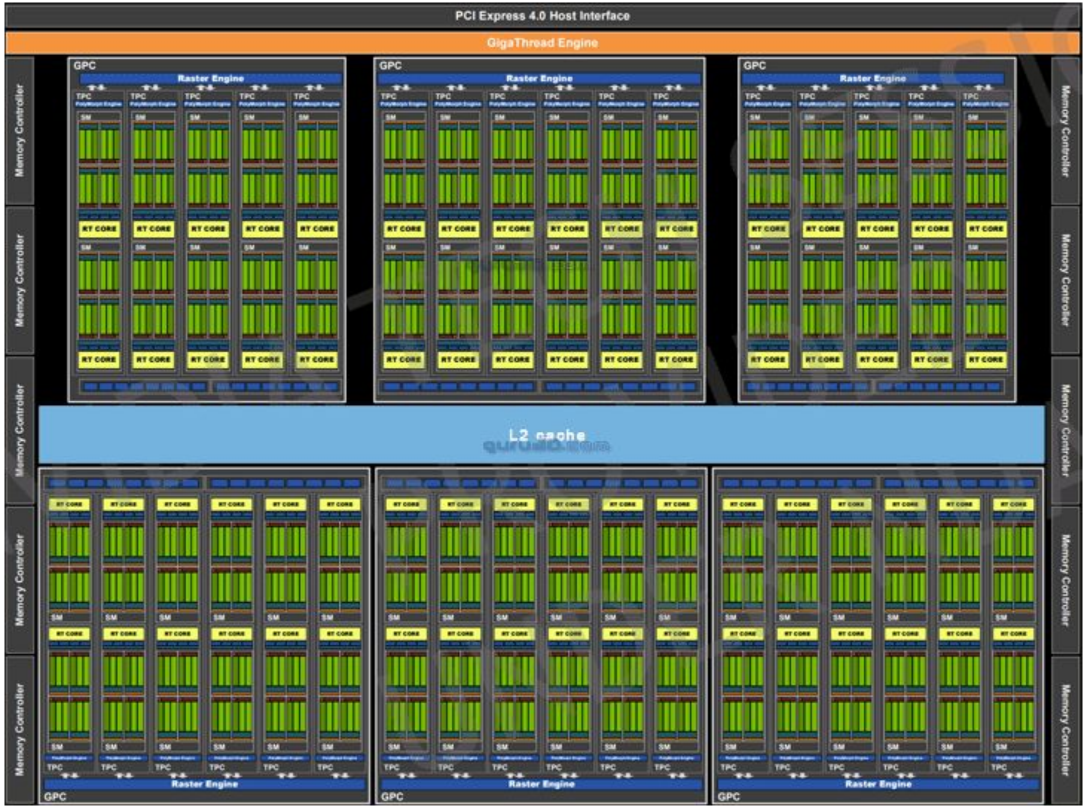
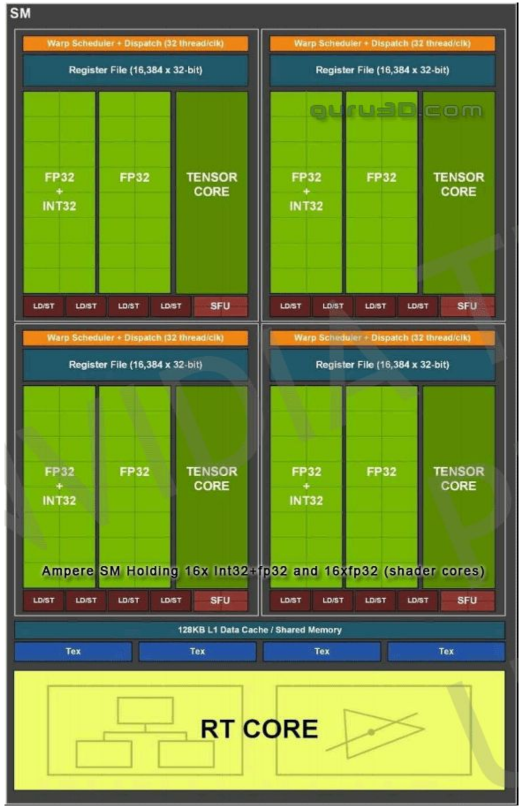
GP-GPU
and
High Performances Computing

Lecture 2
GP-GPU Architecture

Ampere Architecture




SM
Warp Scheduler + Dispatch (32 thread/clk)
Register File (16,384 x 32-bit)
FP32 + INT32
FP32
TENSOR CORE
LD/ST
SFU
Ampere SM Holding 16x Int32+fp32 and 16xfp32 (shader cores)
128KB L1 Data Cache / Shared Memory
Tex
RT CORE

GP-GPU computing in this course?

Programming Interfaces

OpenCL™

NVIDIA.
CUDA®

OpenACC
More Science, Less Programming

Vulkan®

The CUDA Programming Model

- ➔ CUDA is a recent programming model, designed for GPUs with
  - ◆ Manycore architectures
  - ◆ Wide SIMD parallelism
  - ◆ Scalability

- ➔ CUDA provides:
  - ◆ A thread abstraction to deal with SIMD
  - ◆ Synchronization & data sharing between small groups of threads

- ➔ CUDA programs are written in C + extensions
- ➔ OpenCL uses very similar programming model

Nvidia Card Technology

General card (e.g. GeForce RTX 3090)

- ➔ Available cores:
  - ◆ Generic CUDA Cores
  - ◆ Tensor Cores
  - ◆ Ray Tracing Cores
- ➔ Floating-point format:
  - ◆ simple-precision: many units, high perf
  - ◆ double-precision: very few units, low perf

Cannot be inserted into clusters

Computing capabilities are **not certified**

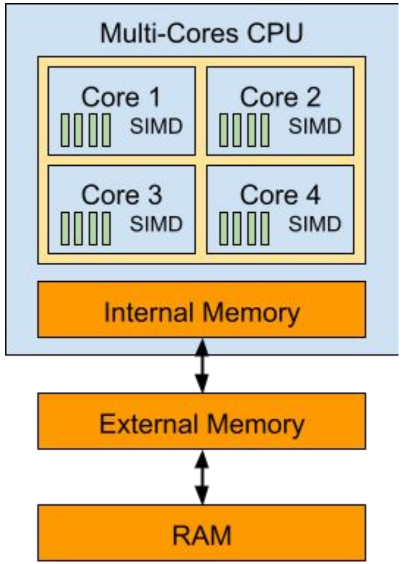
Professional card (e.g. Quadro RTX A6000)

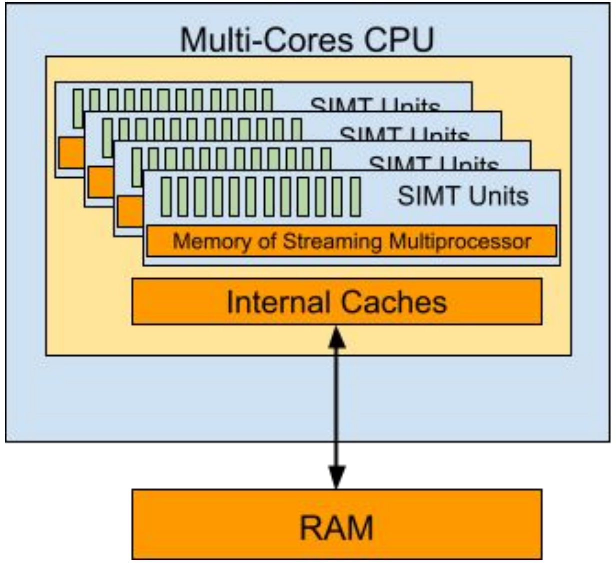
- ➔ Available cores:
  - ◆ Generic CUDA Cores
  - ◆ Tensor Cores
- ➔ Floating-point format:
  - ◆ simple-precision: many units, high perf
  - ◆ double-precision: many units, high perf

Can be used in cluster for high performance computing

Computing capabilities are **certified**

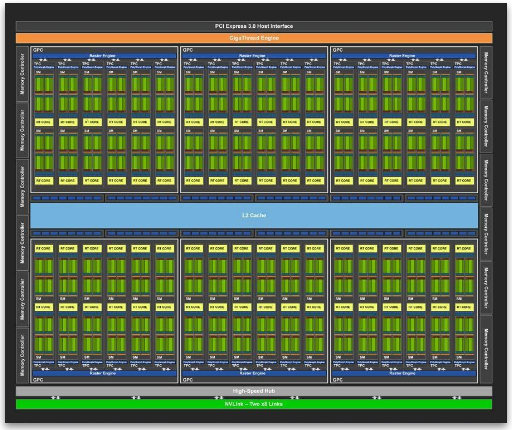
CPU vs GPU - General overview

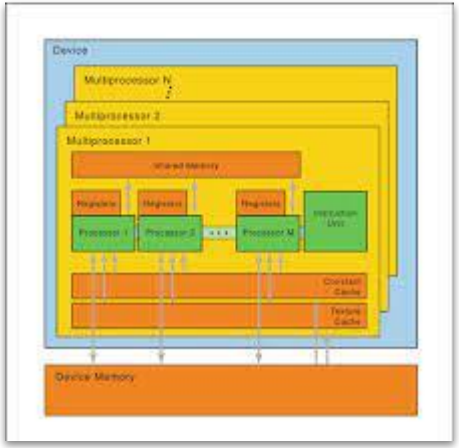


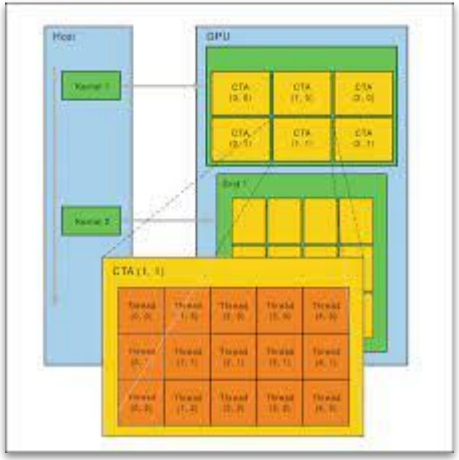


CPU uses GPU as a scientific coprocessor for SIMD computing

From the architecture to the programming







$$\text{GPU Chip} \longleftrightarrow \text{Virtual machine architecture} \longleftrightarrow \text{Programming model and language}$$

The reason why

Floating power cost

A typical CPU chip with 10^4 floating points units

- ➔ Size : 100mm^2
- ➔ Power consumption : 200 W
- ➔ Processing power : 20 TFLOPS

In order to get 1 exa flops, we need

- ➔ 50 000 chips
- ➔ 10 MW

Communication cost (per flops)

- ➔ 64-bit double precision : 26 pJ
- ➔ 256-bit buses : 20 pJ +256 pJ + 1nJ
- ➔ 256-bit access : 50pJ
- ➔ RW DRAM : 16nJ
- ➔ Off-chip Link : 500pJ

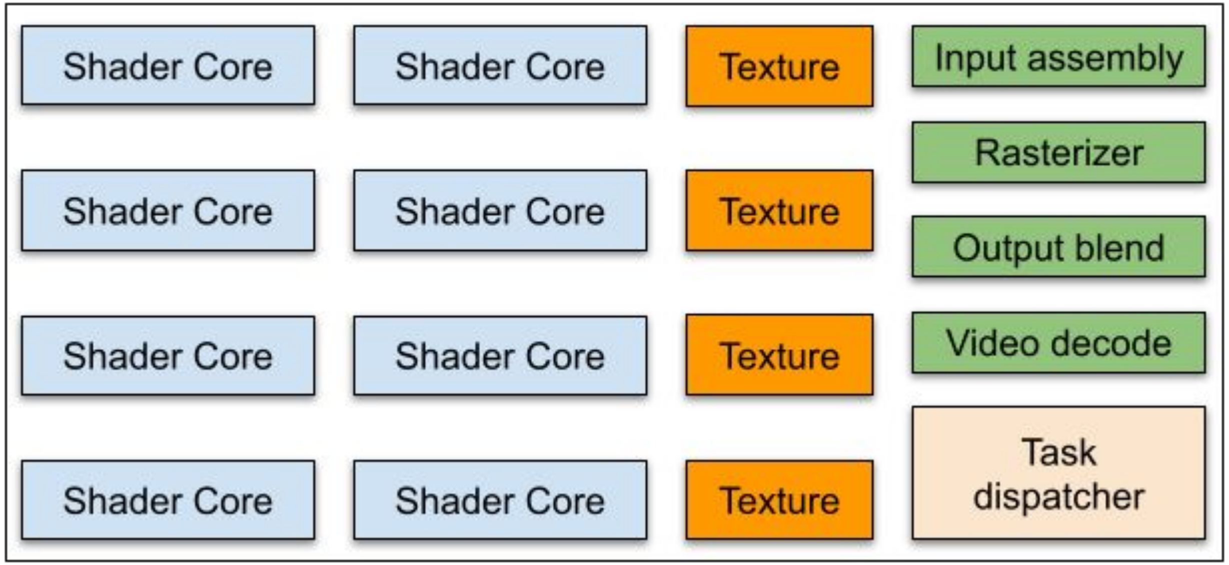
Cost per data movement (roughly, in 2015)

Communication cost (per flops)

- ➔ Move data 1mm on chip : 6 pJ
- ➔ Single floating point operation : 100 pJ
- ➔ Move data 20 mm on chip : 120 pJ
- ➔ Move data off chip (within SMP) : 250 pJ
- ➔ Move data into DRAM : 2000 pJ
- ➔ Move data off chip : > 2500 pJ

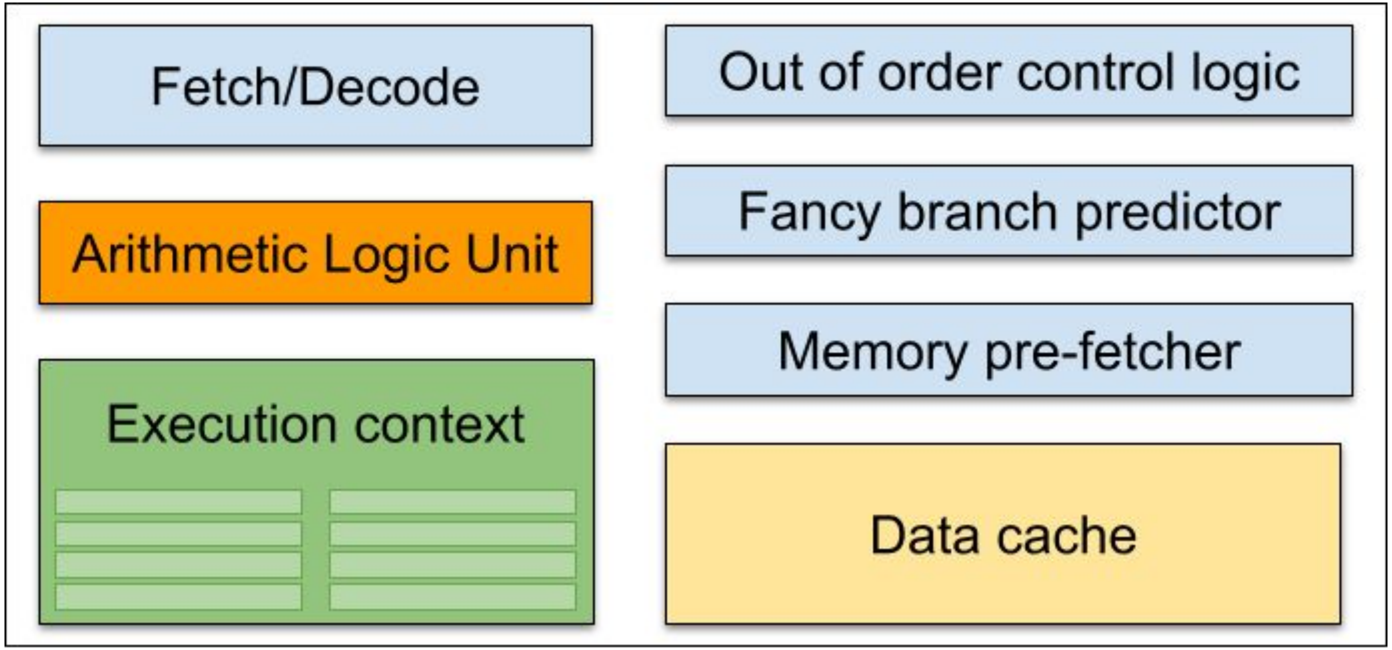
How to decrease energy cost?

What is inside a GPU ?



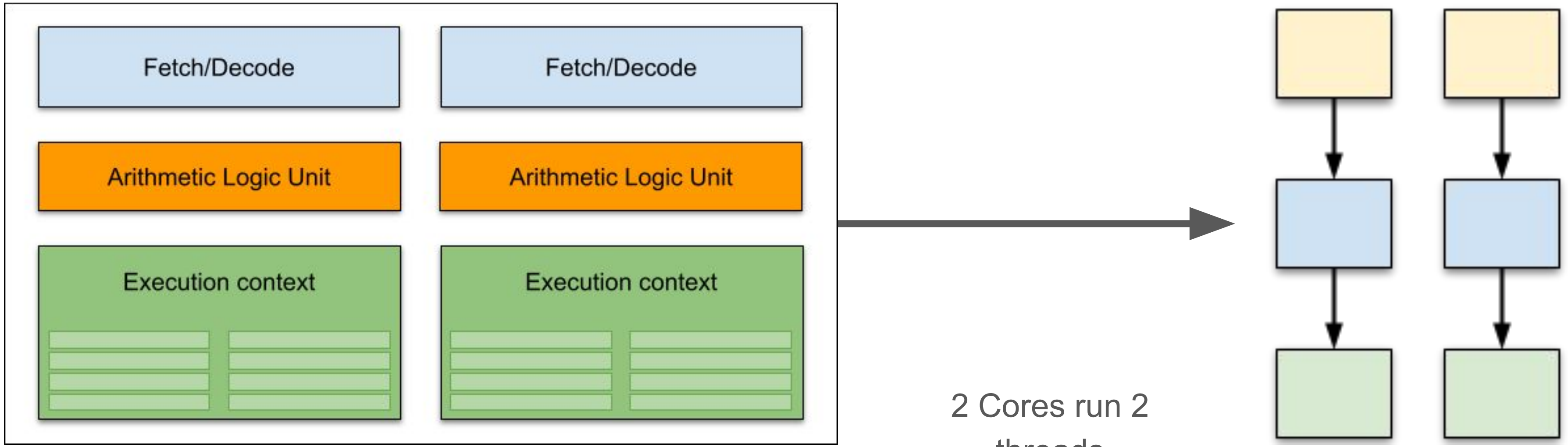
A GP-GPU is a heterogeneous multi-processor optimized for graphics

CPU model working



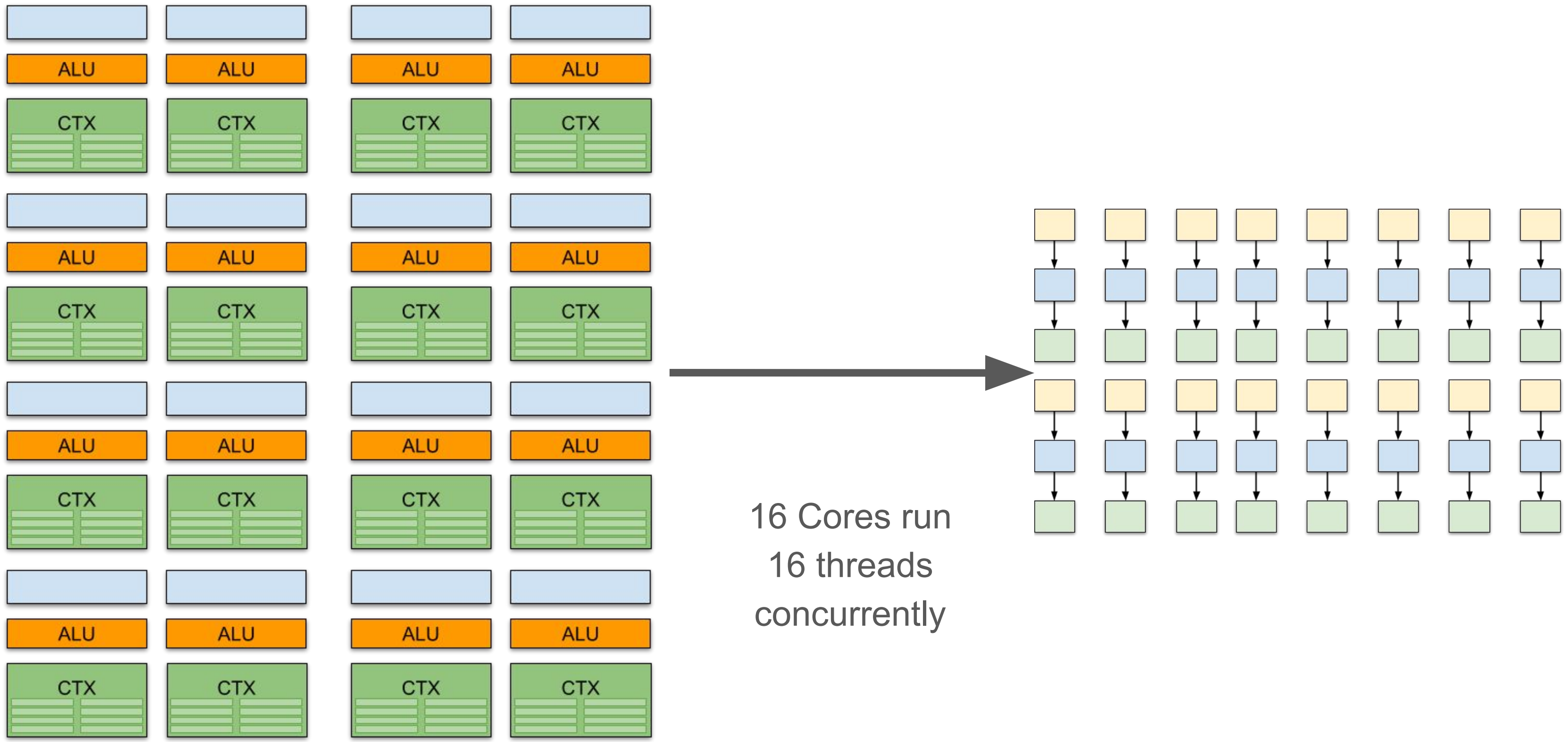
Remove components that are dedicated to on single instruction stream run fast

2 Cores



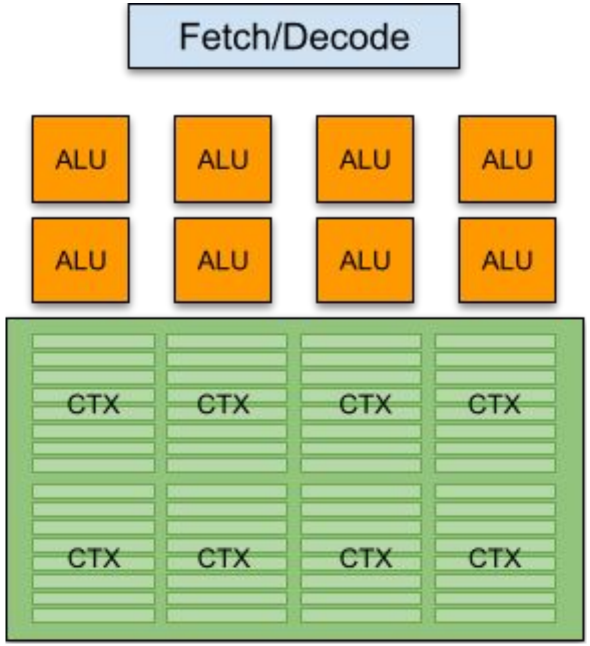
2 Cores run 2 threads concurrently

16 Cores



16 Cores run
16 threads
concurrently

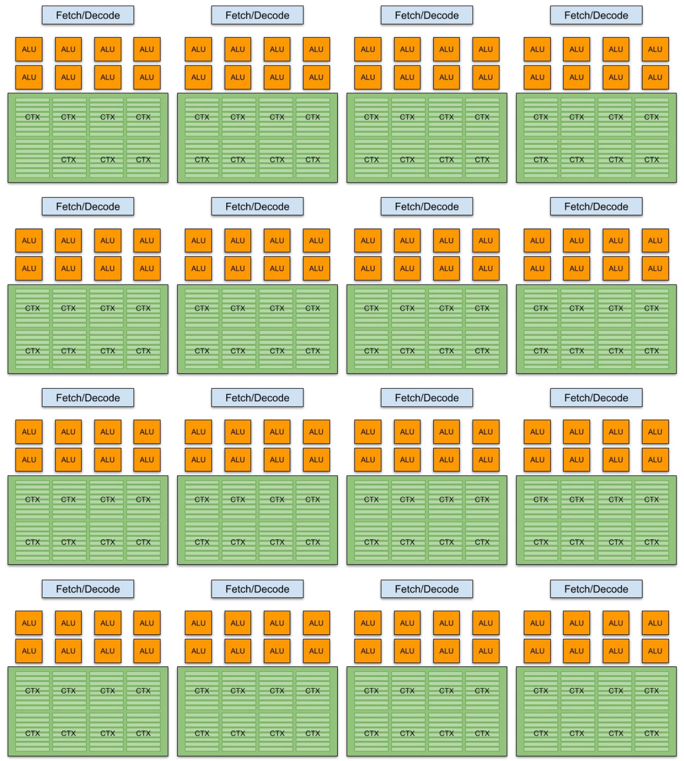
Do we really need multiple decoders?



In classical computer, this organisation is called, single instruction, multiple data (SIMD).

In GPU Computing, this is called warp (Nvidia).

Multilevel parallelism



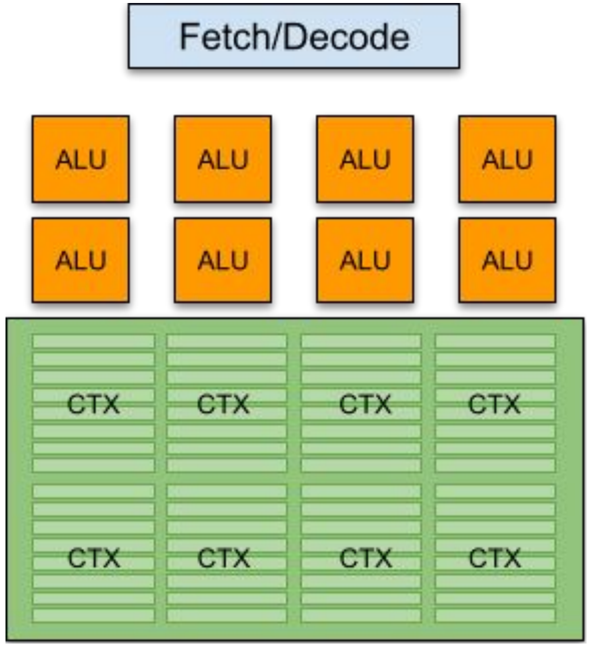
A chip with such a design as :

- 16 cores
- 8 Multiplication and Addition units per core
- 16 Instructions streams
- 256 Gflop/s at 1 Ghz

A real cheap will be at least 64 times larger.

The one we will be using at 10240 cores.

Program execution

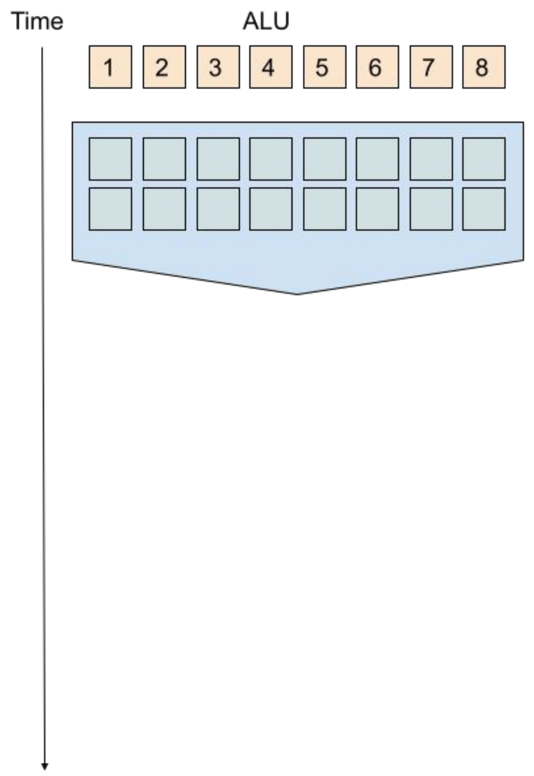


Each thread will receive a copy of the program.

Each program will be executed on a different subset of data.

This is called Single Program Multiple Data (SPMD).

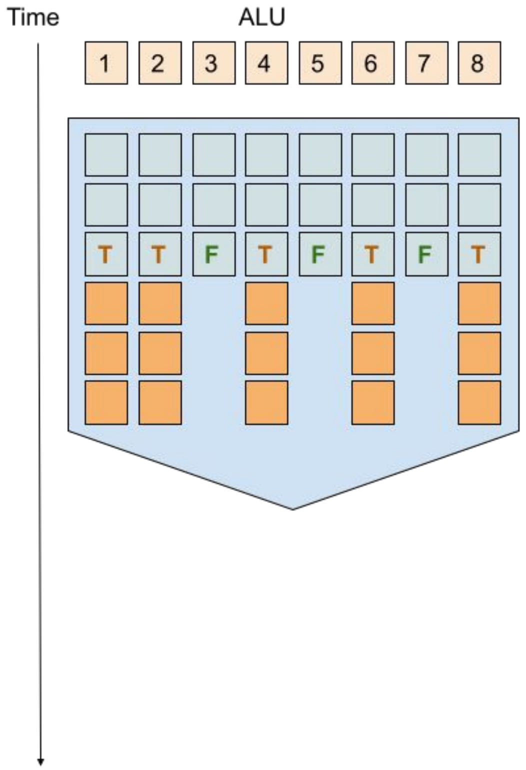
Dealing with instruction flow



```
<unconditional flow>

if (x < 0) {
      y = log10 (x);
      y += Wp;
   out = y * W0;
} else {
      out = W0;
}
<Some unconditional
flow>
```

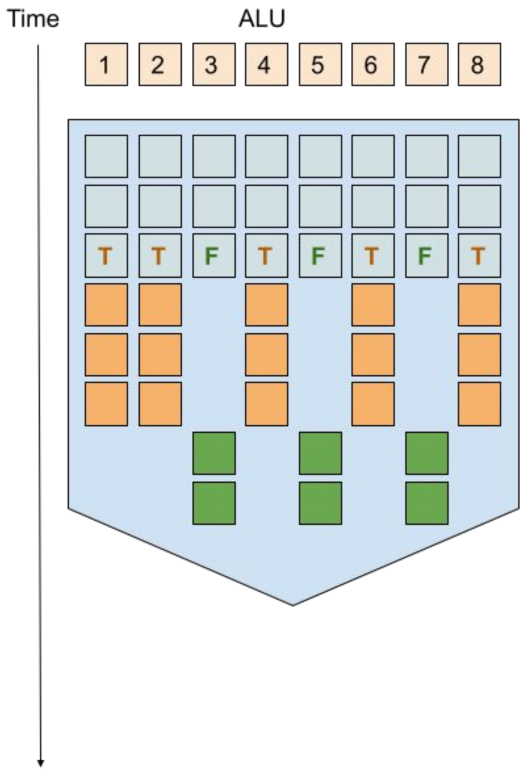
Dealing with instruction flow



```
<unconditional flow>

if (x < 0) {
      y = log10 (x);
      y += Wp;
   out = y * W0;
} else {
      out = W0;
}
<Some unconditional
flow>
```

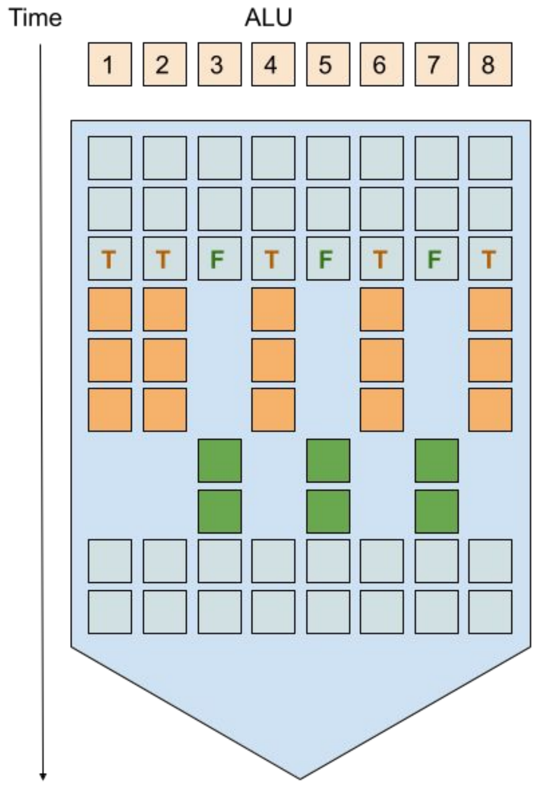
Dealing with instruction flow



```
<unconditional flow>

if (x < 0) {
      y = log10 (x);
      y += Wp;
   out = y * W0;
} else {
      out = W0;
}
<Some unconditional
flow>
```

Dealing with instruction flow



```
<unconditional flow>

if (x < 0) {
      y = log10 (x);
      y += Wp;
   out = y * W0;
} else {
      out = W0;
}
<Some unconditional
flow>
```

The hardware is dealing with non-executing units.

Wait state

- ➔ When a processing element is waiting on data from a previous operation, a waiting state occurs
- ➔ Each memory operation is 100-1000 cycles
- ➔ Stalls are avoided using caches and processors logics

Core organization

Each thread in a warp share

- ➔ An instruction stream to decode
- ➔ An execution context for storage (64kB per thread)
- ➔ 8 SIMD functional unit
- ➔ One control unit

- ➔ Each core can run a group of 32 threads, a warp.
- ➔ Warps can be interleaved to run simultaneously (up to 320)
- ➔ Up to 10240 threads context can be stored

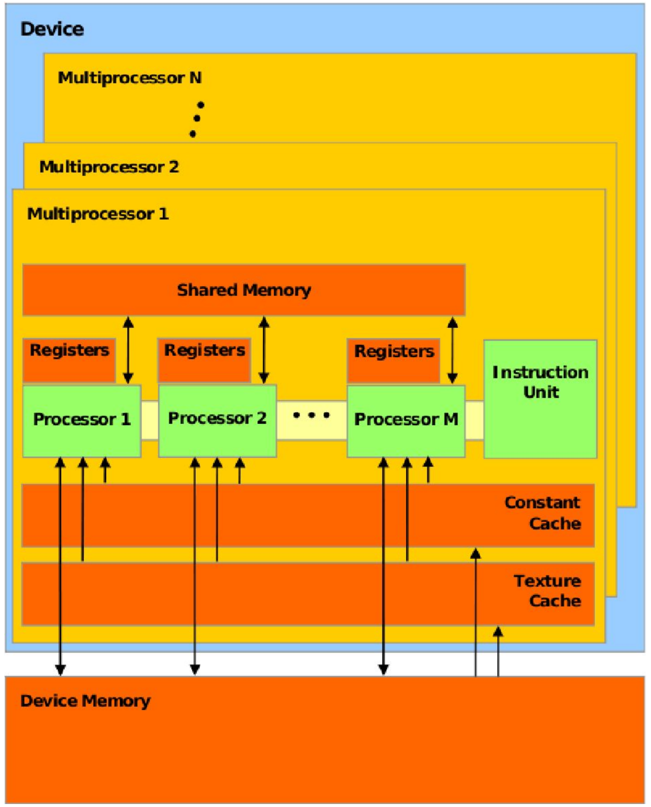
Stream Multiprocessors architecture

A GPU is a set of N Stream multiprocessors

- ➔ N independent « SIMT » machines
- ➔ Sharing the GPU board memory

Current (Ampere) architecture

- ➔ several instruction decoders/units
- ➔ 64 hardware threads
- ➔ warps of 32 threads
- ➔ less strongly synchronized
- ➔ 32K-128K registers distributed among all hardware threads (and not shared)
- ➔ A fast memory shared between all running threads (of the SM)
- ➔ several schedulers of warps of threads



Tensor cores

Ampere architecture has 64 tensor cores

1 Tensor Core is able to achieve a flow of product-add on a flow of 4x4 matrices

➔ D = A.B : produces a flow of D output matrices
➔ D = A.B + C, with accumulation of A.B product flow into C matrix

➔ A tensor core is a hardware implementation of a matrix operator

Some specifics about our cards

- Frequency: 1410-1800 GHz
- Memory size: 46GB
- Memory bandwidth: 768 GiB/s
- Cuda Core: 10752
  - Tensor Core: 336
  - RT Core: 84
- Precision
  - Half: 38.709 TFlops
  - Single: 38.709 TFlops
  - Double: 1.210 TFlops
- Programming interface:
  - Cuda 8.6
  - OpenCL 3.0
  - Vulkan 1.2

Conclusions

A GPU is a CPU

- ➔ Parallel computers need independent work to run their many cores (or other resources) efficiently

Themes of this class

- ➔ Difference between CPU and GP-GPU
- ➔ Different levels in a GPU
- ➔ Overview of main challenges in GP-GPU computing