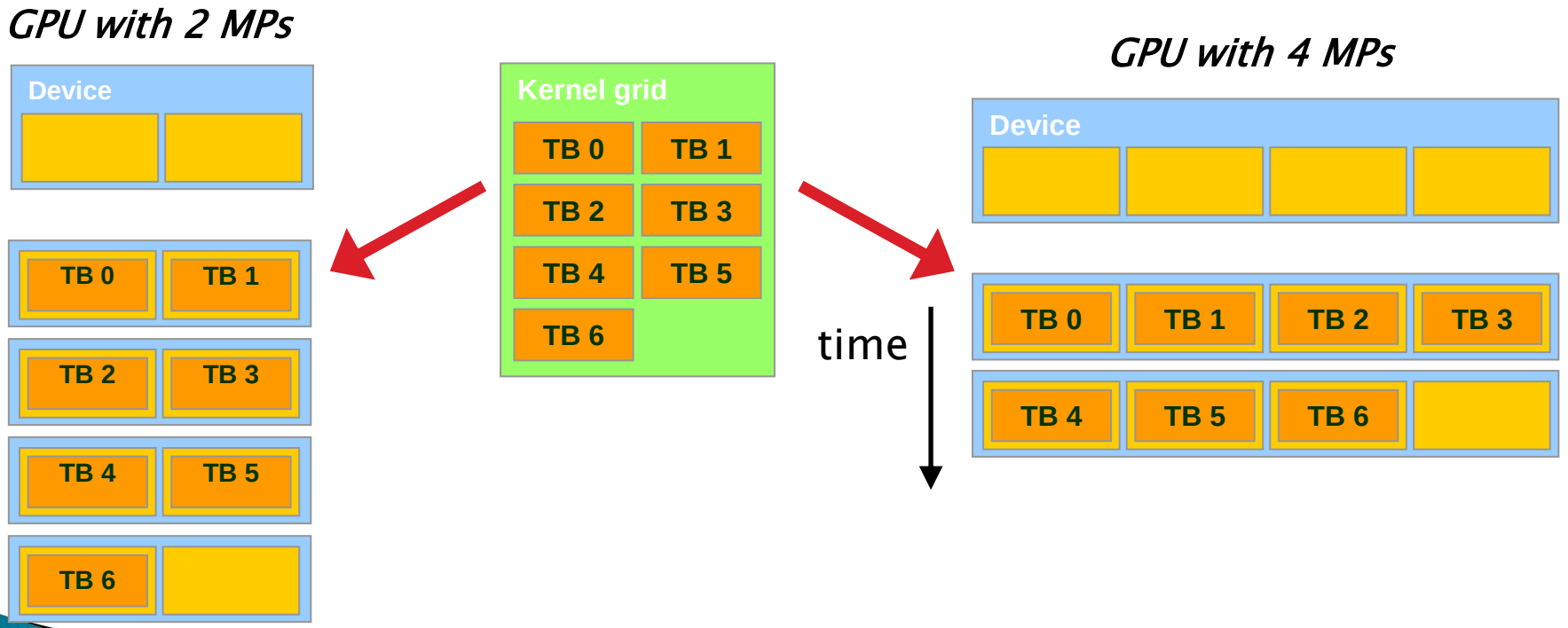
VRIJE UNIVERSITEIT BRUSSEL

Execution Model

- Execution of N blocks of M threads
- Thread blocks are assigned to multiprocessors **(MPs)**
  - A thread block stays there until it completes
- Multiprocessors may execute multiple thread blocks concurrently
- Thread blocks not yet assigned to a multiprocessor must wait
- The order in which thread blocks execute is non-deterministic
- Consequences:
  - There can be no interaction between thread blocks
  - CUDA code scales inherently

Thread block execution

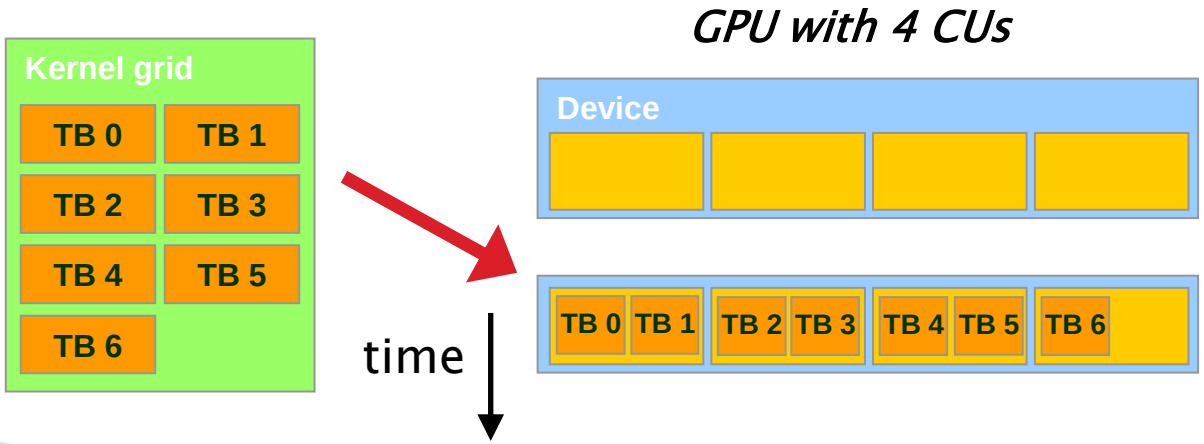
- Simple scheduler
  - Assigns thread blocks to available multiprocessors (MPs)
  - Basically, a waiting queue for thread blocks
- Thread blocks (TBs) execute independently
  - Global Synchronization among thread blocks is not possible!



VRIJE UNIVERSITEIT BRUSSEL

Multiple TBs per MP

- One MP can execute TBs concurrently
- Determined by available resources (hardware limits):
  - *Max. TBs simultaneously on MP: 8*
  - *Max. threads simultaneously on MP : 1024*
  - *Private memory (registers) per MP : 16/48KB*
  - *Shared memory per MP : 16/32KB*



VRIJE UNIVERSITEIT BRUSSEL

Exercise: Matrix Vector Operation

- Matrix A mxn
- Vector B n

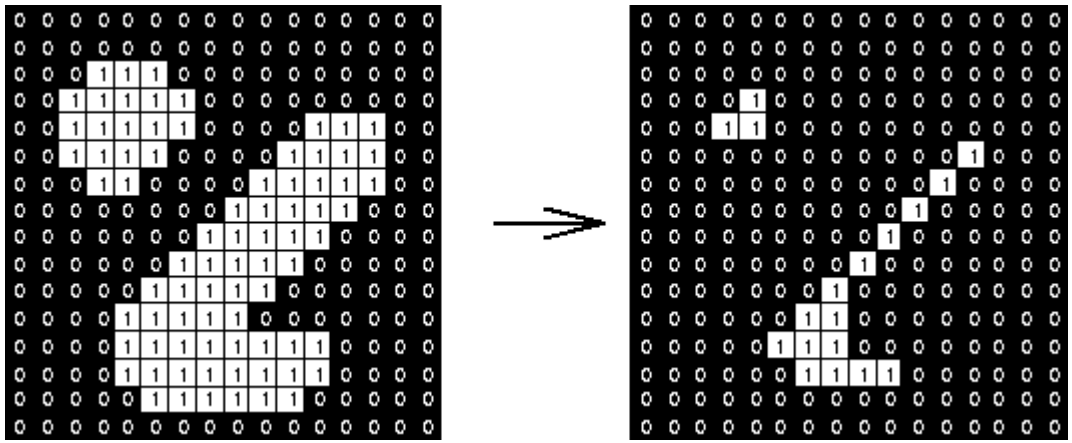
- Computation?
  - Repeat N times:
    - A[i,j] = A[i,j] + A[i,j]*B[j]

- Observe
  - Data throughput in function of N
  - Computational throughput in function of N

VRIJE UNIVERSITEIT BRUSSEL

Exercise: Erosion

- Typical operation in image processing
- Given an input pixel, the value of the corresponding output pixel is the minimum of values of pixels under a mask centered on the input pixel
- Example Erosion with a 3x3 mask on a binary image:



- Implement erosion for one-dimensional data for a parameterizable mask width
  1. Doing everything in global memory
  2. Using local memory
- Try two-dimensional erosion

Level 3

See other chapter