VRIJE UNIVERSITEIT BRUSSEL

ULB UNIVERSITÉ LIBRE DE BRUXELLES

GPU Computing

» Lesson 4: The Pipeline Model and Occupancy

Gauthier Lafruit & Jan Lemeire

2022-2023

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

Intro

VRIJE UNIVERSITEIT BRUSSEL

Goal

- Predict performance
- Understand performance
  - Identify bottlenecks
  - Estimate the effect of changes
- Pedagogical tool

VRIJE UNIVERSITEIT BRUSSEL

The Philosophy of the model

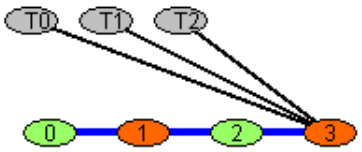
- The GPU is modeled with **pipelines**: one for each susbsystem
  - Basically: the computational and the memory subsystem
- To understand several aspects influencing the performance, one should understand the behavior of *pipelined processors*
- Our performance analysis is based on the **simulation** of a dual pipeline model
  - *It does not intend to reflect a hardware-accurate model nor a cycle-accurate simulation*

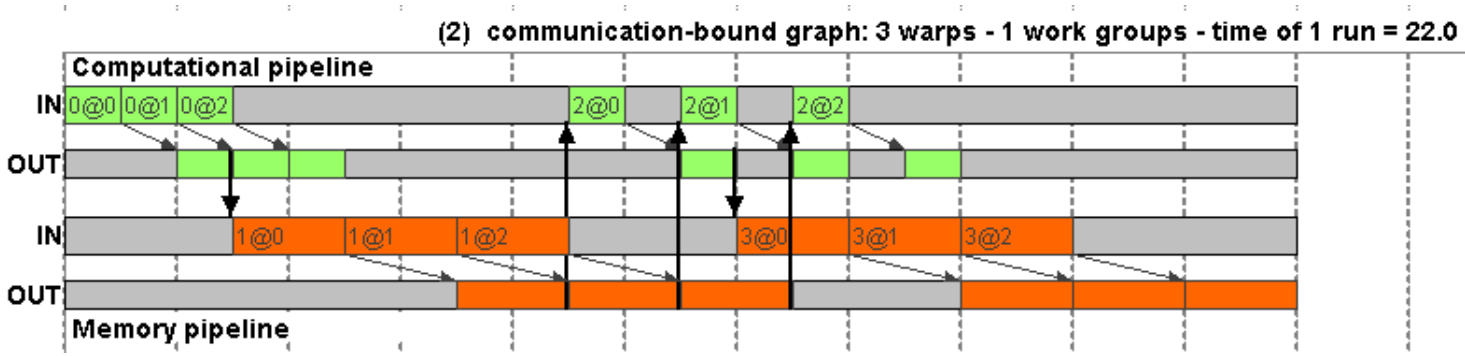
http://parallel.vub.ac.be/pipeline/

VRIJE UNIVERSITEIT BRUSSEL

Pipeline simulator

- Semi-abstract model of a GPU multiprocessor: 1 pipeline for computional subsystem and 1 pipeline for memory subsystem
- Based on instruction dependency graph of program and the latencies of the hardware





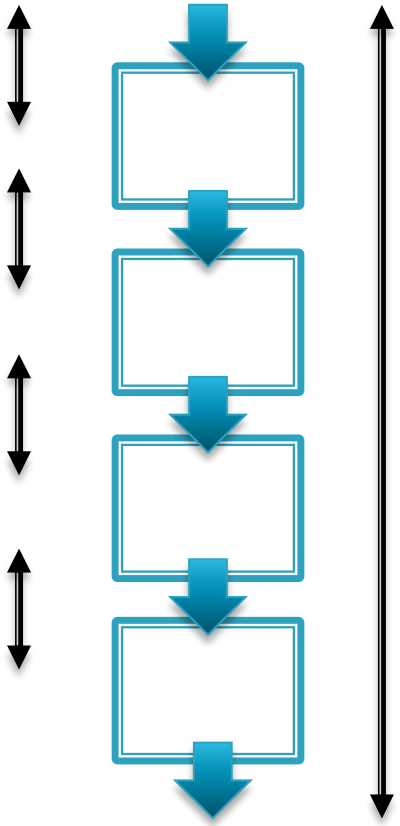
Pipeline Analogy

VRIJE UNIVERSITEIT BRUSSEL

A processing pipeline

Issue latency (λ)
= inverse of **throughput**

Time for 1 stage
Is called a **cycle**
Is equal for all stages

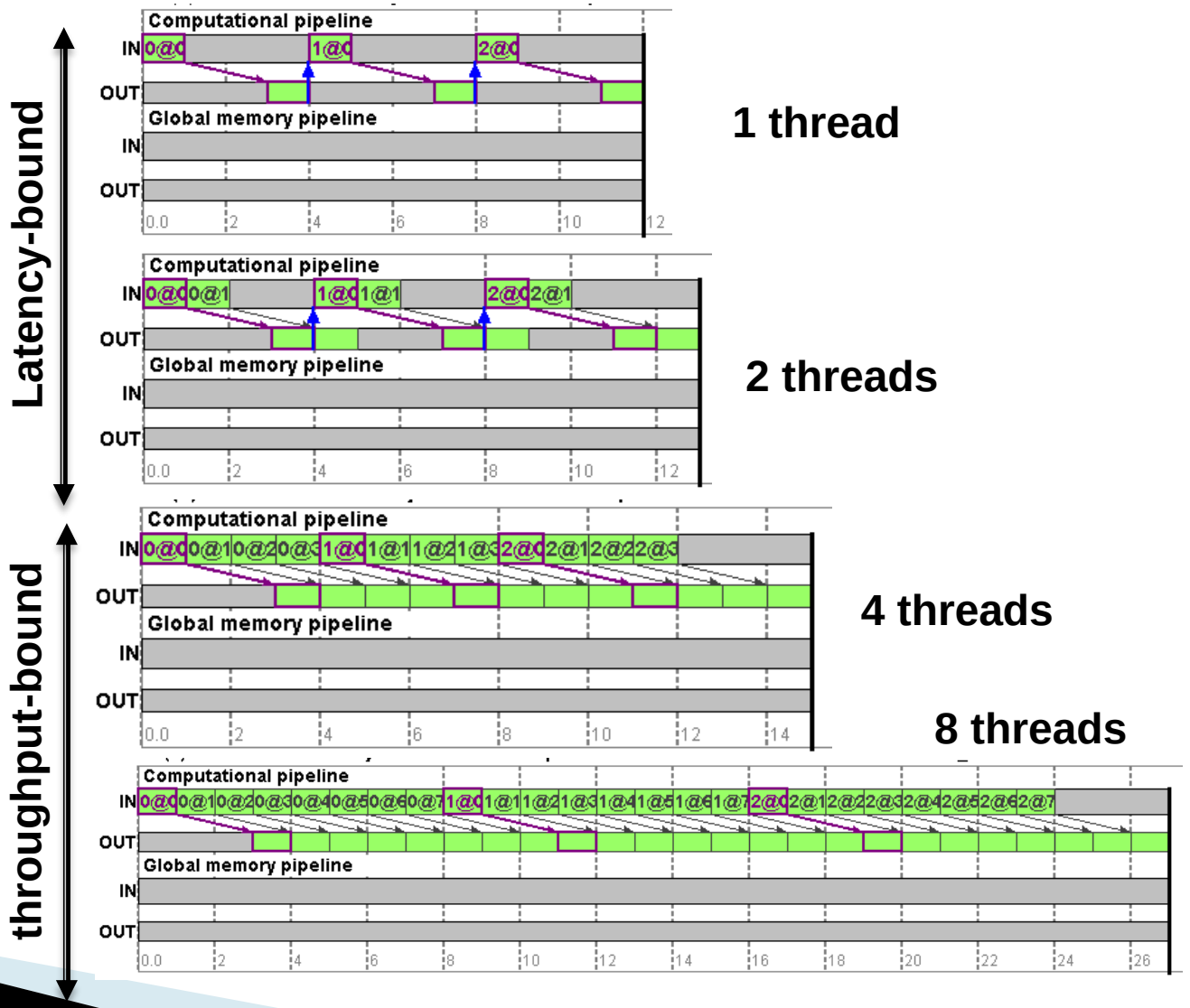


Completion latency (Λ)
= end-to-end latency

Time to complete all stages
Here: Λ = 4 * λ

Single pipeline

- See text of Lesson 3
- Kernel has 3 dependent computations



Full latency hiding due to simultaneous multithreading

VRIJE UNIVERSITEIT BRUSSEL

Single Pipeline

- One warp and only dependent instructions

Software '3 computations (all dependent)' & GPU 'Latencies_1_and_4'

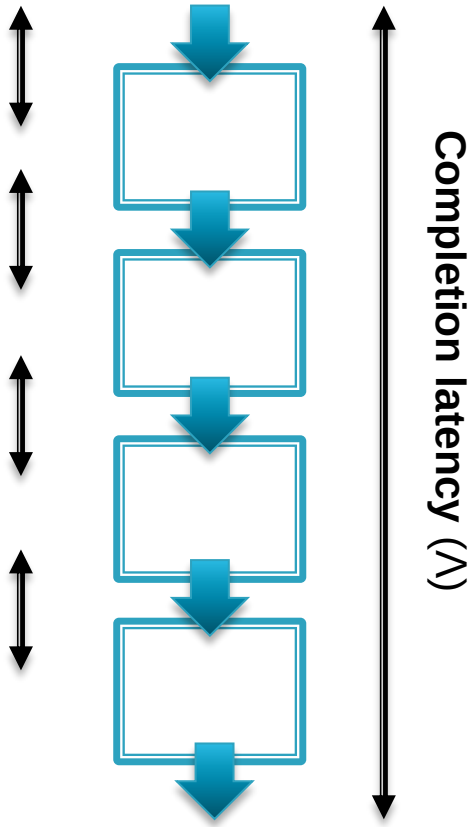
  - Completion latency (Λ) determines performance
    = length of the pipeline
- Increase occupancy:

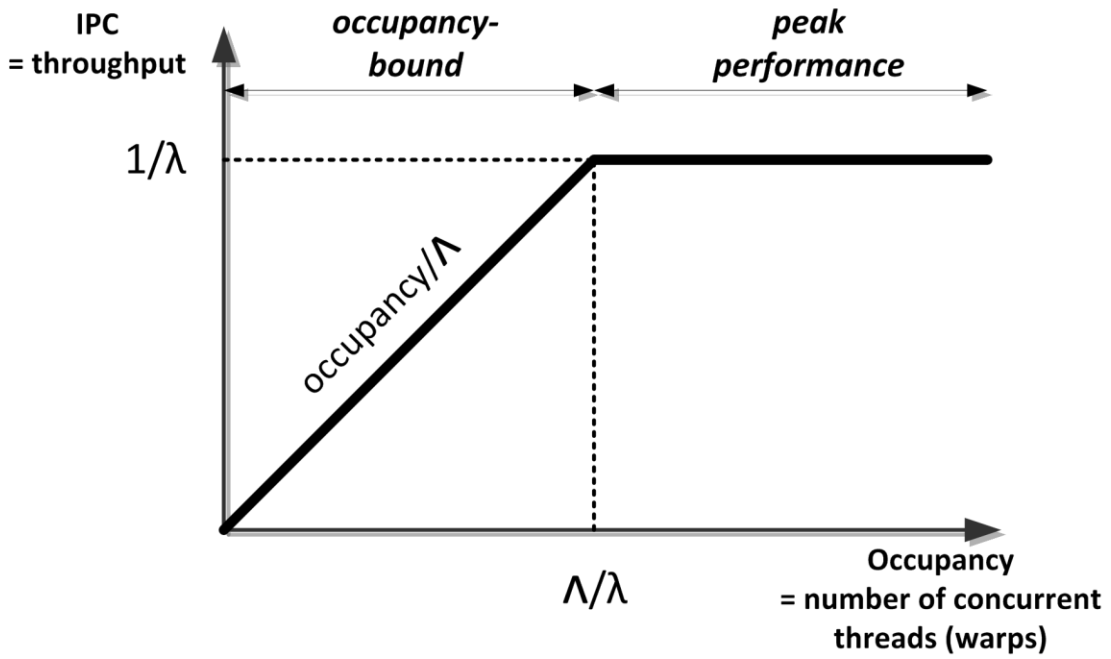
Increase #Threads (warps)

  - latency hiding
  - Issue latency (λ) determines performance
    = 1 cycle for simple pipeline
    Determines the peak performance

Occupancy roofline

Issue latency (λ)



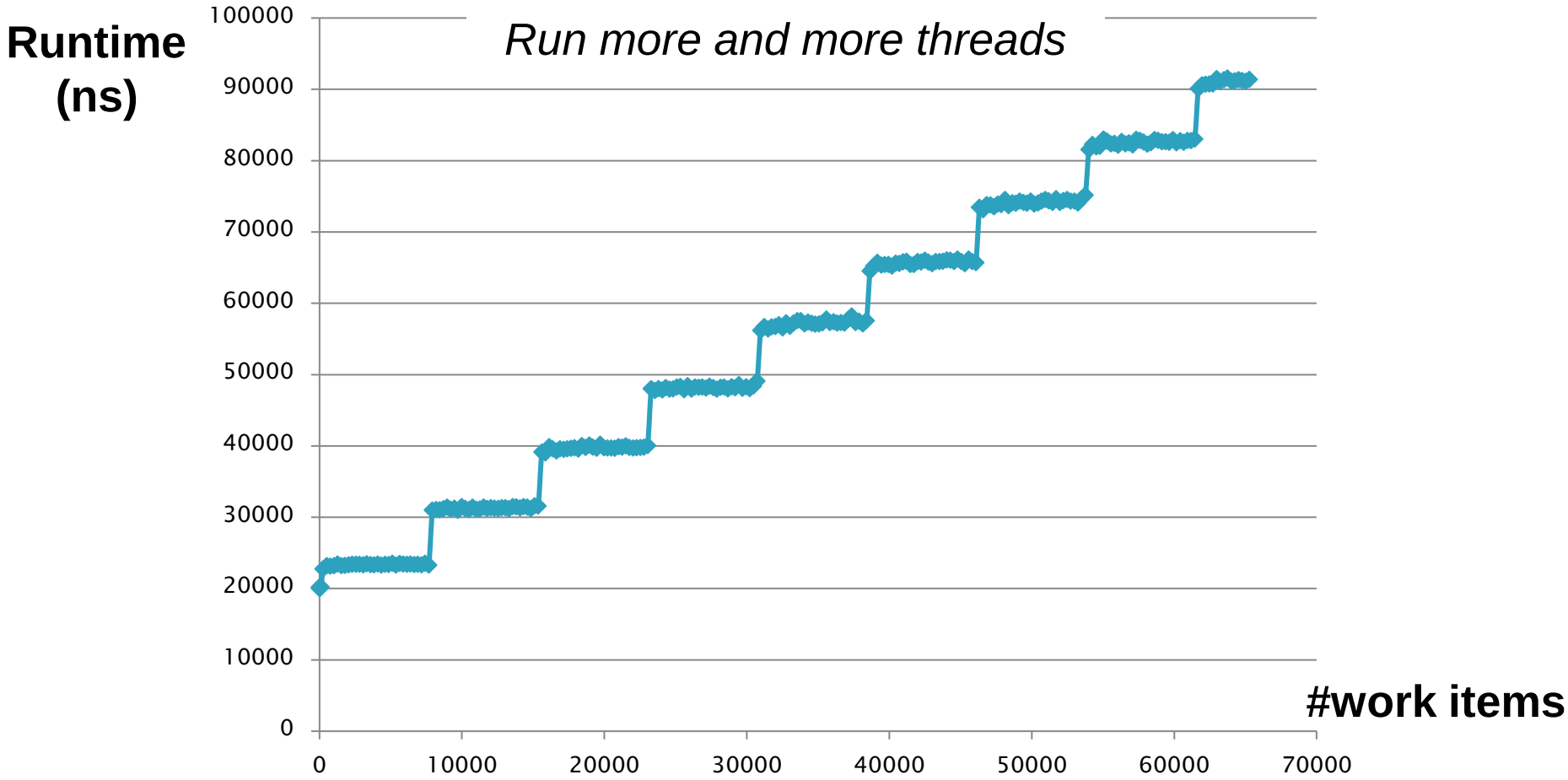


VRIJE UNIVERSITEIT BRUSSEL

Hardware Parameters: Latencies

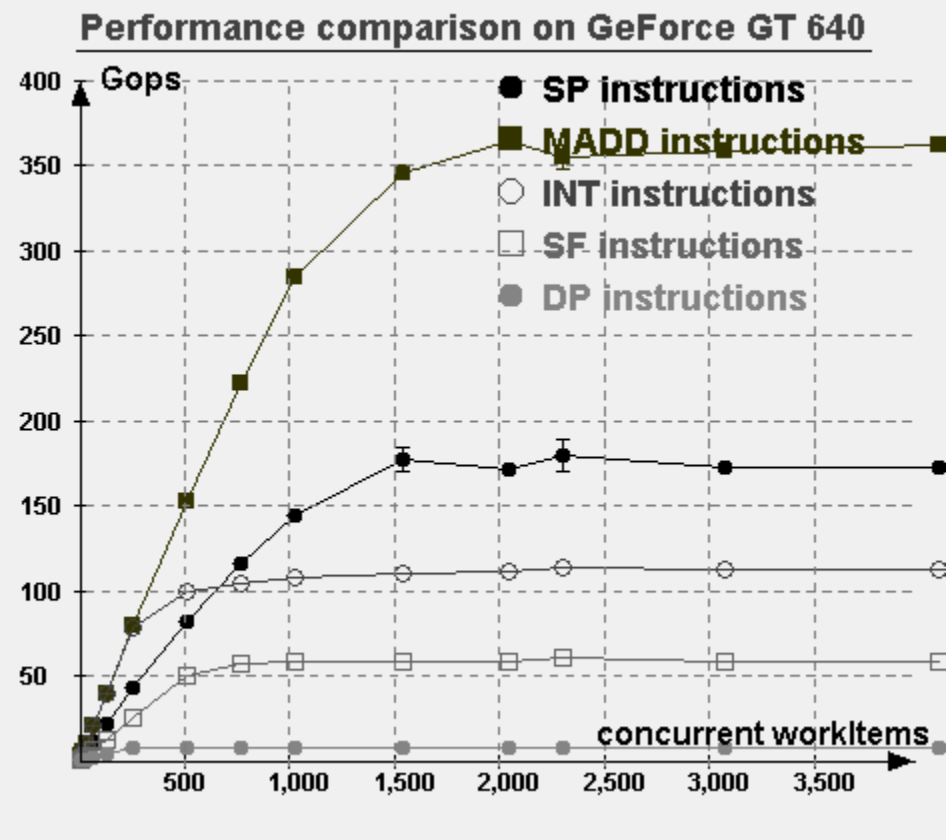
- **Issue latency** (λ): the number of cycles required between issuing two consecutive independent instructions
  - inversely proportional to the peak performance
- **Completion latency** (Λ): the number of cycles until the result of an instruction is available for use by a subsequent instruction
- Both may depend on *context*: instructions may be executed inefficiently, resulting in longer latencies

Running A simple ADDITION kernel



VRIJE UNIVERSITEIT BRUSSEL

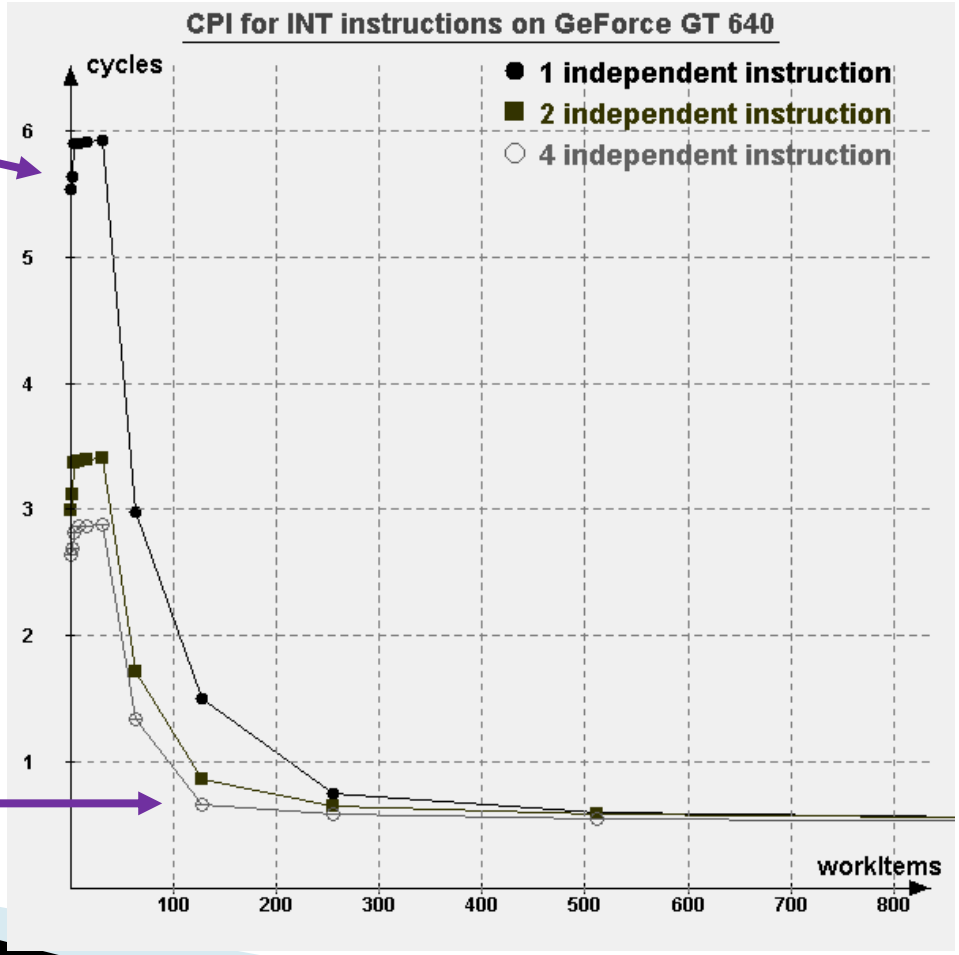
Peak performance



Cycles Per Instruction

- The reversed graph

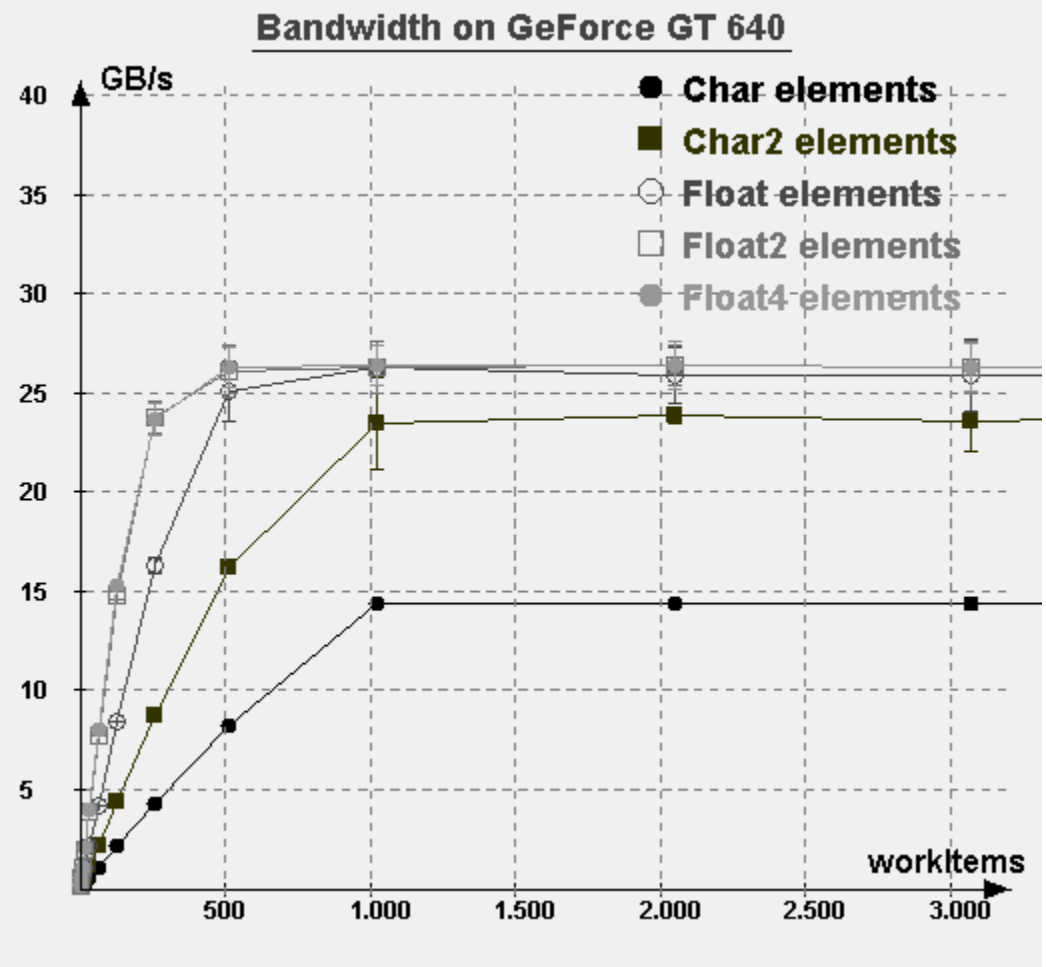
Completion latency



Issue latency

VRIJE UNIVERSITEIT BRUSSEL

Bandwidth



VRIJE UNIVERSITEIT BRUSSEL

Instruction-Level Parallelism (ILP)

- Consider a kernel which has 2 series of 3 independent instructions

Model '3 computations (two independent instructions)'

- Less concurrent warps needed for full latency hiding

VRIJE UNIVERSITEIT BRUSSEL

Dual Pipeline

- Computation and memory
- Memory access is modeled as a single pipeline

Model '3 computations and memory (all dependent)'

- $\Lambda_{mem} >> \Lambda_{comp}$ and $\lambda_{mem} >> \lambda_{comp}$

  ➔ More concurrency needed for peak performance

- Communication vs memory bound

Models 'balanced graph', 'communication-bound graph' and 'computation-bound graph'

- The cost of barrier synchronization

Compare models with and without barrier

VRIJE UNIVERSITEIT BRUSSEL

1 warp applies SIMT

- Since the *scheduling unit* is a warp (hardware thread) consisting of 32 kernel threads, the simulator is based on warps and not on individual work items.
- We are interested in the issue and completion latency of instructions of warps
  - It will generate 32 executions of the same instruction for all the 32 work items of the warp
  - In the microbenchmarks we have to divide the CPI by the warp size (32 for Nvidia)

VRIJE UNIVERSITEIT BRUSSEL

Real GPU is not a simple pipeline

- NVIDIA generations (Processing Elements per Compute Unit)
  - Tesla: 8 PEs/CU ➔ 1 warp instruction every 4 clock cycles
  - Fermi: 32 PEs/CU ➔ 1 warp instruction every clock cycle
  - Kepler: 192 PEs/CU ➔ 6 warp instructions every clock cycle
  - Maxwell: 128 PEs/CU ➔ 4 warp instructions every clock cycle
- Pipeline model:
  - One computation pipeline λ_{comp} = f(generation)
    - λ_{comp}(Tesla) = 4 clock cycles
    - λ_{comp}(Fermi) = 1 clock cycles
    - λ_{comp}(Kepler) = 1/6 clock cycles
    - λ_{comp}(Maxwell) = 1/4 clock cycles
  - One memory pipeline
    - Latencies depend on type of memory request
    - Longer for non-ideal memory access

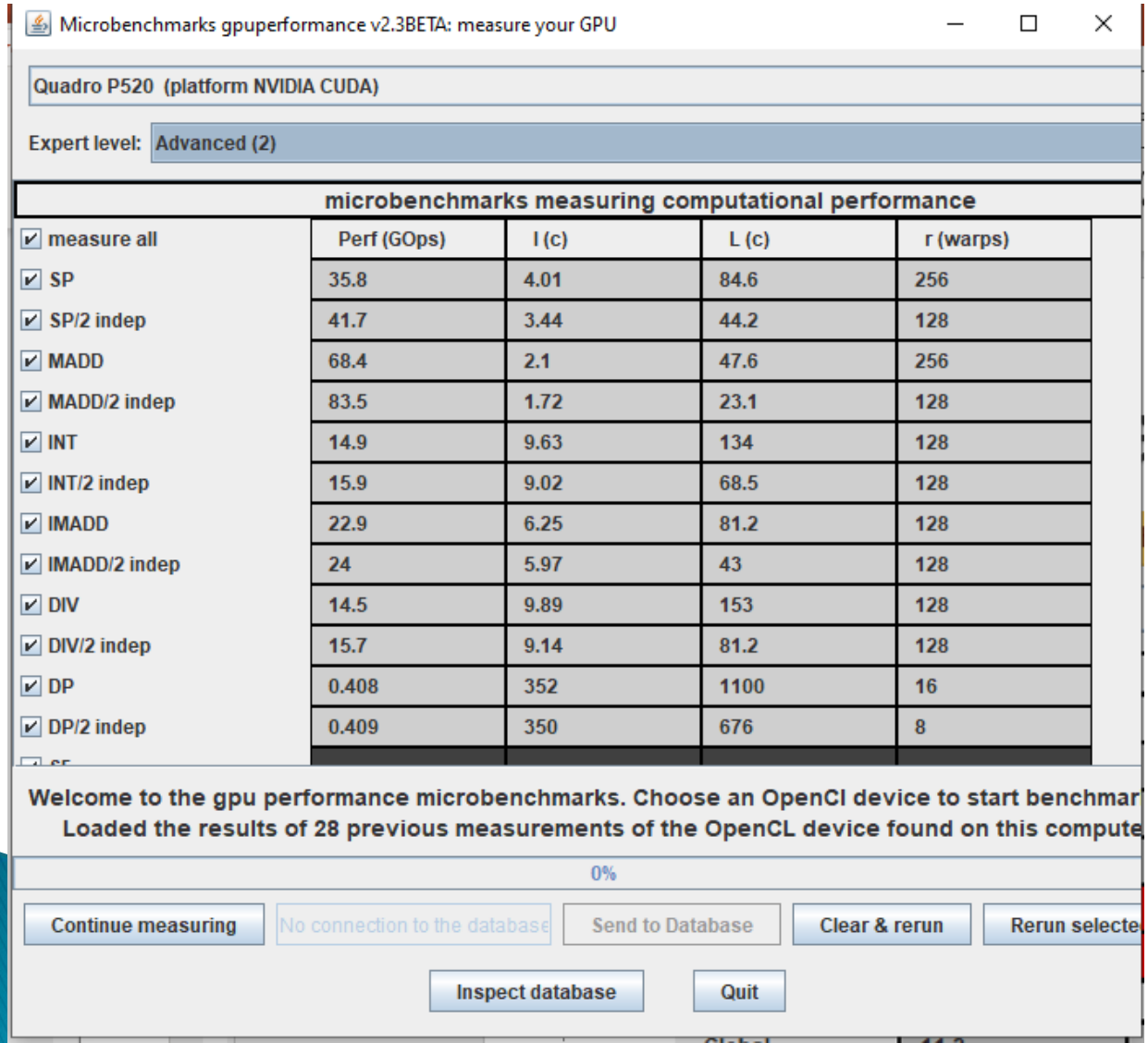
Measuring the parameters with microbenchmarks

www.gpuperformance.org

See paper Lemeire 2016:

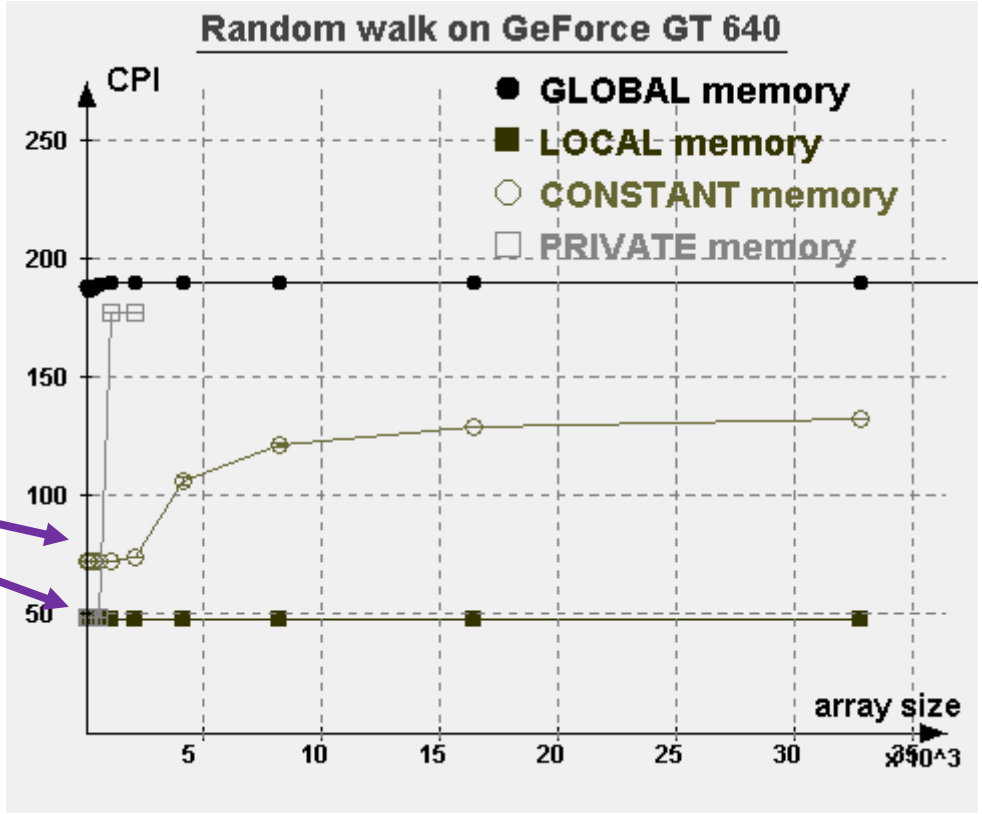
Jan Lemeire, Jan G. Cornelis, Laurent Segers, Microbenchmarks for GPU characteristics: the occupancy roofline and the pipeline model, Procs of 24th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Heraklion, Greece, 2016

Benchmark app



- In the advanced level, also de lambdas are measured.

Latencies for different memory types



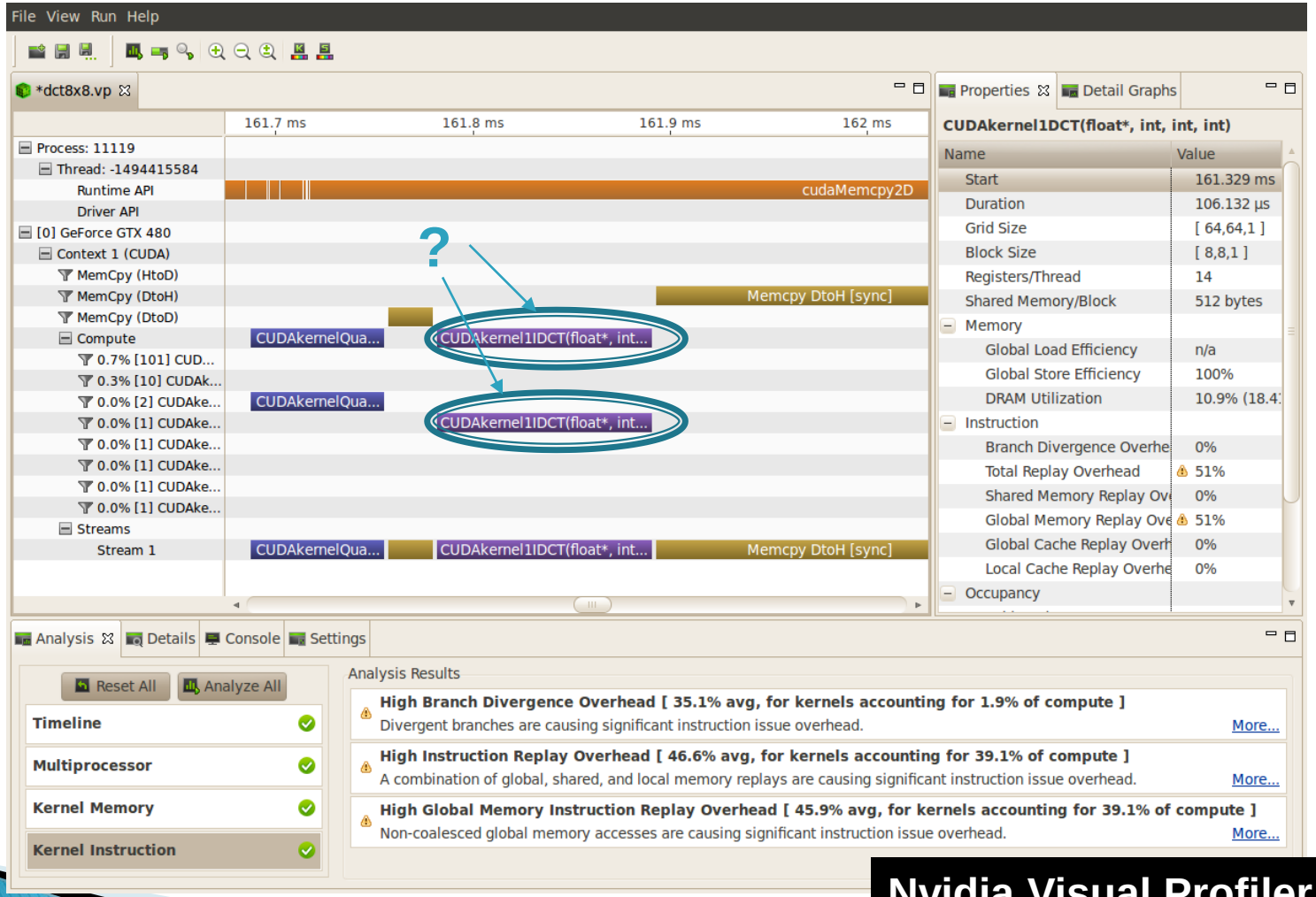
caching

Additional information about the Pipeline Model.
Will not be discussed in the course.
Just for those interested.

The Pipeline Model

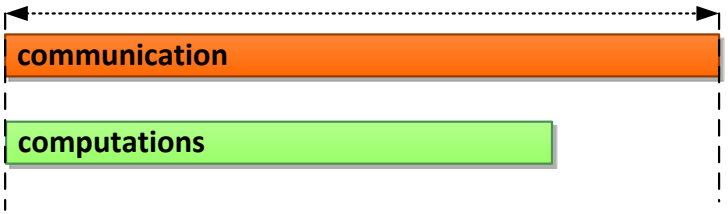
VRIJE UNIVERSITEIT BRUSSEL

What happens inside a kernel?



Nvidia Visual Profiler

A. Peak Performance

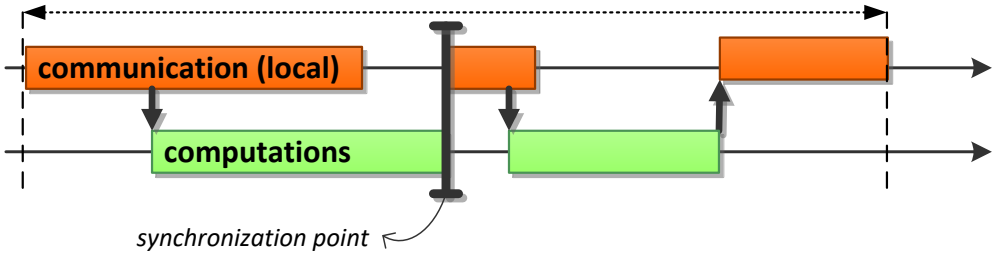


115 GB/s

1 TeraFlops

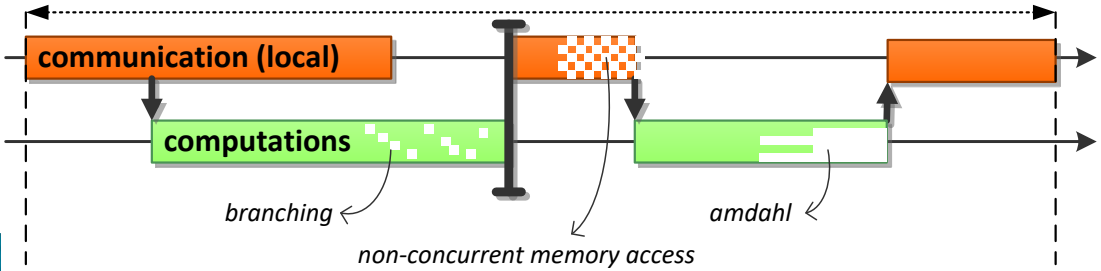
Resource-bound

B. Non-overlap



Non-overlap

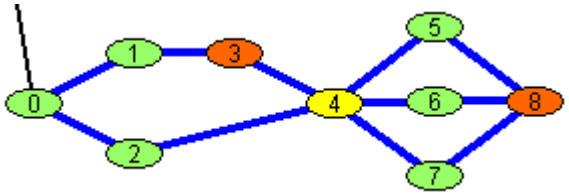
C. Anti-parallel interactions



Anti-parallel patterns
& model for latency hiding

Approach

Software characteristics



instruction dependency graph

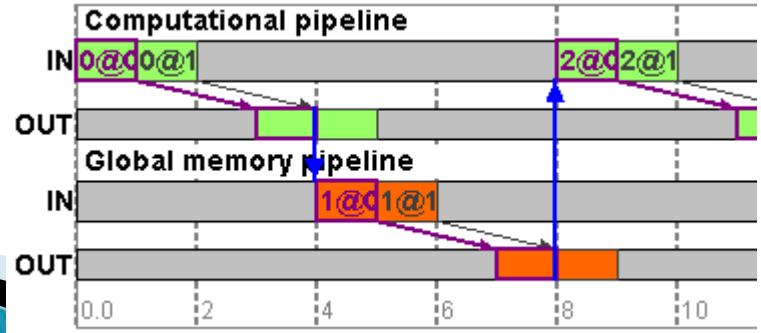
Execution Configuration

Hardware characteristics

Issue latencies (λ)
Completion latencies (Λ)

Pipeline Simulator

Execution Profile

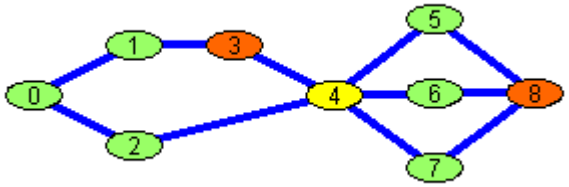


Efficiency report

VRIJE UNIVERSITEIT BRUSSEL

Model input: (1) Software

- **Instruction dependency graph** (IDG):
  - An IDG is a Directed Acyclic Graph in which each vertex of the graph corresponds to a single executed instruction (type of instruction). The edges between the vertices represent the **dependences** between the instructions. They can be either data dependences or control dependences.



Green node: computation
Orange node: memory request
Yellow node: barrier instruction (on the work group level)

Model input: (2) Hardware

- **Core count** (#Compute Units)
- **Clock frequency** f_{clock}
- **Subsystem set** S: the subsystems that can be modelled by an independent pipeline
- **Scheduler**: see later
- **Issue limit**: number of instructions that can be issued within one cycle
- **Available resources** R_{av}: a mapping from R (all types of resources) to the natural numbers, how much of r is available on a single core.

VRIJE UNIVERSITEIT BRUSSEL

Model input: (2) Hardware continued

- **Context set C**: A set of all possible contexts.
- **Context mapper f**: A function that determines the context in which an instruction $i \in I$ is executed given software characterization and the execution configuration
- **Subsystem and latencies mapper g**: A mapping from (instruction, context) to subsystem, issue and completion latency

VRIJE UNIVERSITEIT BRUSSEL

Hardware Parameters: Latencies

- **Issue latency** (λ): the number of cycles required between issuing two consecutive independent instructions
  - Determines peak performance
- **Completion latency** (Λ): the number of cycles until the result of an instruction is available for use by a subsequent instruction

They should not be constants!

- E.g.: Non-coalesced memory reads or bank conflicts
  - ➢ Larger latency

VRIJE UNIVERSITEIT BRUSSEL

Model input: (3) Execution configuration

- **Group size**: The number of work items (kernel threads) of the group.
- **Group count**: The total number of work groups. The total number of kernel threads is the product of group count and group size.
- **Resource requirements R_{req}** of a work group: A mapping from R to the natural numbers.

VRIJE UNIVERSITEIT BRUSSEL

Work groups

- We simulate 1 core/compute unit
- #Work groups to be executed on each core:
  = [Group count/Core count]
- **#Concurrent work groups** on a core

$$= \min_{r \in \boldsymbol{R}} (R_{av}[r] \div R_{req}[r])$$

- Simulation starts with #Concurrent work groups. Once all threads of work group have finished, a new work group is started.

VRIJE UNIVERSITEIT BRUSSEL

Simulator preprocessing step 1

- Context of each instruction is determined
  - f : (i ∈ I, kernel code, execution parameters) → c ∈ C
  - Added to the IDG for each instruction

VRIJE UNIVERSITEIT BRUSSEL

Simulator preprocessing step 2

- Latencies & subsystem are determined
  - $g : (i \in I, c \in C) \rightarrow (s \in S, \lambda, \Lambda)$
- IDG is decorated with:
  - *Issue latency* of the instruction
  - *Completion latency* of the instruction
  - *Subsystem* that executes the instruction

VRIJE UNIVERSITEIT BRUSSEL

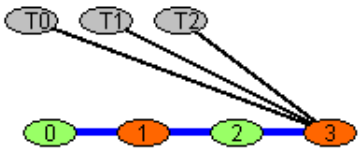
Simulation

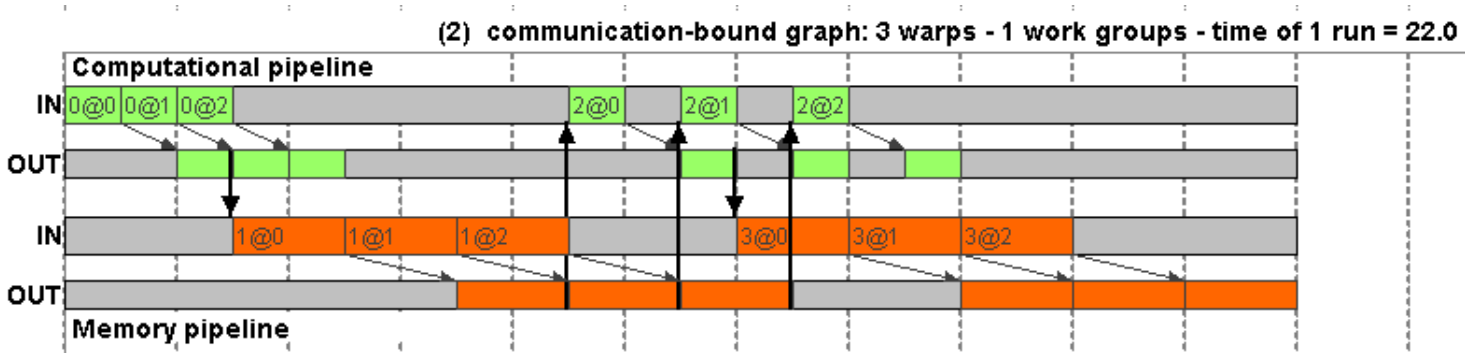
- Discrete Event Simulation: the state of the system changes only at *discrete* moments in time, triggered by events.
- An instruction has one of the following states:
  - **Waiting**: the instruction depends on instructions which have not yet completed execution.
  - **Ready**: all instructions on which the instruction depends have completed, but the instruction has not been issued yet.
  - **Issued**: the instruction has been issued to the appropriate subsystem, but it has not yet completed execution.
  - **Complete**: the instruction has completed execution, its results are available for dependent instructions.

VRIJE UNIVERSITEIT BRUSSEL

Pipeline simulator

- Semi-abstract model of GPU: 1 pipeline for computional subsystem and 1 pipeline for memory subsystem
- Based on instruction dependency graph of program and the latencies of the hardware





VRIJE UNIVERSITEIT BRUSSEL

Instruction scheduler

- The scheduler determines *which of the ready instructions is issued next*
  - Priority to the first work groups? (=WG_PRECEDENCE)
  - Or Round Robin (RR)?
- An important part of the scheduler is the **issue limit**, which determines the maximum number of instructions that can be scheduled in one cycle.

Inefficiencies

VRIJE UNIVERSITEIT BRUSSEL

Lambdas are not constants

- Will be longer for inefficient execution
  - Examples
    - Global vs local memory
    - Bank conflicts
    - Branching within a warp
  - *Is discussed in next lesson*
- Is modelled by the context

Computational microbenchmarks

www.gpuperformance.org

See paper Lemeire 2016:

Jan Lemeire, Jan G. Cornelis, Laurent Segers, Microbenchmarks for GPU characteristics: the occupancy roofline and the pipeline model, Procs of 24th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Heraklion, Greece, 2016

VRIJE UNIVERSITEIT BRUSSEL

Kernel: *fool the compiler*

```
__kernel void independent1madd(
    __global float *src,
    __global float *dst,
    int flag)
  {
    unsigned int index = get_global_id(0);
    float r = src[index];
    float l = get_local_id(0);

    #pragma unroll
    for (int i = 0; i < N; ++i) {
      r = r * l - 0.5f;
    }
    if (flag * r)
       dst[index] = r;
  }
```

VRIJE UNIVERSITEIT BRUSSEL

Java app

Microbenchmarks: measure your GPU

platform 1, device 1: GeForce GTX 650 Ti

			Computational performance	
iType	Peak	lambda	Lambda	Ridge point
SP	467.2	0.28	10.80	11264
MADD	1001.9	0.13	5.89	11264
INT	308.2	0.43	5.73	11264
SF	nope	nope	nope	nope
DP	nope	nope	nope	nope
			Memory performance	
Global	11.3	11.18	...	...
Char	34.0	3.84	80.98	1024
Char2	60.7	4.31	79.68	1024
Float	69.0	7.56	81.93	512
Float2	72.7	14.45	90.58	256
Float4	74.0	28.24	173.09	256
Local	200.0	0.66	...	...
Constant	83.7	1.57	...	...
Private	1820.0	0.07	...	...