VRIJE UNIVERSITEIT BRUSSEL

Computerarchitectuur

The GPU architecture

Jan Lemeire
2024-2025

VRIJE UNIVERSITEIT BRUSSEL

What to know from this chapter

- Why is the potential computational power of the GPU so much higher than that of a CPU?
- Why is it good to have more threads than cores?
- Why is a GPU called a thread engine? Why are fine-grained threads efficient, while not on CPU?
- What are potential performance bottlenecks that prevent from attaining full performance?
- What is the difference between vector processing (SIMD) and GPU thread processing (SIMT)?

versus

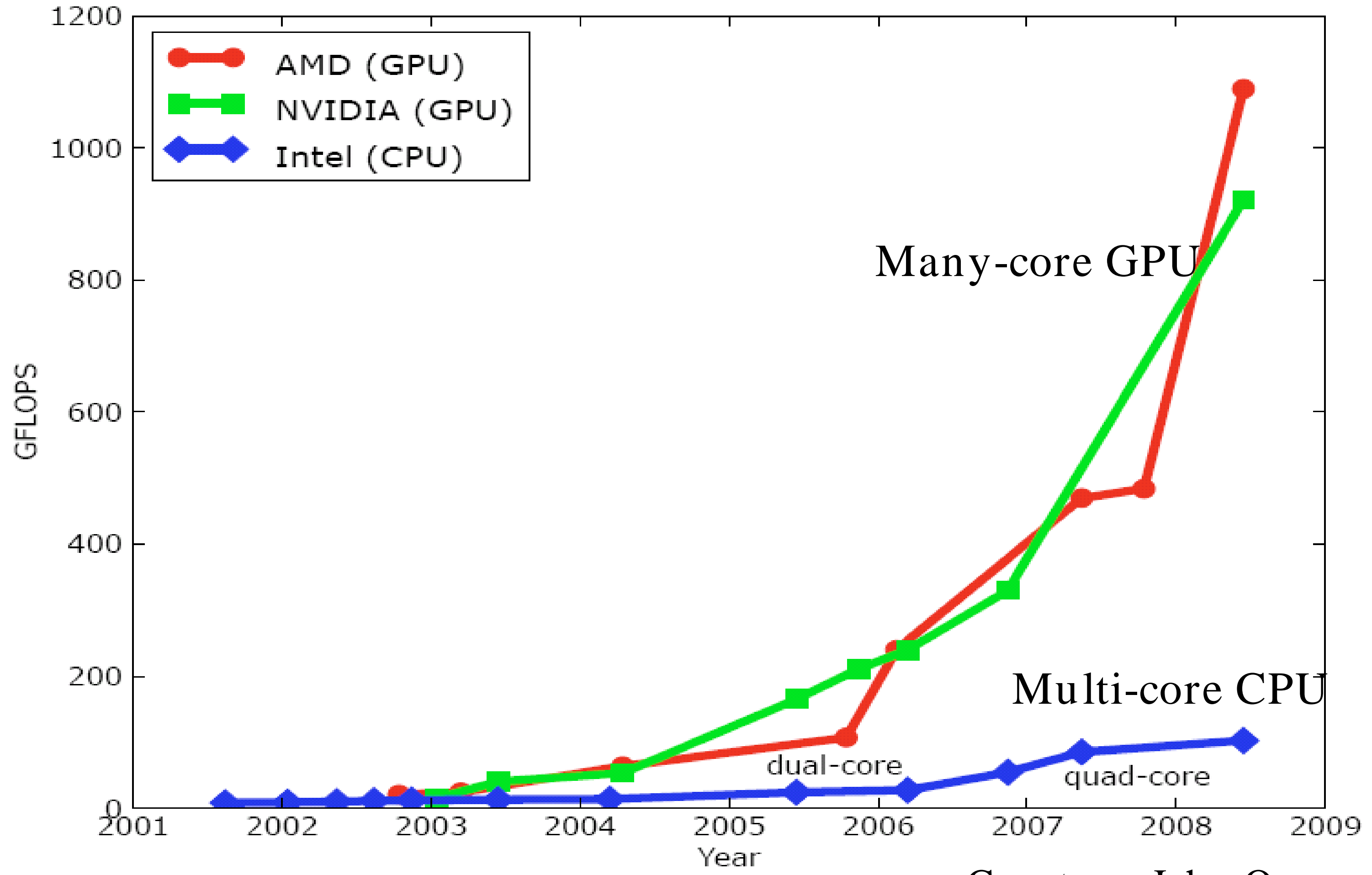
1. The Power of GPUs

2010
350 Million triangles/second
3 Billion transistors GPU
1995
5.000 triangles/second
800.000 transistors GPU
2016
14.000 Million triangles/second
15 Billion transistors GPU

[Beta - MikeTheSempai]

B SYNCHRONIZE

Graphical Processing Units (GPUs)



VRIJE UNIVERSITEIT BRUSSEL

Supercomputing for free

- FASTRA at university of Antwerp

http://fastra.ua.ac.be

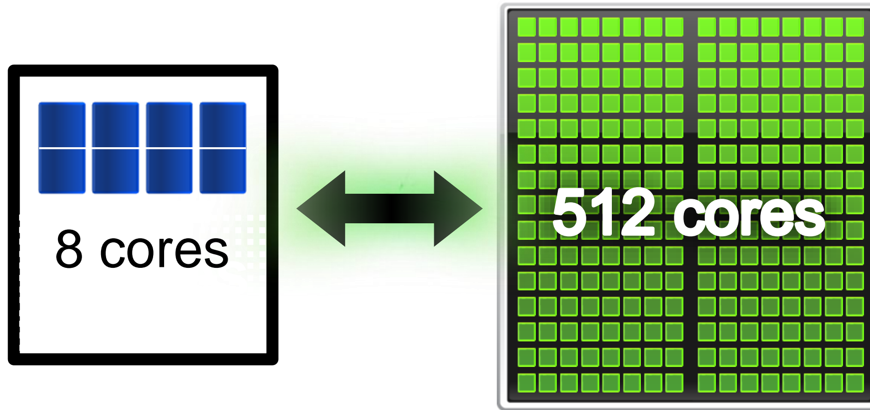
Collection of 8 graphical cards in PC

FASTRA 8 cards = 8x128 processors = 4000 euro

Similar performance as University's supercomputer (512 regular desktop PCs) that costed 3.5 million euro in 2005

"**Supercomputing in a box**": a high-end GPU cost 500 to 2500 euro and has equivalent power as 40 quadcore CPUs

Why are GPUs faster?

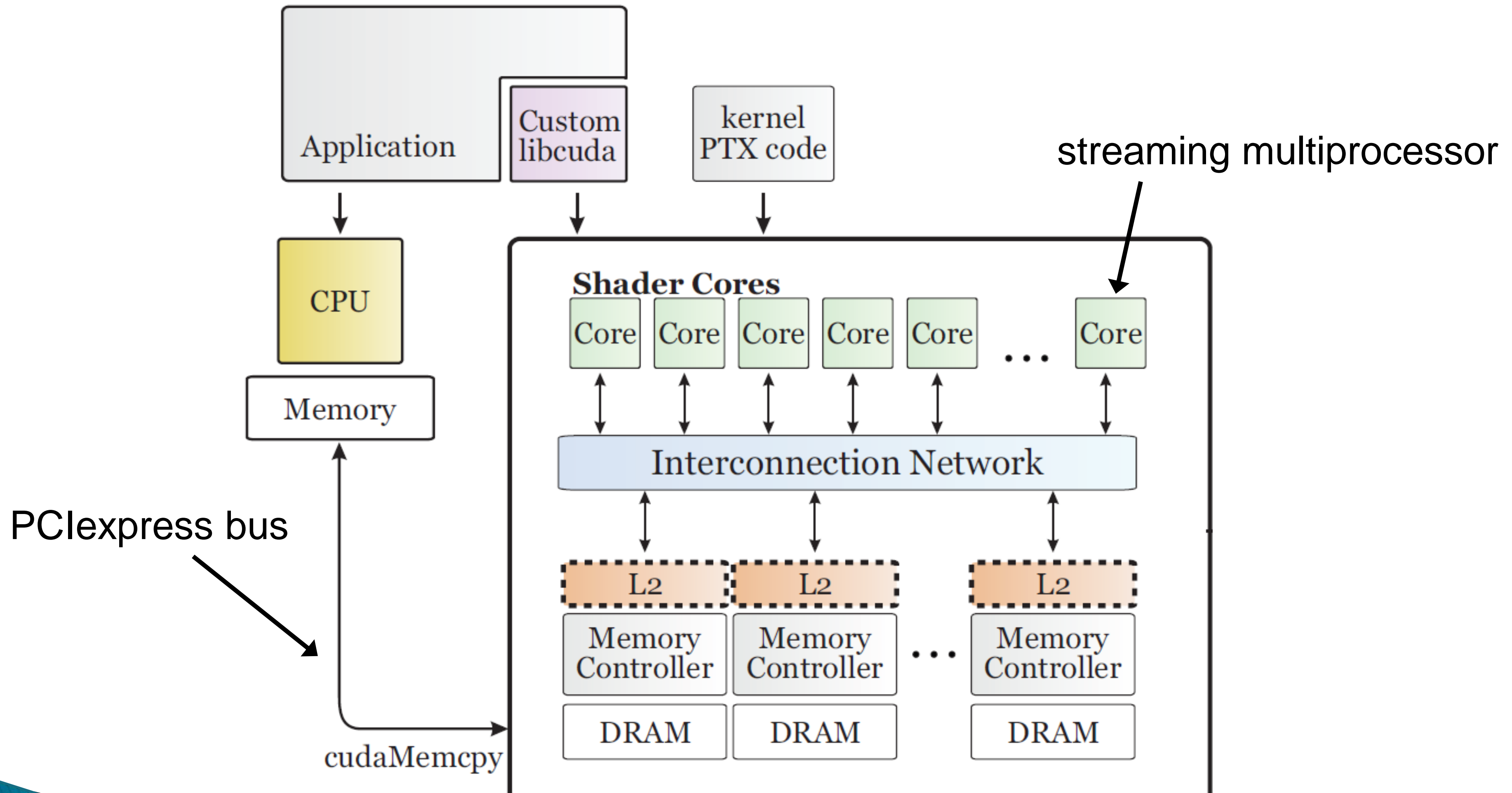


Both, about 15 billion transistors

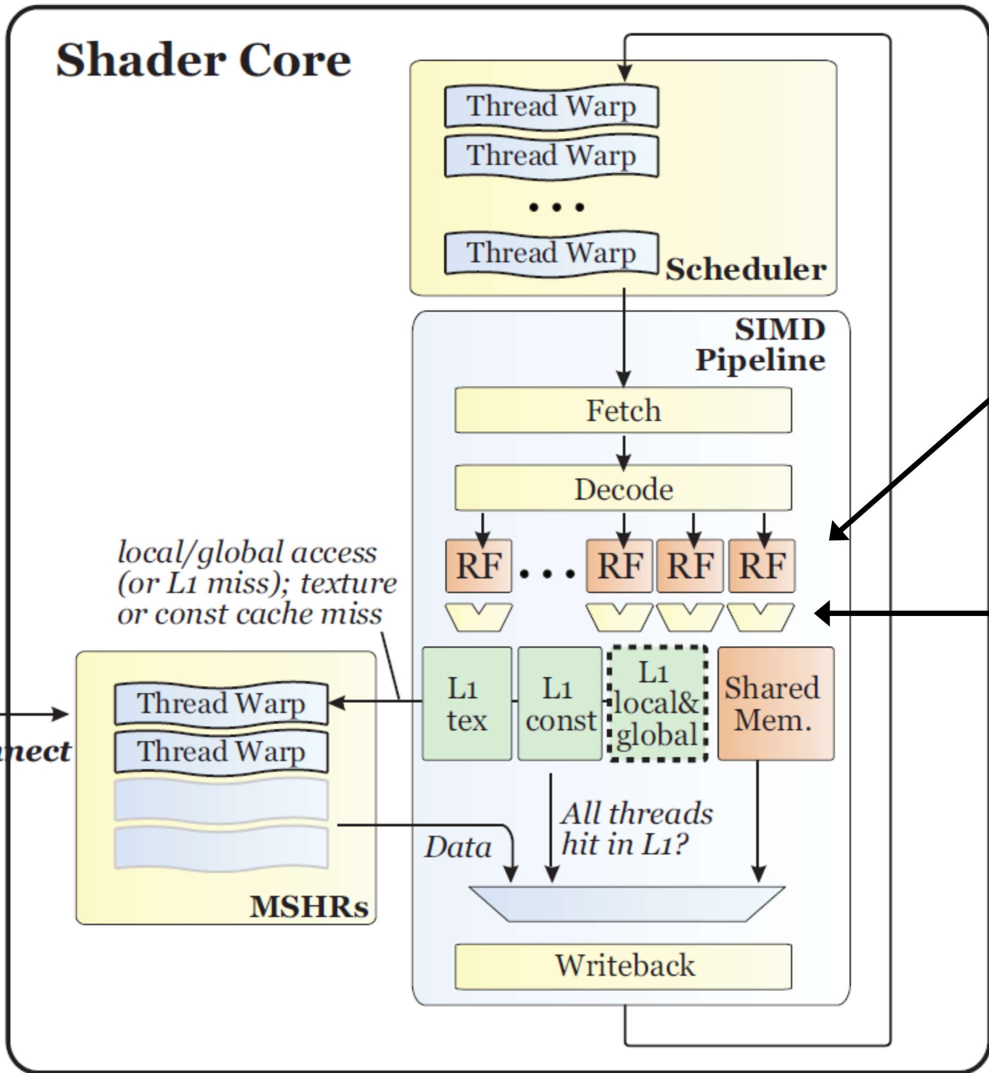
No branch prediction, out-of-order execution, …

Devote transistors to… computation

GPU Architecture



1 Streaming Multiprocessor



The Same Instruction is executed on Multiple Thread (SIMT)

width of pipeline:
8 – 32 – 192 - 128

Scalar processors (called 'cores' by CUDA)

1 multiprocessor

Full name	Abbreviated name
NIVDIA Tesla C2050	Fermi
NIVDIA GeForce GTX 650 Ti	Kepler
NIVDIA Quadro K620	Maxwell
NIVDIA GeForce GTX 1060 6GB	Pascal
AMD Radeon R9 380	Tonga

	Fermi	Kepler	Maxwell	Pascal	Tonga
$N(\Pi)$	14	4	3	10	28
f_{clock} (MHz)	1147	1032	1058	1506	1010
issue limit	1	4	4	4	1
$\lvert\omega\rvert$	32	32	32	32	64
Resources					
Group slots	8	16	32	32	16
Warp slots	48	64	64	64	40
Local memory (KB)	48	48	64	96	64
Registers (KB)	128	256	256	256	

Scalar processors (called 'cores' by CUDA) → ALU count

	Fermi	Kepler	Maxwell	Pascal	Tonga
$max(\lvert\gamma\rvert)$	1024	1024	1024	1024	256
max(local memory) (KB)	48	48	48	48	32
ALU count	32	192	128	128	64
SFU count	8	32	32	32	-
RAM Bandwidth (GB/s)	144	86.4	29	192	176
L2 Cache size (KB)	768	256	2048	1536	512
L2 Cache line size (B)	128	128	128	128	64
L1 Cache size (KB)	16	16	64	48	16
max(global memory) (MB)	1024	672	512	1536	2880
RAM Size (MB)	2688	2048	2048	6144	4096

#LD/STO units = 16 32 32 32

VRIJE UNIVERSITEIT BRUSSEL

Peak GPU Performance

- GPUs consist of MultiProcessors (MPs) grouping a number of Scalar Processors (SPs)
- 1 Multiply-Add instruction performs 2 operations at once
- Nvidia GTX 280:
  - 30MPs x 8 SPs/MP x 2FLOPs/instr/SP x 1 instr/clock x 1.3 GHz

  = **624 GFlops**
- Nvidia Tesla C2050:
  - 14 MPs x 32 SPs/MP x 2FLOPs/instr/SP x 1 instr/clock x 1.15 GHz (clocks per second)

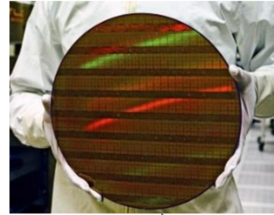
  = **1030 GFlops**

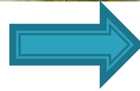
VRIJE UNIVERSITEIT BRUSSEL

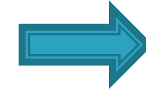
Other limit: memory bandwidth

- Nvidia GTX 280:
  - 1.1 GHz memory clock
  - 141 GB/s
- Nvidia Tesla C2050:
  - 1.5 GHz memory clock
  - 144 GB/s

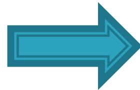
Example: real-time image processing

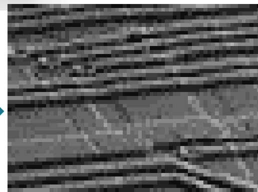




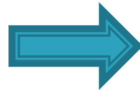


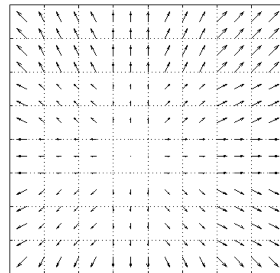
Images of 20MegaPixels



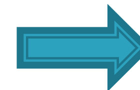


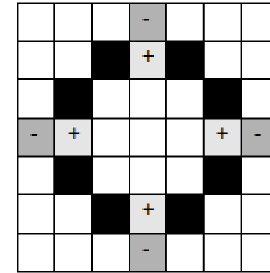
Pixel rescaling





lens correction





pattern detection

CPU gives only 4 fps
next generation machines need 50 fps
GPUs deliver 70 fps

Example: pixel transformation (FPN)

```
usgn_8  transform(usgn_8 in, sgn_16 gain, sgn_16 gain_divide,
sgn_8 offset)
{
    sgn_32 x = (in * gain / gain_divide) + offset;

    if (x < 0)
        x = 0;
    if (x > 255)
        x = 255;
    return x;
 }
```

Pixel transformation

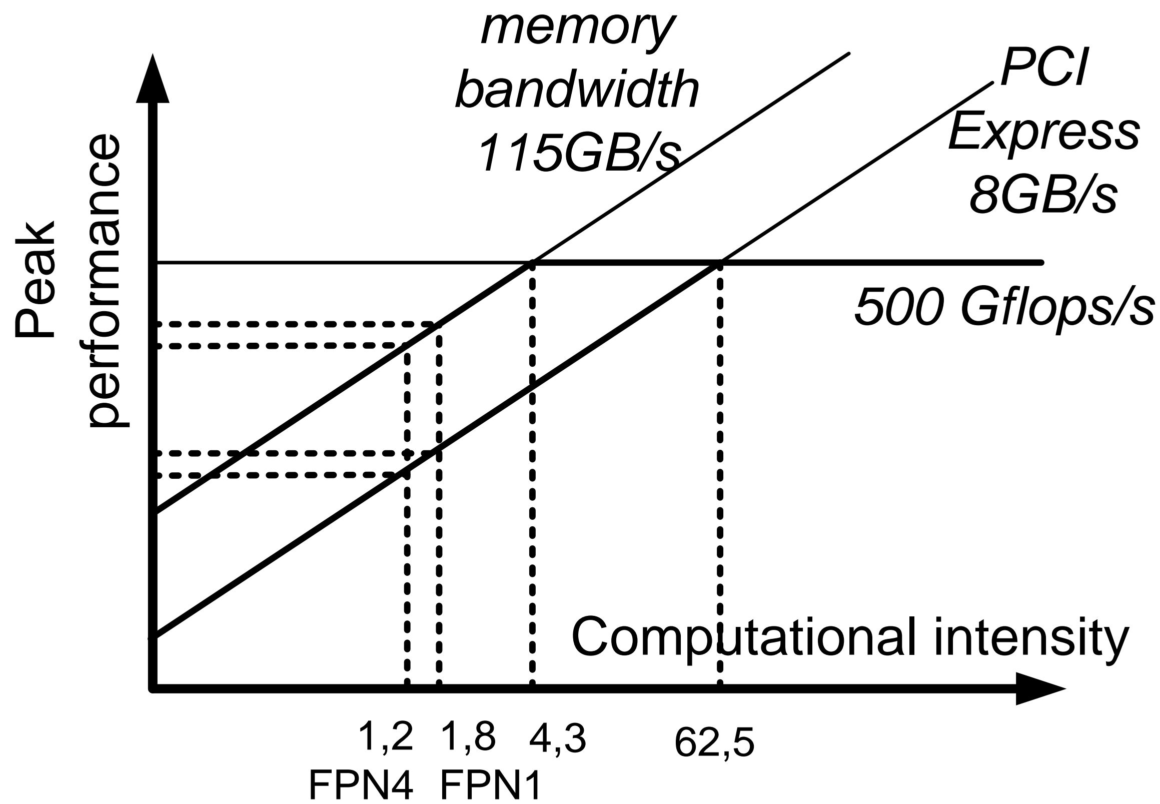
- Performance on Tesla C2050
- 1 pixel is represented by 1 byte [0-255]
  - Per pixel: read 4 bytes (pixel, gain, divide & offset) and write 1 byte
- Integer operations: performance is half of floating point operations
- **Pixel transformation**: typically 6 operations (1 index calculation, 3 integer calculations and 2 comparisons)

P_{mem} *(bytes/s)*	115 GB/s	P_{ops} *(ops/s)*	500 Gops/s	
bytes/pixel	5	*Ops/pix*	6	CI=1,2
P_{mem}xCI *(pix/s)*	**23 Gpix/s**	**Pix/s**	83 Gpix/s	

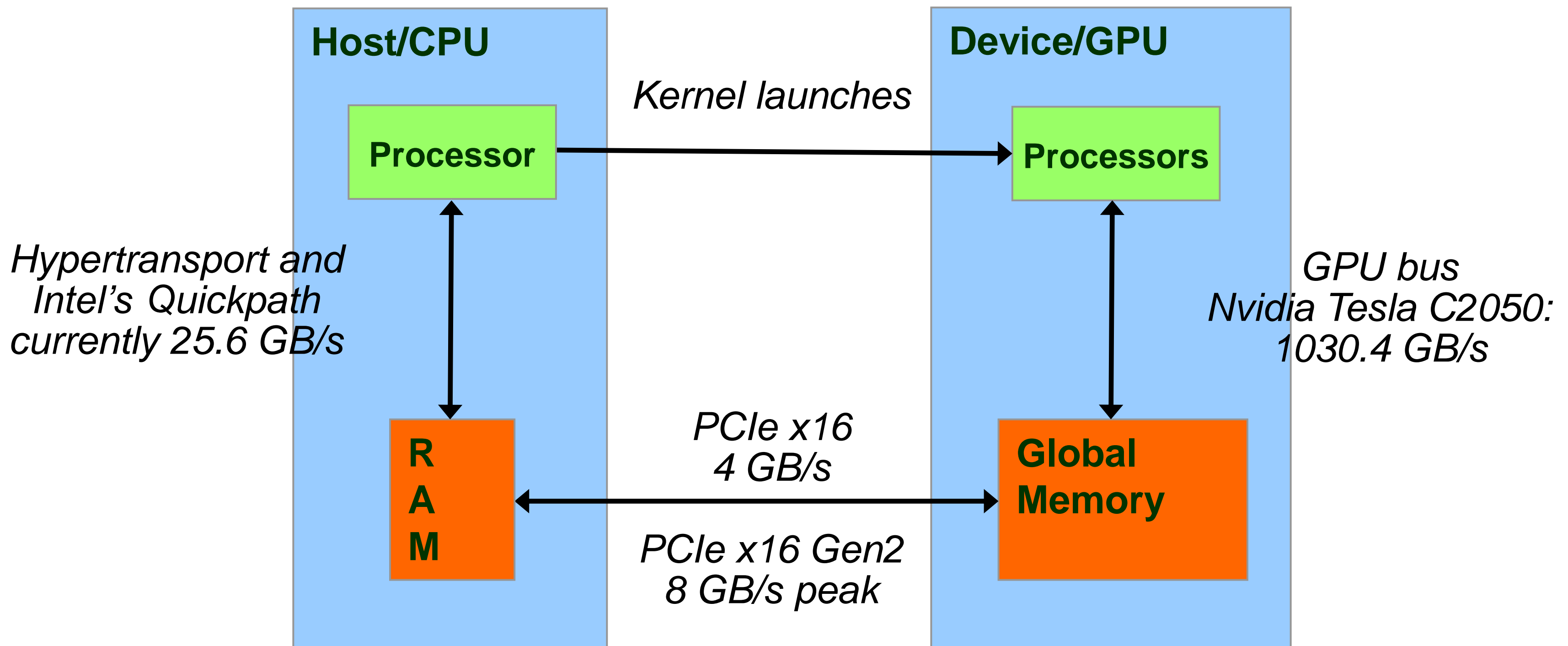
Memory-bound (take minimum)

CI = Computational Intensity

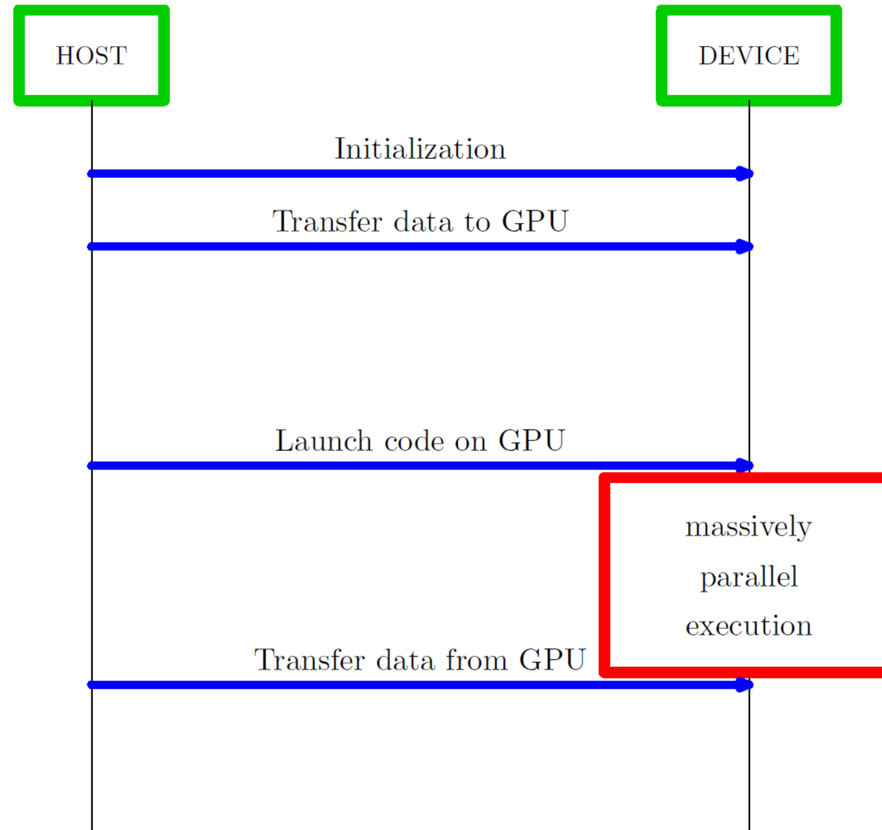
Roofline model applied



Host (CPU) – Device (GPU)



Typical Sequence of Events



2. Massive thread engine

Multithreading

- Performing multiple threads of execution in parallel
  - Replicate registers, PC, etc.
  - Fast switching between threads
- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn’t hide short stalls (eg, data hazards)

VRIJE UNIVERSITEIT BRUSSEL

Multithreading on CPU

- 1 process/thread simultaneously active per core
- When activating another thread: *context switch*
  - Stop program execution: flush pipeline (let all instructions finish)
  - Save state of process/thread into **Process Control Block** : registers, program counter and operating system-specific data
  - Restore state of activated thread
  - Restart program execution and refill the pipeline

Overhead

Fine multi-threading: Hardware threads

- In several modern CPUs
  - typically 2HW threads
- Devote extra hardware for process state
- Thread switching by hardware
  - (almost) no overhead
  - Within 1 cycle!
  - *Instructions in flight from different threads*

Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HyperThreading
  - Two threads: duplicated registers, shared function units and caches

Benefits of fine-grained multithreading

- Independent instructions (no bubbles)
- More time between instructions: possibility for *latency hiding*
  - Hide memory accesses
- If pipeline full
  - Forwarding not necessary
  - Branch prediction not necessary

VRIJE UNIVERSITEIT BRUSSEL

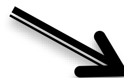
Thread executes kernel

- Massively parallel programs are usually written so that each thread computes one part of a problem
  - For vector addition, we will add corresponding elements from two arrays, so each thread will perform one addition
  - If we think about the thread structure visually, the threads will usually be arranged in the same shape as the data

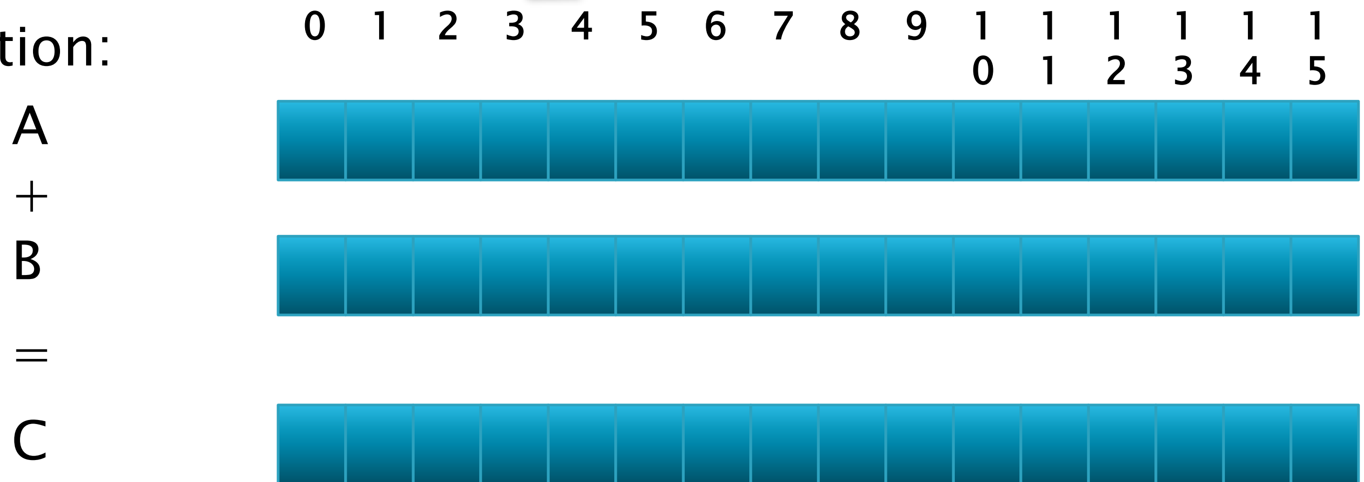
Thread Structure

- Consider a simple vector addition of 16 elements
  - 2 input buffers (A, B) and 1 output buffer (C) are required

Array Indices

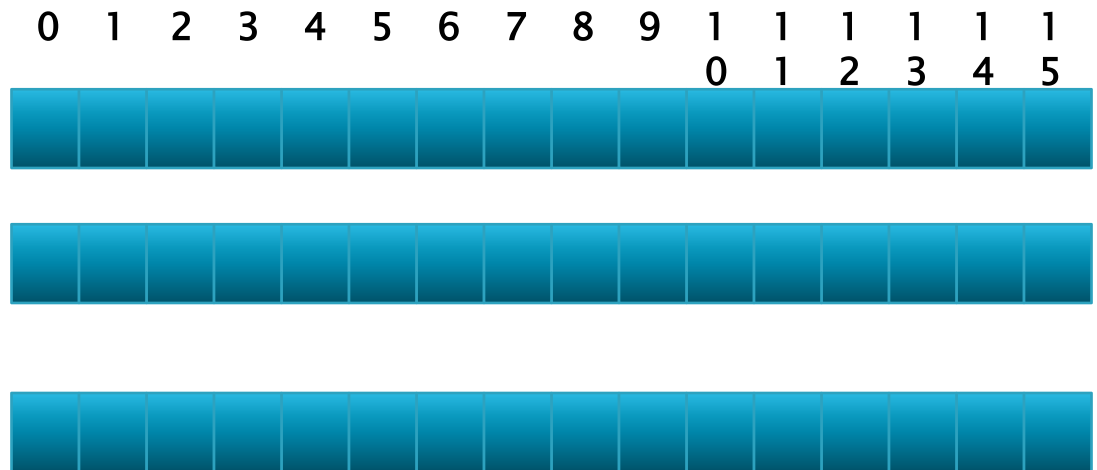


Vector Addition:



Thread Structure

- Create thread structure to match the problem
  - 1–dimensional problem in this case



Thread Structure

- Each thread is responsible for adding the indices corresponding to its ID

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Vector Addition:

A

+

B

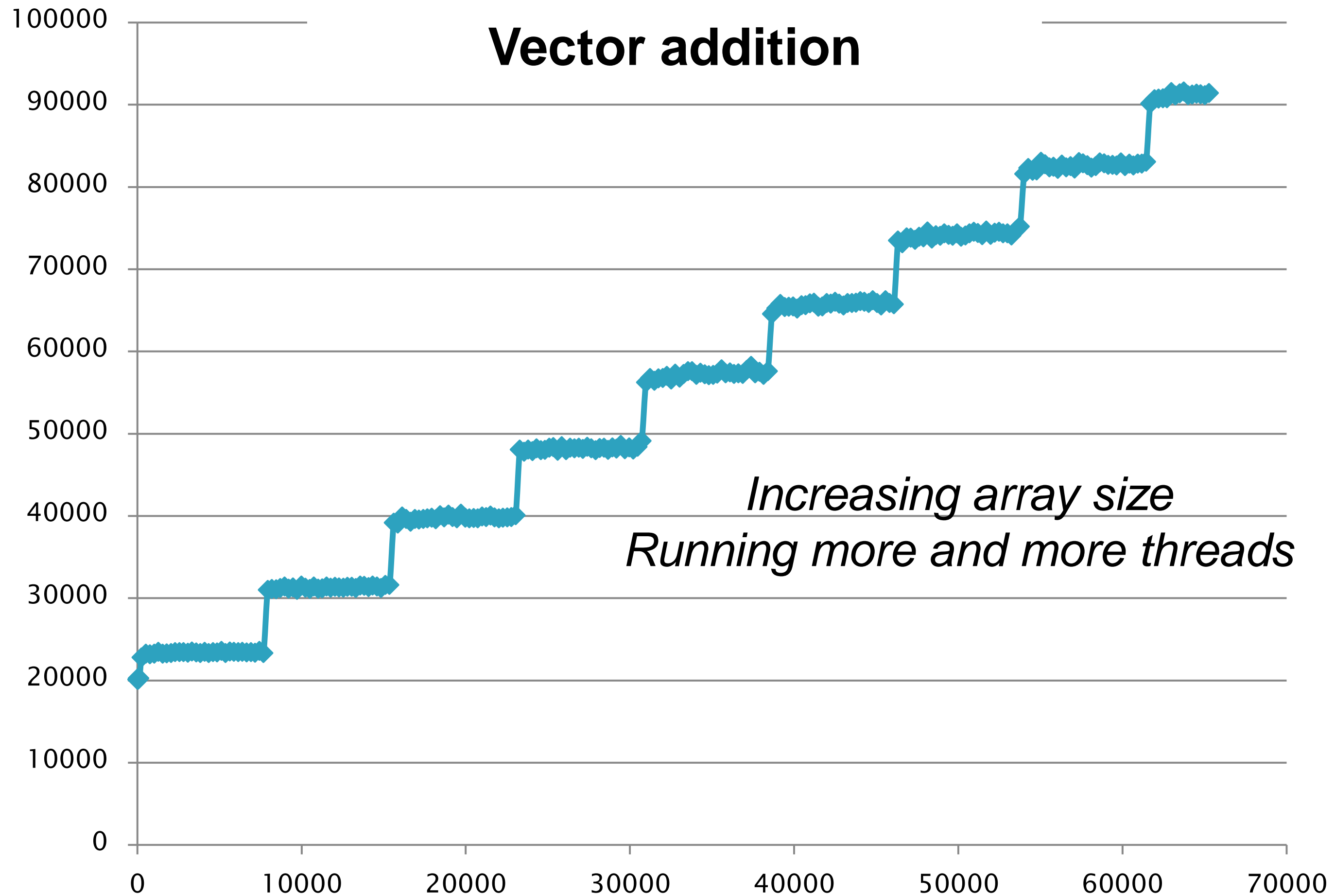
=

C

VRIJE UNIVERSITEIT BRUSSEL

The effect of parallelism

Runtime (ns)



Array size = #threads

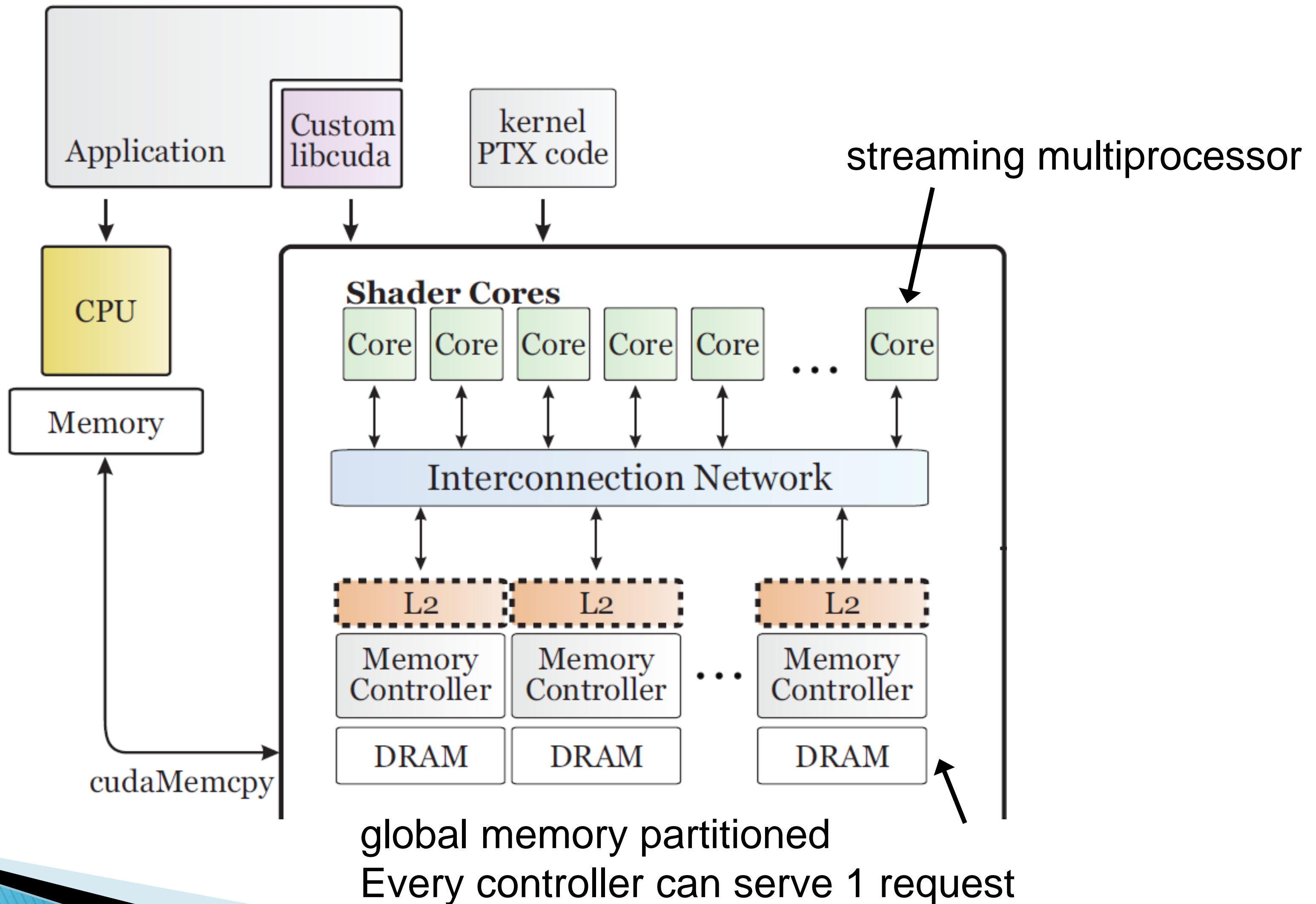
VRIJE UNIVERSITEIT BRUSSEL

Concurrency

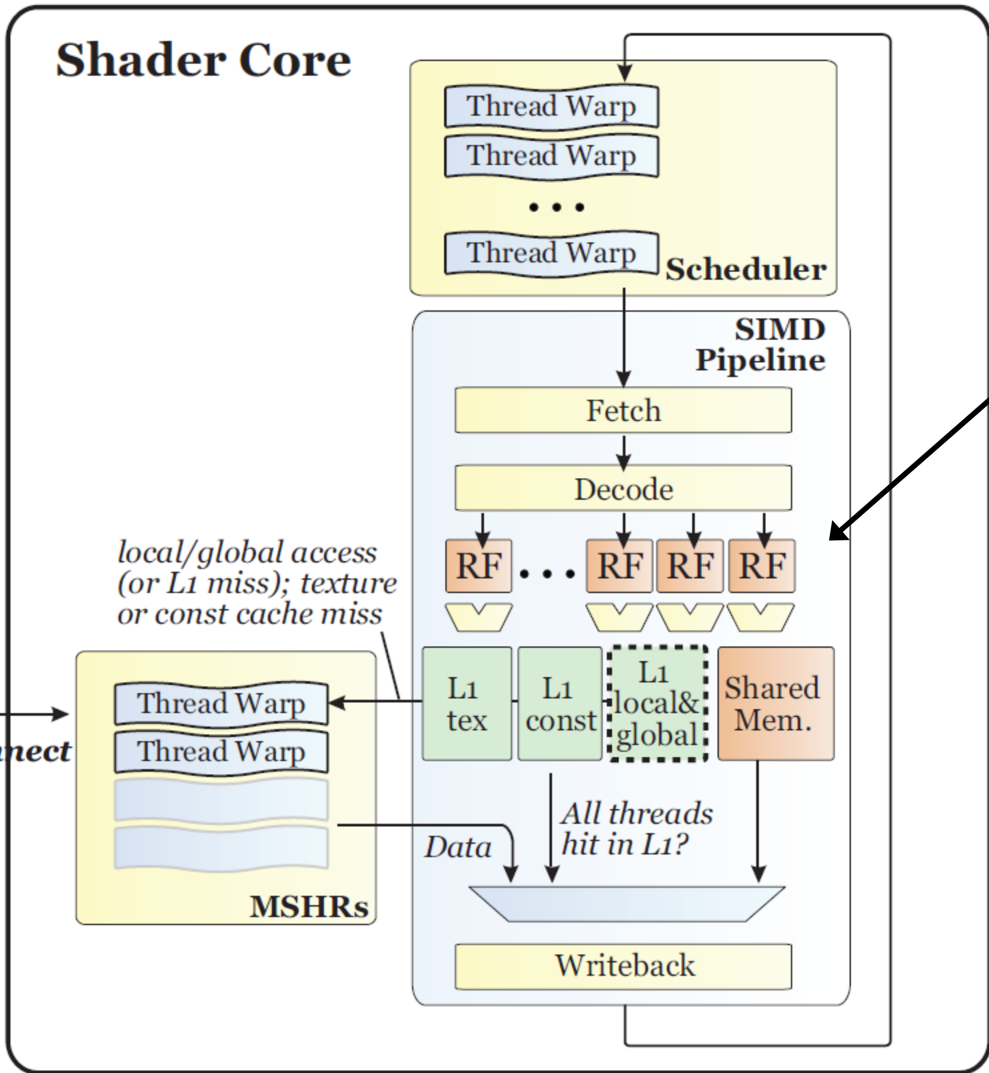
- Keep all processing units busy!
  - Enough threads
- All Multiprocessors (MPs)
- All Scalar Processors (SPs)
- Full pipeline of scalar processor
  - Pipeline of up to 24 stages

3. GPU architectuur

GPU Architecture



1 Streaming Multiprocessor



The Same Instruction is executed on Multiple Thread (SIMT)

width of pipeline:
8 – 32 – 192 - 128

VRIJE UNIVERSITEIT BRUSSEL

Execution Model

- Kernel = smallest unit of execution, like a C function, executed by each **work item** (≈ kernel thread)
- Data parallelism: kernel is run by a grid of work groups
- **Work group** consist of instances of same kernel: work items
- Different data elements are fed into the work items of the work groups

  ➡We talk about *stream computing*

Kernel execution

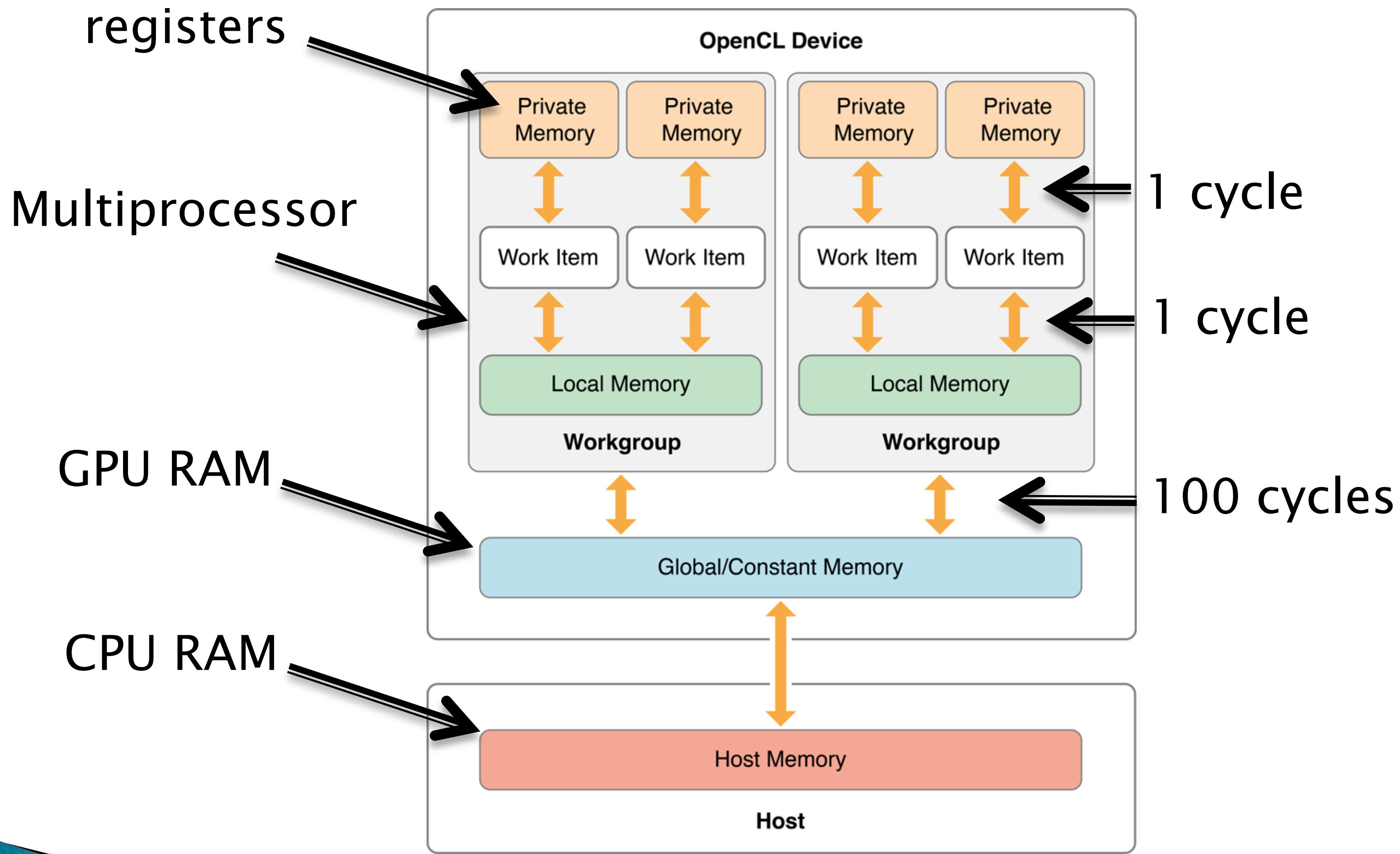
- Simple scheduler
  - Assigns work groups to available streaming MultiProcessors (MPs)
  - Basically, a waiting queue for work groups
- Work groups (WGs) execute independently
  - Global Synchronization among work groups is not possible!

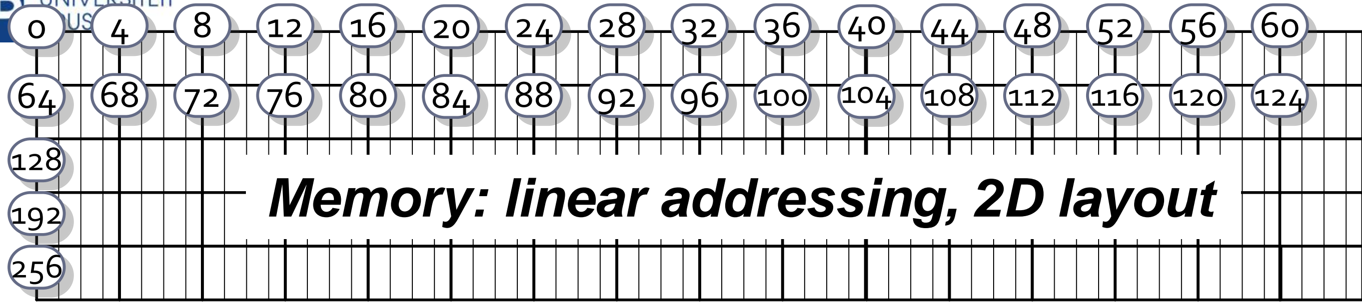
GPU with 2 MPs

time

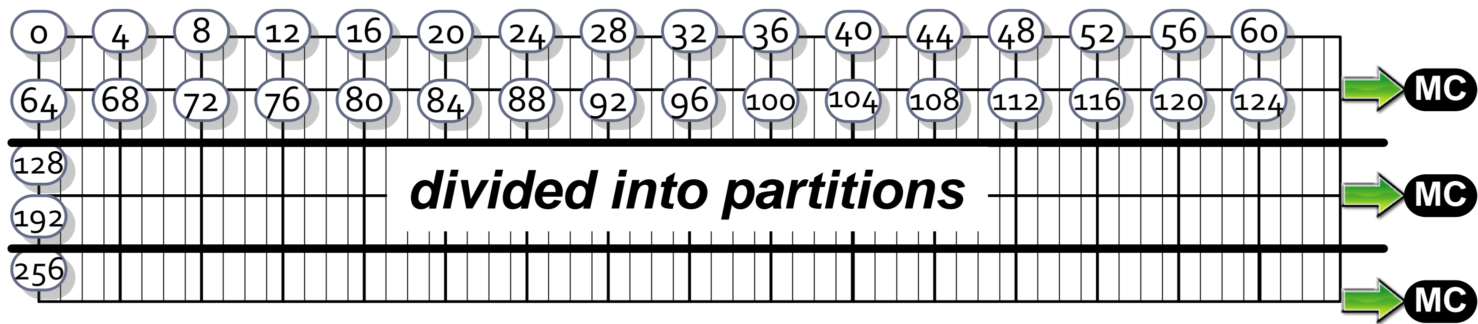
GPU with 4 MPs

Architecture – Memory Model

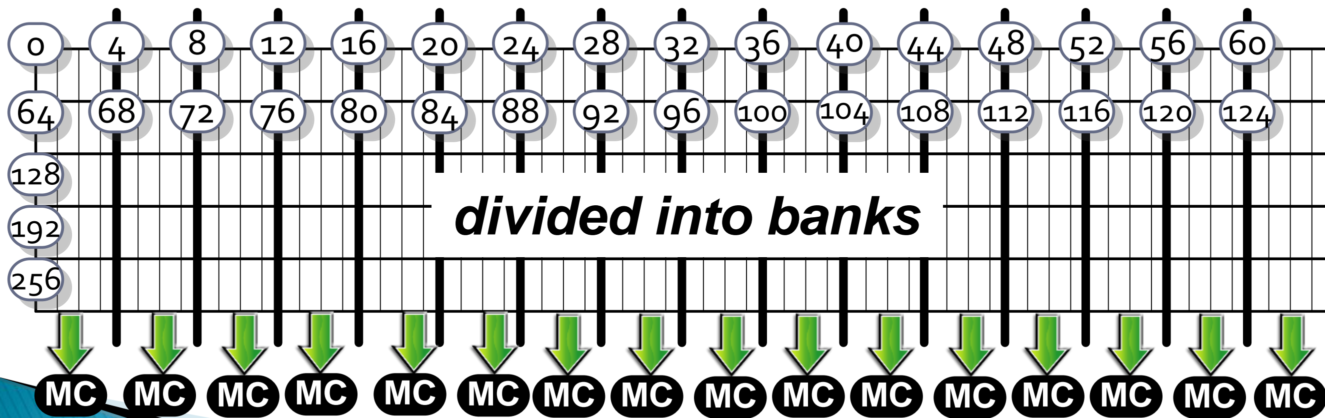




...



...



Memory Controllers: Can handle 1 request at a time

How many memory transactions can be handled simultaneously?

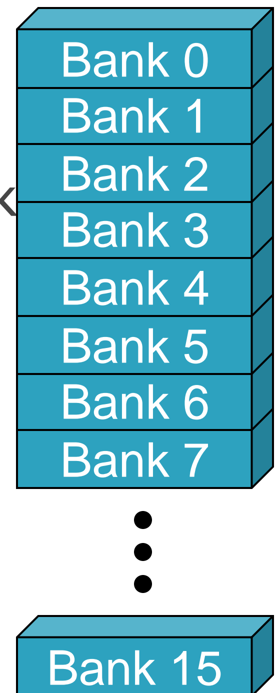
VRIJE UNIVERSITEIT BRUSSEL

Global memory

- Divided into partitions
  - NVIDIA GPUs typically have 8 partitions
- Memory controller can serve 1 segment (≈ cache line of 4x32 Bytes)
- Memory coalescing for warps
  - Accessed elements of a warp should belong to same aligned segment
  - if not (uncoalesced access), memory requests are serialized => wille take more time
- Active warps of different cores/multiprocessors simultaneously access global memory
  - **Partition camping** when they access the same partition => serialization of memory requests

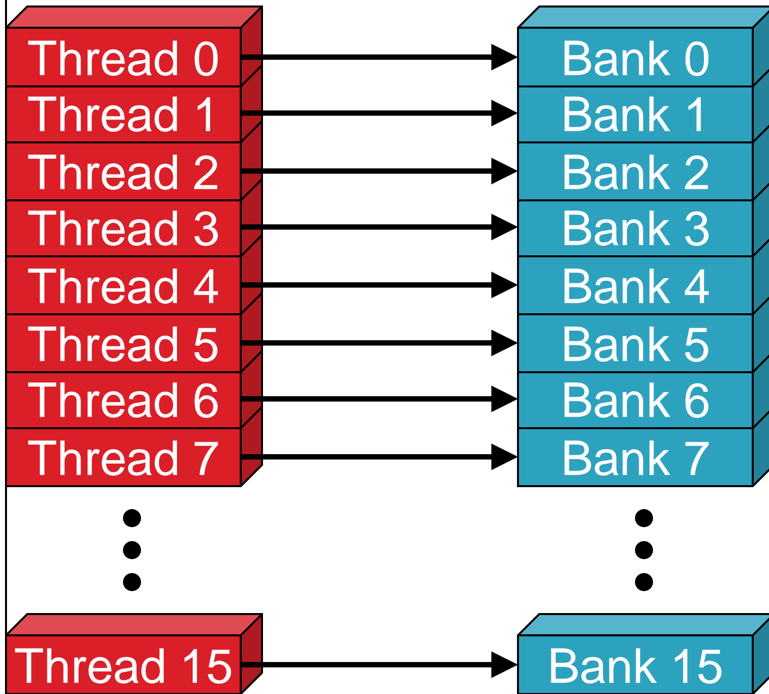
Local/Shared memory

- Local/Shared memory is divided into banks
- Each bank can service one address per cycle
- Multiple simultaneous accesses to a bank result in a bank conflict
  - Conflicting accesses are serialized
  - Cost = max # simultaneous accesses to a single bank
- No bank conflict:
  - all threads of a half-warp access different banks,
  - all threads of a half-warp access identical address, (broadcast)

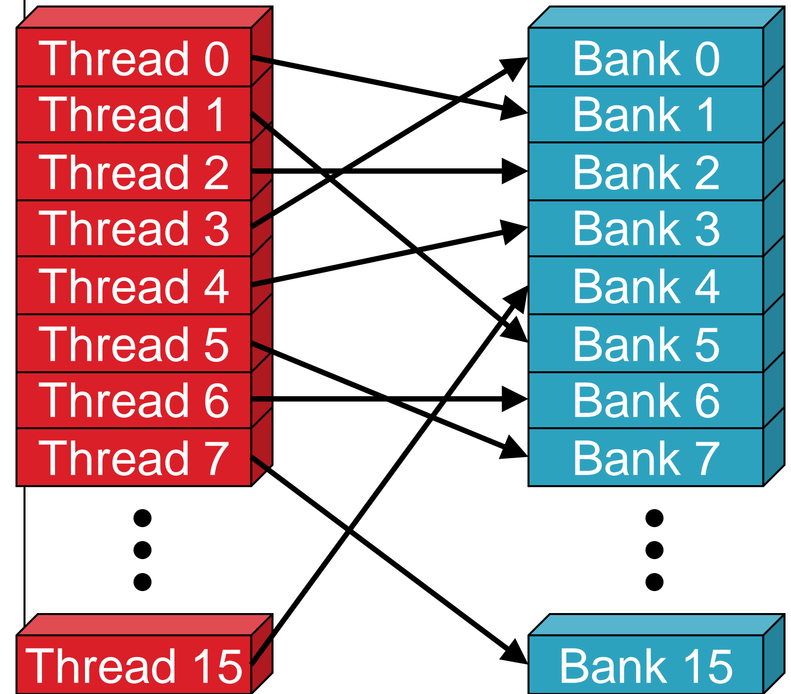


Bank Addressing Examples

- No Bank Conflicts
  - Linear addressing stride of 1



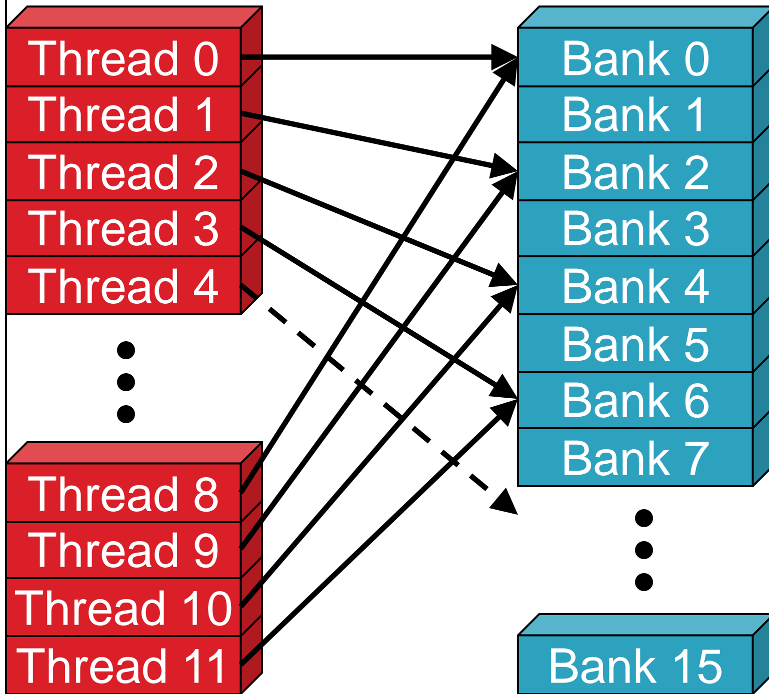
- No Bank Conflicts
  - Random 1:1 Permutation



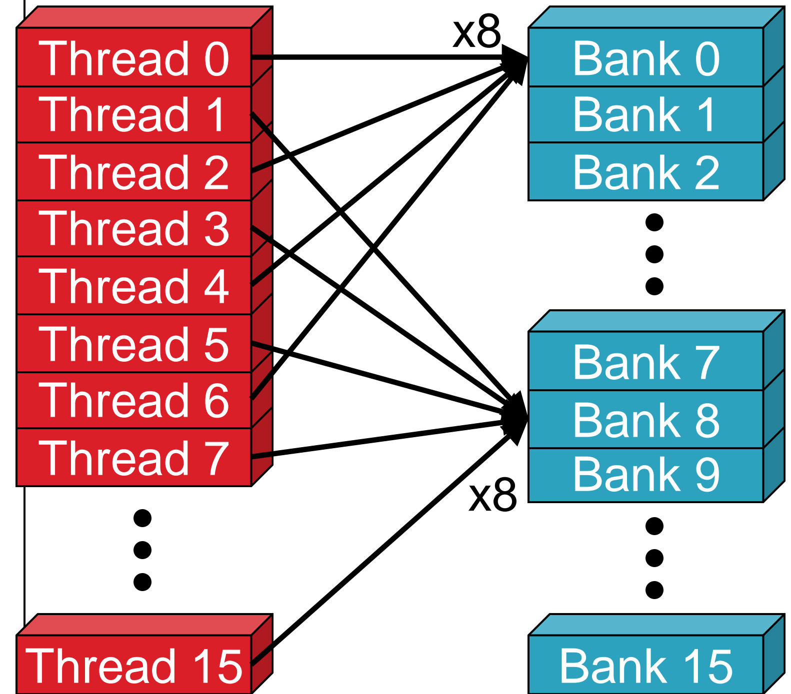
VRIJE UNIVERSITEIT BRUSSEL

Bank Addressing Examples

- 2-way Bank Conflicts
  - Linear addressing stride of 2



- 8-way Bank Conflicts
  - Linear addressing stride of 8



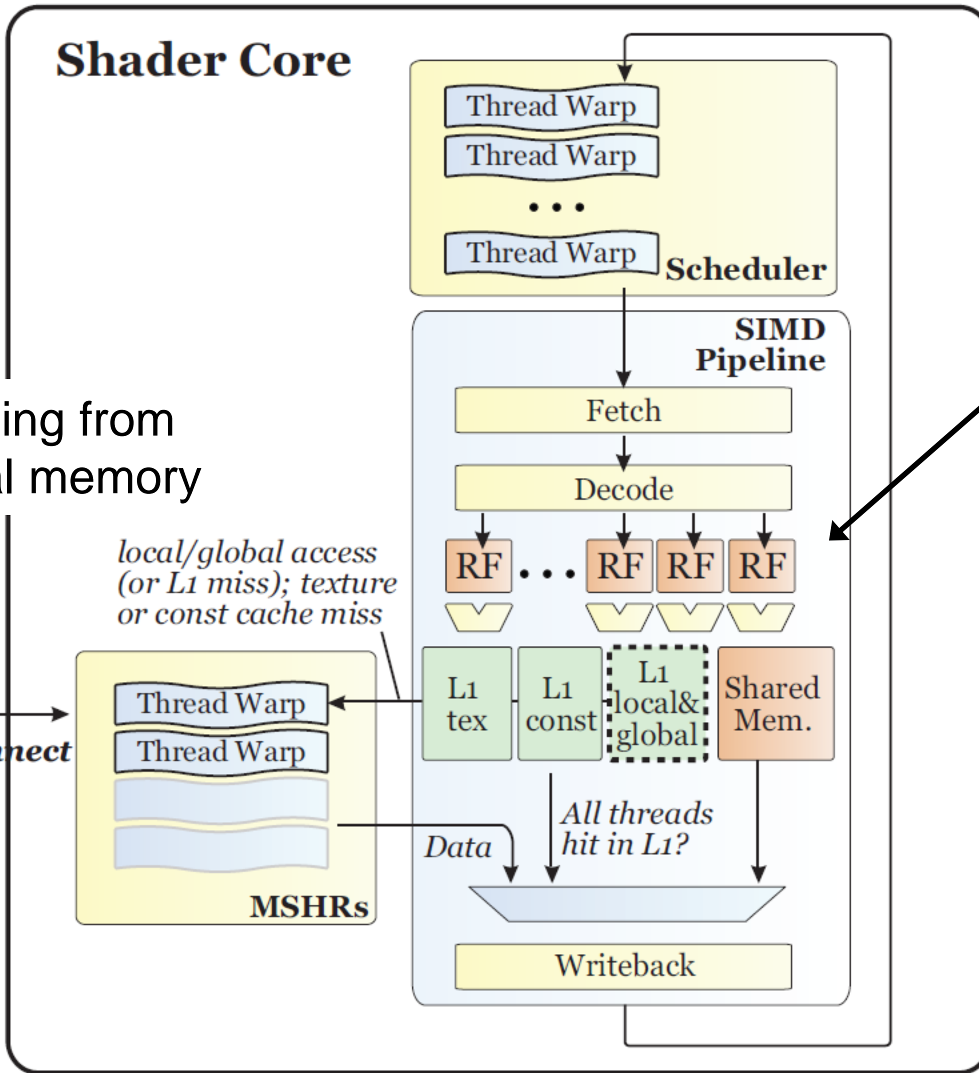
VRIJE UNIVERSITEIT BRUSSEL

Worst case

- Threads of the same warp accessing the same column of a matrix having a width of a multiple of 16

4. Hardware Threads & SIMT

1 Compute Unit (core)

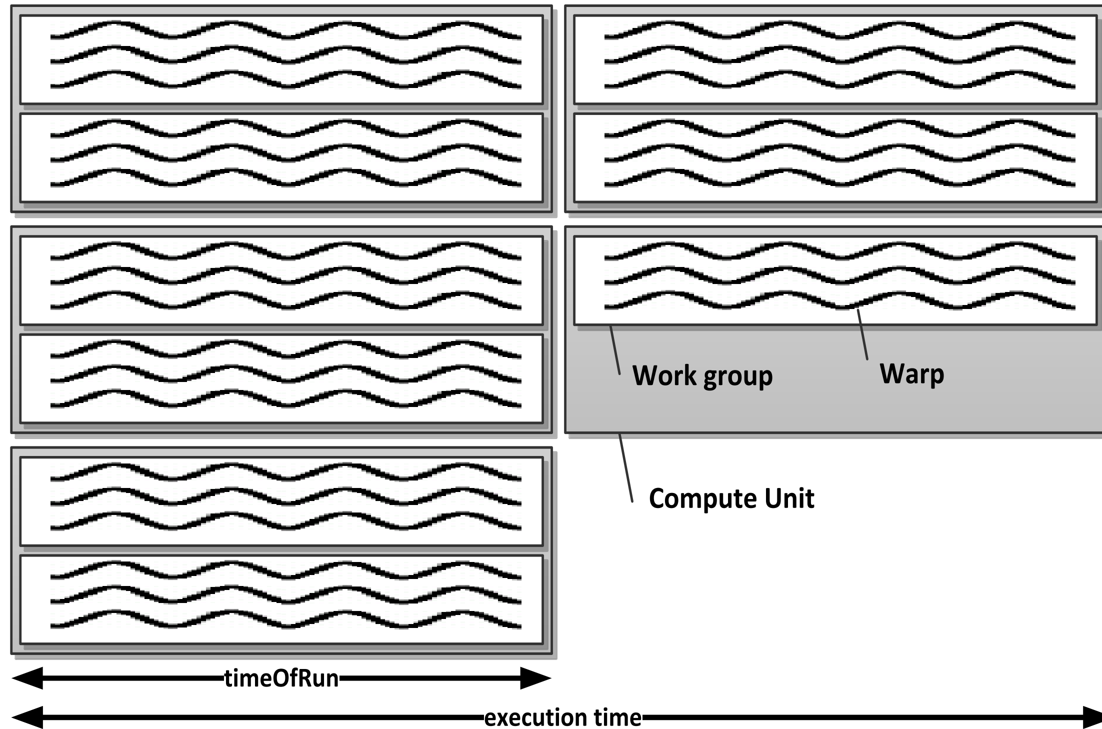


Reading from global memory

The Same Instruction is executed by Multiple Threads (SIMT)

width of pipeline depends on Nvidia architecture: 8 – 32 – 192 - 128

The execution on a GPU



- Work groups are scheduled on compute units (cores).
- Warps of active work groups are scheduled on the core

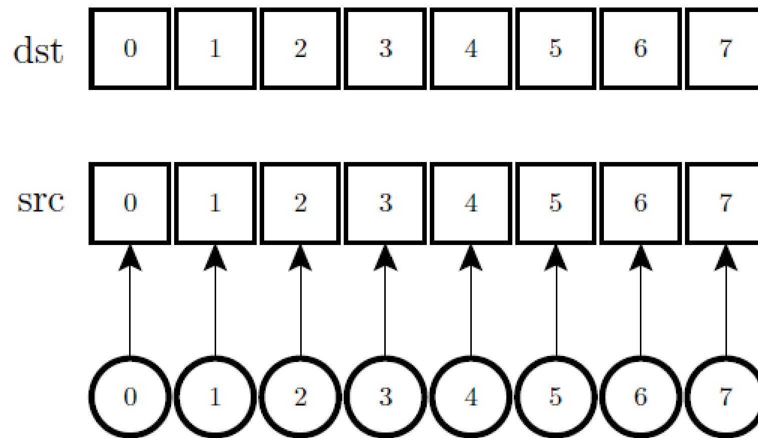
VUB VRIJE UNIVERSITEIT BRUSSEL

Warp executes work items in lock step

- **Hardware thread (called warp by Nvidia):**
  - Work items are executed together in groups, the instructions of the kernel are executed at the same time they will execute the same instruction
  - Nvidia: 32; AMD: 64; Intel: variable number (8/16/24/32)
- **Consequences:**
  1. Running 1 work item or 32 work items takes the same amount of time
     - Create work groups which are multiples of 32 or 64 (AMD)
  2. Branching: if work items of the same warp take different branches, all branches will be executed after each other
     - Performance loss
  3. Concurrent memory access: if work items access memory, all work items of the same warp do it simultaneously
     - Not all memory access can be done with the same speed

When is SIMT = vector processing?

- Contiguous data access (See lesson 2)



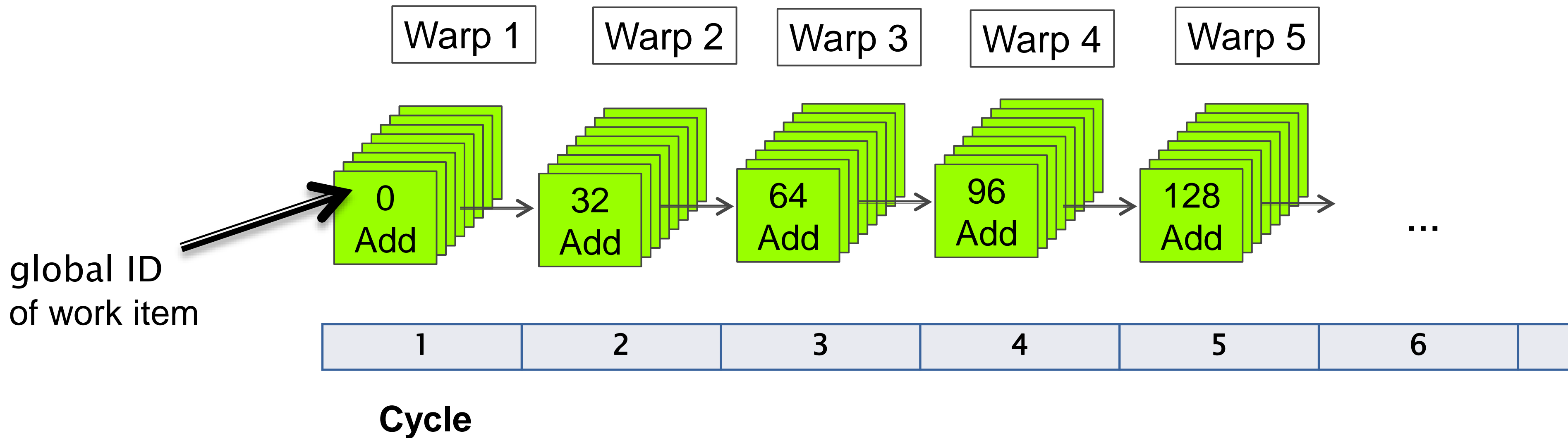
- Warp execution of instructions on the data is similar to vector instructions operating on vector registers.

Vectors versus SIMT

- Vectors
  - Data should be stored in vector register
  - Instructions are performed onto these registers
  - Harder to program
- SIMT
  - Each thread of a warp can choose on which data it works
  - Easier to program: programmer does not have to worry about *work item-data mapping*

Warp execution

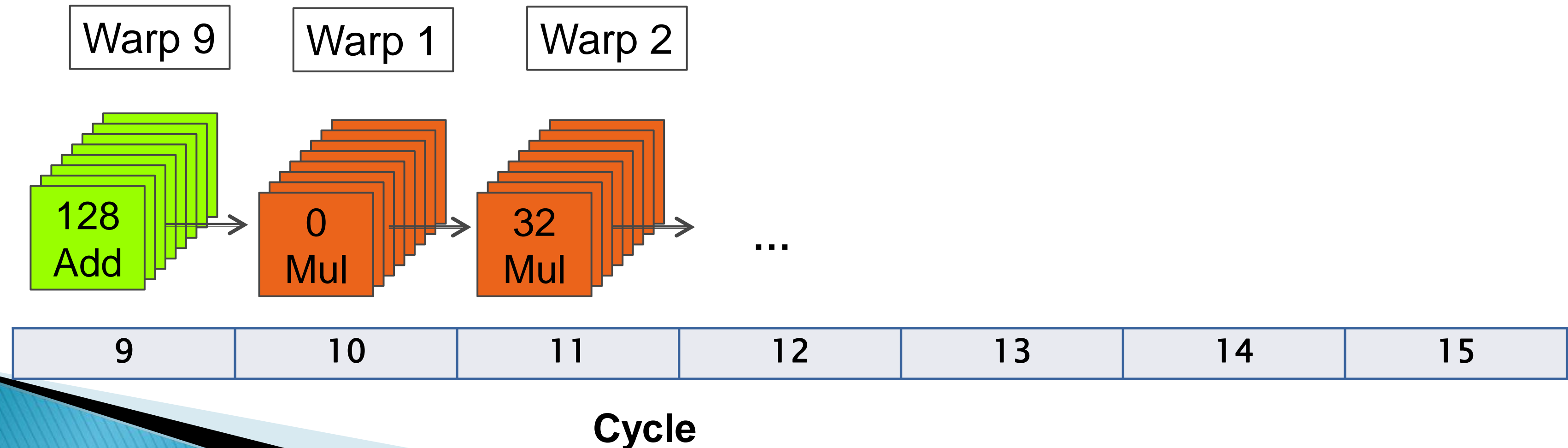
- Work items are sent into pipeline grouped in a warp
  - ALUs all execute the same instruction in `lockstep': Single Instruction, Multiple Threads (SIMT)
  - Every cycle a new warp can issue an instruction



Warp execution

- On an Nvidia Kepler architecture, a single precision floating point instruction (add or multiplication) takes 9 cycles, which is the length of the pipeline.
  - 8 other warps can be scheduled in the mean the mean time
  - After 9 cycles, the second instruction of the first warp (multiplication) can be issued, next the second warp and so on

  ⇒ With 9 warps the pipeline is completely filled, no stalling/idling, the completion latency of 9 cycles is hidden.

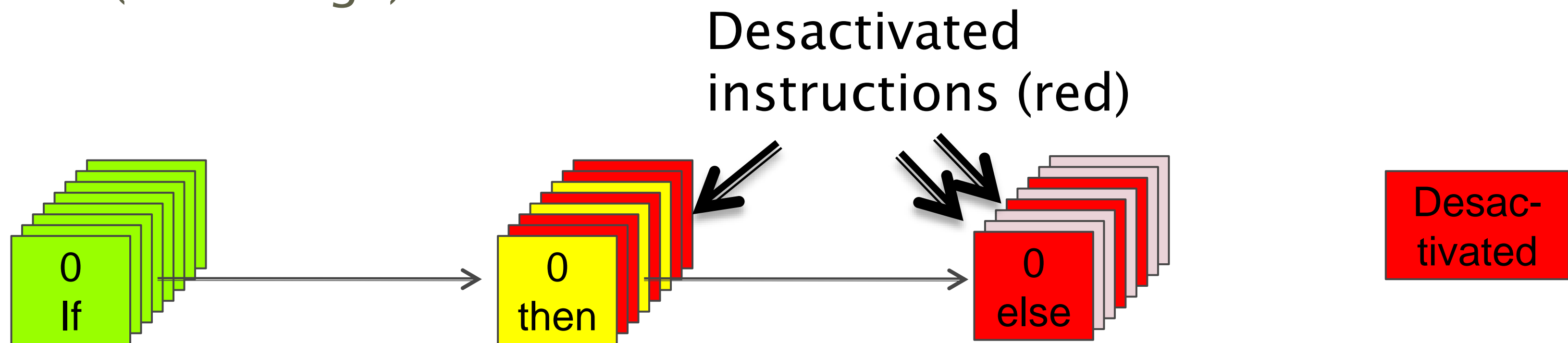


SIMT Conditional Processing

- If work items of a warp follow different branches, the instructions of both branches have to be executed, but are desactivated for some threads.

  => Performance loss!

- **Example**: assume 8 threads, one instruction in if-clause, one in then-clause
  - 3 cyles in which 24 instructions are executed, 8 lost cycles (66% usage)



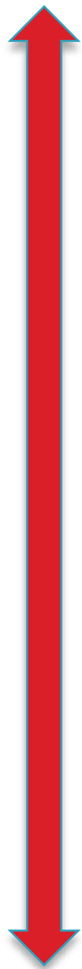
ARM’s conditional instructions

- The condition is tested against the current processor flags and if not met the instruction is treated as a no-op.
- removes the need to branch, avoiding pipeline stalls and increasing speed. It also increases code density.
- By using suffix EQ, NE, GT, ...

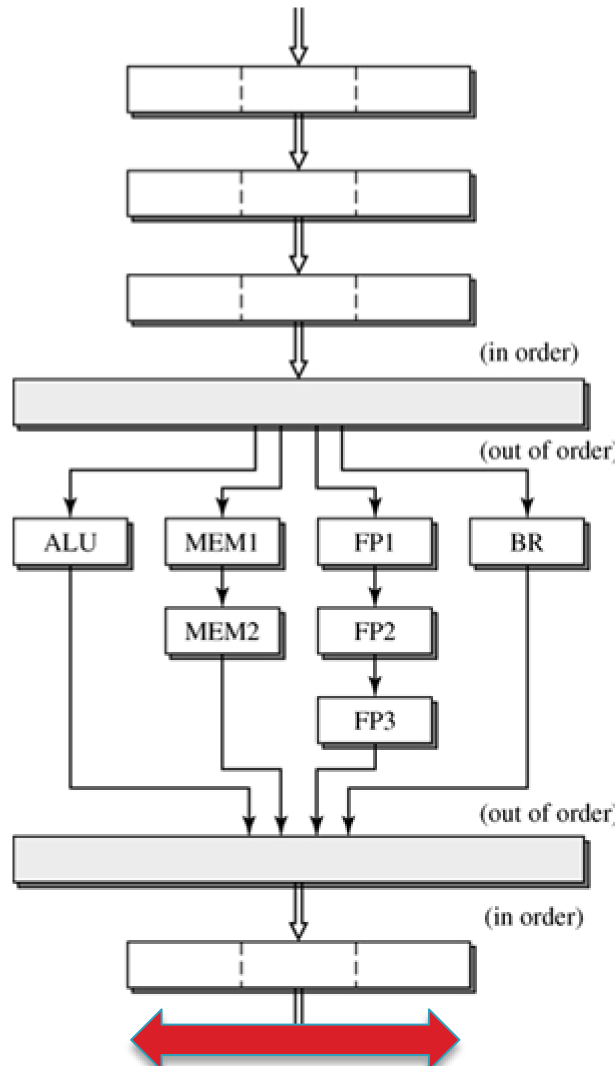
```
CMP r0, #5          ; if (a == 5)
MOVEQ r0, #10       ; only executed if equality
BLEQ fn             ; fn(10)
```

5. Conclusions

‘Sequential’ processor: super-scalar out-of-order pipeline



Pipeline depth



Pipeline width

Different processing units

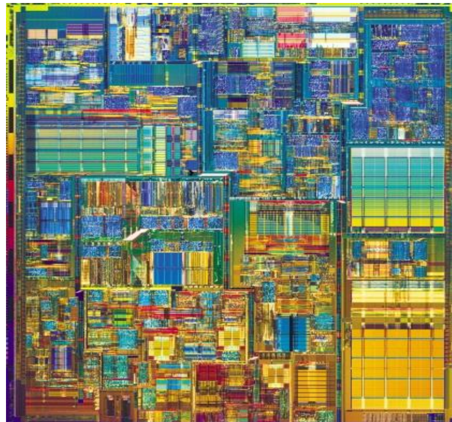
Out-of-order execution
Branch prediction
Register renaming
…

CPU computing

Algorithm

Implementation

Compiler



manual

automatic

Write once
Run everywhere
efficiently!

Automatic optimization

Low latency of each instruction!

VRIJE UNIVERSITEIT BRUSSEL

GPU architecture strategy

- Light-weight threads, supported by the hardware
  - Thread processors, upto 96 threads per processing element
  - Switching between threads can happen in 1 cycle!
- No caching mechanism, branch prediction, …
  - GPU does not try to be efficient for every program, does not spend transistors on optimization
  - Simple straight-forward sequential programming should be abandoned…
- Less higher-level memory:
  - GPU: 16KB shared memory per SIMD multiprocessor
  - CPU: L2 cache contains several MB's
- Massively floating-point computation power
- RISC ISA instead of CISC
- Transparent system organization

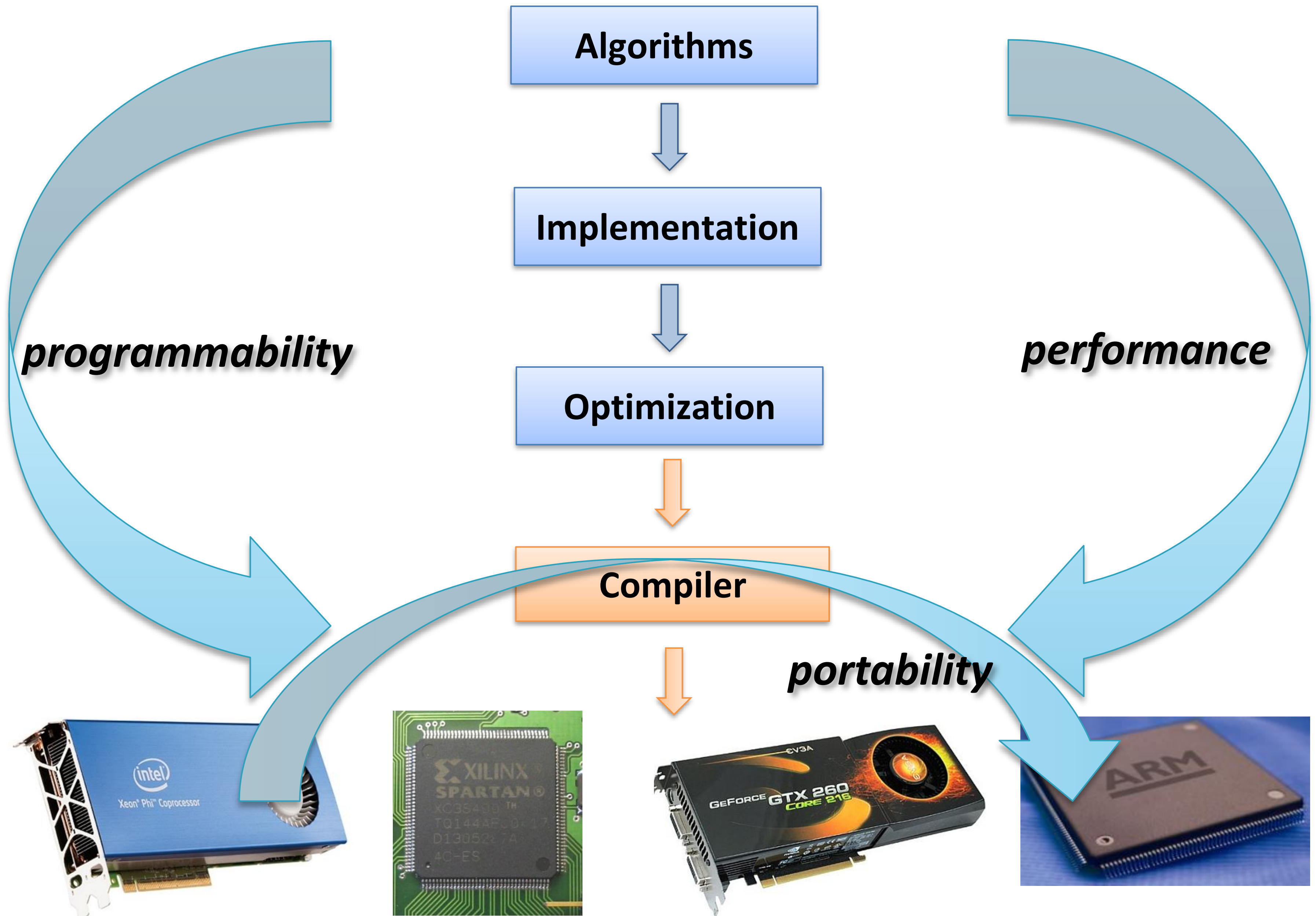
↔ Modern (sequential) CPUs based on simple Von Neumann architecture

VRIJE UNIVERSITEIT BRUSSEL

GPU processor pipeline

- 6-24 stages
- in-order execution!!
- no branch prediction!!
- no forwarding!!
- no register renaming!!
- Memory system:
  - relatively small
  - Until recently no caching
  - On the other hand: much more registers (see later)
- No program call stack
  - All functions inlined
  - No recursion possible

Challenges of GPU computing



VRIJE UNIVERSITEIT BRUSSEL

GPU processing power is not for free

Obstacle 1

Hard(er) to implement

Obstacle 2

Hard(er) to get efficiency

*Discussed in detail in my course **GPU Computing** (2nd semester)*