Special thanks to Assoc Prof Nicholas Vun (Associate Chair in Academic)
for his initiative and contribution to this module

CE/CZ 3001:
Advanced Computer Architecture

Module 6: GPU Architecture and CUDA Programming

- GPU System Architecture

Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Outline

- Motivation for GPU programming
- Evolution of GPU Architectures
- Programming on GPU (CUDA)
- Basic internal operations of GPUs

GPUs for Scientific Computing

GPU is (originally) designed to accelerate 3D graphics renderings on PC

- such as for 3D games like Minecraft and Fortnite

But GPU has now also been used extensively for many mathematical and scientific computing purposes

- in particular, AI applications

Case Study: Computer Vision based AI applications

One particular area that AI has been used extensively

- computer vision-based applications
- using machines to classify images (e.g. name what they see), cluster similar images (e.g. photo search), perform object recognition within scenes, etc

Some examples:

- Identify individuals, street signs
- Medical diagnosis
- Natural-language processing
- Self-driving cars, Robotics, Drones

Convolutional Neural Networks (CNN) in AI

CNN is used extensively in many Computer Vision applications

- CNN based neural networks techniques have been at the heart of spectacular advances in deep learning.
- particularly suitable to be implemented using GPUs

During the 2012 ImageNet computer image recognition competition

- Alex Krizhevsky used GPUs to implement CNN based deep learning
- First time that Deep learning algorithm beat — by a huge margin — handcrafted software written by computer vision experts.

Discrete Convolution* operations are used in various stages in CNN.
(*more correctly, Discrete Cross Correlation)

Discrete Convolution

Mathematically: $x[n] * h[n] = \sum_{k=-N}^{+N} x[k]\ h[n-k]$

Example:

$$\begin{bmatrix} x1 & x2 & x3 \\ x4 & x5 & x6 \\ x7 & x8 & x9 \end{bmatrix} * \begin{bmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{bmatrix}$$

(Discrete Cross Correlation)

$$x[n] * h[n] = \sum_{k=-N}^{+N} x[k]\ h[k]$$

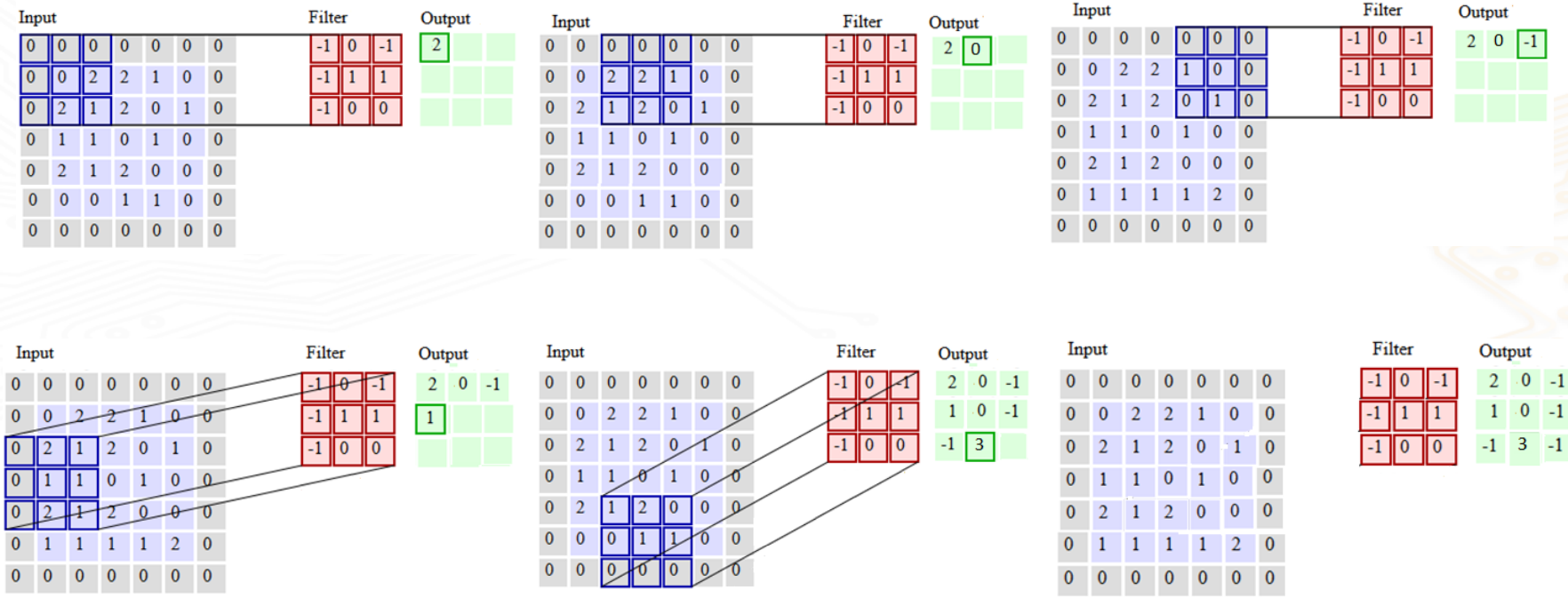
$$= (x1 \cdot h9) + (x2 \cdot h8) + (x3 \cdot h7) + (x4 \cdot h6) + (x5 \cdot h5) + (x6 \cdot h4) + (x7 \cdot h3) + (x8 \cdot h2) + (x9 \cdot h1)$$

Consists of Multiplication and Accumulation (MAC) vector operations

- adding each element x (pixel) of the input matrix (image),
- weighted by the respective elements in the matrix h (digital filter, kernel)

Image (2D) Convolution in CNN

Applying ‘convolutions’ to an image consisting of 7x7 pixels

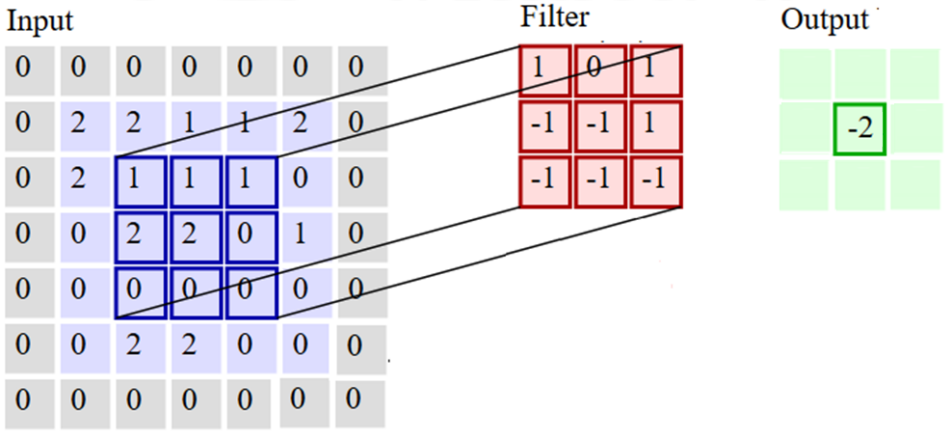


Convolution Process in CNN

'Convolution' is the process of adding each element of the image to its local neighbours, weighted by the filter (kernel) entries

- position the center of the kernel on each of the boundary points of the image, and compute a weighted sum
- to extract certain key features from the image

(Note: In CNN, we actually perform Cross-correlation)



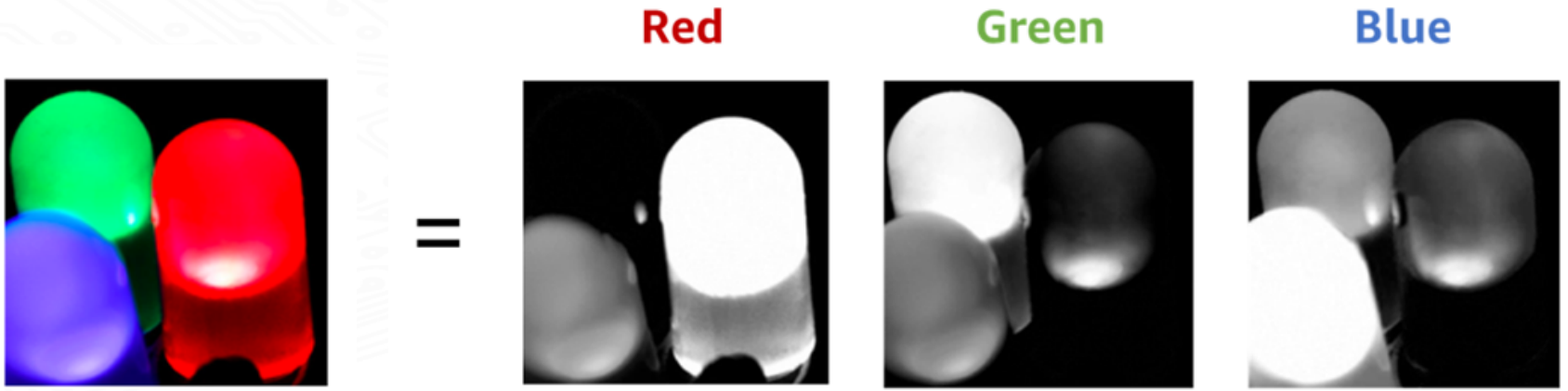
Kernel for detecting vertical edge

+1	0	-1
+1	0	-1
+1	0	-1

Kernel for detecting horizontal edge

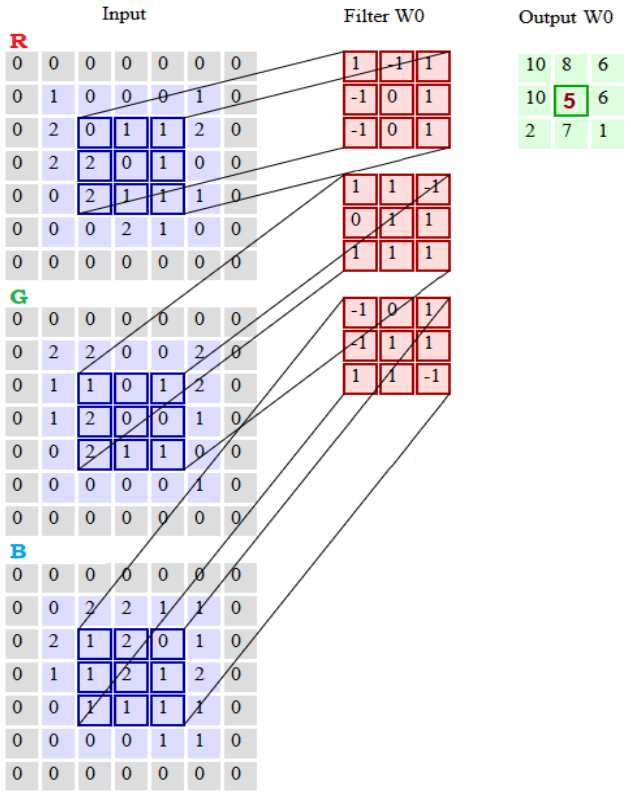
+1	+1	+1
0	0	0
-1	-1	-1

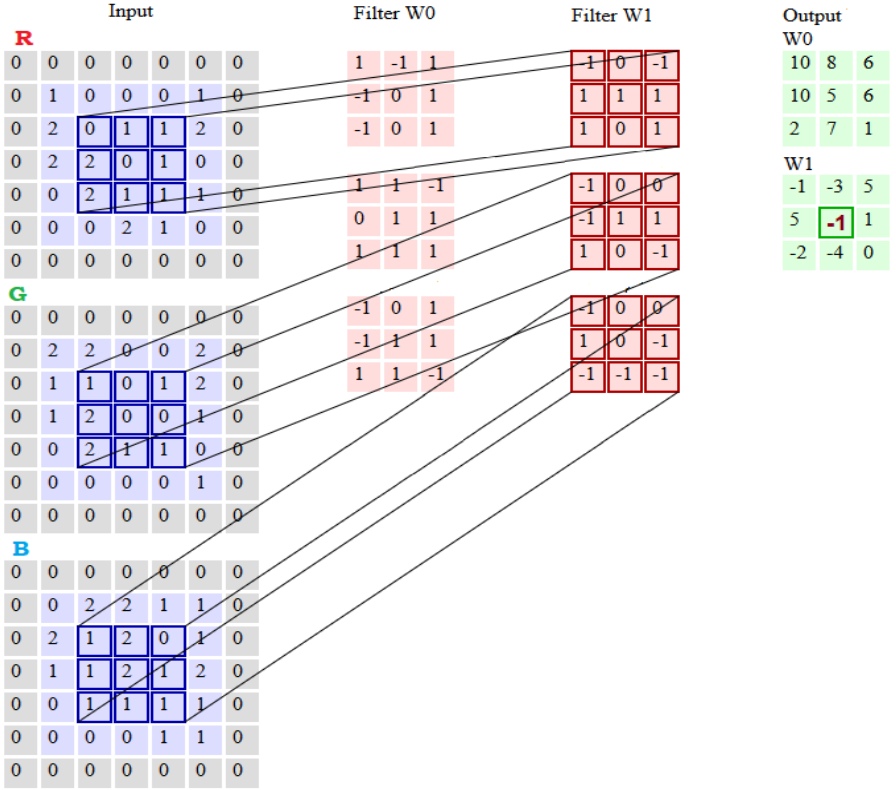
Colour Image



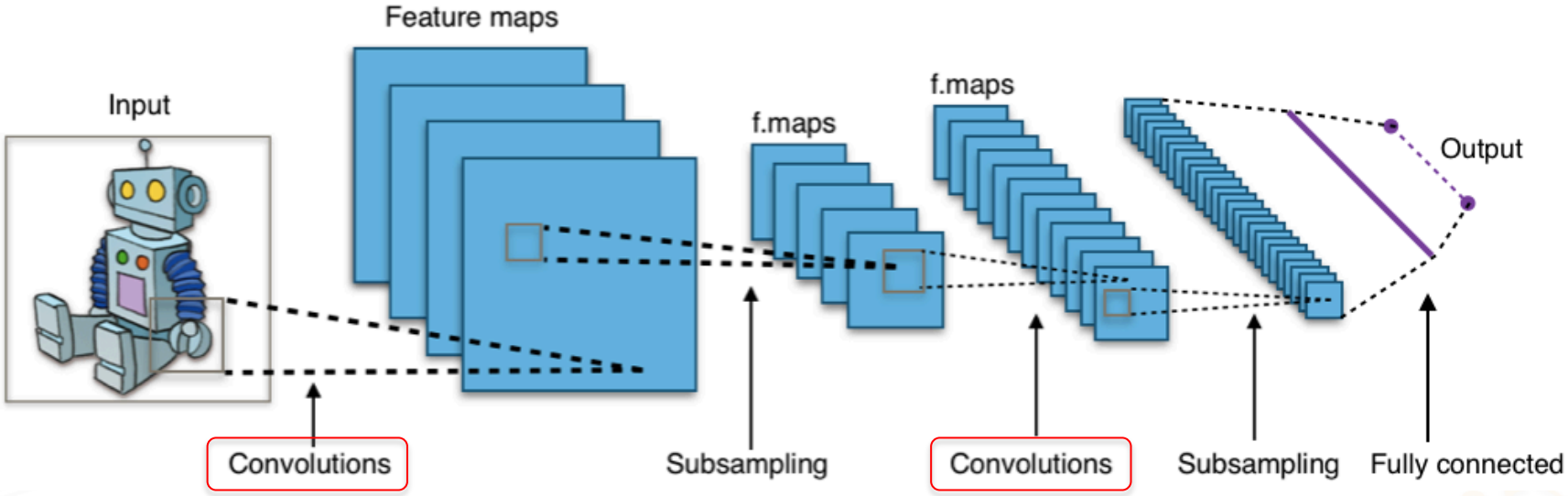
Source:
https://miro.medium.com/max/2441/1*oV0W0fzYBs3wqMajhC2nLQ.png

Convolution for Colour Image





CNN for Image Recognition



Source: Wikipedia

Matrix Operation in CNN

CNN involves massive amount of convolutions/correlations

- hence depends on efficient implementation of the matrix operations which mainly consists of multiplication and addition/accumulation (MAC) operations

Applying to independent sets of data

- can hence be performed in parallel (Data parallelism)

Will be very efficient if we have suitable processor architectures

- that support massive SIMD operations

Recap – Data Level Parallelism

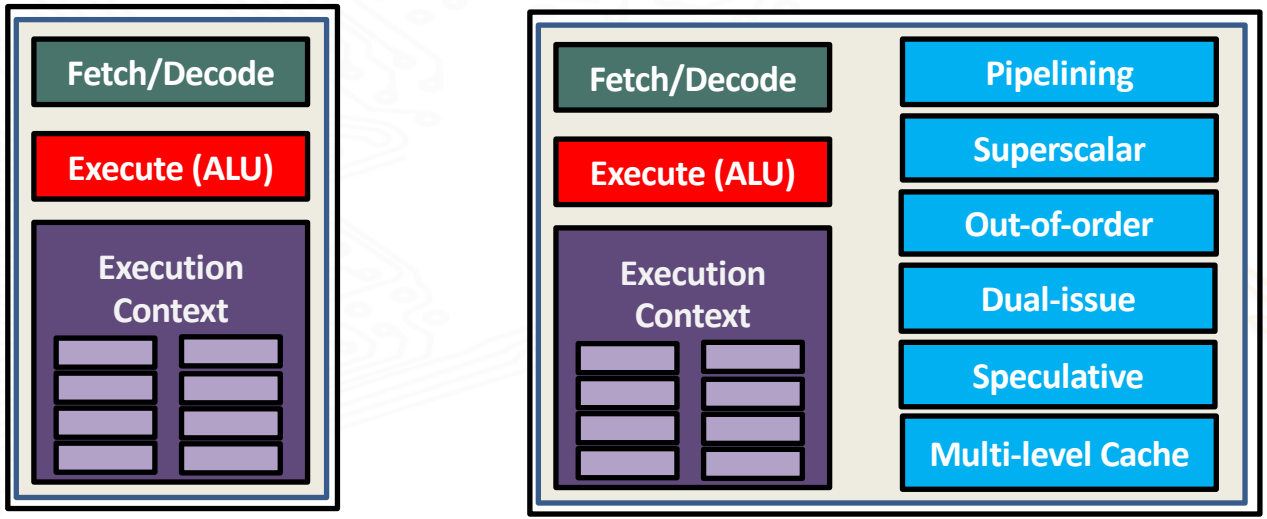
Same operation is performed on multiple data values

- can be executed concurrently with multiple processing units
- Single Instruction Multi Data - SIMD

SIMD execution

- Vector Processor
- Array Processor
- SSE, AVX for multimedia support on modern CPUs
- Graphic Processing Unit - GPU

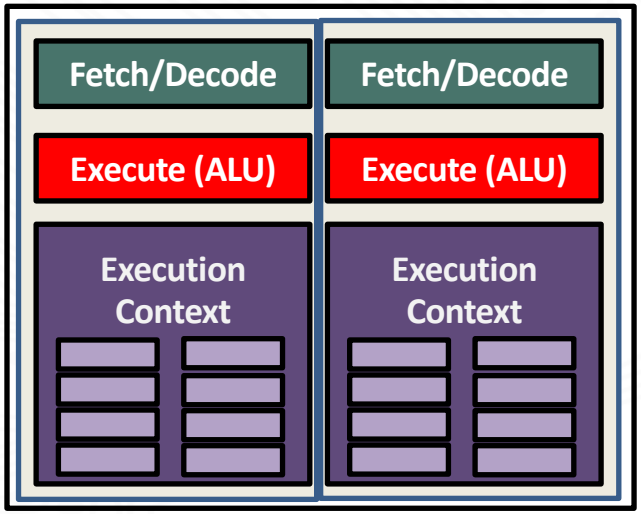
Basic CPU Architecture – Single Core

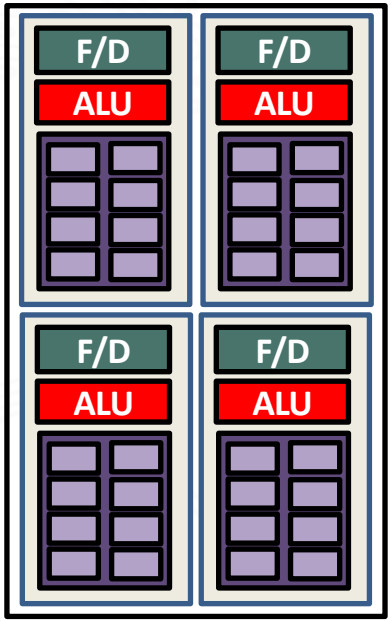


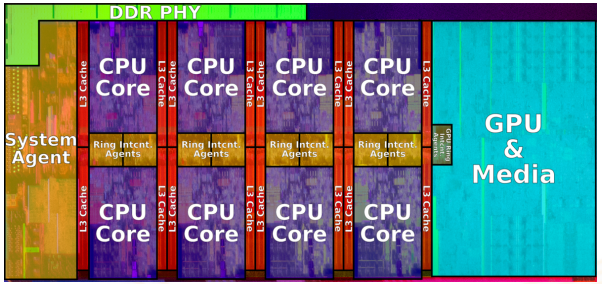
Designed for single-threaded code optimized for low latency

- by using various performance enhancement techniques
- through sophisticated processor logic circuitry

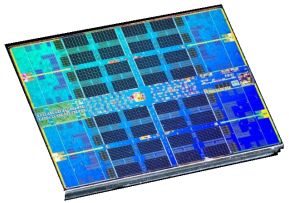
Multicore CPU Architecture







Intel Coffee Lake octa-core i9 processor (Source: wikichip)



AMD Epyc Rome 64-core processor (39.54 billion transistors)

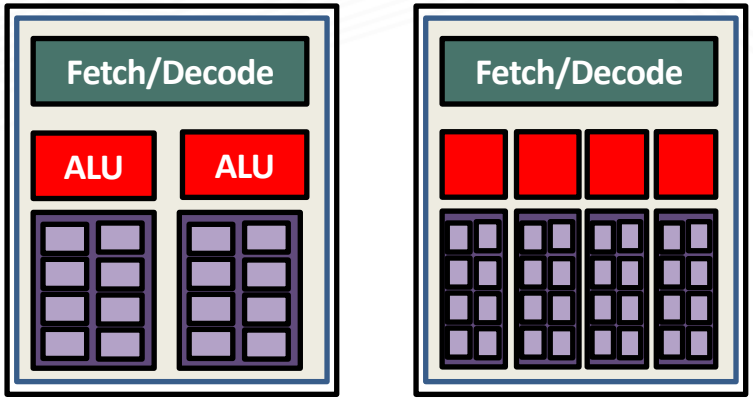
Adding more (and simpler) cores

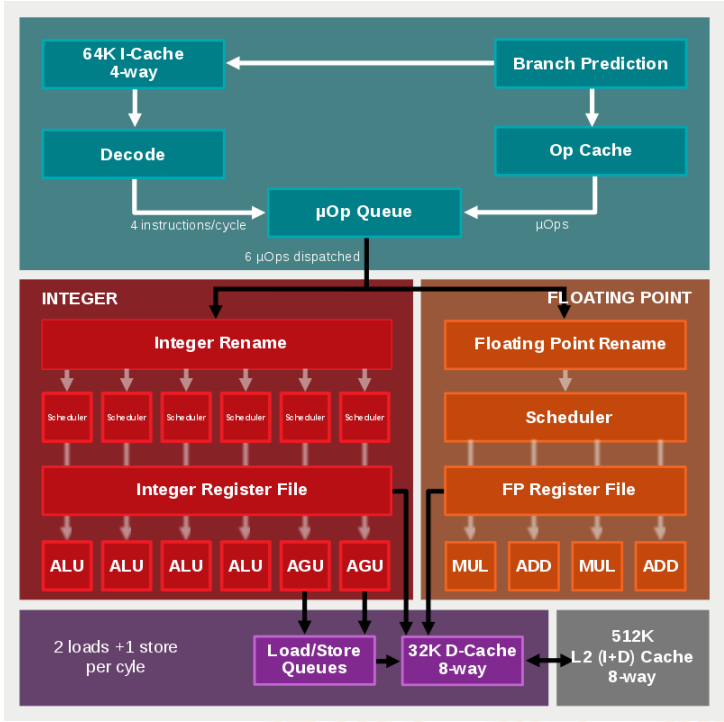
- able to execute multiple instructions in parallel
- effective if we can write efficient concurrent code (e.g. Pthread based multithreaded programming)

CPU Architecture with SIMD ALUs

To better support SIMD operations

- duplicate the ALU within each core
- all ALUs within each core must execute the same instruction simultaneously



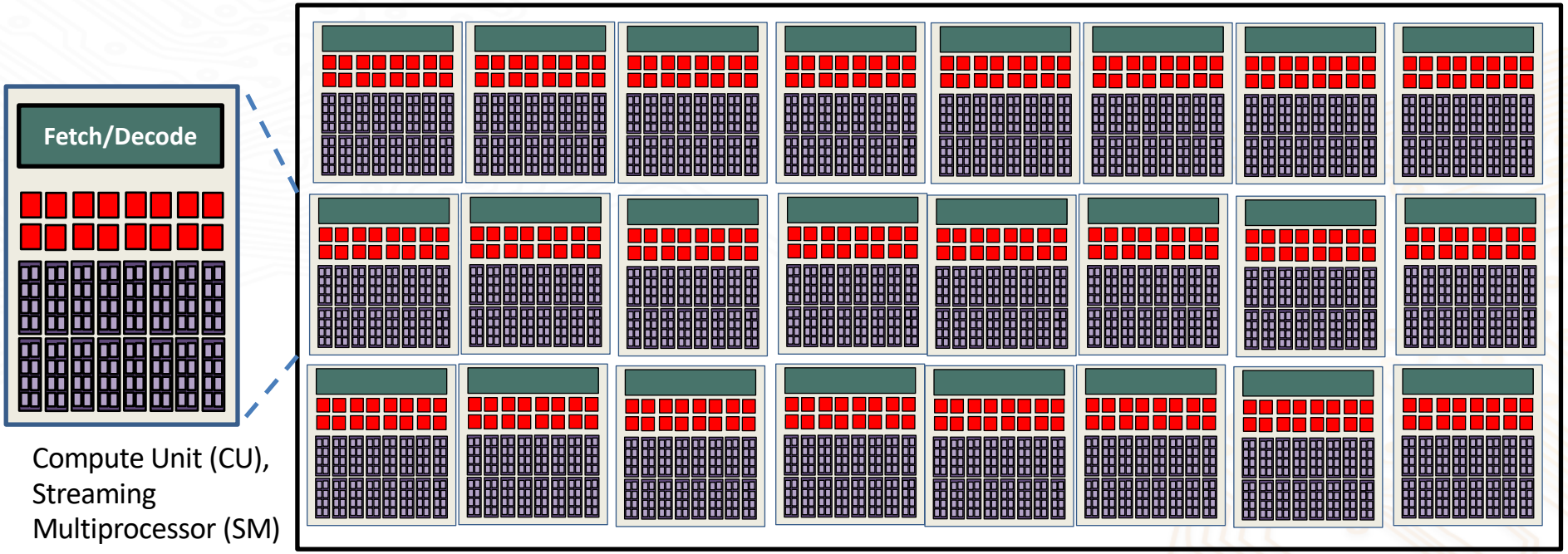


AMD Zen microarchitecture
(4 ALUs and 2 FPUs)

Basic GPU Architecture

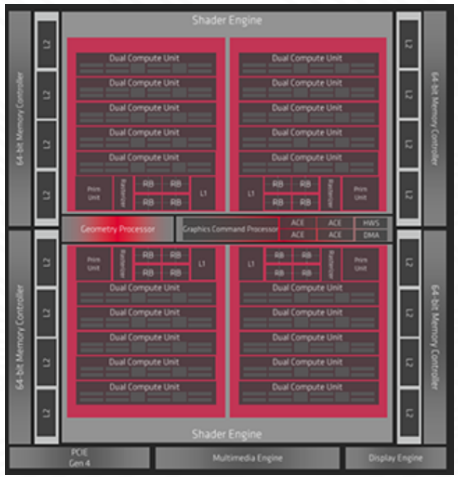
Consists of many-cores (SM, CU), each with many 'ALUs' (Streaming Processors - SP) to support SIMT (Single Instruction Multiple Threads)

- enable massive parallel MAC (floating points) operations



Compute Unit (CU), Streaming Multiprocessor (SM)

AMD Navi GPU



40 CUs, each with 64 streaming processors (SP cores)

- total of 2560 SP cores.

10.3 billion transistors.

NVIDIA Turing GPU

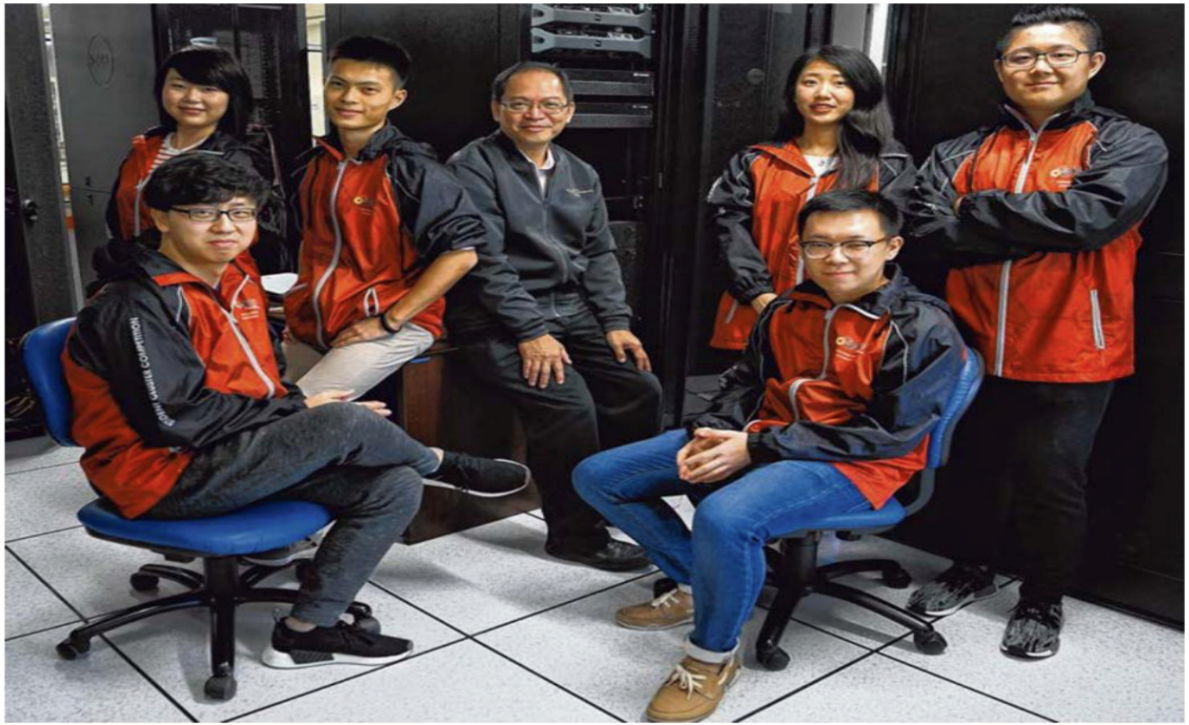
72 SMs, each with 64 streaming processors (CUDA cores)

- total of 4608 CUDA cores.

576 Tensor cores (for 4x4 Matrix operation.)

18.6 billion transistors.

| THE STRAITS TIMES | THURSDAY, NOVEMBER 23, 2017 |



Associate Professor Francis Lee (centre) with the NTU team that bagged the top prize, comprising undergraduates (clockwise from bottom left) Shao Yiyang, Hao Meiru, Chen Hailin, Tang Shuqian, team captain Liu Siyuan and Shi Ziji. ST PHOTO: MARK CHEONG

NTU takes top spot in US supercomputing challenge

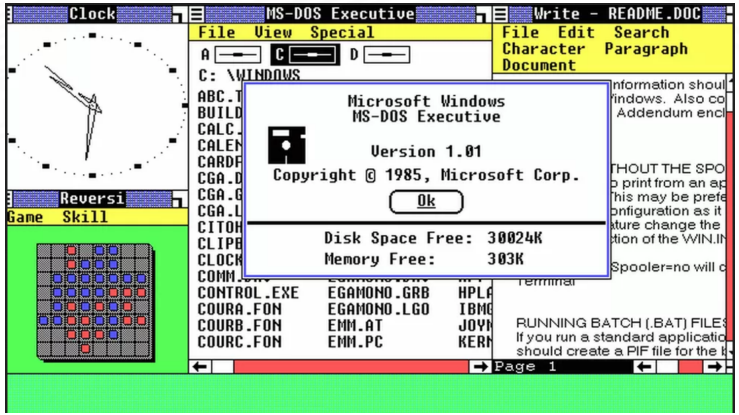
2x nodes.
16x Nvidia V100 GPUs.
2x Intel Xeon E5 2699 CPUs per node.

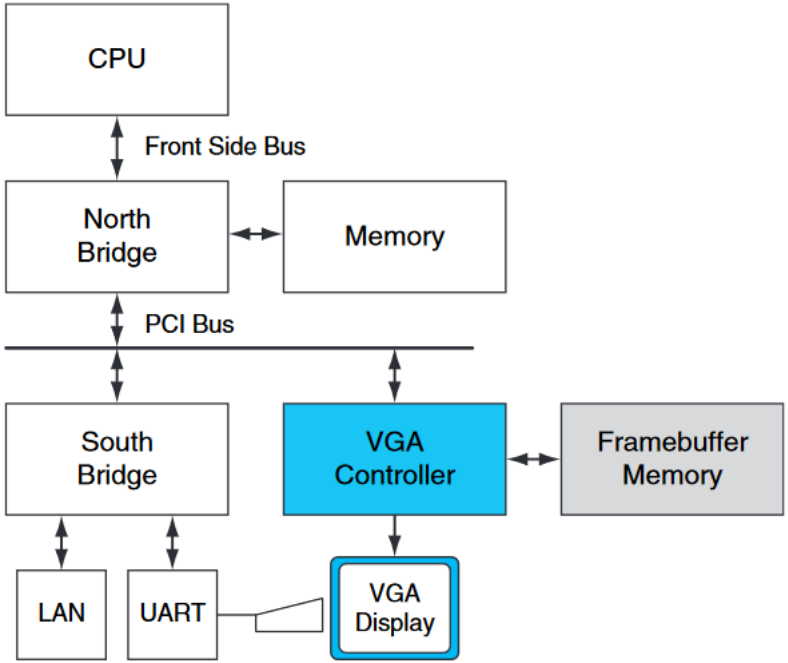
Set world record of Linpack (51.8 Tflops) at SC'17

Evolution of GPU Computing

Before the GPU (circa 1990)

- graphics display is based on a simple framebuffer subsystem standard known as the VGA (video graphics array)
- graphics are stored as bitmap based images in the framebuffer





(Source: Nickolls, J. Graphics and Computing GPU)

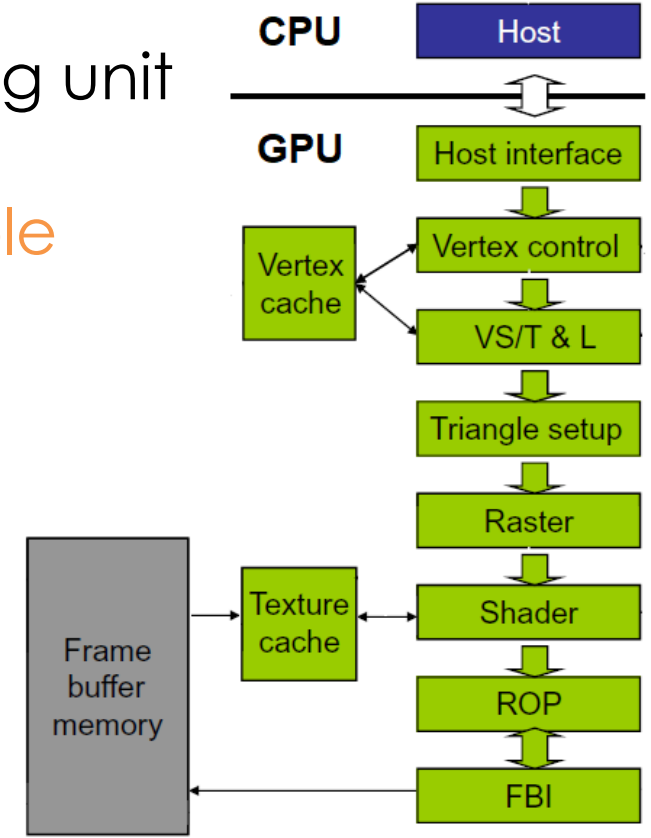
Fixed Function Pipelines

Original design of graphics processing unit

- based on fixed-function pipelines
- configurable but not programmable
- inflexible

Host interface receives graphics API commands and data from CPU

- e.g. through DMA transfer



(Source: Kirk, D. Programming Massively Parallel Processors)

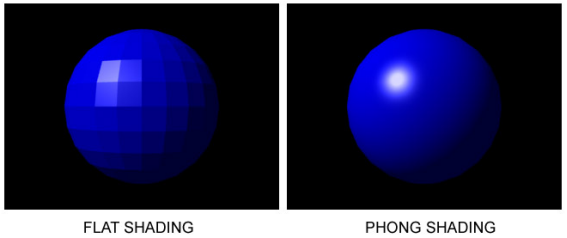
Programmable Graphics Pipeline

Certain functions executed at a few graphics pipeline stages vary with rendering algorithms

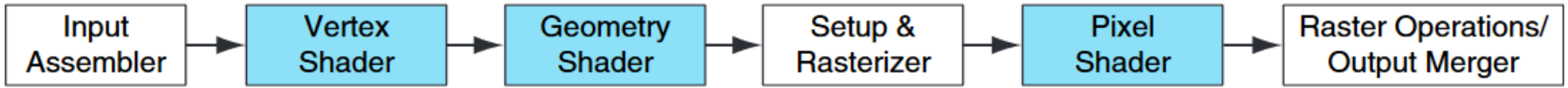
- motivated the hardware designers to make those pipeline stages programmable.

Example:

- vertex shader and pixel shader.



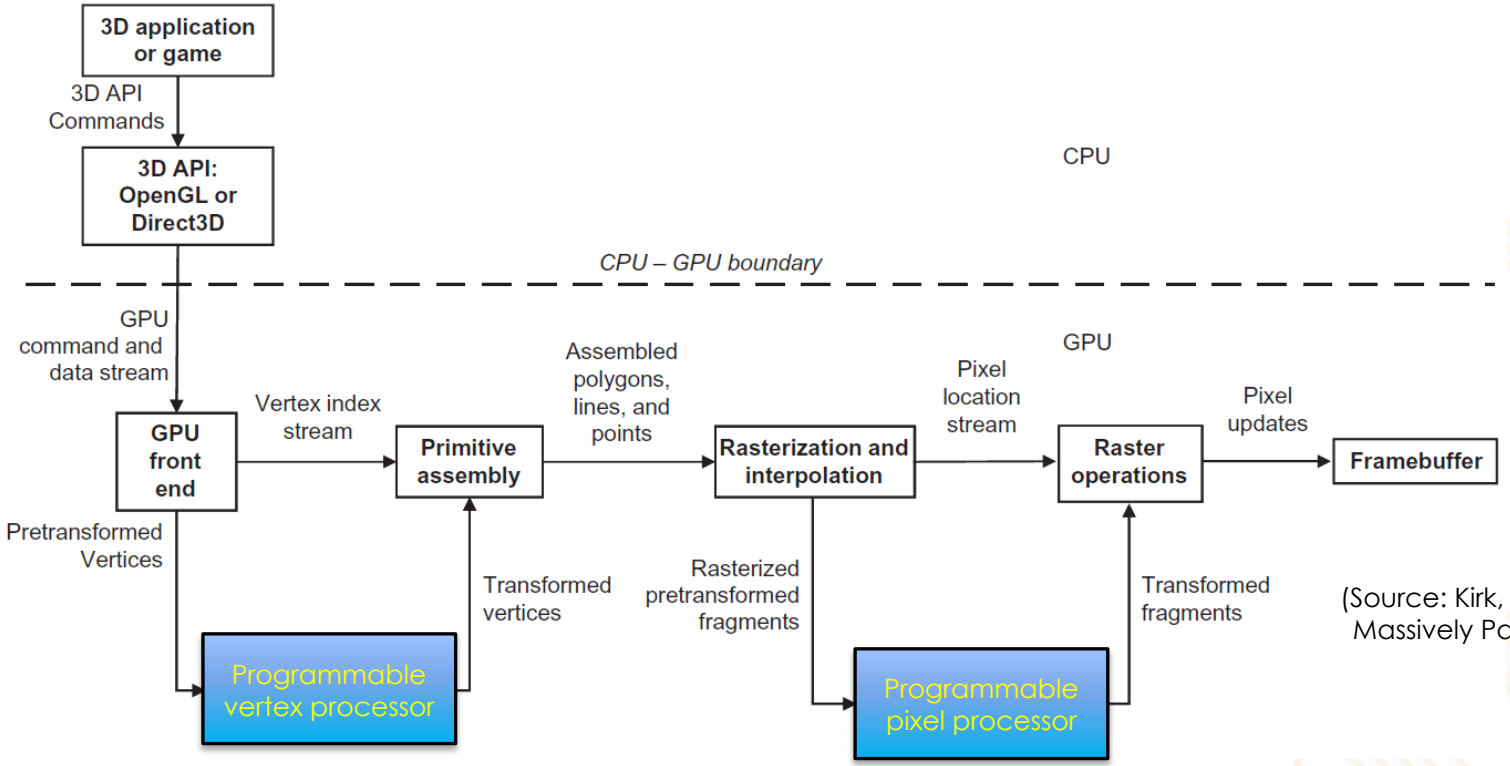
(Source: Wikipedia)



(Source: Nickolls, J. Graphics and Computing GPU)

Processors based GPU

Introduces programmable vertex and pixel processors

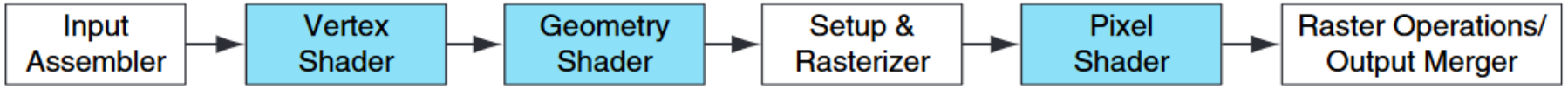


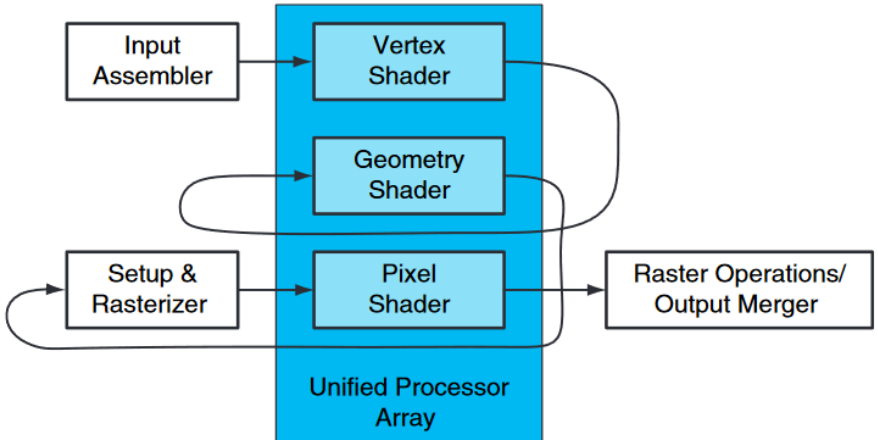
(Source: Kirk, D. Programming Massively Parallel Processors)

Unified GPU Architecture

Instead of separate processors for each processing type

- use the same processor core

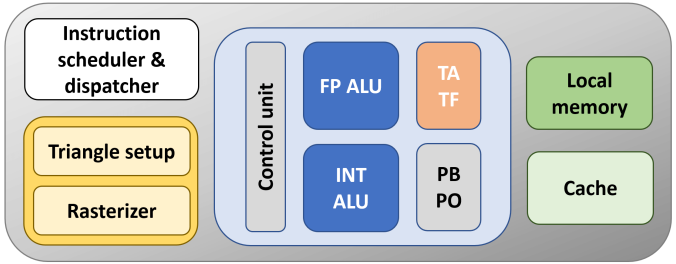




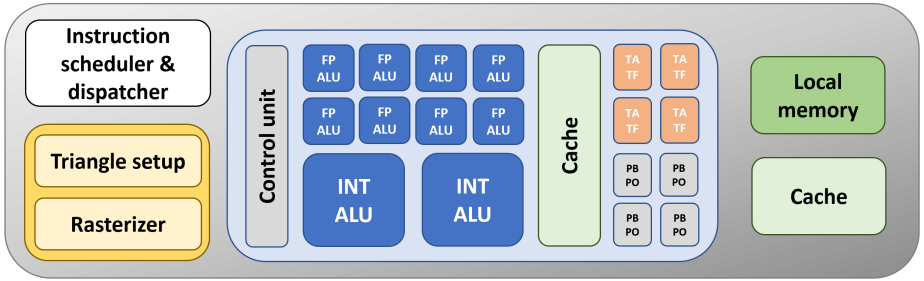
(Source: Nickolls, J. Graphics and Computing GPU)

GPU Processor Internal

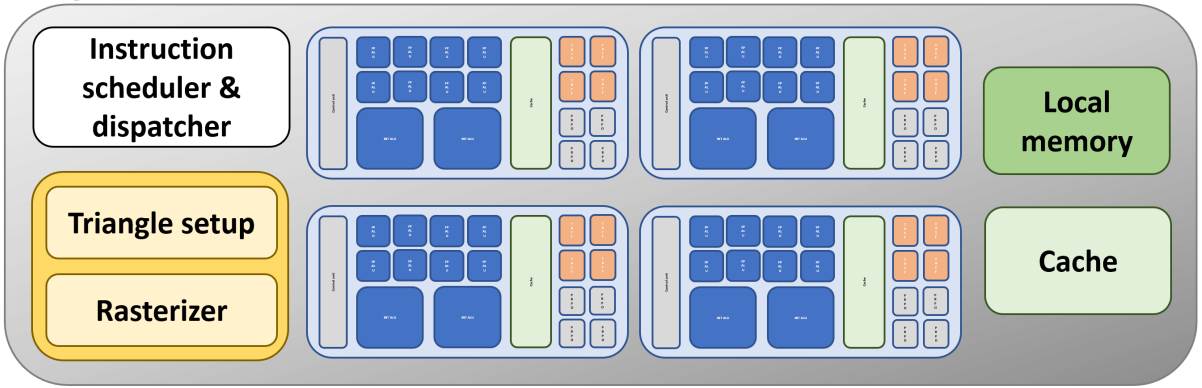
a) Basic GPU core



b) Add more ALUs



c) Add more independent blocks of ALUs



(Source: Evanson, Nick. Navi vs Turing: Architecture Comparison(online))

Example of a Unified GPU Architecture (NVIDIA's Tesla)

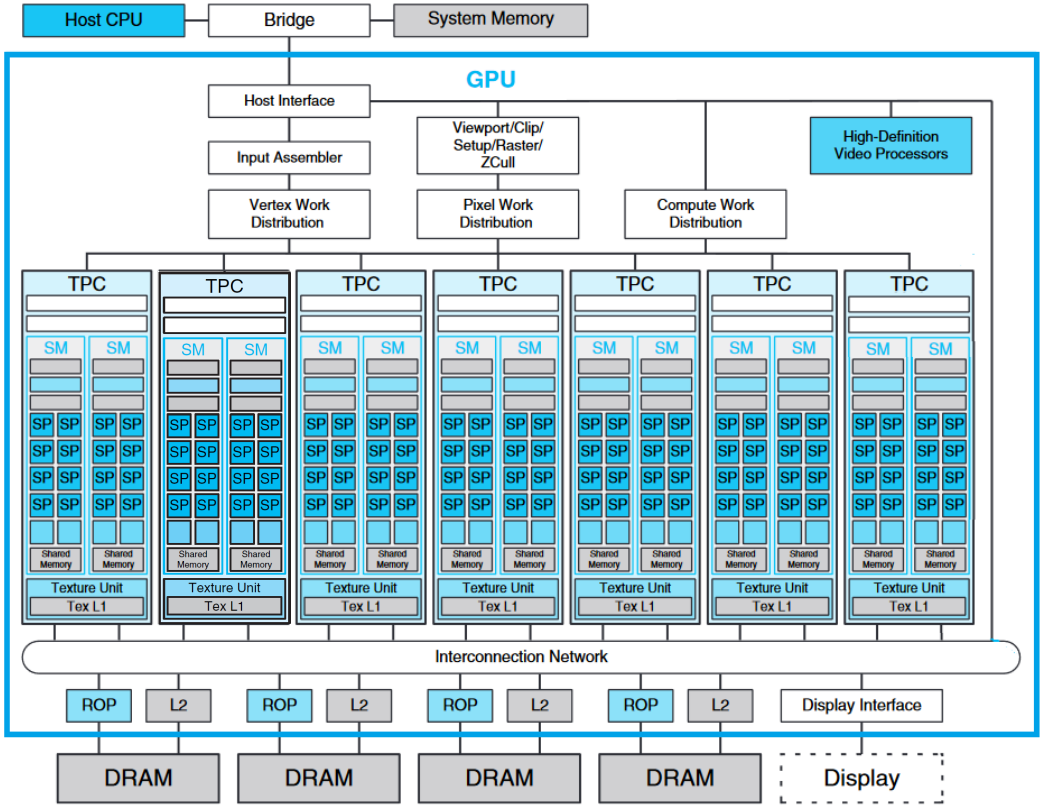
Consists of processor array in the form of

- streaming multiprocessors (SM)

Each SM contains multiple processor cores

- streaming processor (SP) cores
- each SM can manage many concurrent SIMD threads

Easily scalable to smaller or bigger GPU configuration



(Source: Nickolls, J. Graphics and Computing GPU)

Programming the GPU

For graphics rendering, graphics libraries are used to program the GPU through their APIs, such as

- Microsoft DirectX (Direct3D)
- OpenGL

As GPU hardware design evolves to include more unified processors

- increasingly resembled high-performance parallel computers

Researchers took notice of the raw performance of GPUs

- started to explore the use of GPUs to solve non-graphical compute-intensive science and engineering problems
  - E.g. protein folding, stock options pricing, SQL queries, and MRI reconstruction.

General-purpose Computing on GPU

But, to access the GPU's processing units

- have to cast the problem into graphics operations and launch through OpenGL or DirectX API calls.
- input data has to be stored in texture images and issue to the GPU (in form of triangles).
- output will be in the form of pixels generated from the raster operations.
- very restricted as there is no support for general read/write and data types used in typical program development.

This technique is known as GPGPU

- for "General-Purpose Computing on GPUs"

Parallel Programming on GPU

In 2007, NVIDIA released CUDA (Compute Unified Device Architecture)

- a software platform that gives direct access to (NVIDIA) GPU's internal processing units to support parallel programming

In 2009, an industry consortium jointly developed a standardized programming model, OpenCL (Open Computing Language)

- for developing parallel programs that can execute not only on GPU,
- but also across heterogeneous platforms consist of CPU, DSP, FPGA and hardware accelerators

Both can be coded based on C or C++ programming standards

- and is known as GPU Computing.

Summary

- GPU consists of many SIMD processing cores.
- Designed to support high throughput parallel processing such as for vectors and matrices based calculations.
- Adopted for solving scientific and engineering problems that display parallelism.
- Parallel programming standards such as CUDA and OpenCL enable direct access to internals of GPUs.