# GPGPU Computing and SIMD

Wei Wang

# From Single-processing to Multi-processing

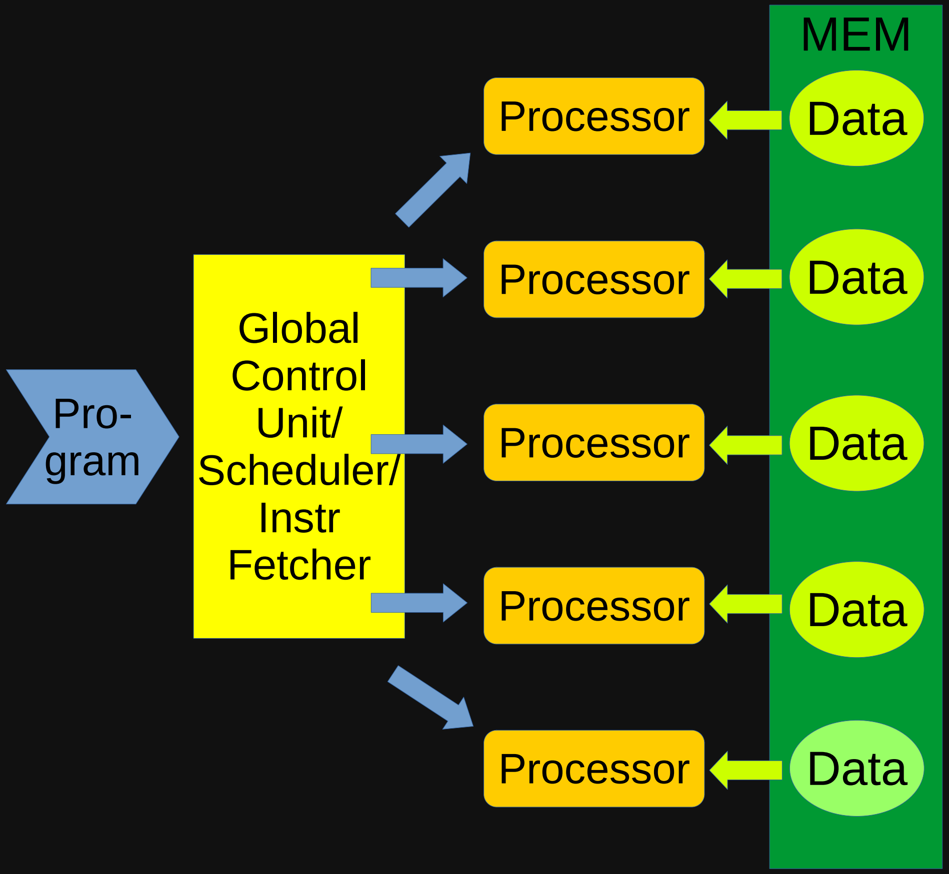
- Due to the failure of Dennard Scaling, today's CPUs are all multi-core processors.
- However, even before multi-core processors, a set of programs also called for multi-processing processors
  - These programs are graphics programs.
- Multi-processing processors usually have complete different architectural characteristics than single-processing processors.

# Control Structure of Parallel Platforms

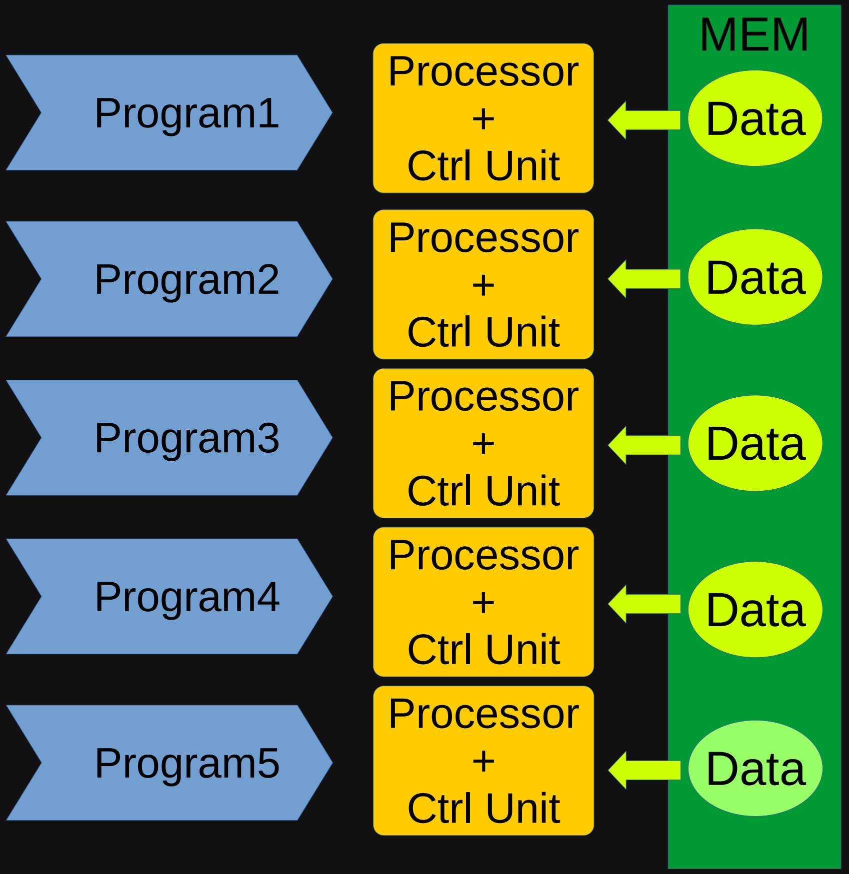
- Processor control structure alternatives
  - work independently
  - operate under the centralized control of a single control unit
- MIMD
  - Multiple Instruction streams
    - each processor has its own control unit
    - each processor can execute different instructions
  - Multiple Data streams
    - processors work on their own data
- SIMD
  - Single Instruction stream
    - single control unit dispatches the same instruction to processors
  - Multiple Data streams
    - processors work on their own data
- SIMT
  - Similar to SIMD, single instruction stream and multiple data streams
  - SIMT is an extension of SIMD that allows programming SIMD with threads

# SIMD and MIMD Processors

- SIMD




- MIMD



# SIMD Control

- SIMD excels for computations with regular structures
  - media processing, scientific kernels (e.g., linear algebra, FFT)
  - Image processing
  - Machine learning algorithms
  - These workloads are also parallel-friendly
- Most SIMD architectures forgo complex branch/control logics and cache/memory management, and dedicate all transistors to processing units
  - Allowing a large number of processing units on a single chip

# SIMD/SIMT Example: Nvidia Pascal Ampere P102 (2020)

- Each Streaming Multiprocessors (SM):
  - 64 FP32/INT32 Cores
    - INT32 cores support INT4, INT8 and INT32 operations
    - FP32 cores support FP32 and FP16 operations
  - 64 FP32 Cores
  - 2 FP64 cores (not in the figure)
  - 4 Tensor Cores
  - 1 RT (Ray Tracing) Core
  - 256KB Register File
  - 128KB L1 cache/Shared memory

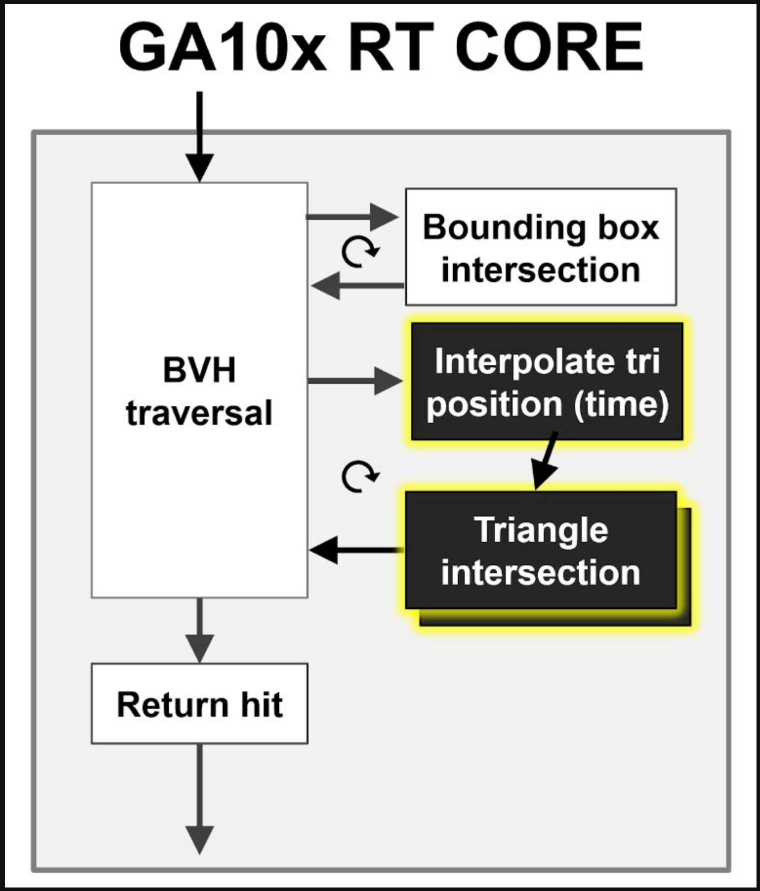
# SIMD/SIMT Example: Nvidia Pascal Ampere P102 (2020)

# SIMD/SIMT Example: Nvidia Pascal Ampere  P102 (2020)

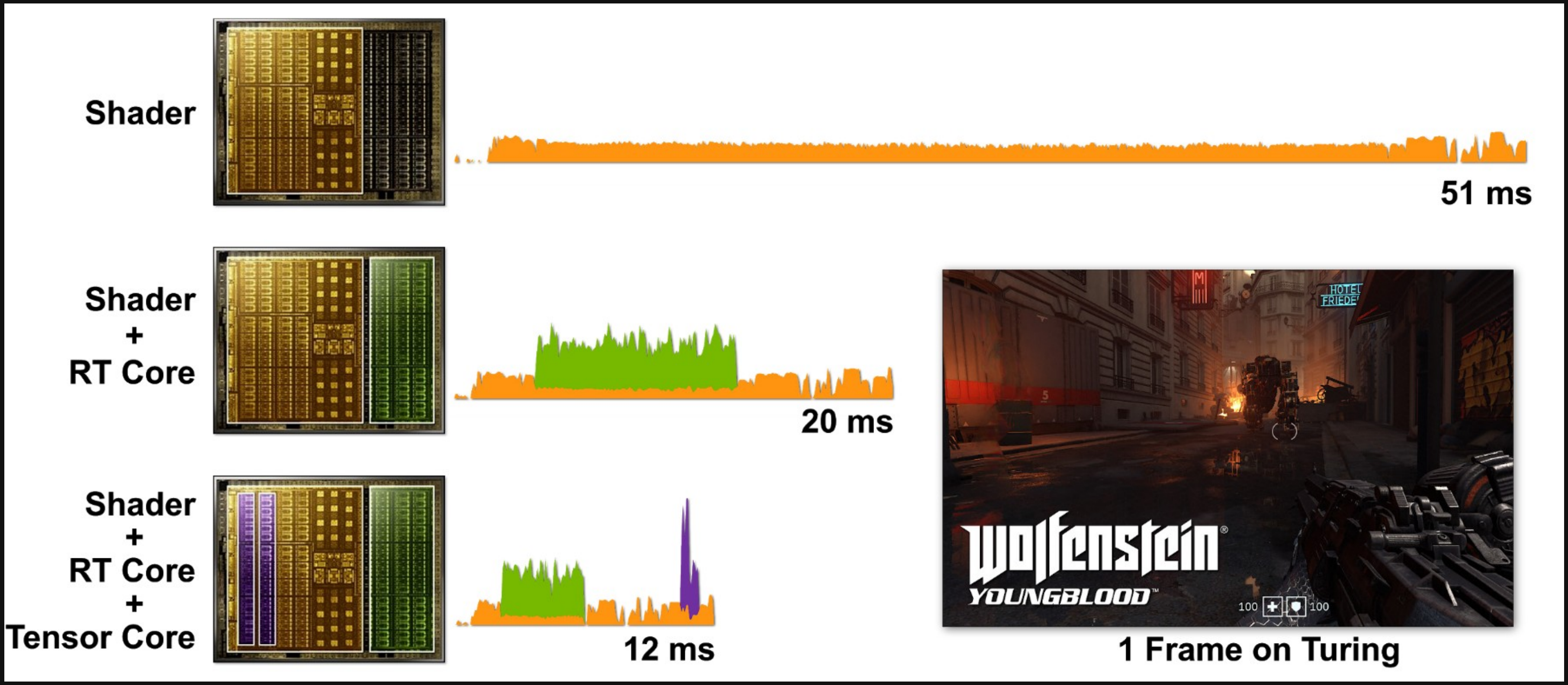
- Whole Chips
  - 7 GPCs (Graphics Processing Clusters)
  - 42 TPCs (texture Processing Clusters ), 84 SMs (two per TPC)
  - Peak FP32/16 TFLOPS (non tensor): 29.8
  - Peak FP16 TFLOPS (w. tensor): 119
  - Peak INT32 TFLOPS (non tensor): 14.9
  - Peak INT8 TFLOPS (w. tensor): 238
  - Memory bandwidth: 760GB/sec
  - size: 28.3 Billion transistors, 628.4 $mm^2$, 8nm process

# RT Core

- RT Core
  - ASIC for Ray Tracing
  - Quote from Nvidia:
    ” RT Core in GA10x includes dedicated hardware units for BVH traversal and ray-triangle intersection testing. Once the SM has cast the ray, the RT Core will perform all of the calculations needed for BVH traversal and triangle intersection tests, and will return a hit or no hit to the SM. ”

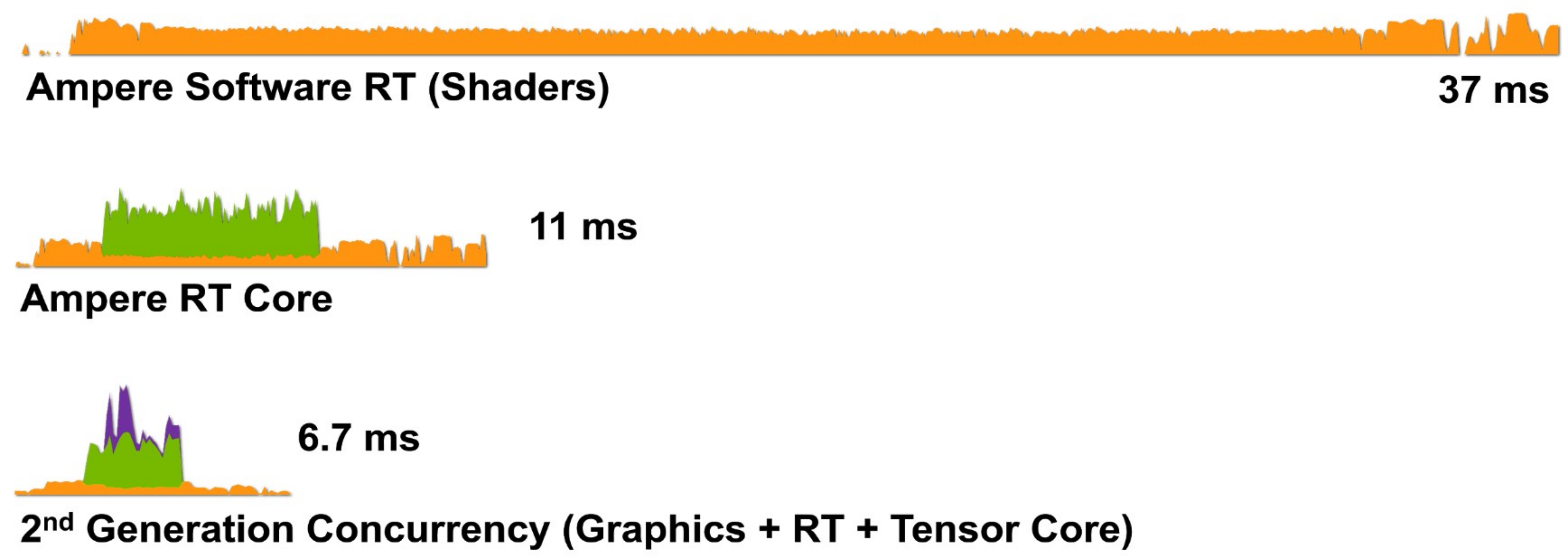


# RT Core




- Results for GTX 2080

# RT Core

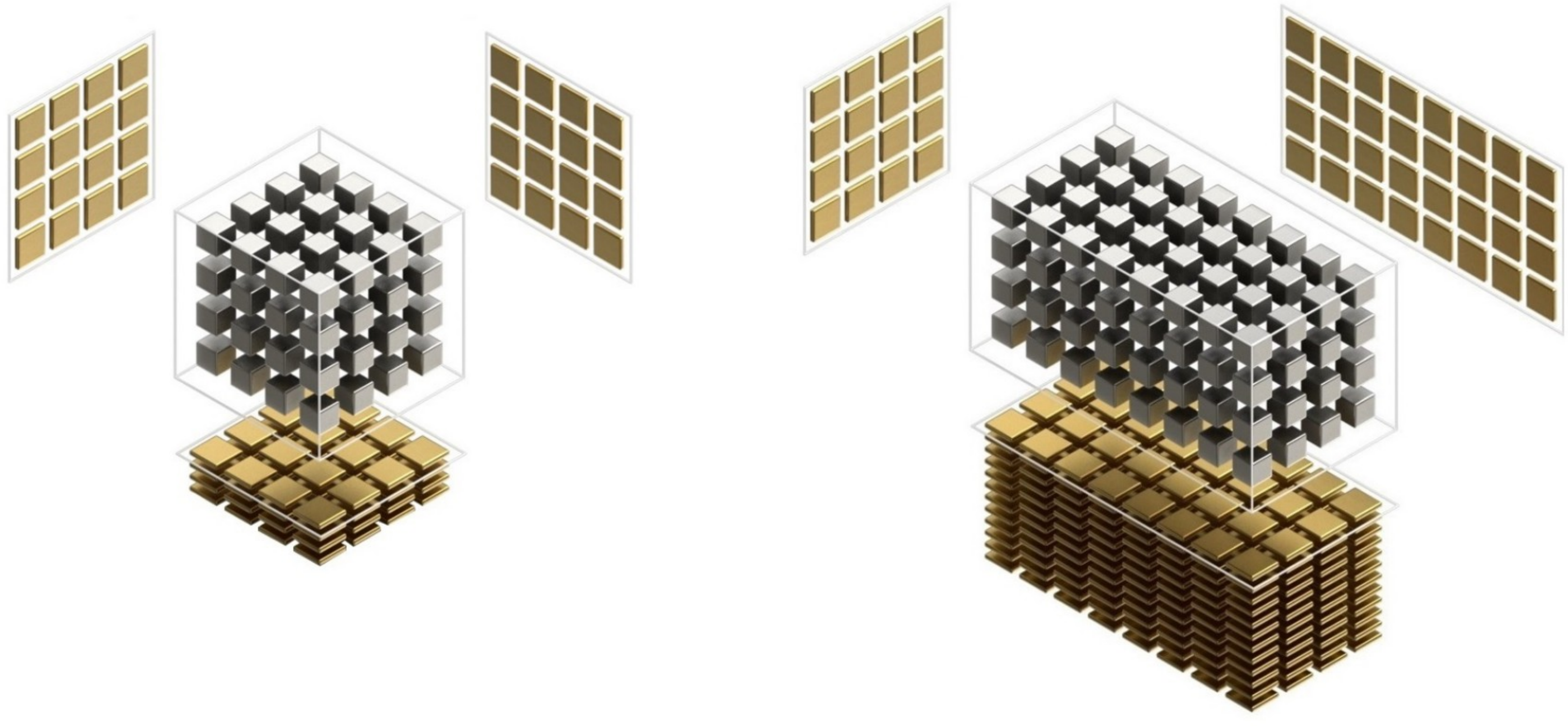



- Results for GTX 3080

# Tensor Cores

- A function unit for 8x4 to 4x4 matrix multiplication
  - Implements simplified GEMM: D= A*B+C
    - Where A, B, C and D are 4x4 matrices
    - A and B have to be FP16, C and D may be FP16 or FP32.
    - Perform FMA (fused mul-add) Operations
  - Gen 1 (2080) does 4x4 matrix multiplication
- Per core: 128 FMA Ops for Dense matrix and 256 Ops for sparse matrix
- Matrix multiplication is one of the most basic operation for machine-learning. Tensor cores were added to speedup deep learning.
- To use Tensor Cores, directly call CUDA SDK’s GEMM kernels.

# Tensor Cores



TURING ARCHITECTURE TENSOR CORE
(GeForce RTX 2080 Super)

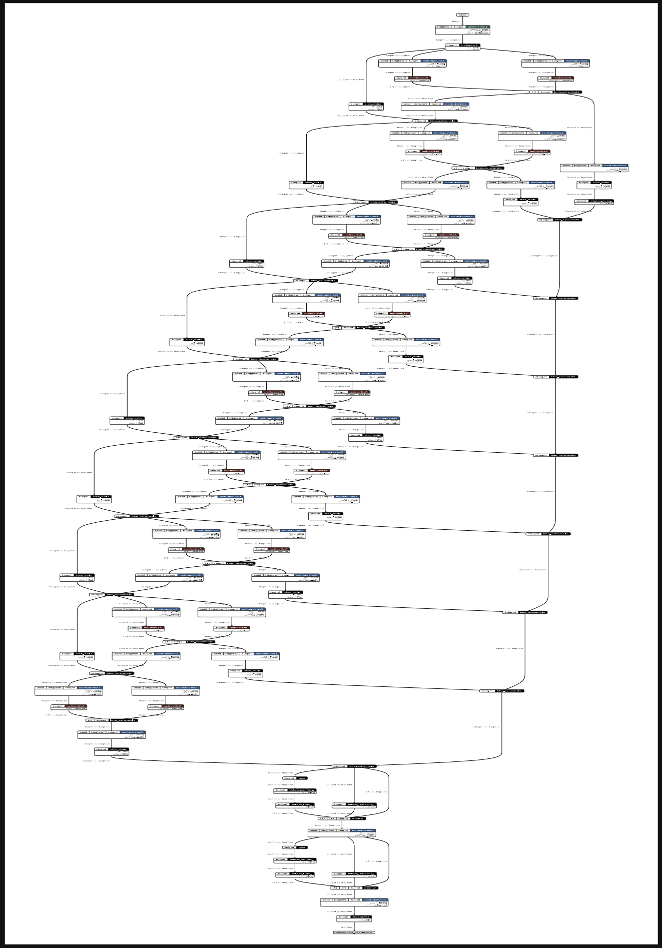
AMPERE ARCHITECTURE TENSOR CORE with Sparsity
(GeForce RTX 3080)

# TensorRT

- Seems to be a compiler-assisted auto-tuning optimization
  - DNN models are trimmed and transformed to use pre-optimized CUDA machine-learning kernels
    - With dropouts (?), lower precision and fused layers
    - Only for inference (work on trained models)
  - Compiler auto-tuning techniques are used to find the best transformation
- Integrated into TensorFlow and MXNet

# TensorRT Example

- WaveNet before TRT



- WaveNet after TRT

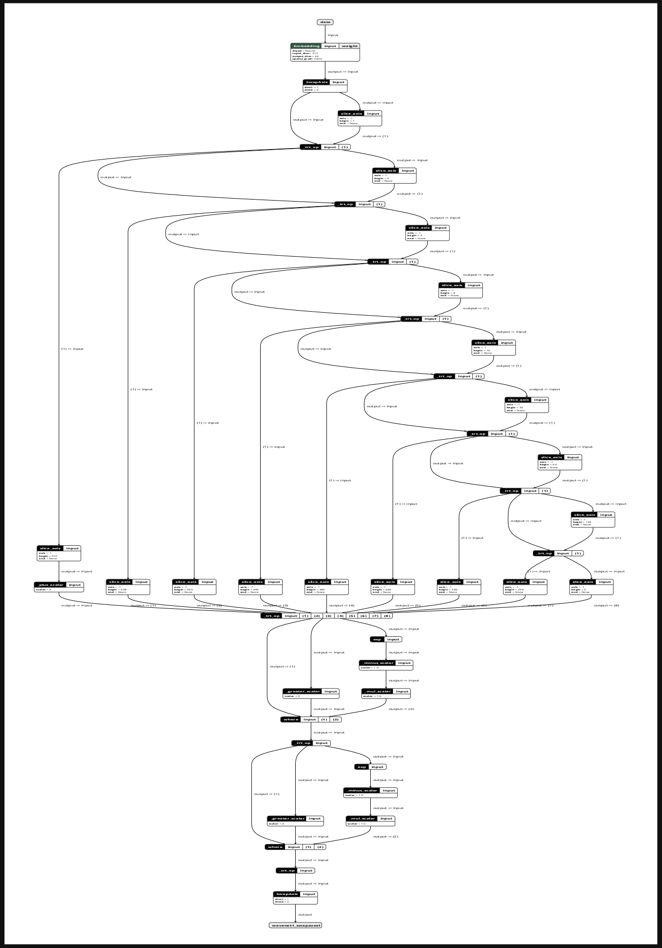
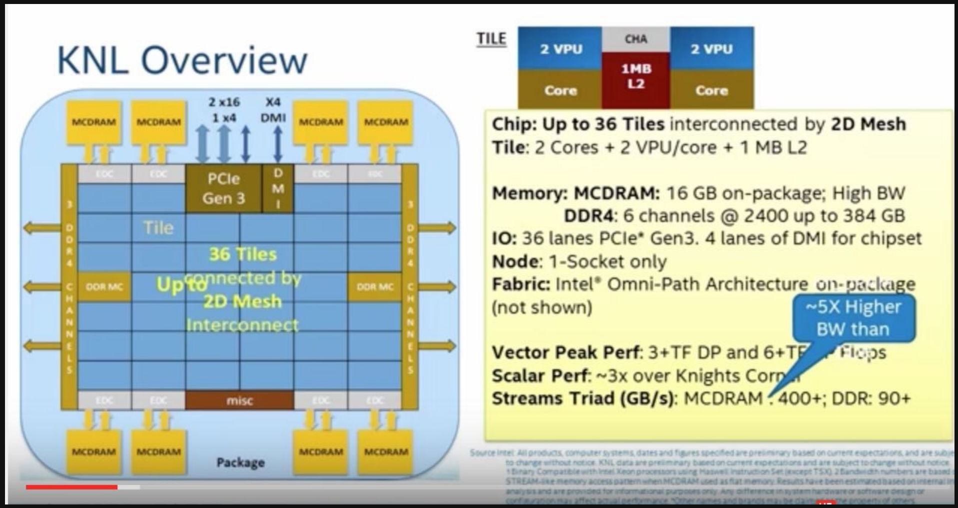


Image from MXNet Example, https://mxnet.incubator.apache.org/versions/master/tutorials/tensorrt/inference_with_trt.html

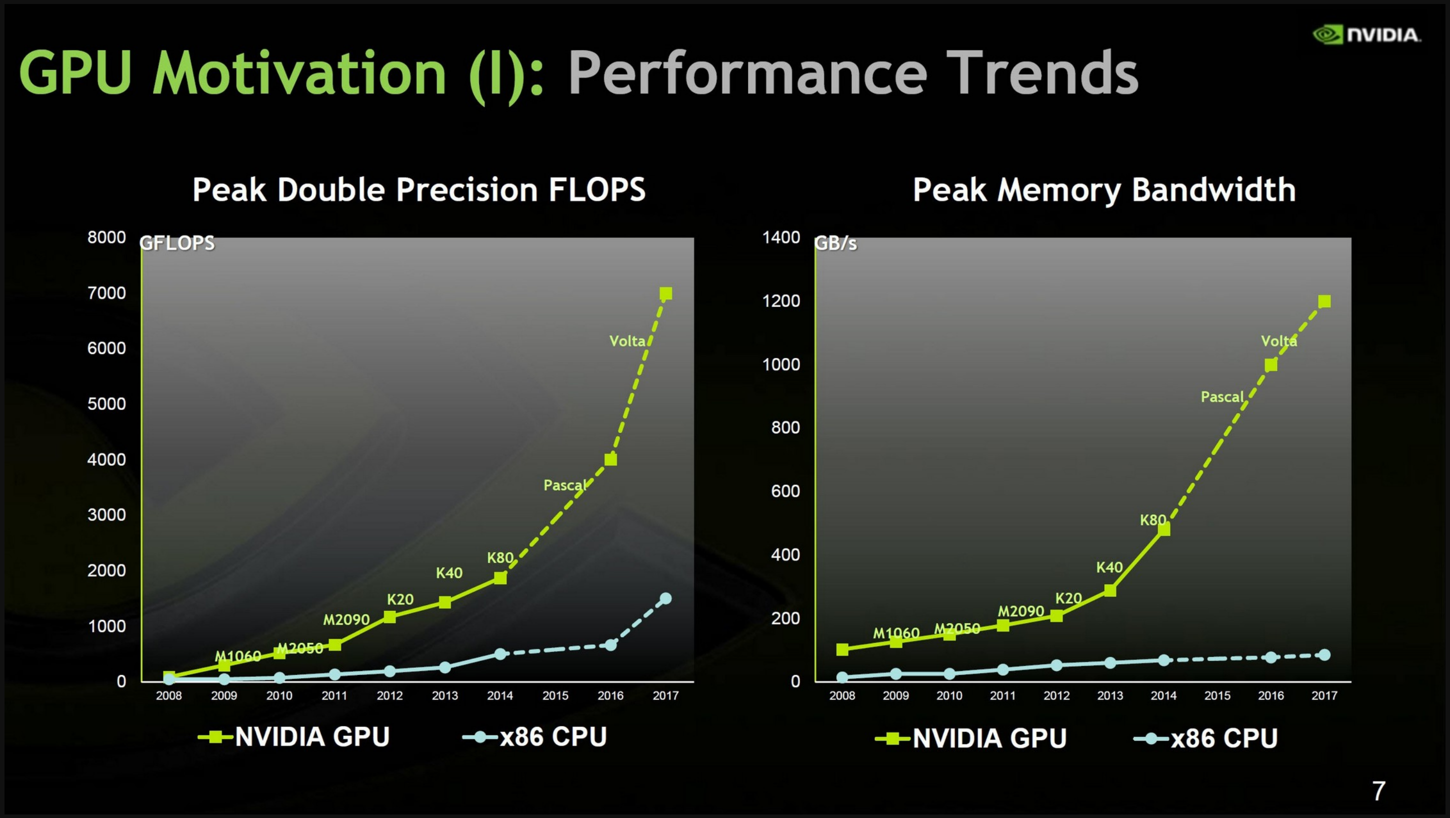
# SIMD Example: Intel Xeon Phi 7290 Knights Landing

- 72 cores
- Each core
  - Four SMT threads
  - 512-bit vector units
  - 32KB L1 cache
  - 1MB L2 cache
- Max GFLOPS: 3000



# Why SIMD?

- SIMD offers much higher theoretical peak performance over MIMD (CPU) per watt



# The Actual Difference Between CPU and GPU

- A 2010 Intel study suggests that GPU is only 2.5x faster than CPU on average
- A 2015 study shows that GPU is about 0 to 60x faster than CPU for several machine learning workloads
  - Note that the implementation is probably not optimized
  - These are the results of one GPU vs one CPU.

# CPU Core V.S. GPU Core

- For an Nvidia GPU, a SM core has
  - 64 32-bit floating point units (FPU)
  - 64 32-bit floating point / Integer units
  - Additionally, a few special functional units are located outside GPU cores
  - Newer versions of GPUs also add caches (but caches are small)
    - 16~128KB L1 cache per SM, 256KB~4MB L2 cache per chip.
- For an Intel Processor, a core typically has
  - 4 ALUs
  - 2 256-bit FPU
  - 4 256-bit Vector ALU
  - 2-4 LD/ST units, LEA units
  - Complex out-of-order execution management, branch prediction and memory disambiguation
  - Large and complex caches:
    - 64KB L1 cache per core; 256KB L2 cache per core; 1.5~2MB L3 cache per core

# CPU Core V.S. GPU Core cont'd

- For an Nvidia GPU, a core has
  - Designed and optimized for graphic processing;
  - most transistors are devoted to floating-point functional units.
  - Relatively higher Power consumption
- For an Intel Processor, a core typically has
  - Designed and optimized for general computing;
  - most of the transistors are devoted:
    - to find out-of-order execution opportunities
    - to dynamically schedule instructions.
    - and to caches
  - Relatively lower power-consumption

# CPU vs GPU Design Philosophy

- CPUs are designed for general purpose applications. These applications have
  - Low instruction-level parallelism (ILP)
    - So CPU has fewer ALUs/FPUs
    - CPU has complex control logic to extract all potential ILP
  - High data locality
    - So CPU has larger caches to exploit data locality

- GPUs are designed for graphics applications and applications with many SIMD operations. These applications have
  - High DLP
    - So GPU has simple control logic, but many ALUs/FPUs to exploit high DLP
  - Low data locality.
    - So GPU have smaller caches.
    - GPU also has many memory controllers/channels to improve DRAM performance.

# GPGPU Programming

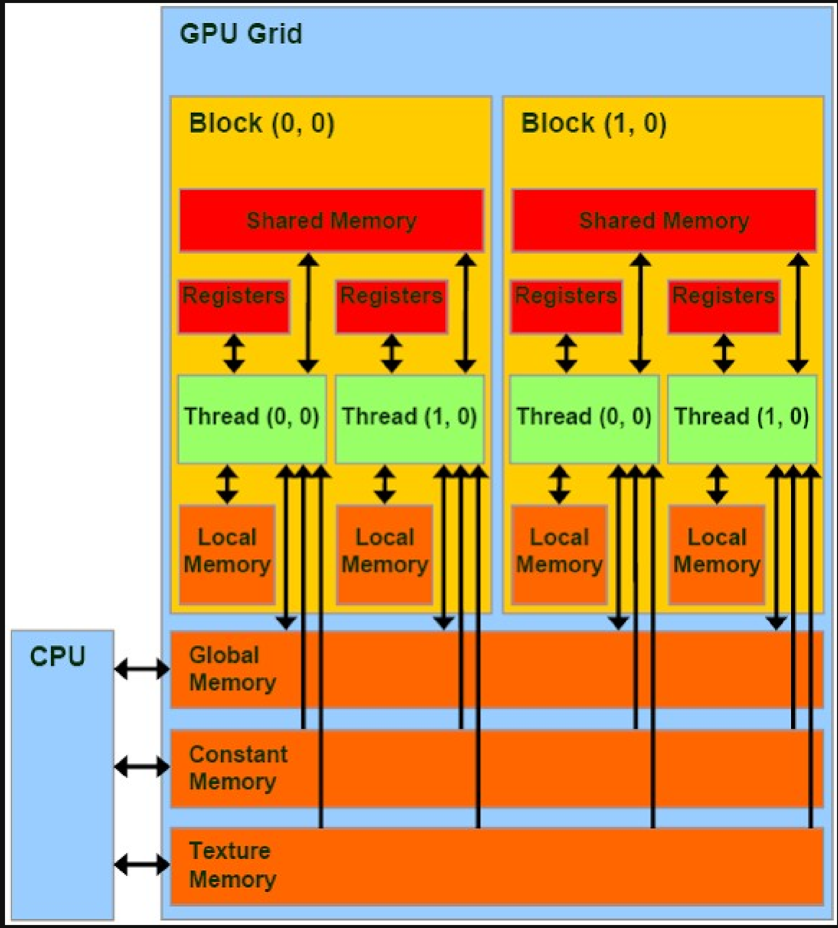
- GPGPU Programming: General-purpose computing on graphics processing units
- Motivation
  - Certain problems are similar to graphic applications in that they involve significant number of linear algebra operations and stream data processing
  - These problems also have limited data reuse and branches, similar to graphic applications
  - GPUs are faster than CPUs with these problems because the large number of processing units
- Therefore, It is both viable and beneficial to solve these problems on GPU

# Caching on GPU

- Traditionally, GPU does not have hardware managed caches
  - Graphic applications do not need hardware managed caches
  - Saved transistors are devoted to CUDA cores
  - There are software managed caches: shared memory, texture cache and constant cache
- New generations of GPU provides L1 and L2 caches
  - Motivated by GPGPU workloads
  - One L1 cache per SM, shared with shared memory or texture cache
  - One global L2 cache shared by all SMs

# A Historical View of Memory Structure of GPU

- Shared memory:
  - Practically a software managed L1 cache
- Local memory:
  - a storage for local variables that cannot be put in registers
  - Originally not cached, now cached through new L1 and L2 cache
  - Today local memory is mostly a concept than a real storage
- Global memory:
  - Main memory of a GPU
  - Originally not cached, now cached through new L1 and L2 cache
- Constant memory:
  - Used to stored constant data, read-only
  - Can be cached in constant cache
  - Incorporated into main memory and L1/L2 cache in newer GPUs
- Texture memory:
  - Used to store read-only data
  - Can be cached in texture cache
  - Incorporated into main memory and L1/L2 cache in newer GPUs



# Memory Structure of Current GPU (Nvidia Pascal)


# Shared Memory

- Used to store data that are shared by the cores within a SM processor
- Shared memory is the fundamental hardware for the communication among GPU cores
- Shared memory has limited size
  - For some GPUs, shared memory shares hardware with L1 cache.
- Shared memory data are declared with key word __shared__

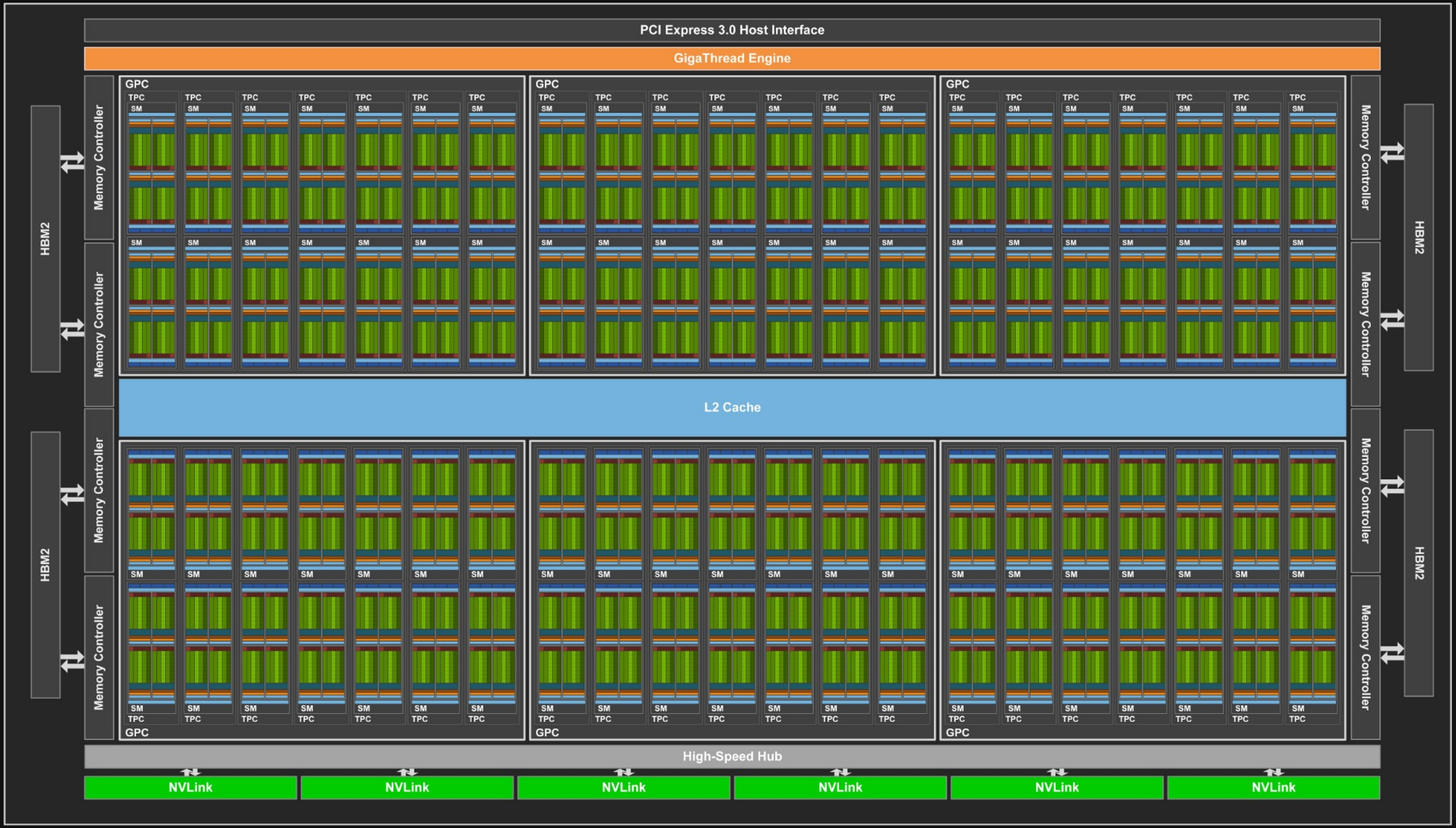
# Constant Memory

- Used to store read-only data
- Constant memory has limited size
- Constant memory data are cached in constant cache (now incorporated into L1/L2 caches)
  - Traditionally, most data are not cached, i.e., data are discarded after use
  - Reused data are declared as constant memory for fast reuse
- Constant memory data are declared with key word __constant__
- Although all data are cached now, GPU may still optimized read-only operations.
  - Therefore, it may still be beneficial  to use constant memory

# Texture Memory

- Used to store texture data for graphic applications
  - Read-only data
- Texture memory data are cached in texture cache (now incorporated into L1/L2 caches) for fast access
  - Unlike constant memory, texture memory data are expressed in 1D, 2D or 3D arrays to represent 2D/3D data locality
  - 1D/2D/3D Data are preloaded to texture cache to improve performance
- Texture memory data are declared with texture keyword, and need to be explicitly bound with data in main memory using function cudaBindTexture
- Although all data are cached now, GPU may still optimized texture-like memory reads (e.g., with prefetches)
  - Therefore, it may still be beneficial  to use texture memory

# NVidia Volta GV100



# NVidia Volta GV100

# NVidia Volta GV100 cont'd

- 84 SMs in 42 TPCs in 6 GPCs
  - TPC: Texture Processing Clusters
  - GPC: GPU Processing Clusters
- Each SM has
  - 64 FP32 cores
  - 64 INT32 cores (FP32 and INT32 can operate at the same time)
  - 32 FP64 cores
  - 8 Tensor cores
  - 4 Texture units
  - 256KB registers
  - 128KB L1 cache/shared memory
  - 4 execution blocks with 32-threads (a warp) issues each

# NVidia Volta GV100 cont’d

- A GPU has
  - 5376 FP32 cores,
  - 5376 INT32 cores,
  - 2688 FP64 cores,
  - 672 Tensor Cores,
  - and 336 texture units
  - 8 memory controllers with HBM2 (3D-stacked) memory and 4096 bits bus width
  - 6144KB L2 cache

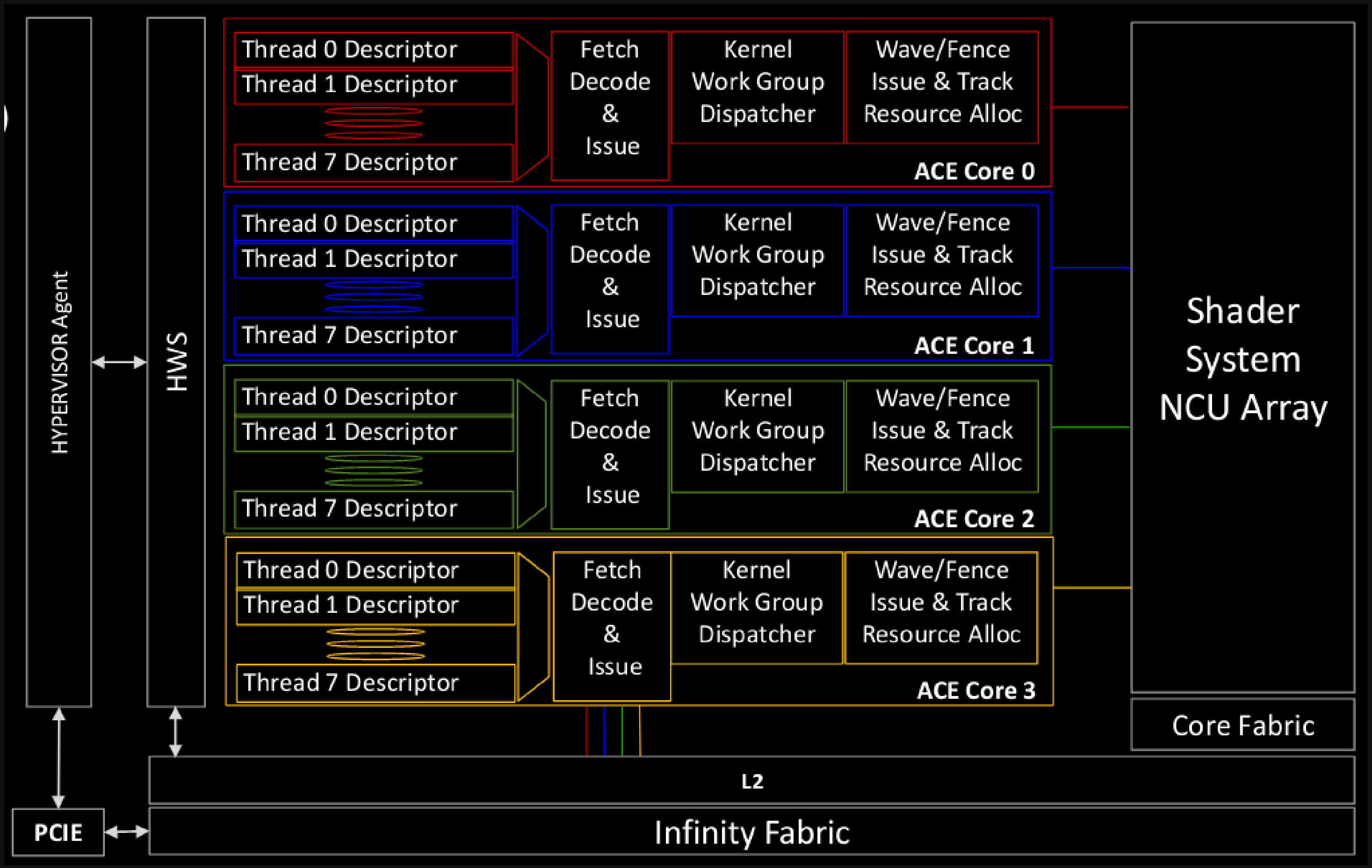
# NVidia Volta V100

- Note that, first Volta chip V100 has only 80 SMs
  - FP16: 30 TFLOPS
  - FP32: 15 TFLOPS
  - FT64: 7.5 TFLOPS
  - Tensor: 125 TFLOPS
  - Memory bandwidth: 900GB/sec
  - 300 watt TDP, 815mm$^2$ chip size and 21.2 bn transistors (TSMC 12nm)
  - 16GB memory

# AMD Vega 10

- Switched from VLIW to general SIMD design
- FP32 13.7 TFLOPS
- FP16/INT16 27.5 TFLOPS
- HBM2 memory up to 16GB, with 484GB/s
- 4MB L2 cache
- Four ACE (Accelerated Compute Engine) cores, each supports 8 Threads
  - All connected to a NCU (Next-Generation Compute Unit) array, which has a cluster of vector ALUs, Pixel units, Texture units and a L1 cache

# AMD Vega10 ACE

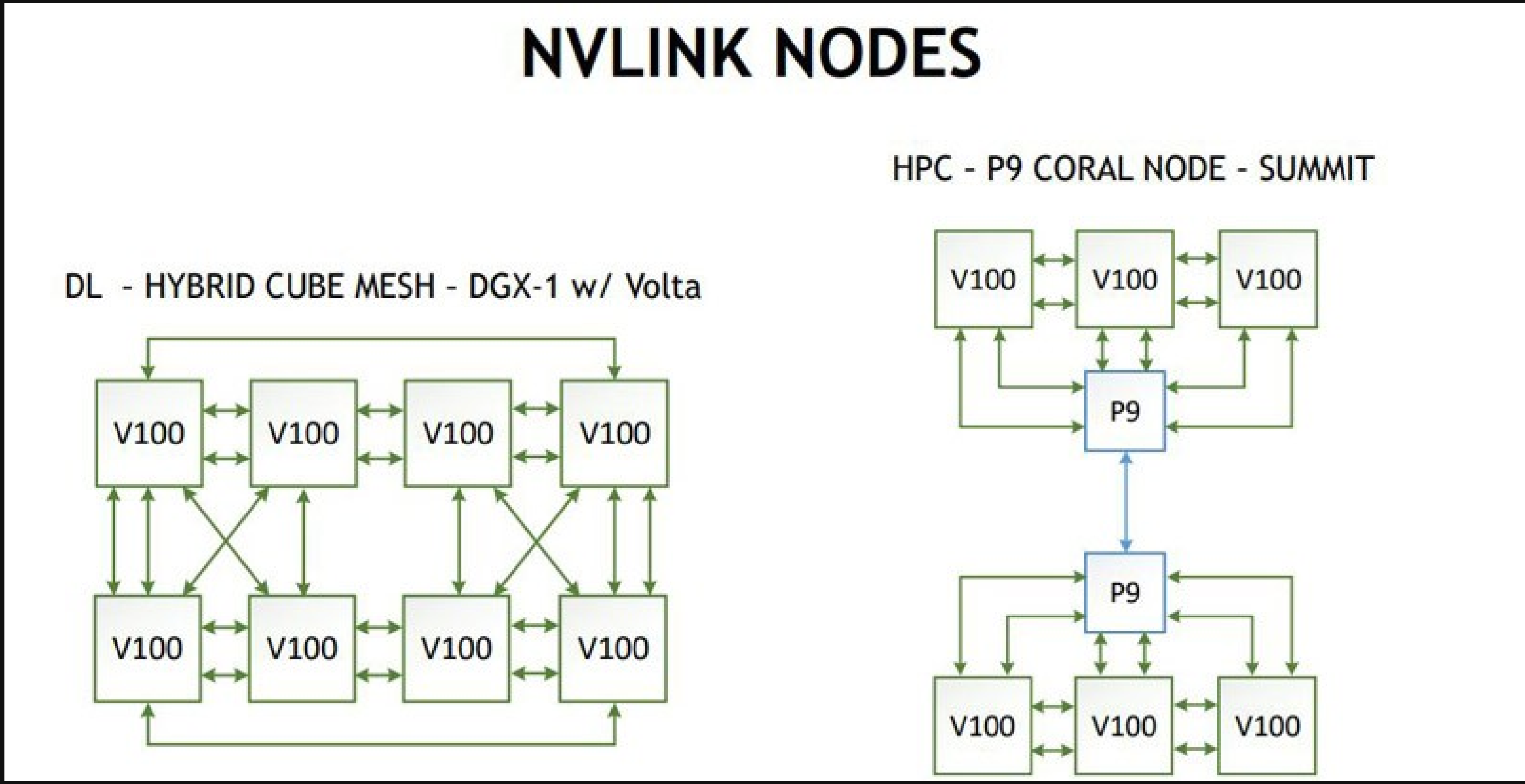


# Multi-GPU Support

- One main limitation for GPU in machine-learning is memory size
  - Current GPUs have 16GB at maximum
- Multiple GPUs are required if the data set or model is large
- Either through
  - Parallelization framework on clusters
  - Directly connected GPUs

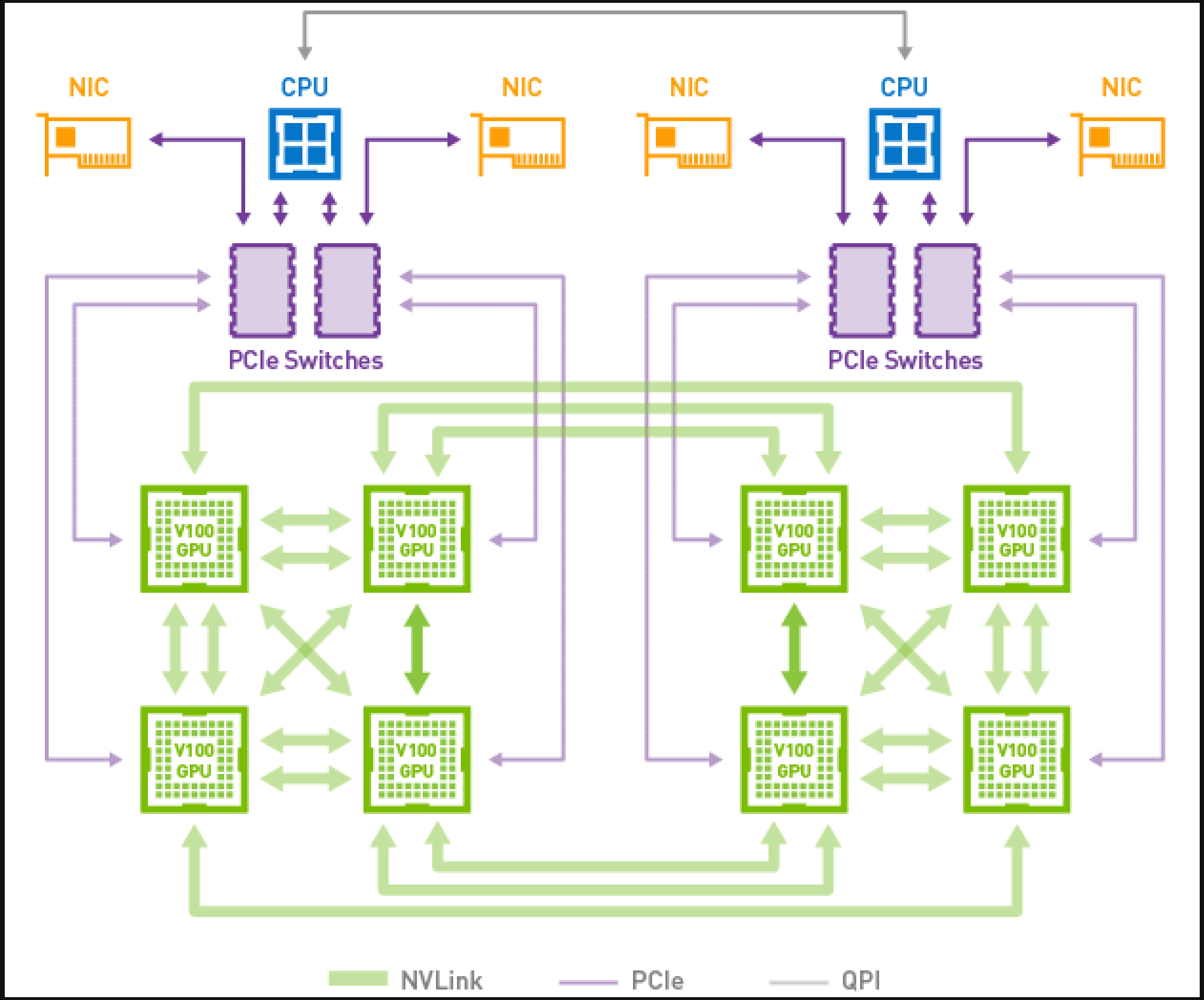
# Nvidia Multi-GPU Support

- GPUs are connected using NVLink connection, forming a network similar to NUMA systems.



# Nvidia Multi-GPU Support

- Another view



# GPU Virtualization

- GPUs for deep learning are too expensive for ordinary users
- Cloud-based GPU solutions are inevitable
  - Either through containers or full VMs.
- Both NVidia and AMD start to support GPU virtualization
  - For Nvidia, most Tesla and some Quadro cards support VGPU
    - NVidia also has a GRID product line that is discontinued
  - For AMD, VEGA is supposed to support VGPU with SR-IOV. Some Fire Pro and Radeon Pro cards also supports VGPU.
- Extra logics are required to support context switches