Introduction: GPU Architecture

Ivy Peng
Assistant Professor in Computer Science
Scalable Parallel System (ScaLab)
Department of Computer Science, KTH

Who are we?

Scalable Parallel System (KTH-ScaLab)

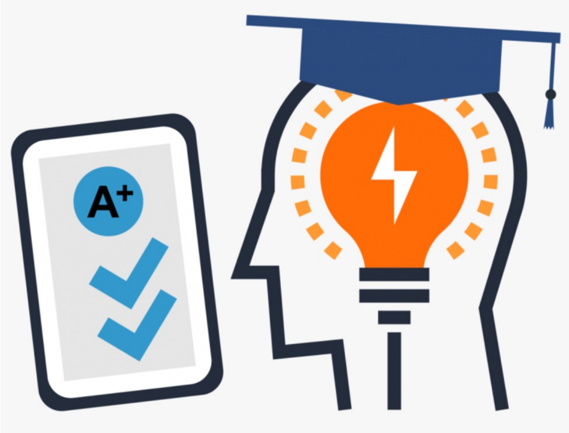
- **Converged Cloud and HPC systems**
  - Kubernetes
  - Architecture
  - Workflows
- **Memory Systems**
  - Heterogeneous memory
  - Disaggregated memory
- **Heterogeneous Computing**
  - GPU
  - RISC-V
  - Quantum Computing

ScaLab is a research group focusing on designing and optimizing algorithms, software, and computer architecture to improve the scalability of future computing systems and accelerate scientific discovery. Our research revolve around large-scale parallel systems, including high-performance computing and data centers. Our main research areas include:

Intended Learning Outcomes (ILO)

At the end of this course, you will be able to:

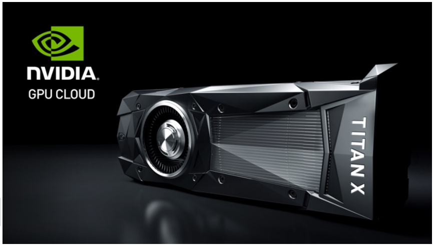
1. Describe the **architecture** of recent AMD and Nvidia GPUs
2. Compile and run **CUDA** and **HIP** kernels on GPUs on a cluster
3. Use **profiling** tools to measure and analyze the performance of CUDA and HIP codes
4. Transform and **implement** a serial kernel into CUDA and HIP code on GPU



Course Organization: Computer Resources

- For AMD GPU: you will be given access to the PDC Dardel Supercomputer with an allocation
  - You will learn how you connect and run jobs on supercomputers as part of this course

- For Nvidia GPU: Google Colab on Google Cloud



Outline

Thursday – part 1	GPU Architecture: AMD and Nvidia CUDA Programming Hands-on
Thursday – part 2	CUDA Programming Hands-on
Friday – part 3	AMD GPU Hands-on
Friday – part 4	HIP Programming Hands-on

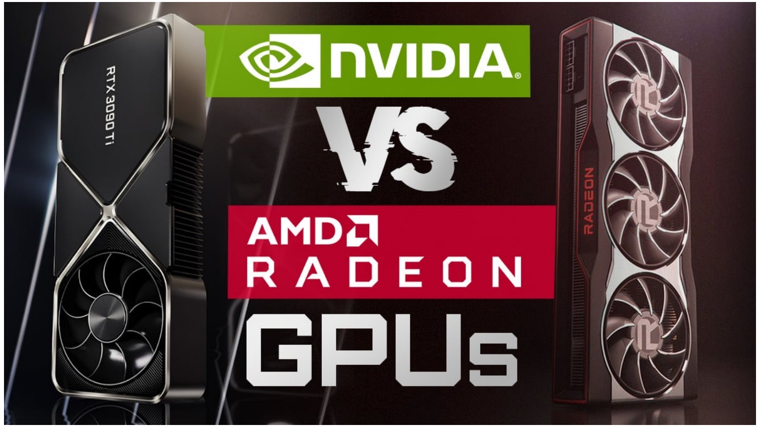
Graphical Processing Unit (GPU)

GPU = specialized accelerator for processing images in video frame for display devices.

GPUs are used in game consoles, embedded systems (like systems on cars for automatic driving), computers, and **supercomputers**.

- Since 2012, GPUs are the main workforce for training deep-learning networks

Some important GPU vendors: **NVIDIA**, **AMD**, ...



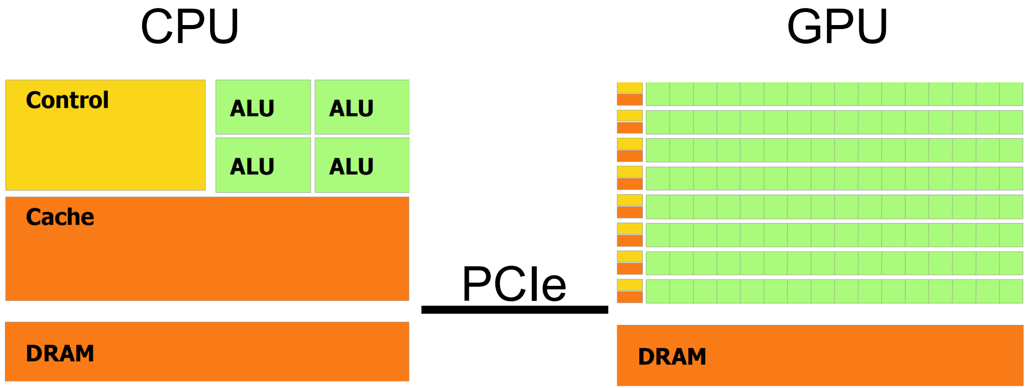
Integrated GPU v.s. Dedicated GPU

- The main difference lies in the memory:
  - Integrated GPU shares the system memory with CPU
  - Dedicated GPU has its own memory
- Integrated GPU is often found in laptops, more power efficient, e.g., Intel HD or Iris Graphics.
- Dedicated GPUs are often removable and need more power, and provide higher performance
- In HPC, we focus on dedicated GPUs

CPUs with integrated graphics

Discrete graphics card

GPU v.s. CPU Architecture



CPU has tens of massive cores, CPU excels at irregular control-intensive work

- Lots of hardware for control, fewer ALUs

GPU has thousands of small cores, GPU excels at regular math-intensive work

- Lots of ALUs, little hardware for control

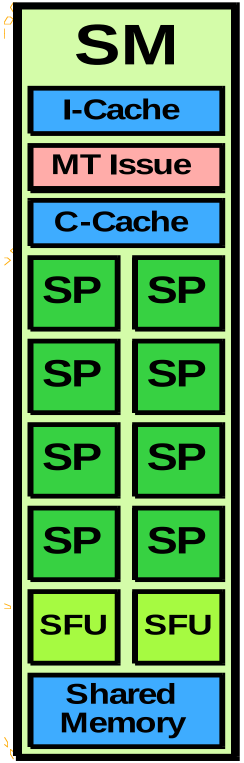
GPU Hardware Model for Nvidia GPUs

The fundamental computing entity is

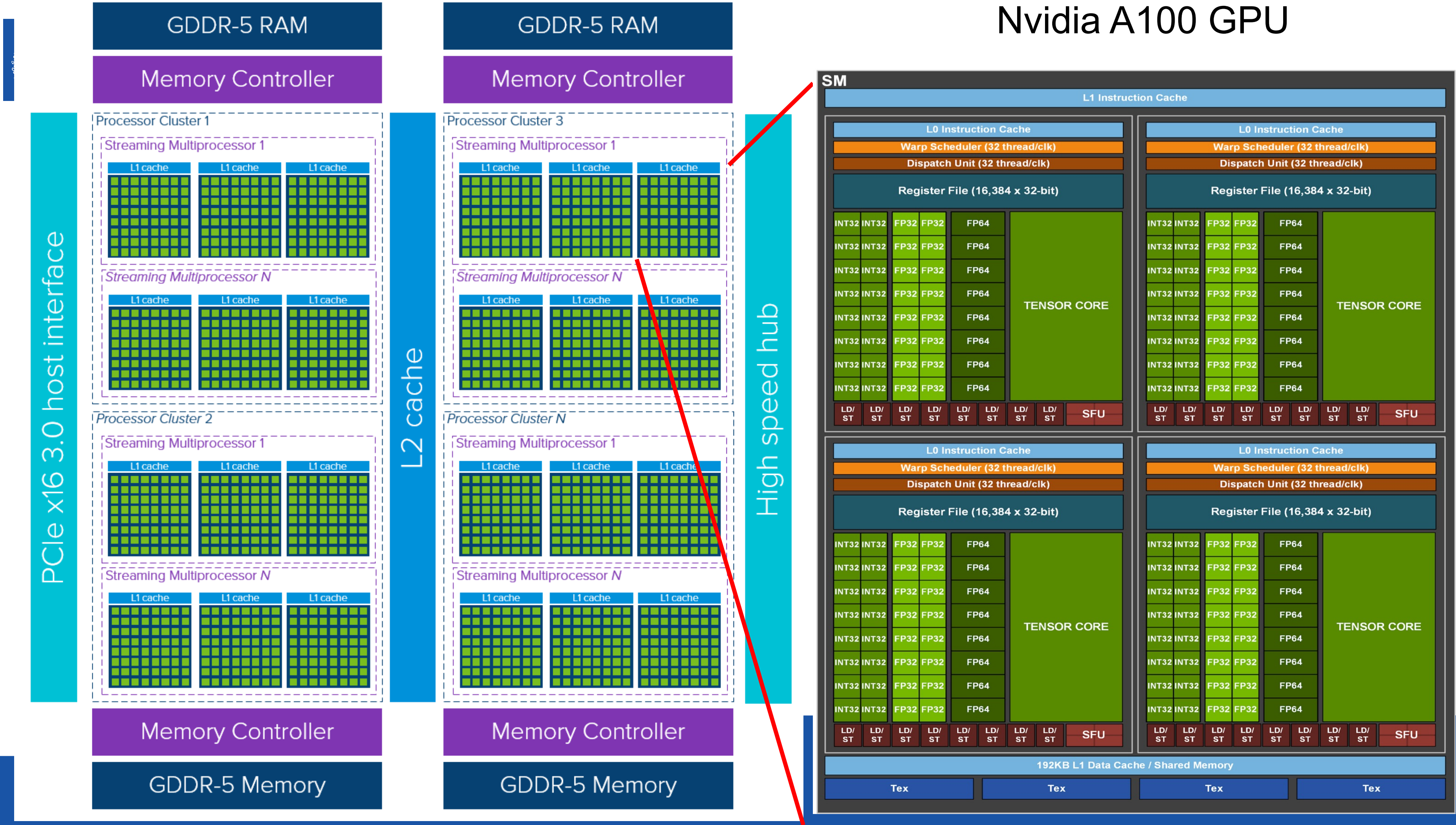
- **Streaming Processor** (**SP**) or **CUDA core**

A **Streaming Multiprocessor** (**SM**):

- A collection of 8/32/192 CUDA Cores (depends on SM architecture)
- Has some fast cache shared memory
- Can synchronize



Nvidia A100 GPU



GDDR-5 RAM

Memory Controller

GDDR-5 RAM

Memory Controller

PCIe x16 3.0 host interface

Processor Cluster 1

Streaming Multiprocessor 1

L1 cache L1 cache L1 cache

Streaming Multiprocessor N

L1 cache L1 cache L1 cache

Processor Cluster 2

Streaming Multiprocessor 1

L1 cache L1 cache L1 cache

Streaming Multiprocessor N

L1 cache L1 cache L1 cache

L2 cache

Processor Cluster 3

Streaming Multiprocessor 1

L1 cache L1 cache L1 cache

Streaming Multiprocessor N

L1 cache L1 cache L1 cache

Processor Cluster N

Streaming Multiprocessor 1

L1 cache L1 cache L1 cache

Streaming Multiprocessor N

L1 cache L1 cache L1 cache

High speed hub

Memory Controller

GDDR-5 Memory

Memory Controller

GDDR-5 Memory

SM

L1 Instruction Cache

L0 Instruction Cache

Warp Scheduler (32 thread/clk)

Dispatch Unit (32 thread/clk)

Register File (16,384 x 32-bit)

INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64

TENSOR CORE

LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST SFU

L0 Instruction Cache

Warp Scheduler (32 thread/clk)

Dispatch Unit (32 thread/clk)

Register File (16,384 x 32-bit)

INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64

TENSOR CORE

LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST SFU

L0 Instruction Cache

Warp Scheduler (32 thread/clk)

Dispatch Unit (32 thread/clk)

Register File (16,384 x 32-bit)

INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64

TENSOR CORE

LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST SFU

L0 Instruction Cache

Warp Scheduler (32 thread/clk)

Dispatch Unit (32 thread/clk)

Register File (16,384 x 32-bit)

INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64
INT32 INT32 FP32 FP32 FP64

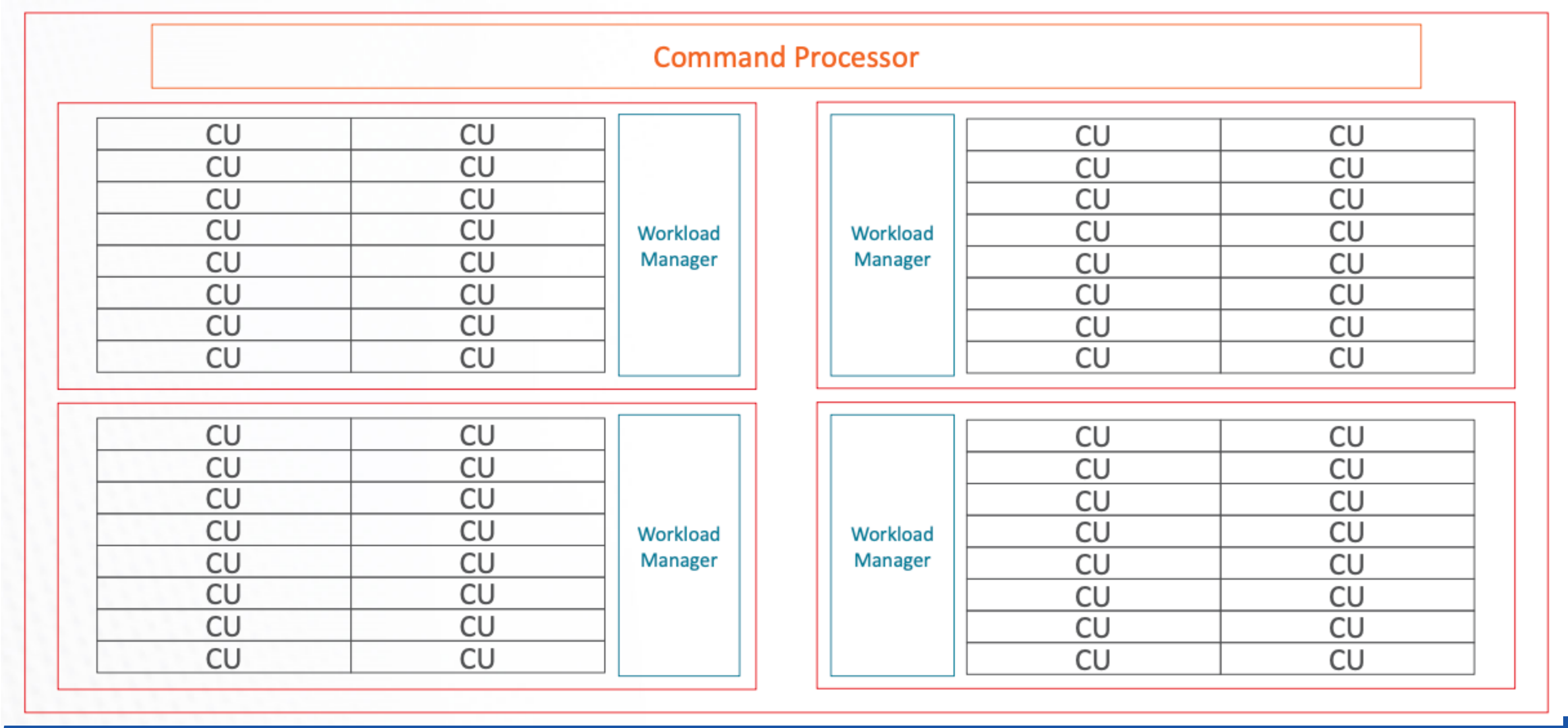
TENSOR CORE

LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST SFU

192KB L1 Data Cache / Shared Memory

Tex Tex Tex Tex

AMD Graphics Core Next (GNC) Architecture



Command Processor

CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU

Workload Manager

Workload Manager

CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU

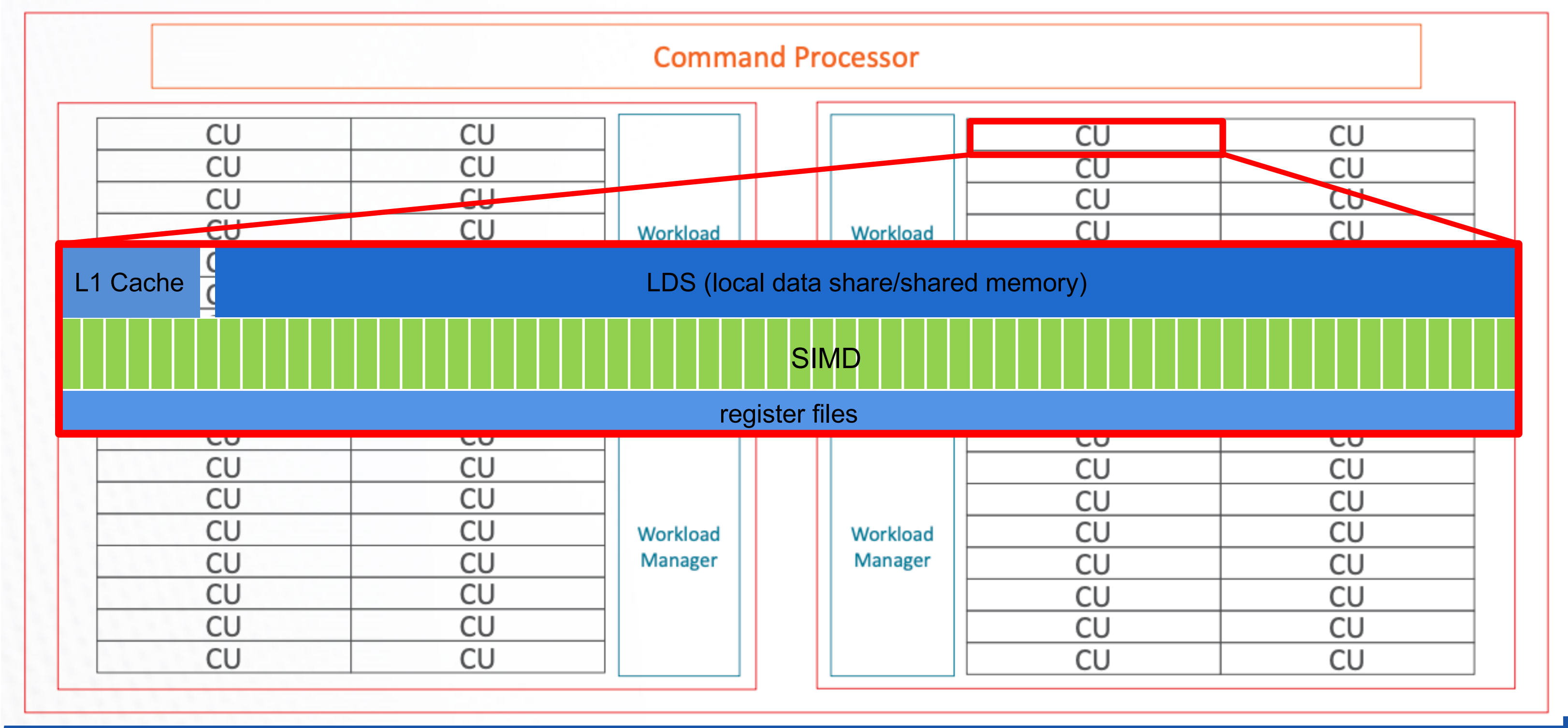
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU

Workload Manager

Workload Manager

CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU
CU	CU

AMD Graphics Core Next (GNC) Architecture



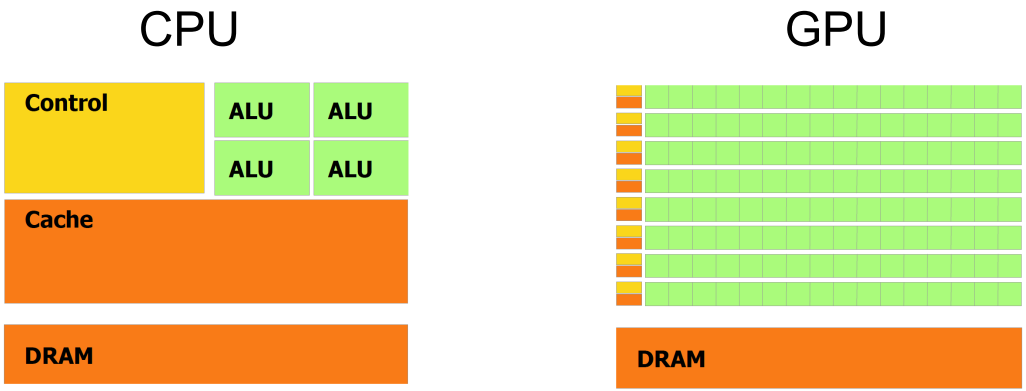
Recent Nvidia GPU Architecture

- Nvidia Volta Architecture, tensor cores, mixed precision
  - the GA100 GPU has 128 SMs, 64 FP32 CUDA Cores/SM
- Nvidia Ampere Architecture, 3rd gen NVLink
- Nvidia Hopper Architecture

Questions: how many cores in GA100?
2 x 13 x 192 = **4992!**
Questions: how many cores per node on Dardel host?
2 x 64 = 128

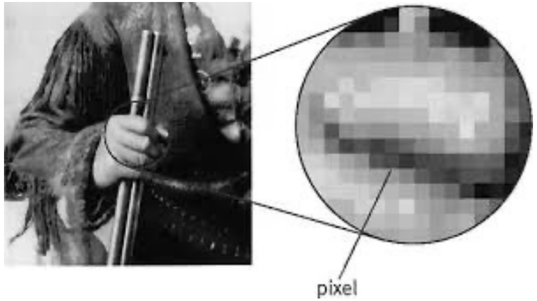
GPU is Throughput-oriented Architecture

- GPUs focus on executing many computation in parallel to maximizing the **total throughput**
  - GPUs do not target to minimize the latency of a single task



GPU Design Motivation: Process Pixels in Parallel

1. Data parallel
   - In 1080i and 1080p videos, 1920 x 1080 pixels = 2M pixels per video frame → compute intensive
   - Lots of parallelism at low clock speed → power efficient
2. Computation on each pixel is independent from computation on other pixels
   - No need for synchronization
3. Good data-locality = access to data is regular
   - No need for large caches

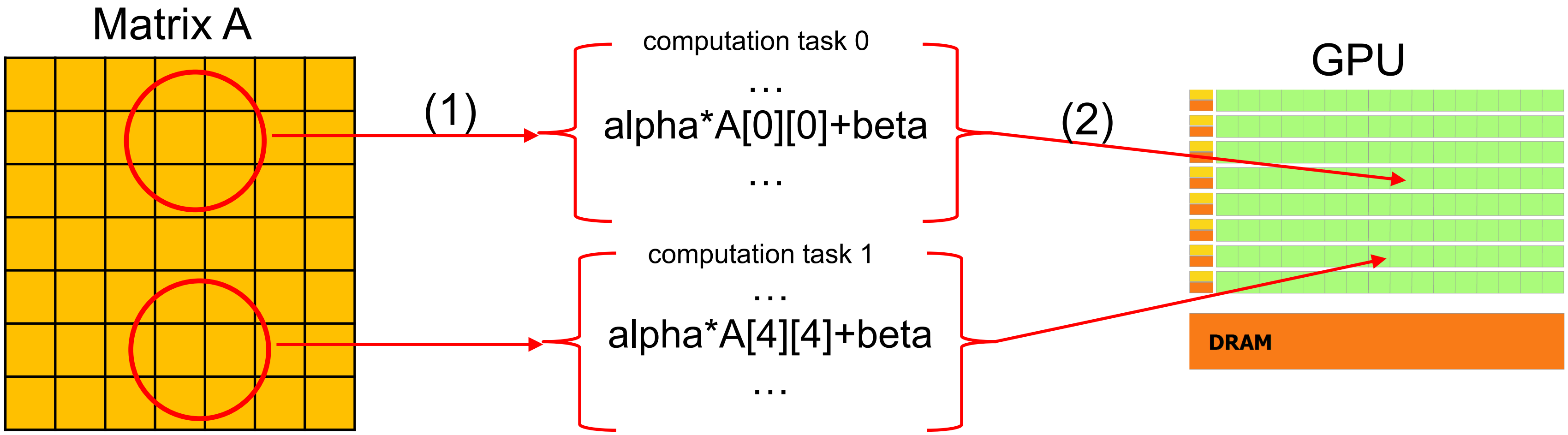


Mapping computation tasks to GPU hardware

(1)Define computation tasks – done by you

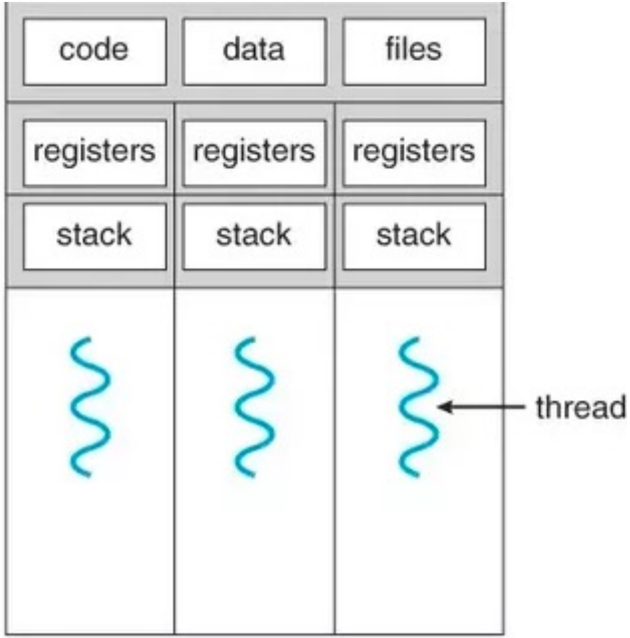
(2)Schedule computational tasks on GPU cores – done by CUDA runtime

- Intuitively, higher throughput when more GPU cores are busy – **How?**



Hardware Multithreading

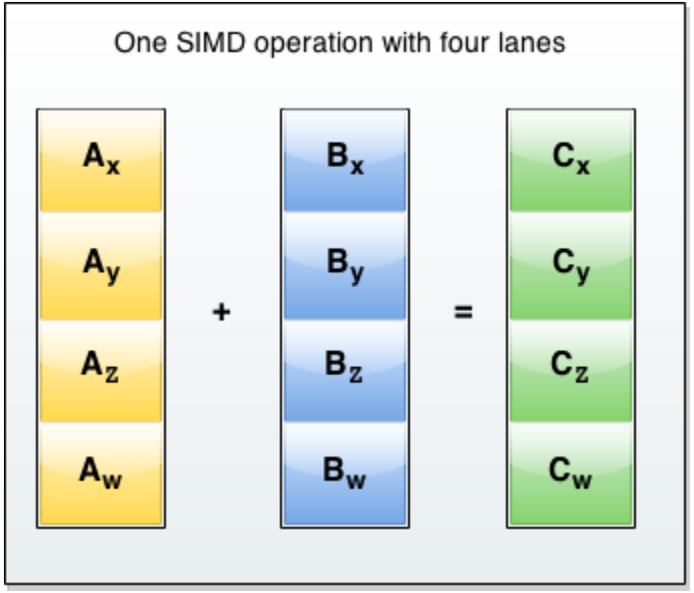
- Computation can be divided into a collection of many concurrent sequential tasks that executed across many threads
  - E.g., decompose a large matrix
- Thread can be seen as **virtualized scalar processor** with a program counter, register file and associate processor state
- Multithreading can be implemented in software (OS) or hardware
  - E.g., hyper-threading on Intel processor
- **Throughput-oriented architectures have implemented in hardware**



SIMD Execution

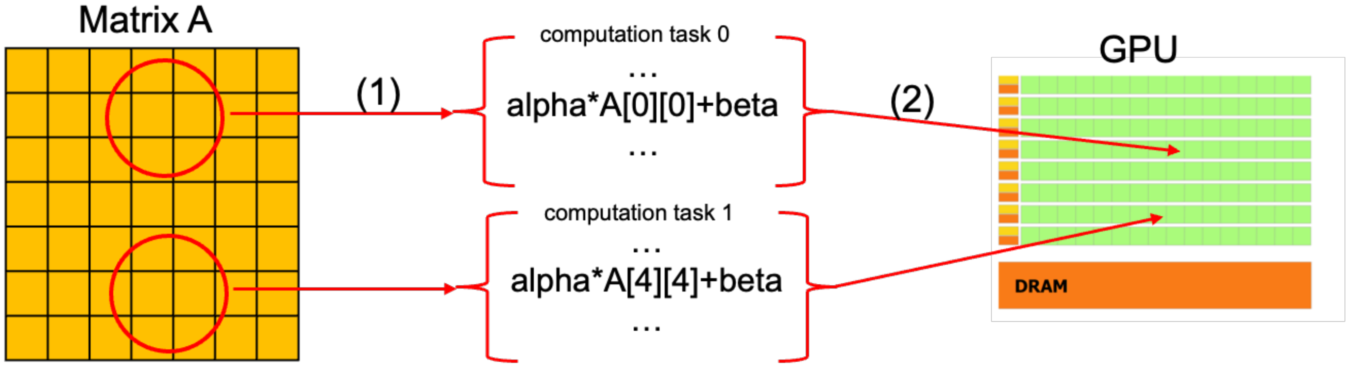
Parallel processors employ some of form of Single-Instruction, Multiple Data (**SIMD**) execution to **increase the throughput:**

- Issuing a single instruction in a SIMD machine applies the given operation to potentially many data operands
- E.g. : a large matrix



How to improve GPU utilization?

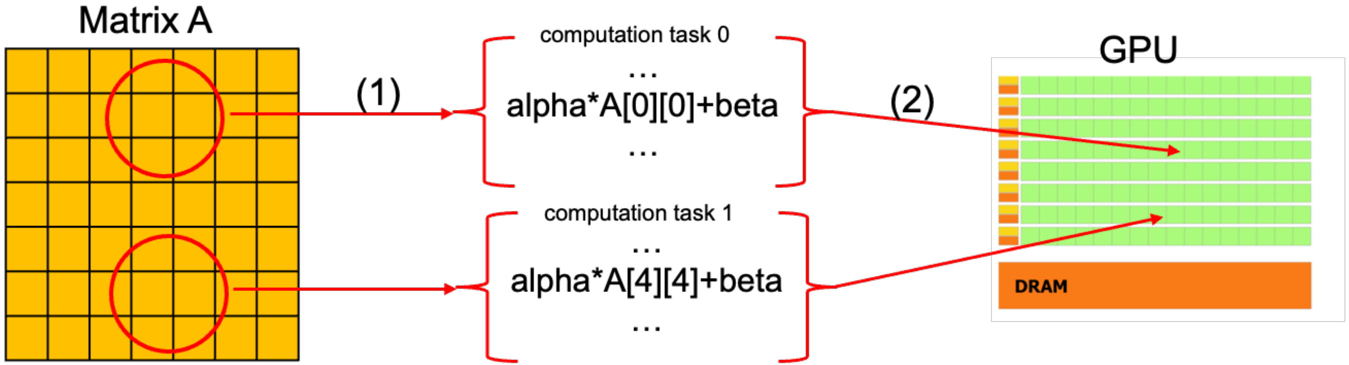
- Increase # of CUDA cores
  - High-end GPUs have a large number of cores
- Define computation tasks with low dependency and synchronization
  - If task 0 needs to wait for task 1, adding CUDA cores won’t help
- Define many tasks to **oversubscribe**
  - Define #tasks >> #cores



Hardware Multithreading Hides Latency

Long-latency operations of a single-thread can be hidden or covered by **ready-to-run work** from another thread, examples:

- Thread 1 cannot run because waiting data from DRAM
- Thread 2 can run because all its required operands are ready
- Switch to run thread 2 while overlapping data fetching for thread 1



Q & A