Master Informatics Eng.

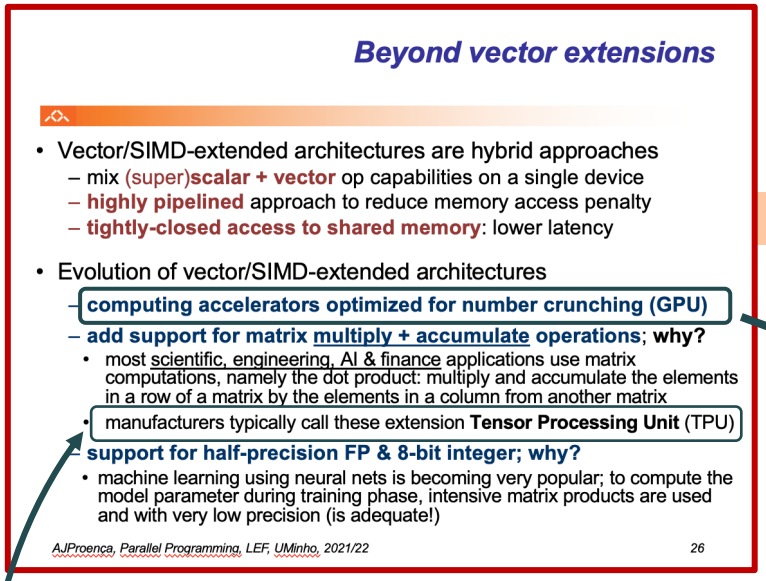
2022/23

A.J.Proença

Computing accelerators: GPU & CUDA

(most slides are borrowed)

Compute accelerators



Beyond vector extensions

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of vector/SIMD-extended architectures
  - **computing accelerators optimized for number crunching (GPU)**
  - **add support for matrix multiply + accumulate operations; why?**
    - most scientific, engineering, AI & finance applications use matrix computations, namely the dot product: multiply and accumulate the elements in a row of a matrix by the elements in a column from another matrix
    - manufacturers typically call these extension **Tensor Processing Unit** (TPU)
  - **support for half-precision FP & 8-bit integer; why?**
    - machine learning using neural nets is becoming very popular; to compute the model parameter during training phase, intensive matrix products are used and with very low precision (is adequate!)

AJProença, Parallel Programming, LEF, UMinho, 2021/22 26

Best accelerator for number crunching, namely intensive vector/matrix computing: **GPU**

Other common compute accelerators:

- DSP: Digital Signal Processor, mostly used in telecommunication equipments, from cell phones to radio systems and TVs
- TPU: Tensor Processing Units, optimized for operations with tensors (vector and n-dimensional matrices), popular in AI app's, namely in autonomous driving
- FPGA: Field Programmable Gate Arrays, reconfigurable h/w; can be configured in runtime to behave according to a given specification

Data Parallelism: SIMD CPU vs. GPU

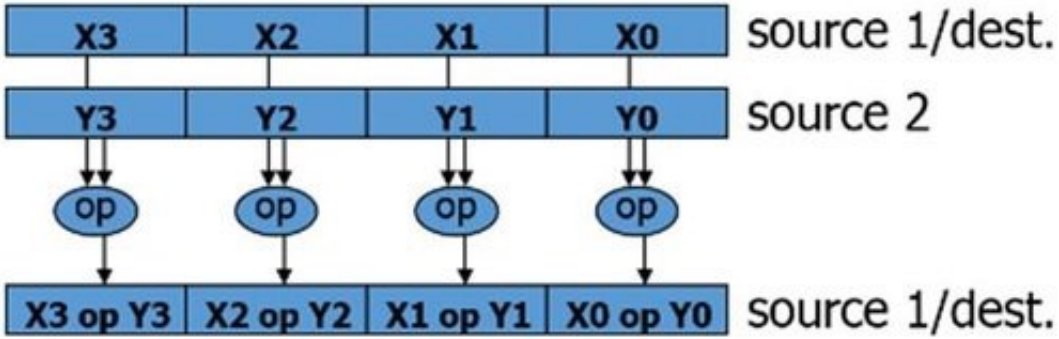
CPU

SIMD

1 instruction – multiple data

SSE2/3/4 – Neon – Altivec

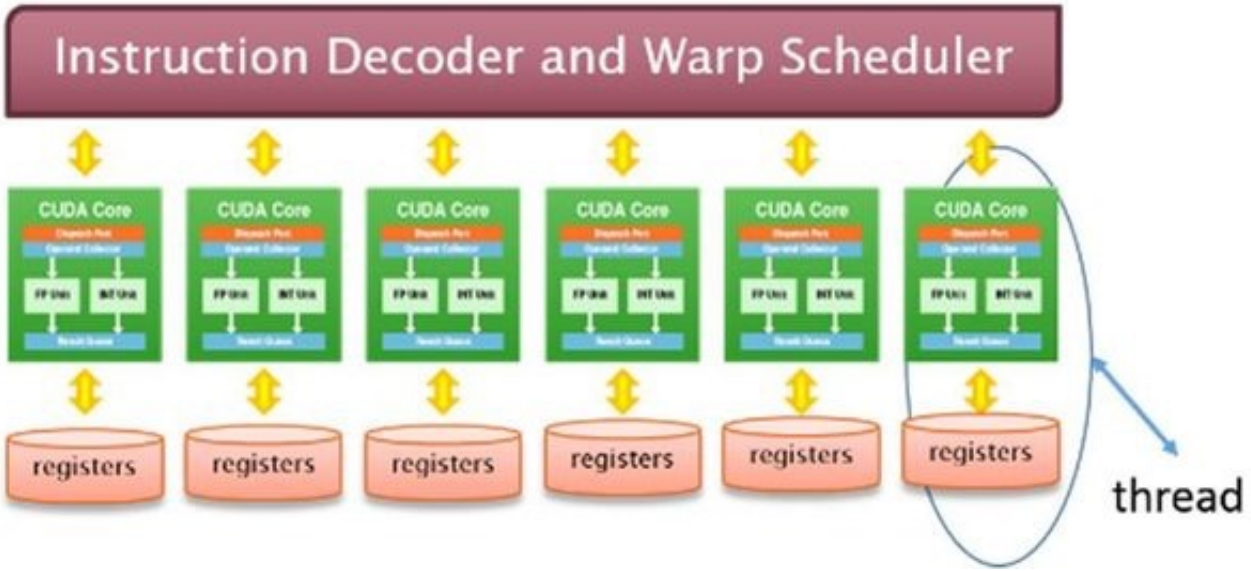
AVX – AVX2...

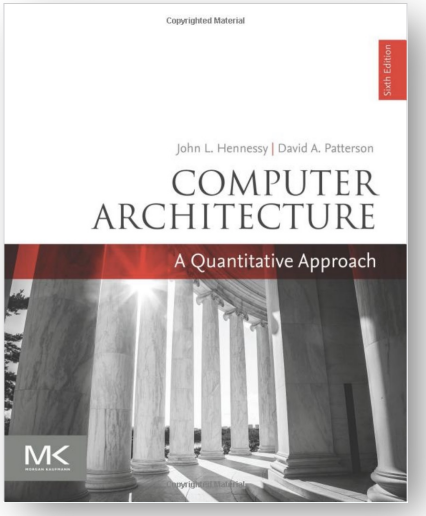


GPU

SIMT

1 instruction – multiple threads

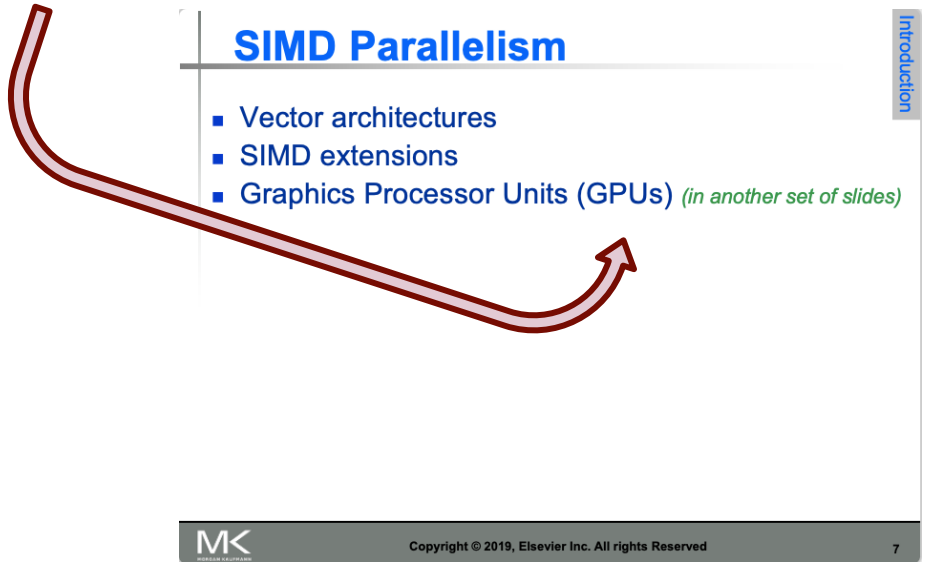




Chapter 4

Data-Level Parallelism in Vector, SIMD, and GPU Architectures

Graphical Processing Units



- Basic idea:
  - Heterogeneous execution model
    - CPU is the *host*, GPU is the *device*
  - Develop a C-like programming language for GPU
  - Unify all forms of GPU parallelism as *CUDA thread*
  - Programming model follows SIMT:
    "*Single Instruction Multiple Thread*"

cores/processing element in several computing devices

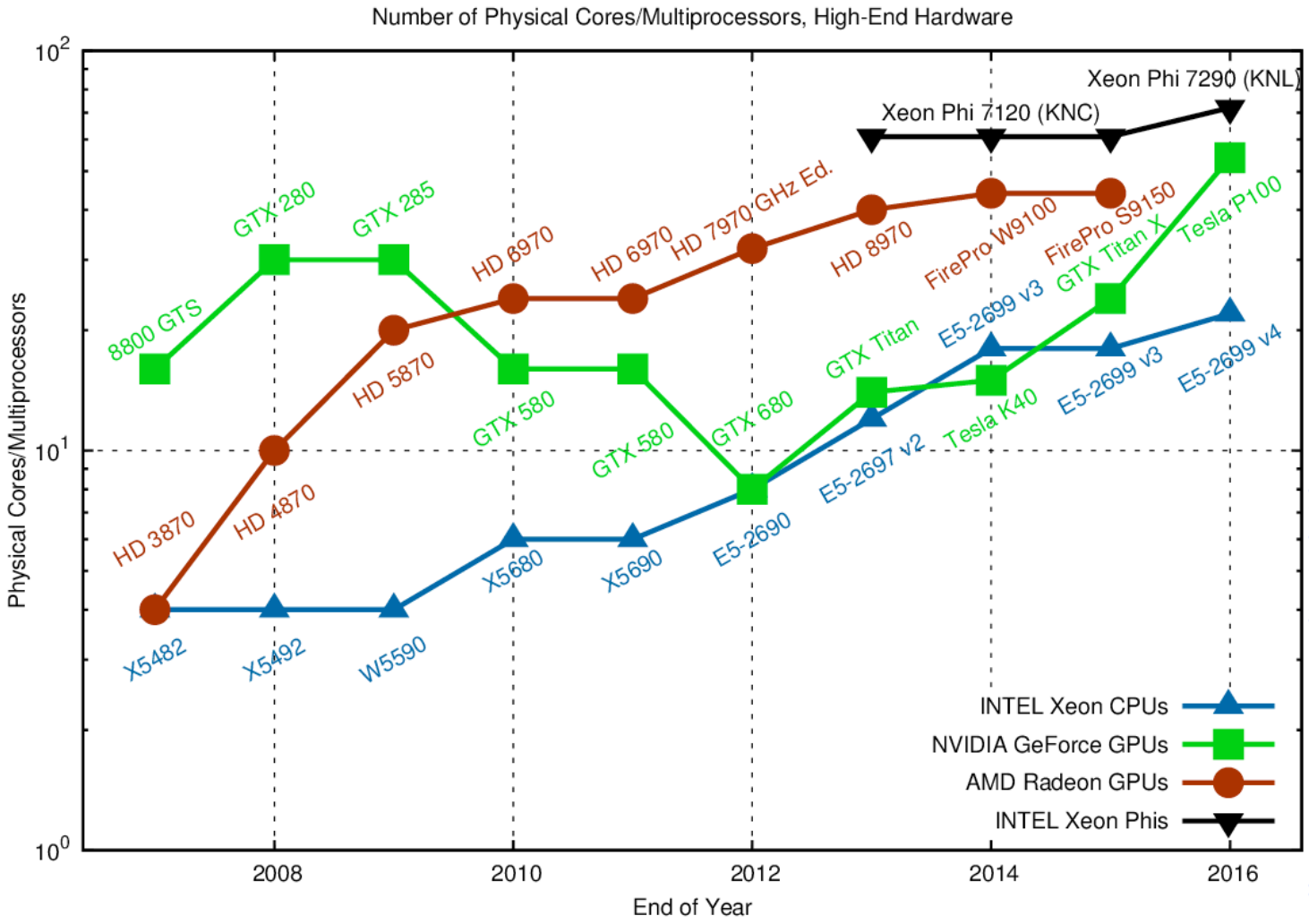
Key question: what is a **core**?

a) IU+FPU?
GPU-type...

b) A SIMD processor?
CPU-type..

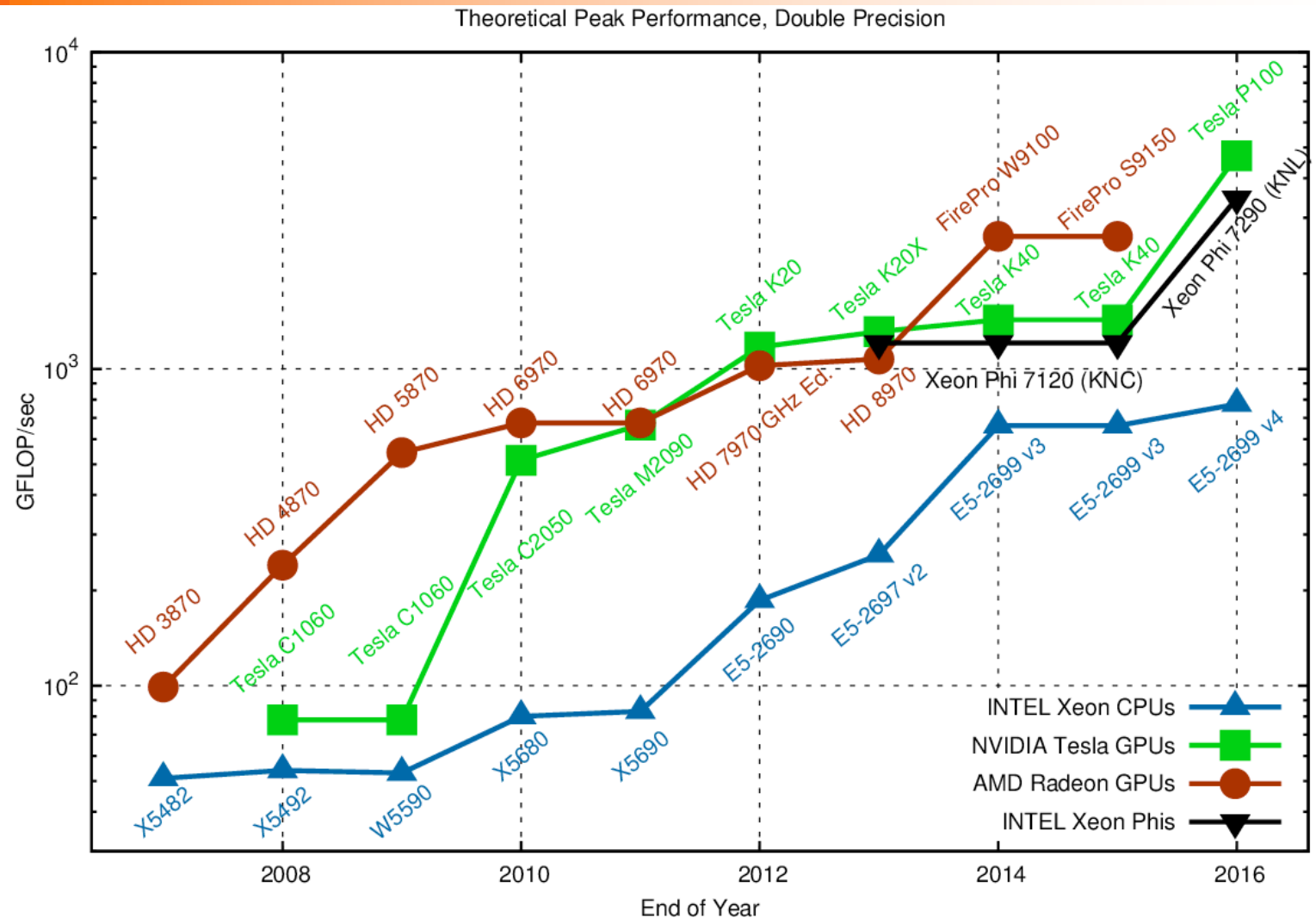
This updated slide and in this course:
- **b)**

Note: the web link with these plots was updated in Aug'16



http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Theoretical peak performance in several computing devices (DP)



http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

NVIDIA GPU Architecture

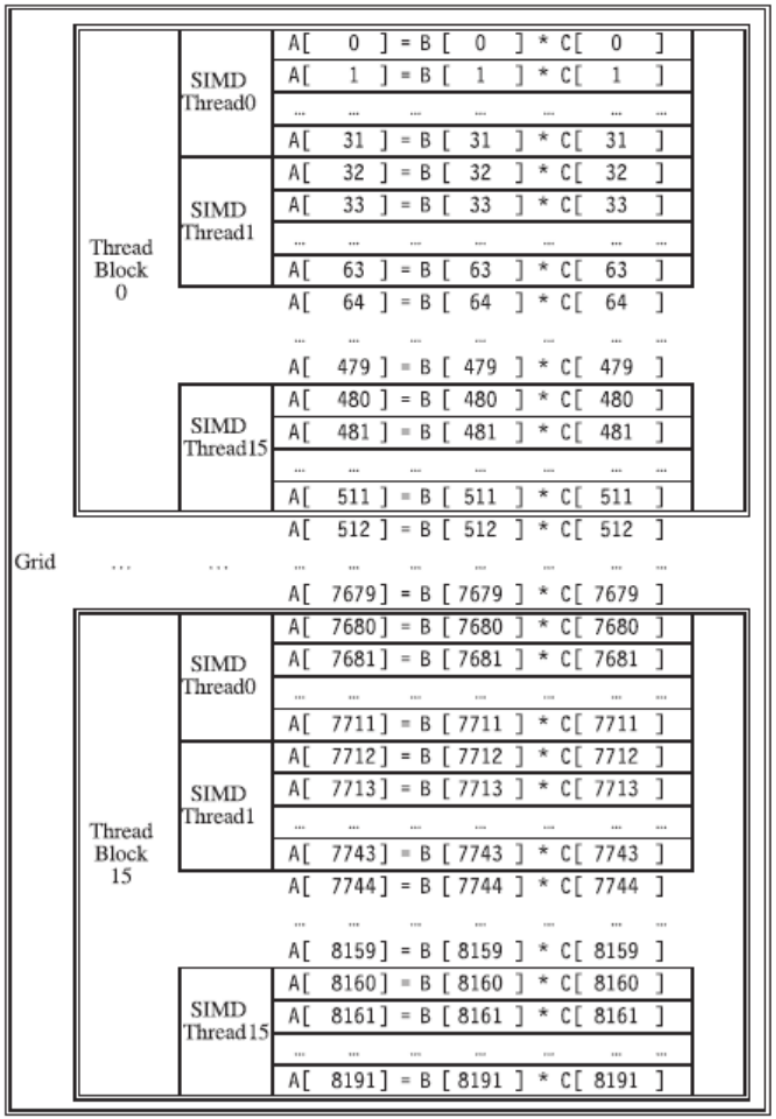
- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files

- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units, behaving as a vector processor

Terminology

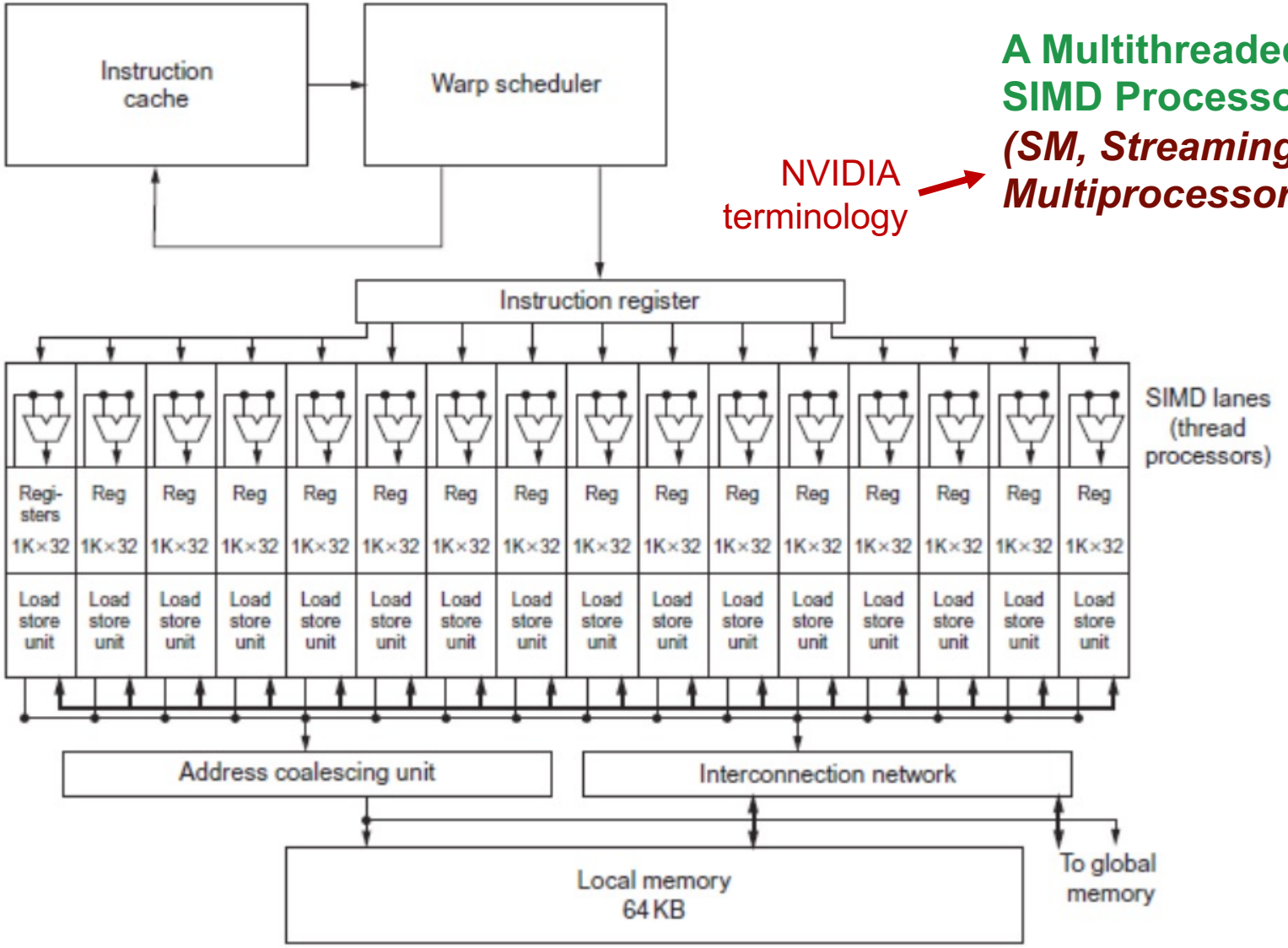
- Each thread is limited to 64 registers
- Groups of 32 threads combined into a SIMD thread or "**warp**"
  - Mapped to 16 physical lanes
- Up to 32 warps are scheduled on a single SIMD processor (**SM**)
  - Each warp has its own PC
  - Thread scheduler uses scoreboard to dispatch warps
  - By definition, no data dependencies between warps
  - Dispatch warps into pipeline, hide memory latency
- Thread block scheduler schedules blocks to SIMD processors
- Within each SIMD processor (**SM**):
  - 32 SIMD lanes
  - Wide and shallow compared to vector processors

Example



- Code that works over all elements is the grid
- Thread blocks break this down into manageable sizes
  - 512 threads per block
- SIMD instruction executes 32 elements at a time *(warp)*
- Thus grid size = 16 blocks
- Block is analogous to a strip-mined vector loop with vector length of 32
- Block is assigned to a multithreaded SIMD processor by the thread block scheduler
- Early-generation GPUs had 7-15 multithreaded SIMD processors (**SM**)

GPU Organization *(NVIDIA)*

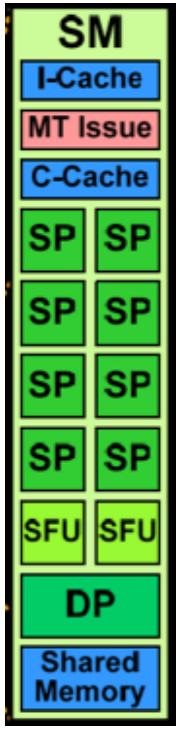


NVIDIA terminology → **A Multithreaded SIMD Processor *(SM, Streaming Multiprocessor)***

NVIDIA GPU Memory Structures

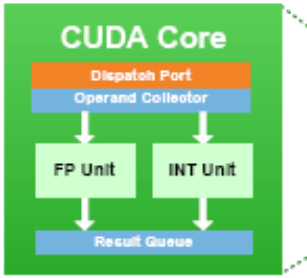
- Each SIMD Lane has private section of **off-chip DRAM**
  - "Private memory" *(Local Memory)*
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor *(SM)* also has local memory *(Shared Memory)*
  - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors *(SM)* is GPU Memory, off-chip DRAM *(Global Memory)*
  - Host can read and write GPU memory

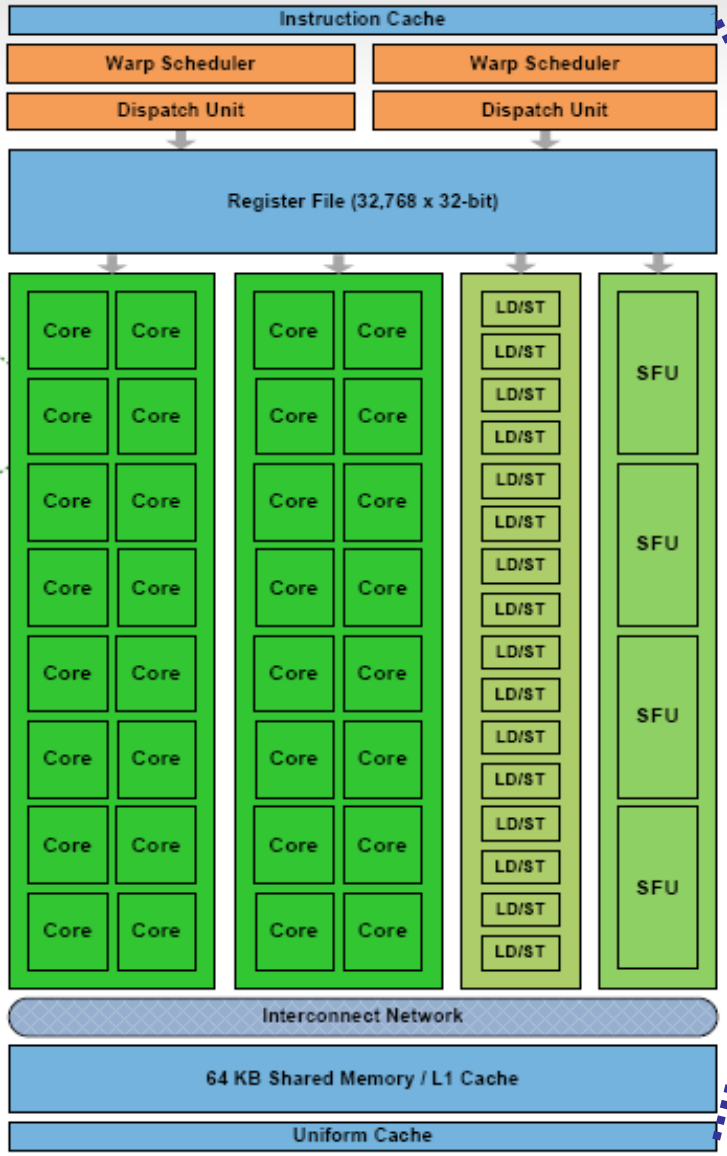
NVIDIA terminology



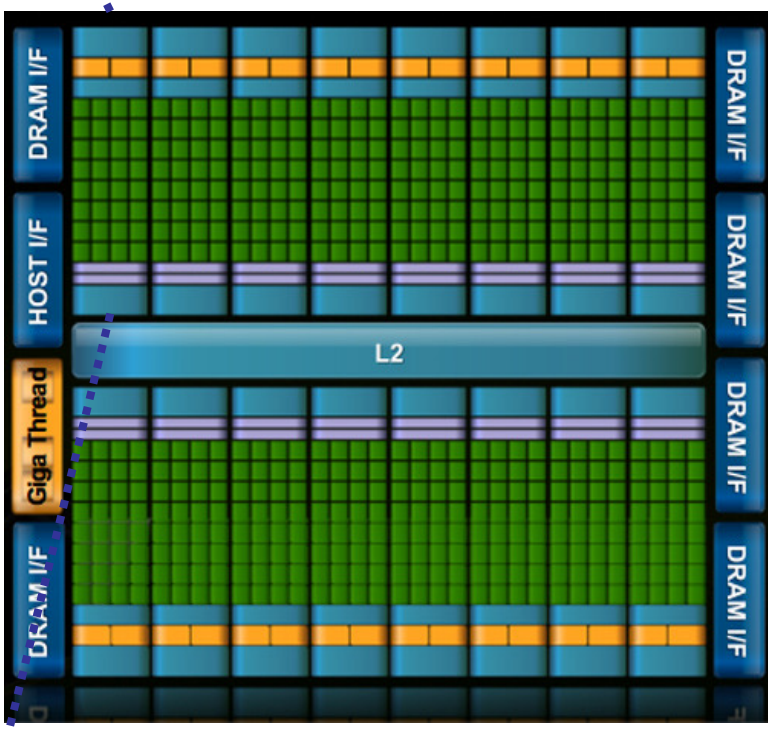
The NVidia Fermi architecture

Warp: a 32-wide SIMT instruction





Fermi Multithreaded SIMD Processor *(SM, Streaming Multiprocessor)*



Fermi Architecture:
GF110: 512 CUDA-cores
July'11

Pascal Architecture Innovations

May'16

- Each SIMD processor has
  - Two or four SIMD thread schedulers, two instruction dispatch units
  - Four 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 16 special function units
  - Two threads of SIMD instructions are scheduled every two clock cycles
- Fast single-, double-, and half-precision
- High Bandwith Memory 2 (HBM2) at 732 GB/s
- NVLink between multiple GPUs (20 GB/s in each direction)
- Unified virtual memory and paging support

Pascal Multithreaded SIMD Proc.

Vector Architectures *vs.* GPUs

- SIMD processor analogous to vector processor, both have MIMD
- Registers
  - RV64V register file holds entire vectors
  - GPU distributes vectors across the registers of SIMD lanes
  - RV64 has 32 vector registers of 32 elements (1024)
  - GPU has 256 registers with 32 elements each (8K)
  - RV64 has 2 to 8 lanes with vector length of 32, chime is 4 to 16 cycles
  - SIMD processor chime is 2 to 4 cycles
  - GPU vectorized loop is grid
  - All GPU loads are gather instructions and all GPU stores are scatter instructions

SIMD Architectures *vs.* GPUs

- GPUs have more SIMD lanes
- GPUs have hardware support for more threads
- Both have 2:1 ratio between double- and single-precision performance
- Both have 64-bit addresses, but GPUs have smaller memory
- SIMD architectures have no scatter-gather support

The GP100 Pascal Architecture

Pascal: 60 SMs, 3840 CUDA-cores, 4 HBM2 on-package

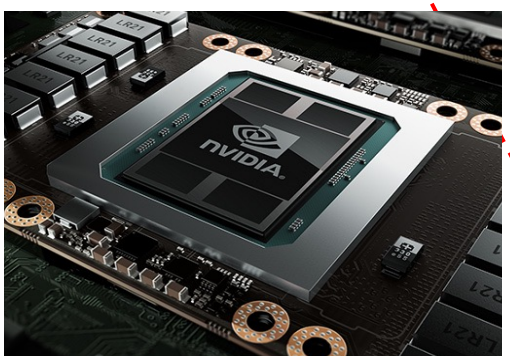
*released on **May'16***

Pascal Architecture: 6x GPCs, 60 SMs

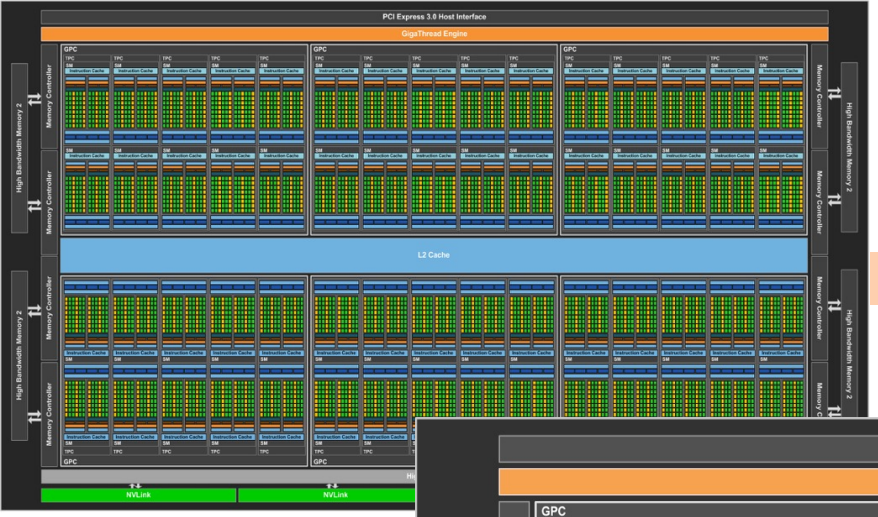
Pascal SM:
4x16 CUDA-cores

Ratio **DP**unit : **SP**unit —> **1 : 2**

Pascal P100 w/ 16GiB HBM2



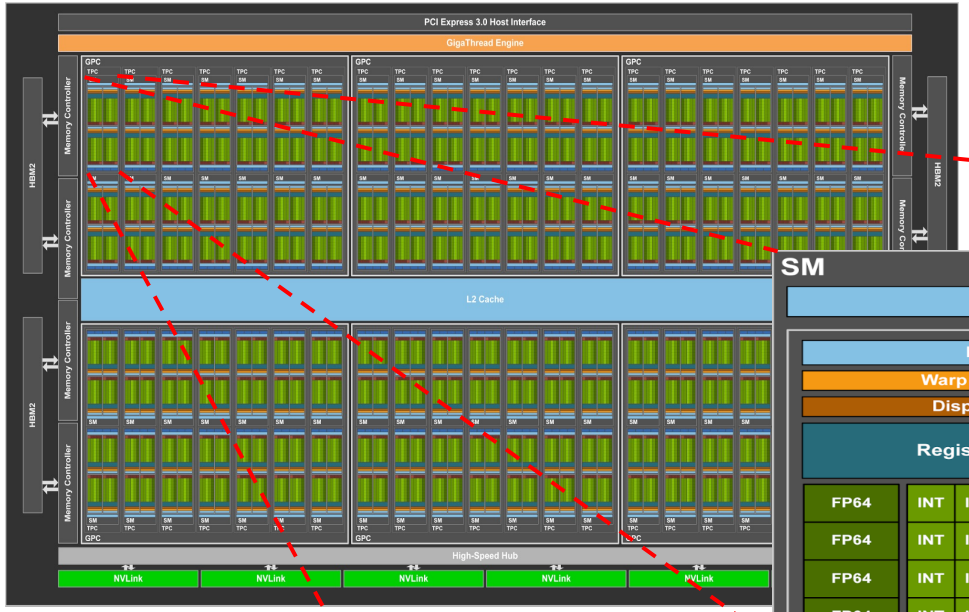
From the GP100 to the GV100 Volta Architecture

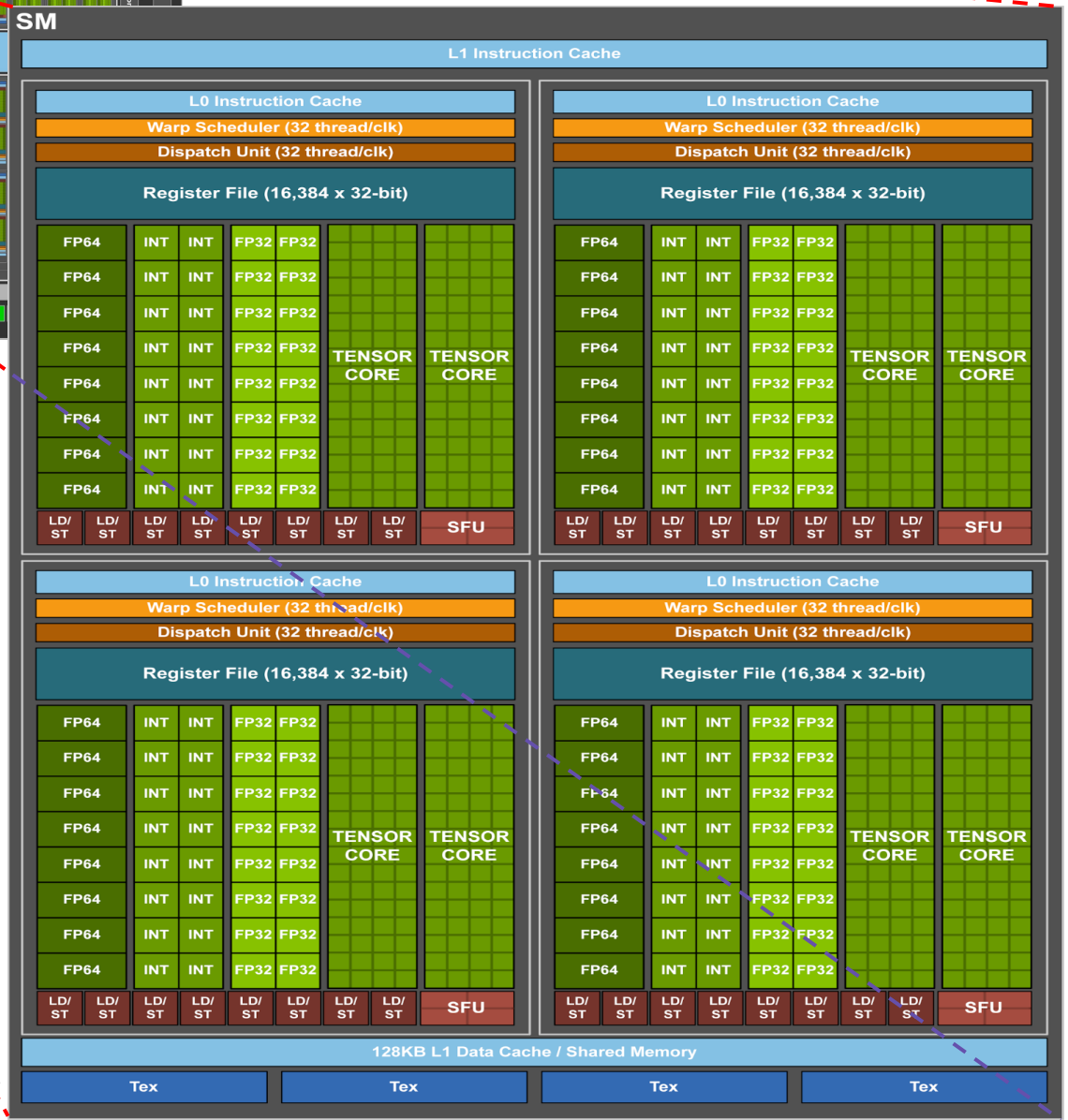


Pascal:
60 SM
3840 CUDA-cores
May'16

Volta:
84 SM
5120 CUDA-cores
4 HBM2 on-package
released Dec'17

Volta Architecture: 6x GPCs, 84 SMs



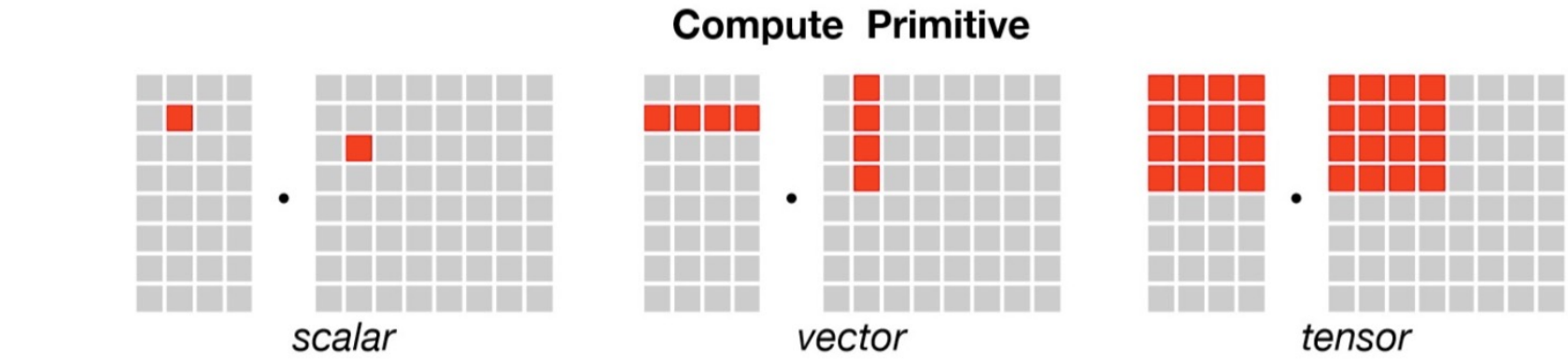


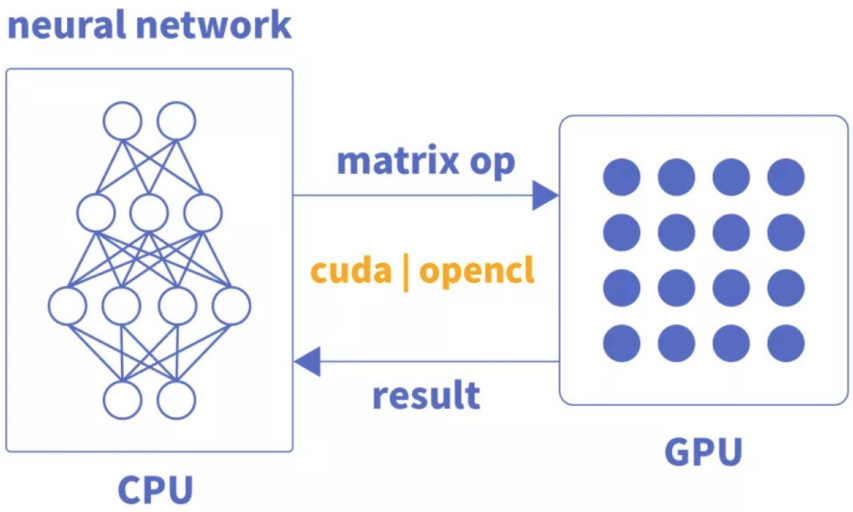
Volta SM:
4x16 (INT+FP32)-cores
New: 4x2 Tensor-cores

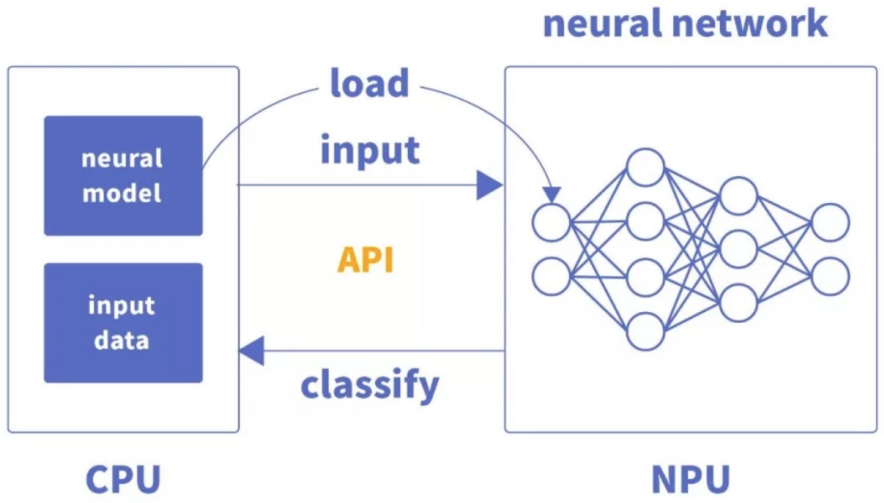
Ratio **DP**unit : **SP**unit —> **1 : 2**

Volta V100 w/ 16GiB HBM2

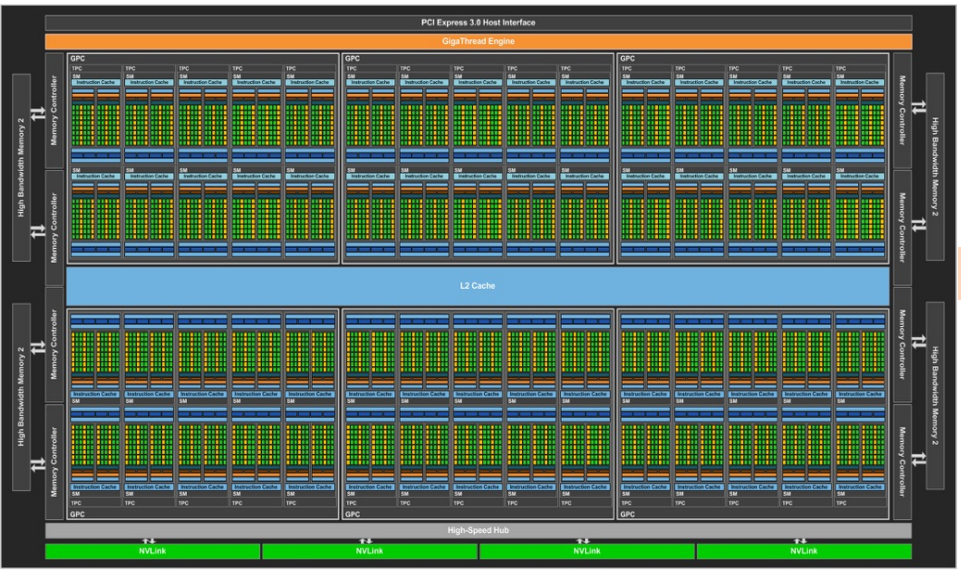
Compute primitive (smallest unit) *in CPU, GPU and TPU/NPU*







From GV 100 to Ampere: up to 8 GPC, 128 SMs total

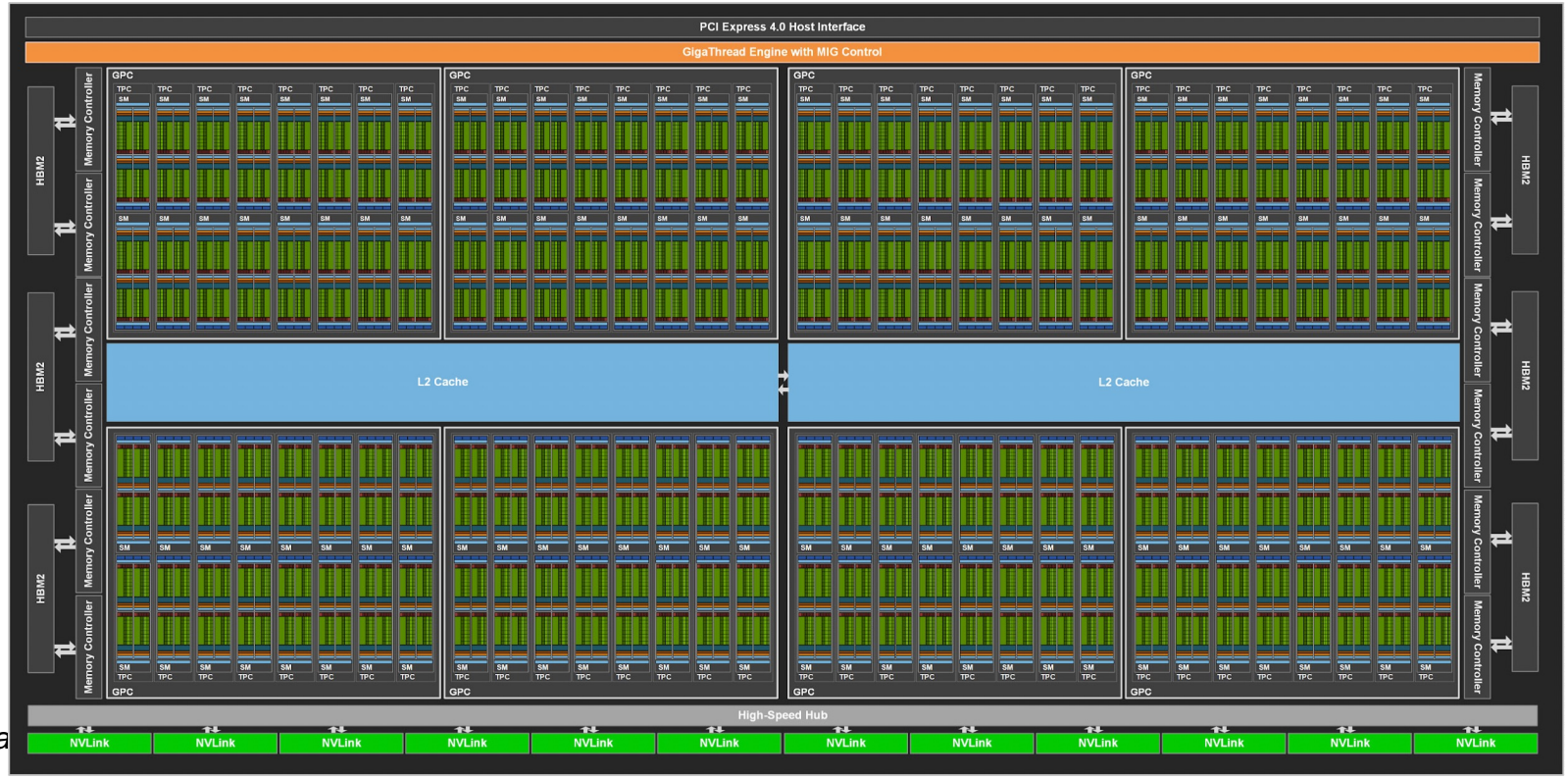


Ampere: NVidia GA100
8192 FP32 CUDA Cores
512 3rd generation Tensor Cores
5 HBM2e (up to 80 GiB)
10 512-bit mem controllers
May'20

Volta:
84 SM
3584 CUDA-cores
December'17

Ampere:
GA100
for graphics
w/ 8 GPC

A100
for HPC & AI
w/ 7 GPC



Ampere Architecture

Ampere SM:

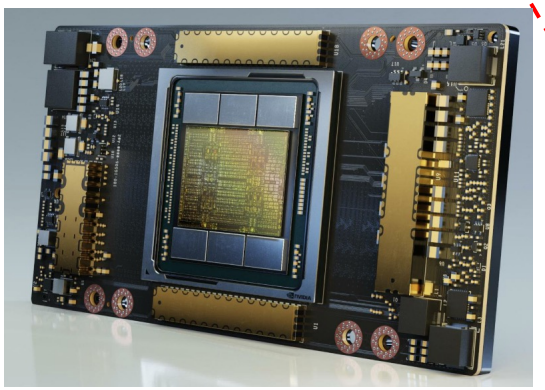
64x FP32 CUDA Cores/SM

32x FP64 CUDA Cores/SM

4x 3rd generation Tensor Cores

Tensor Cores support
FP64, FP32, TF32, FP16, BF16, INT8…

1024 dense FP16/FP32 FMA op's/cycle



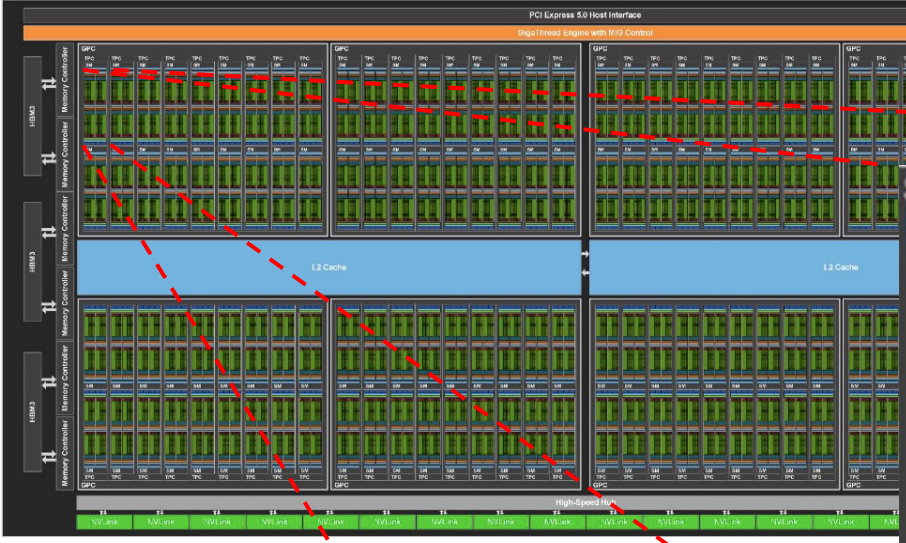
From GA 100 to Hopper: up to 8 GPC, 132 SMs total

Hopper: NVidia GH100
16 896 FP32 CUDA Cores
528 4th generation Tensor Cores
5 HBM3 (up to 80 GiB)
10 512-bit mem controllers
announced March'22

Ampere:
128 SM
8192 CUDA-cores
May'20

Hopper:
GH100
w/ 8 GPC
full w/ 144 SMs

Hopper Architecture



Hopper SM:

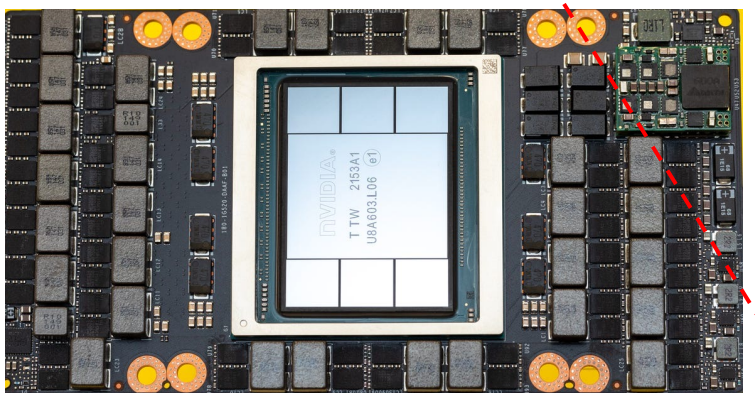
128x FP32 CUDA Cores/SM

64x FP64 CUDA Cores/SM

4x 4th generation Tensor Cores

Tensor Cores support (2x faster)
FP64, FP32, TF32, FP16, BF16, **FP8,** INT8…

1024 dense FP16/FP32 FMA op's/cycle



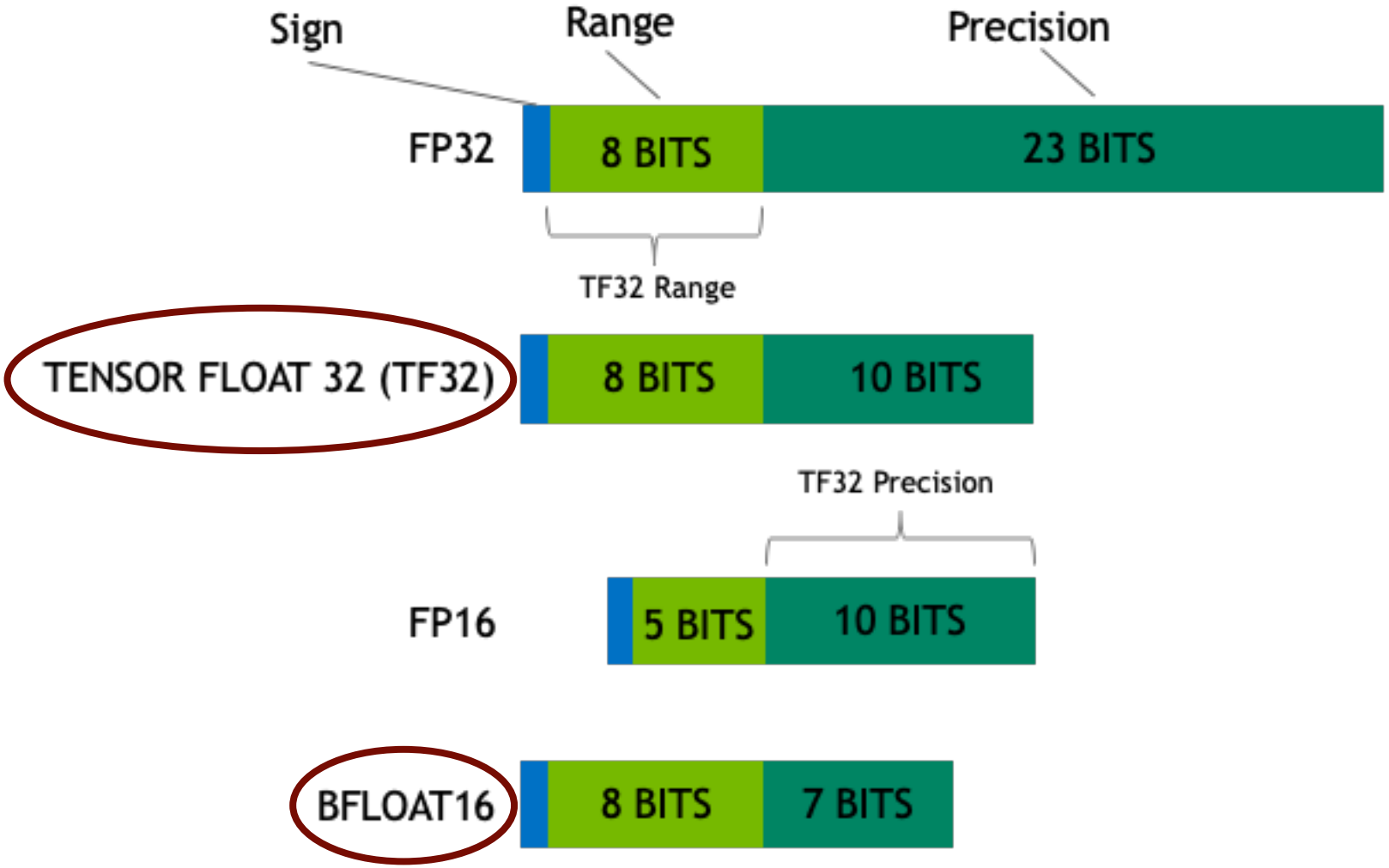
Compute Capability & Precision Support Matrix: GP100 vs GV100 vs GA100 vs GH100

GPU features	NVIDIA Tesla **P100**	NVIDIA Tesla **V100**	NVIDIA **A100**	NVIDIA **H100**
GPU codename	GP100	GV100	GA100	GH100
GPU architecture	NVIDIA **Pascal**	NVIDIA **Volta**	NVIDIA **Ampere**	NVIDIA **Hopper**
Transistors	15.3 billion	21.1 billion	54.2 billion	80 billion
Process	**16nm**	**12nm**	TSMC **7nm**	TSMC **4nm**
Die size	610 mm^2	828 mm^2	815 mm^2	814 mm^2
Compute capability	6.0	7.0	8.0	9.0
Threads / warp	32	32	32	32

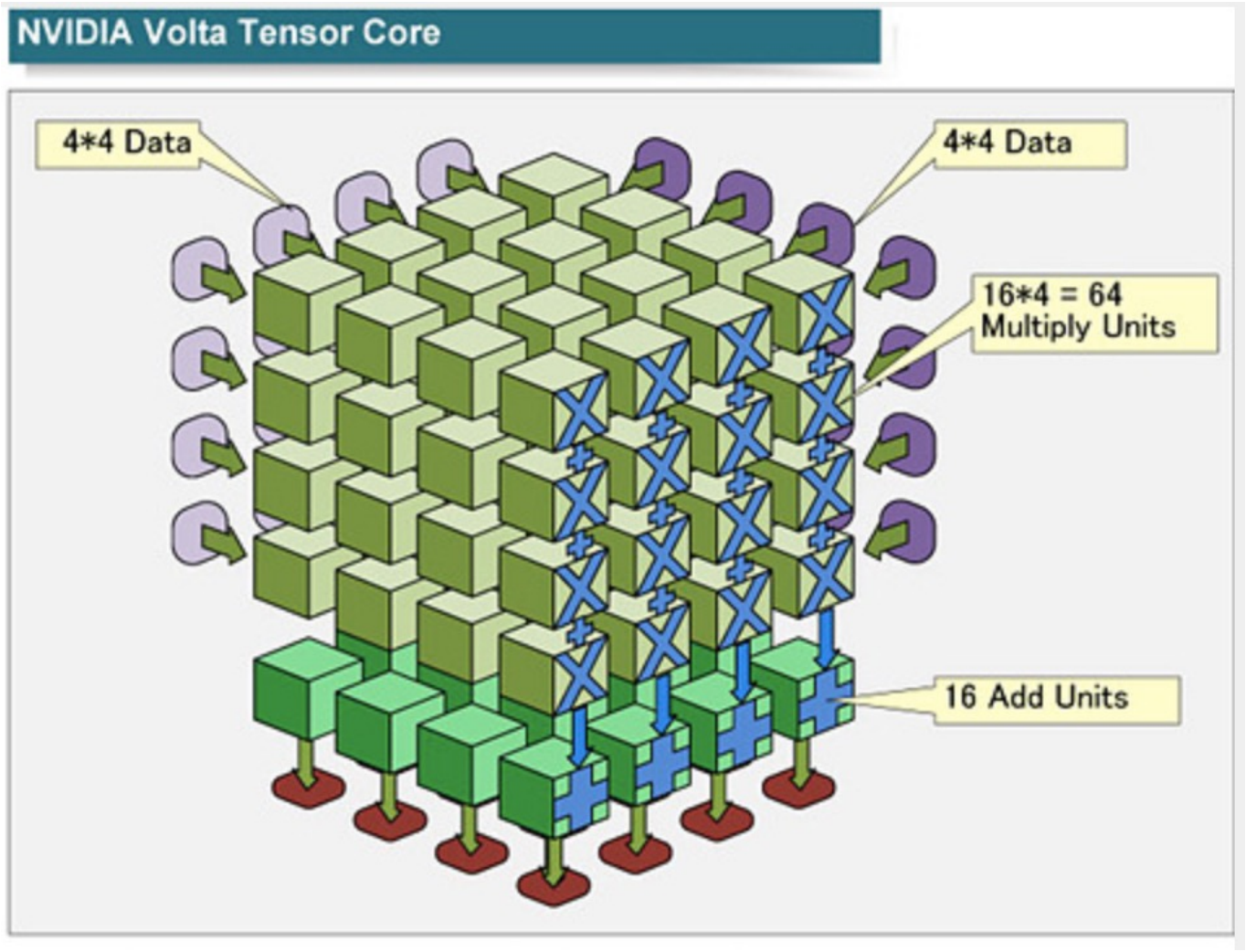
	Supported CUDA Core Precisions									Supported Tensor Core Precisions								
	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16	FP8	FP16	FP32	FP64	INT1	INT4	INT8	TF32	BF16
NVIDIA Tesla P4	No	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
NVIDIA P100	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
NVIDIA Volta	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No	No
NVIDIA Turing	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	Yes	No	No
NVIDIA A100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
NVIDIA H100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes

https://en.wikipedia.org/wiki/Hopper_(microarchitecture)

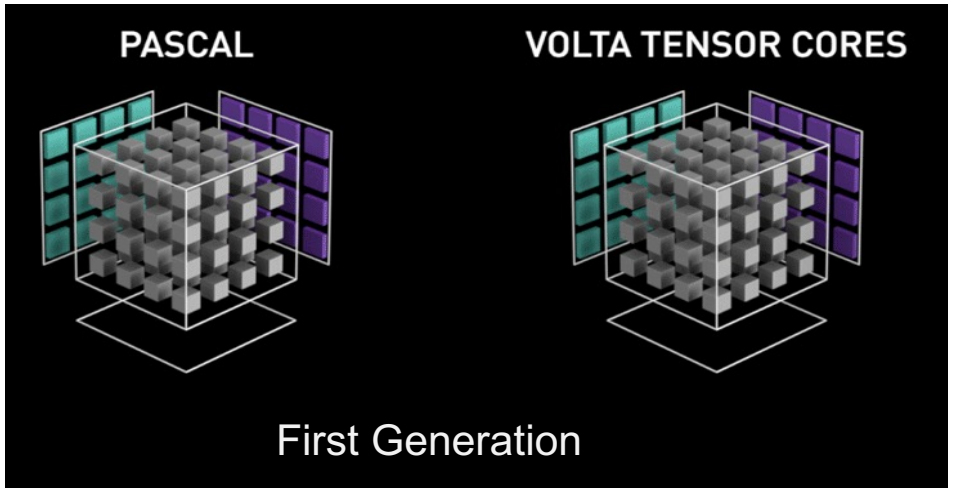
Novel Floating-Point formats

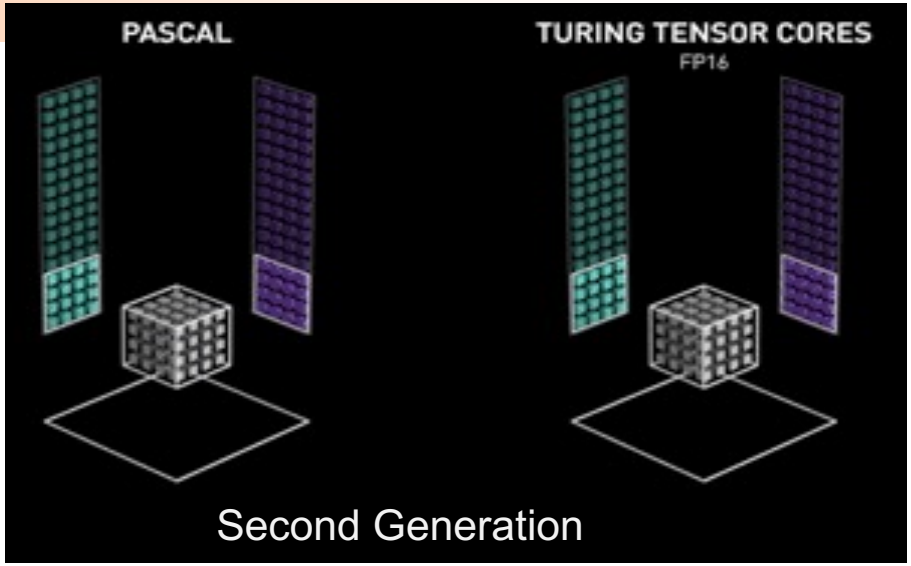


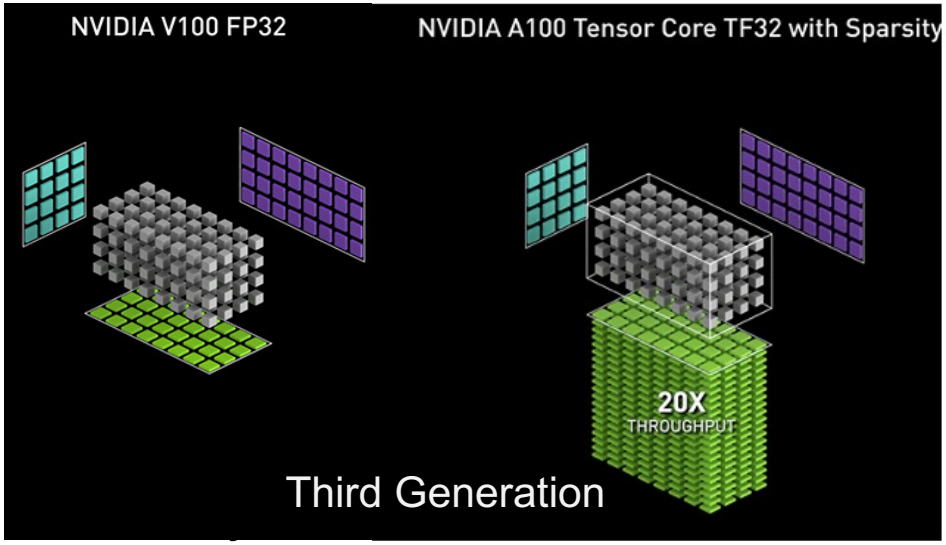
Matrix FMA in a single clock cycle with FP32

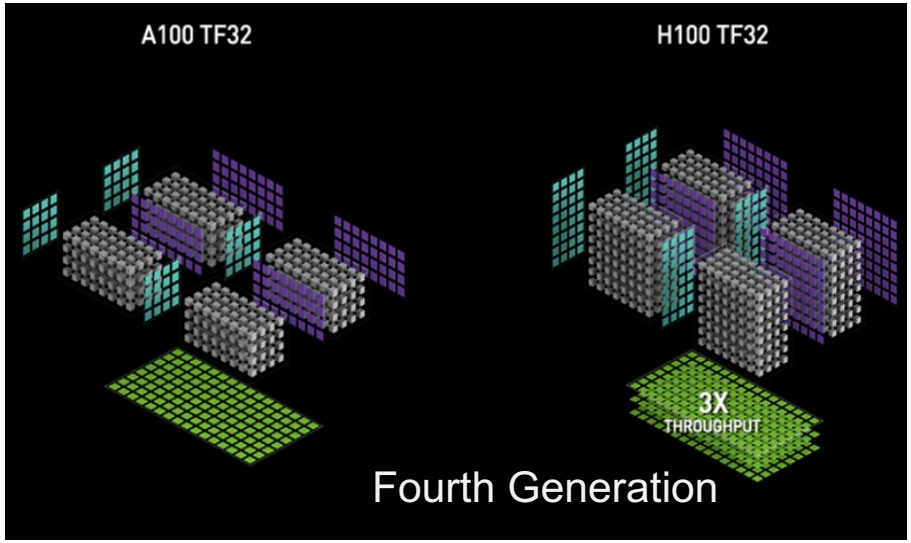


NVidia Tensor cores: 4 generations

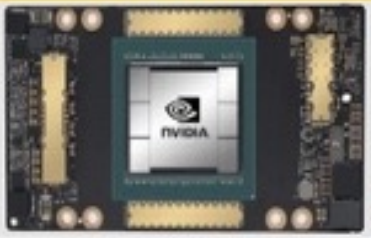








Tesla evolution (1)

NVIDIA Data-Center GPUs Specifications				
VideoCardz.com	NVIDIA H100	NVIDIA A100	NVIDIA Tesla V100	NVIDIA Tesla P100
Picture				
GPU	**GH100**	GA100	GV100	GP100
Transistors	**80B**	54.2B	21.1B	15.3B
Die Size	**814 mm²**	828 mm²	815 mm²	610 mm²
Architecture	**Hopper**	Ampere	Volta	Pascal
Fabrication Node	**TSMC N4**	TSMC N7	12nm FFN	16nm FinFET+
GPU Clusters	**132/114***	108	80	56
CUDA Cores	**16896/14592***	6912	5120	3584
L2 Cache	**50MB**	40MB	6MB	4MB
Tensor Cores	**528/456***	432	320	-
Memory Bus	**5120-bit**	5120-bit	4096-bit	4096-bit
Memory Size	**80 GB HBM3/HBM2e***	40/80GB HBM2e	16/32 HBM2	16GB HBM2
TDP	**700W/350W***	250W/300W/400W	250W/300W/450W	250W/300W
Interface	**SXM5/*PCIe Gen5**	SXM4/PCIe Gen4	SXM2/PCIe Gen3	SXM/PCIe Gen3
Launch Year	**2022**	2020	2017	2016

https://videocardz.com/press-release/nvidia-announces-h100-hopper-gpu-with-up-to-16896-fp32-cores-80gb-hbm3-memory-700w-of-tdp

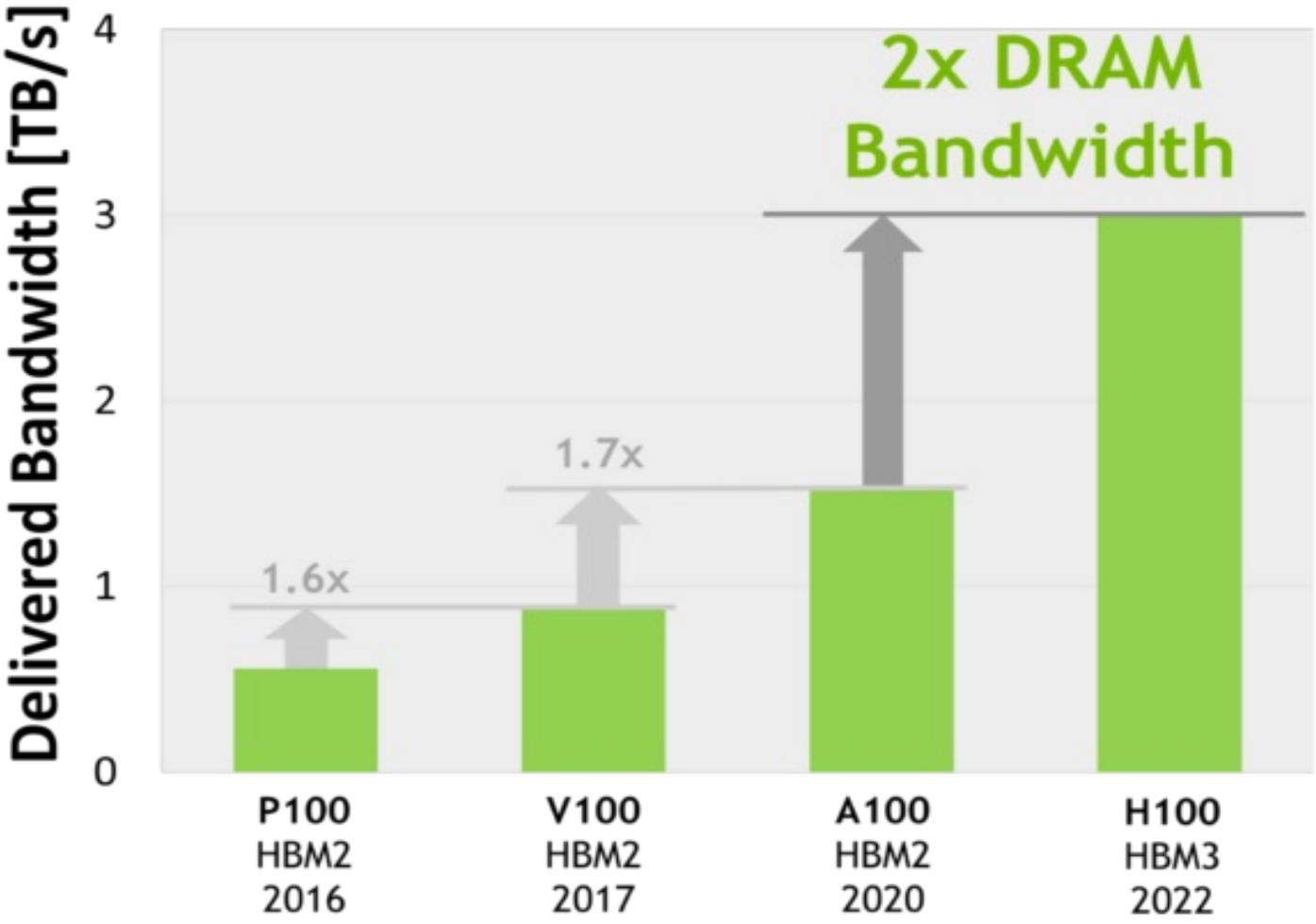
Tesla evolution (2)

Nvidia Datacenter GPU	Nvidia A100 SXM	Nvidia H100 SXM	Nvidia H100 PCIe
GPU codename	GA100	GH100	GH100
GPU architecture	Ampere	Hopper	Hopper
GPU board form factor	SXM4	SXM5	PCIe Gen5
Launch date	May 2020	March 2022	March 2022
GPU process	TSMC 7nm N7	custom TSMC 4N	custom TSMC 4N
Die size	826mm2	814 mm2	814 mm2
Transistor Count	54 billion	80 billion	80 billion
FP64 CUDA cores	3,456	8,448	7,296
FP32 CUDA cores	6,912	16,896	14,592
Tensor Cores	432	528	456
Streaming Multiprocessors	108	132	114
Peak FP64	9.7 teraflops	30 teraflops	24 teraflops
Peak FP64 Tensor Core	19.5 teraflops	60 teraflops	48 teraflops
Peak FP32	19.5 teraflops	60 teraflops	48 teraflops
Peak FP32 Tensor Core	156 teraflops | 312 teraflops*	500 teraflops | 1,000 teraflops*	400 teraflops | 800 teraflops*
Peak BFLOAT16 Tensor Core	312 teraflops | 624 teraflops*	1,000 teraflops | 2,000 teraflops*	800 teraflops | 1,600 teraflops*
Peak FP16 Tensor Core	312 teraflops | 624 teraflops*	1,000 teraflops | 2,000 teraflops*	800 teraflops | 1,600 teraflops*
Peak FP8 Tensor Core	-	2,000 teraflops | 4,000 teraflops*	1,600 teraflops | 3,200 teraflops*
Peak INT8 Tensor Core	624 TOPS | 1,248 TOPS*	2,000 TOPS | 4,000 TOPS*	1,600 TOPS | 3,200 TOPS*
Peak INT4 Tensor Core	1,248 TOPS | 2,496 TOPS*	-	-
Interconnect	NVLink: 600GB/s PCI Gen4: 64GB/s	NVLink: 900GB/s PCI Gen5: 128GB/s	NVLink: 600GB/s PCI Gen5: 128GB/s
Max TDP	400 watts	700 watts	350 watts

*Effective TFLOPS or FLOPS using the Sparsity feature

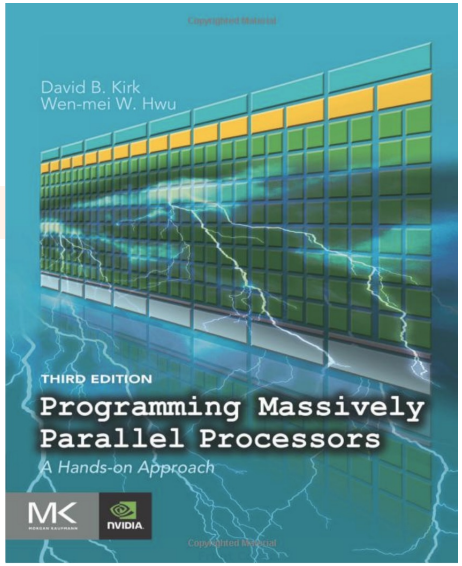
https://www.hpcwire.com/2022/03/22/nvidia-launches-hopper-h100-gpu-new-dgxs-and-grace-megachips/

Performance evolution of NVidia HBM



https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

The CUDA programming model



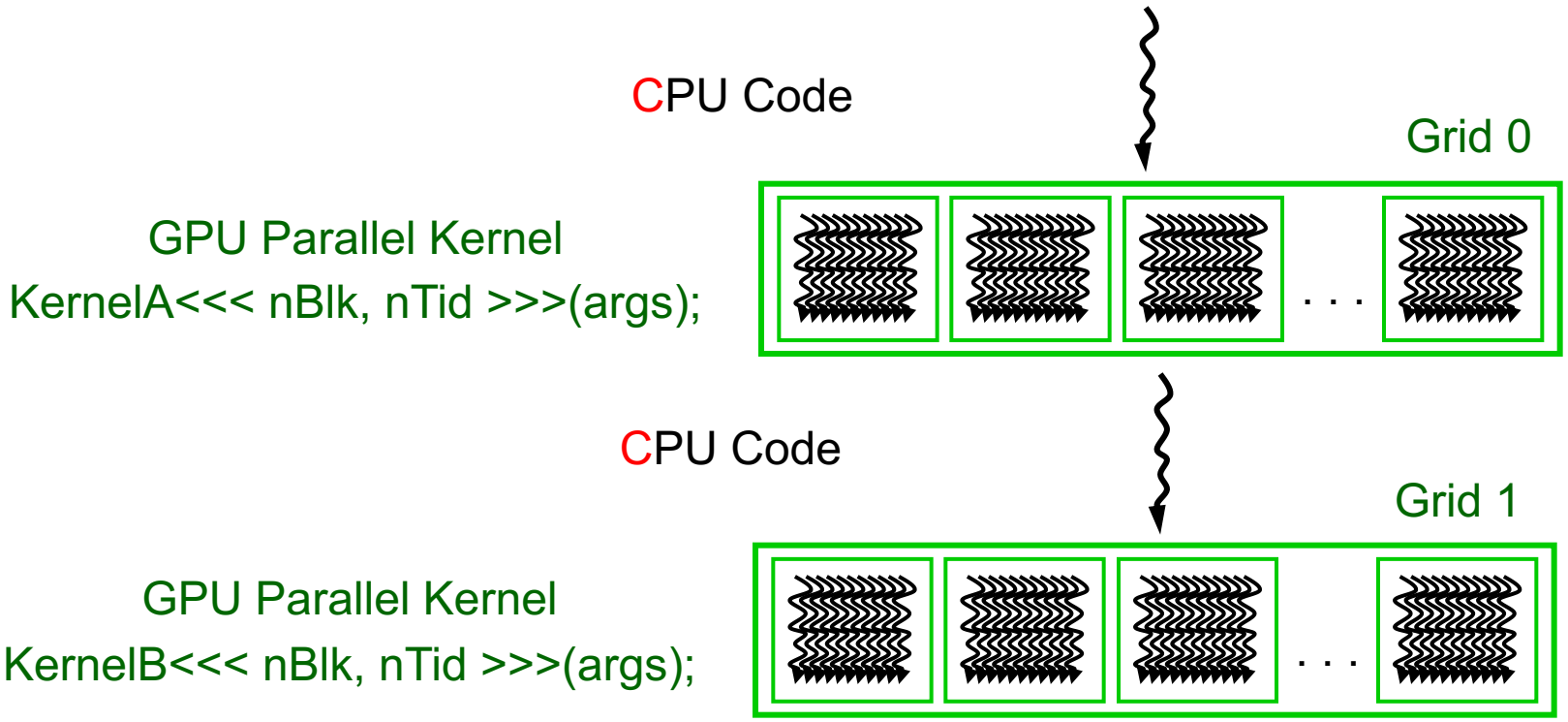
- ***C**ompute **U**nified **D**evice **A**rchitecture*
- CUDA is a recent programming model, designed for
  - a multicore CPU ***host*** coupled to a many-core ***device***, where
    - *devices* have wide SIMD/SIMT parallelism, and
    - the *host* and the *device* do not share memory
- CUDA provides:
  - a thread abstraction to deal with SIMD
  - synchr. & data sharing between small groups of threads
- CUDA programs are written in C with extensions
- OpenCL inspired by CUDA, but hw & sw vendor neutral
  - programming model essentially identical

CUDA Devices and Threads

- A compute device
  - is a coprocessor to the CPU or host
  - has its own DRAM (device memory)
  - runs many threads in parallel
  - is typically a GPU but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device kernels which run on many threads - **SIMT**
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - very little creation overhead, requires LARGE register bank
  - GPU needs 1000s of threads for full efficiency
    - multi-core CPU needs only a few

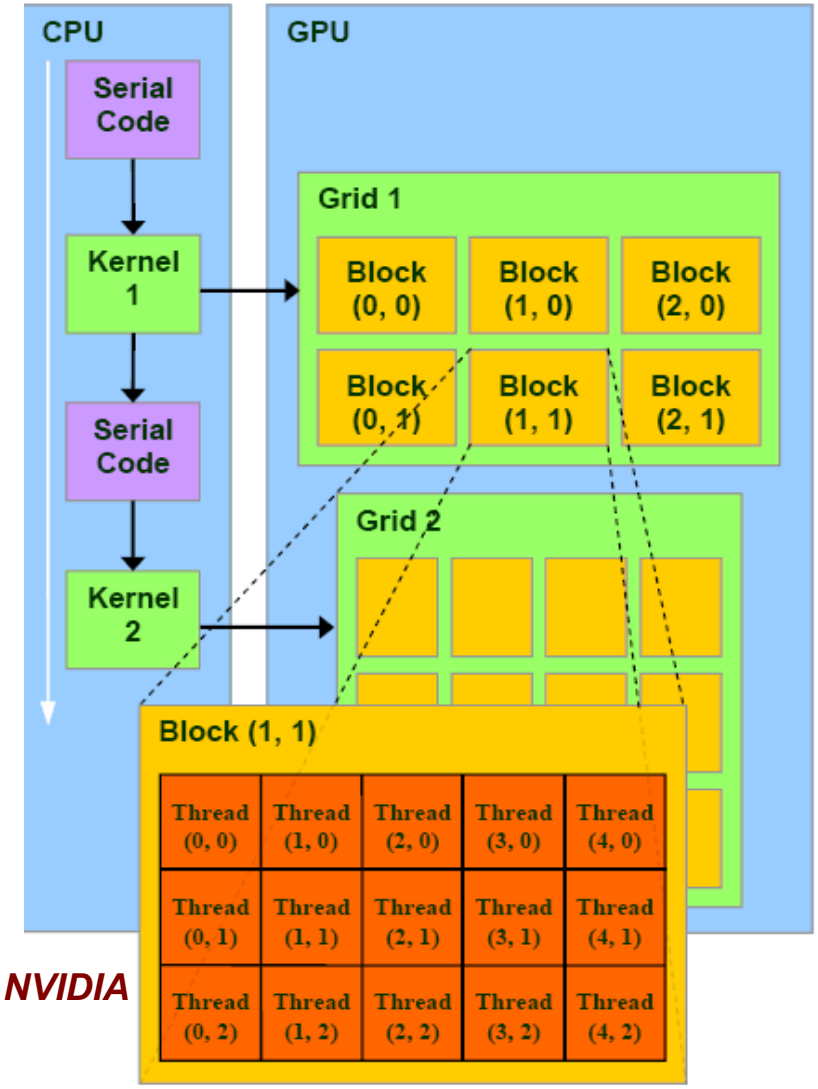
CUDA basic model: Single-Program Multiple-Data (SPMD)

- CUDA integrated CPU + GPU application C program
  - Serial C code executes on CPU
  - Parallel Kernel C code executes on GPU thread blocks



Programming Model: SPMD + SIMT/SIMD

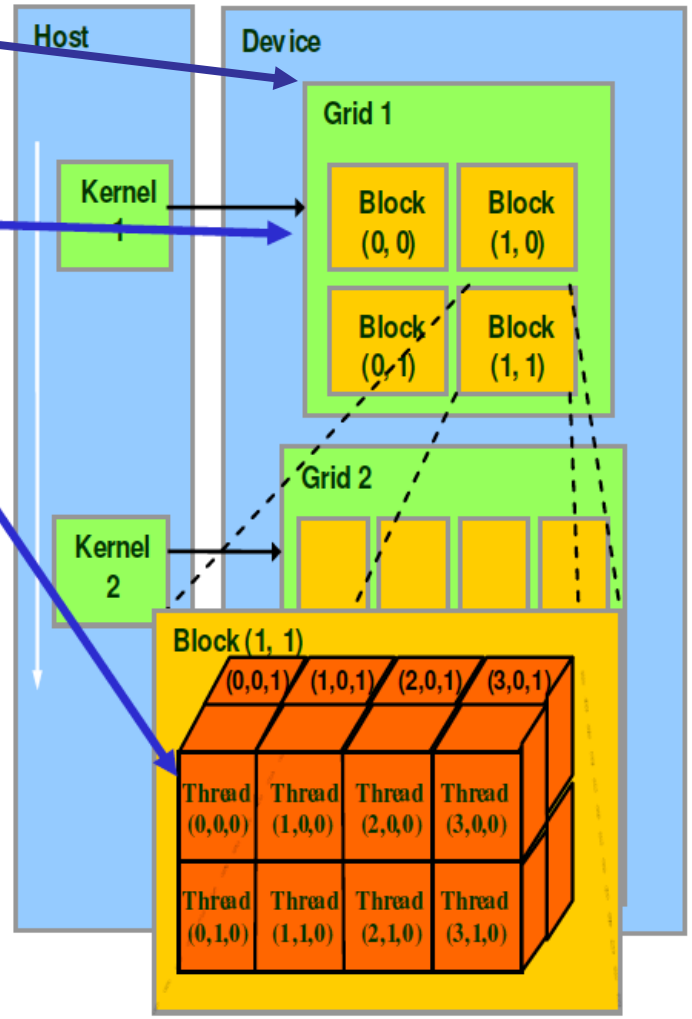
- Hierarchy
  - Device => Grids
  - Grid => Blocks
  - Block => Warps
  - Warp => Threads
- Single kernel runs on multiple blocks (SPMD)
- Threads within a warp are executed in a lock-step way called single-instruction multiple-thread (SIMT)
- Single instruction are executed on multiple threads (SIMD)
  - Warp size defines SIMD granularity (32 threads)
- Synchronization within a block uses shared memory



Courtesy NVIDIA

The Computational Grid: Block IDs and Thread IDs

- A kernel runs on a computational grid of thread blocks
  - Threads share global memory
- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- A thread block is a batch of threads that can cooperate by:
  - Sync their execution w/ barrier
  - Efficiently sharing data through a low latency shared memory
  - Two threads from two different blocks cannot cooperate



Code example

C with CUDA Extensions: C with a few keywords

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Standard C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

NVIDIA Confidential

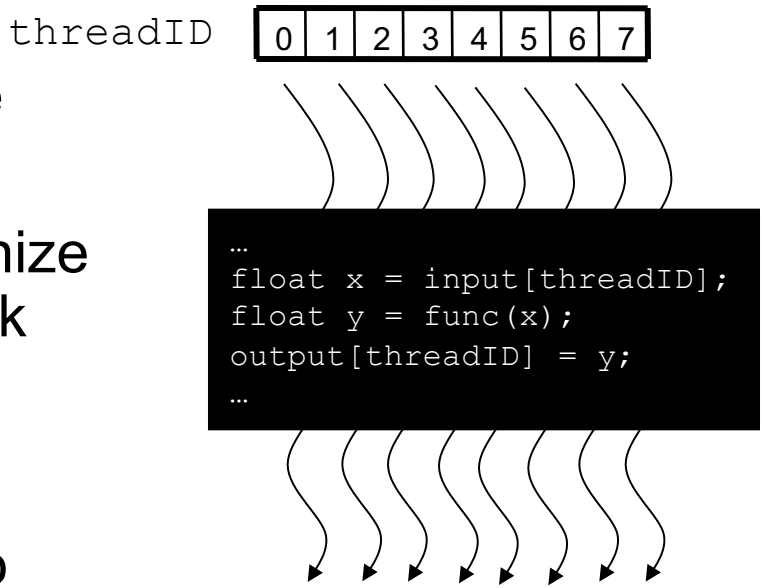
Terminology *(and in NVidia)*

- *Threads of SIMD instructions (**warps**)*
  - Each has its own IP (up to 48/64 per SIMD processor, Fermi/Kepler)
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Threads are organized into blocks & executed in groups of 32 threads *(**thread block**)*
    - Blocks are organized into a grid
- The thread block scheduler schedules blocks to SIMD processors *(**Streaming Multiprocessors**)*
- Within each SIMD processor:
  - 32 SIMD lanes *(**thread processors**)*
  - Wide and shallow compared to vector processors

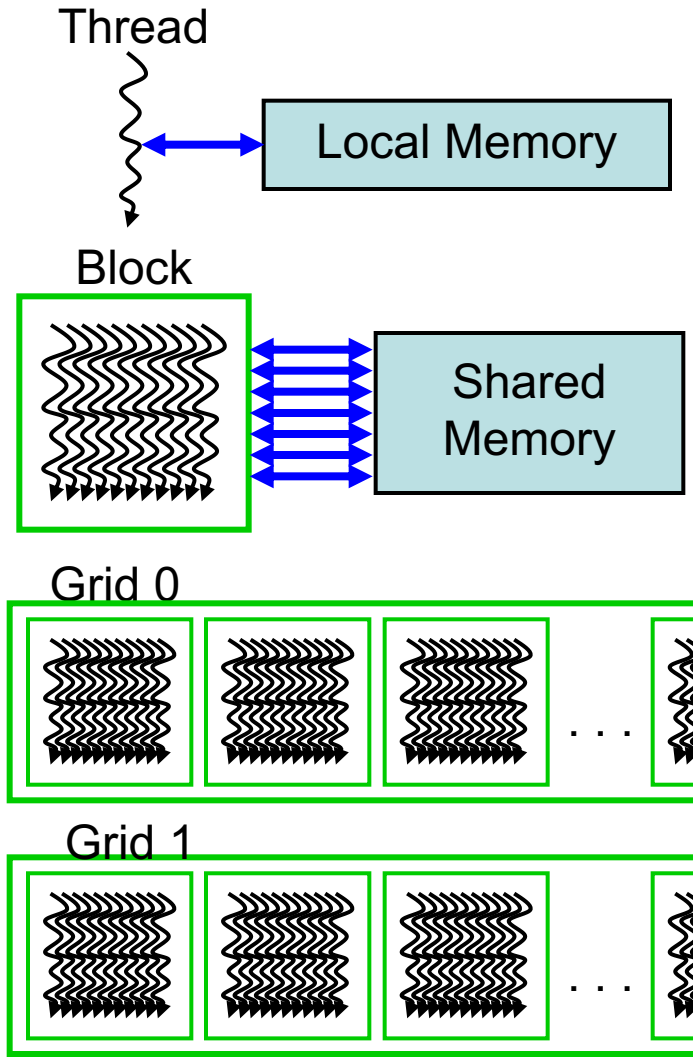
CUDA Thread Block

- Programmer declares (Thread) Block:
  - Block size 1 to **512** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have thread id numbers within Block
- Thread program uses thread id to select work and address shared data

CUDA Thread Block



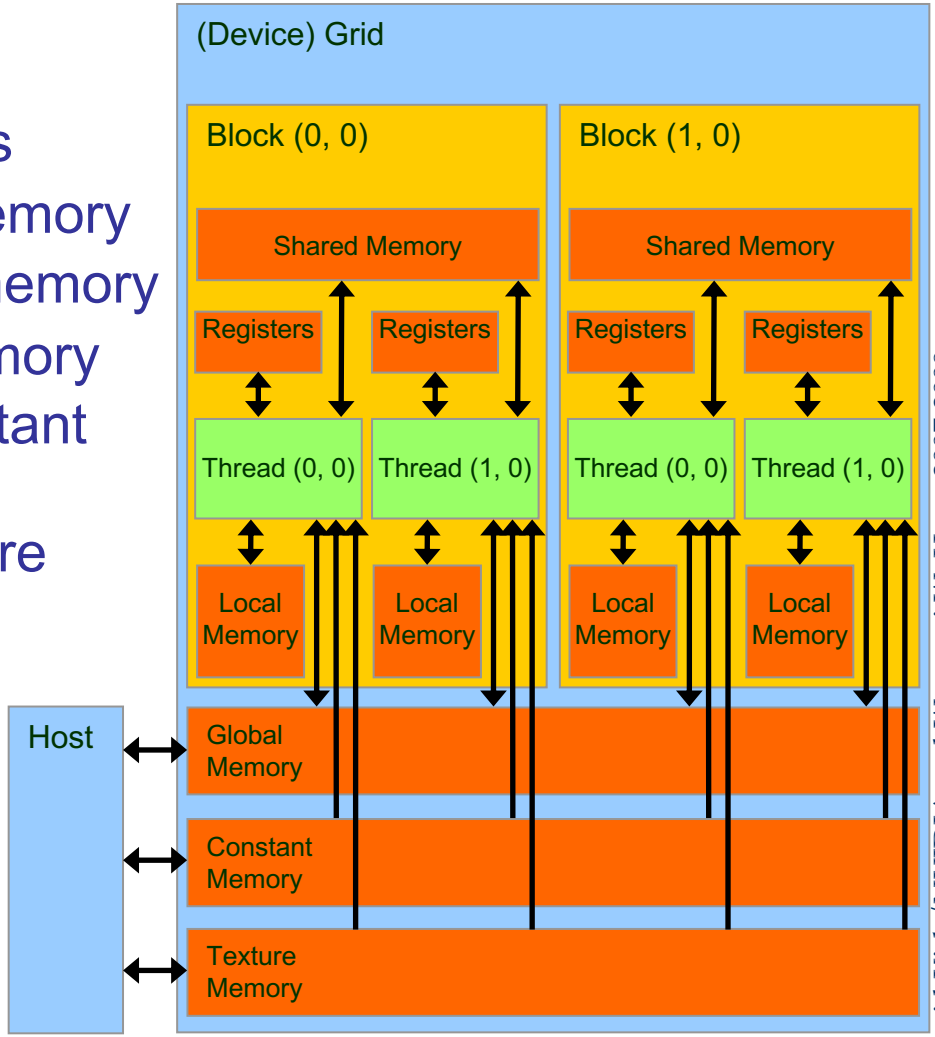
Parallel Memory Sharing



- Local Memory: **per-thread**
  - Private per thread
  - Auto variables, register spill
- Shared Memory: **per-block**
  - Shared by threads of the same block
  - Inter-thread communication
- Global Memory: **per-application**
  - Shared by all threads
  - Inter-Grid communication

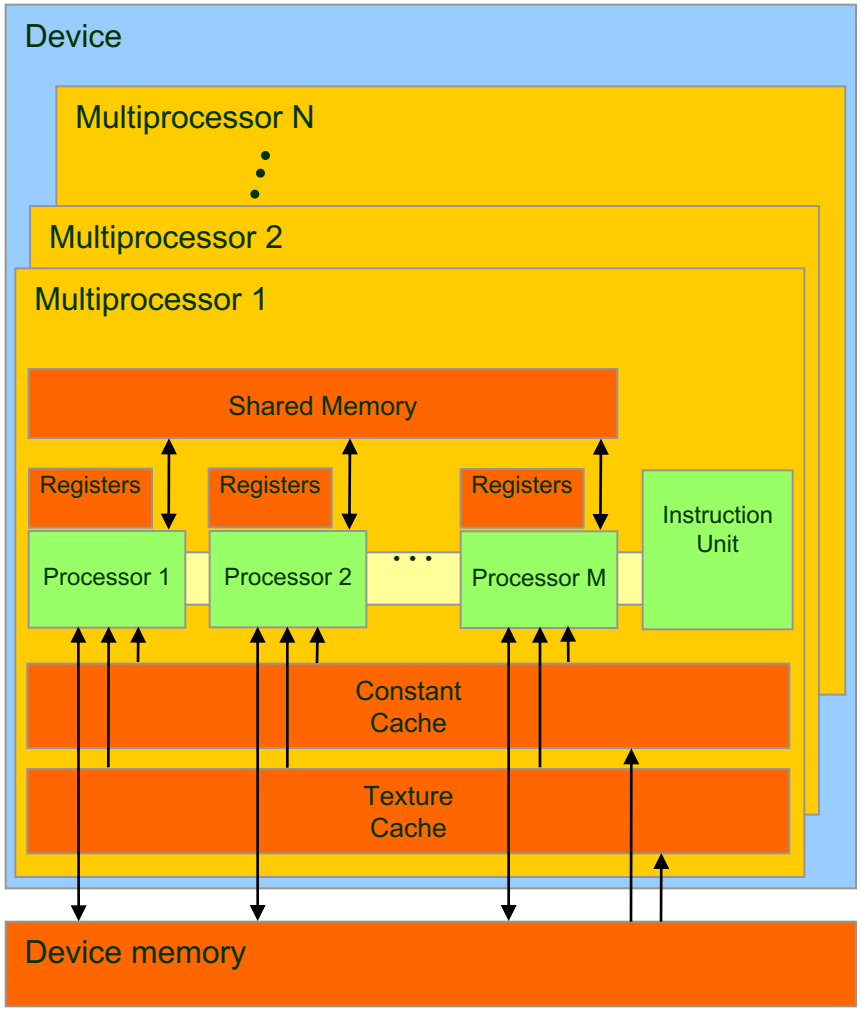
CUDA Memory Model Overview

- Each thread can:
  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory
- The host can R/W global, constant, and texture memories



Hardware Implementation: Memory Architecture

- Device memory (DRAM)
  - Slow (2~300 cycles)
  - Local, global, constant, and texture memory
- On-chip memory
  - Fast (1 cycle)
  - Registers, shared memory, constant/texture cache



Courtesy NVIDIA

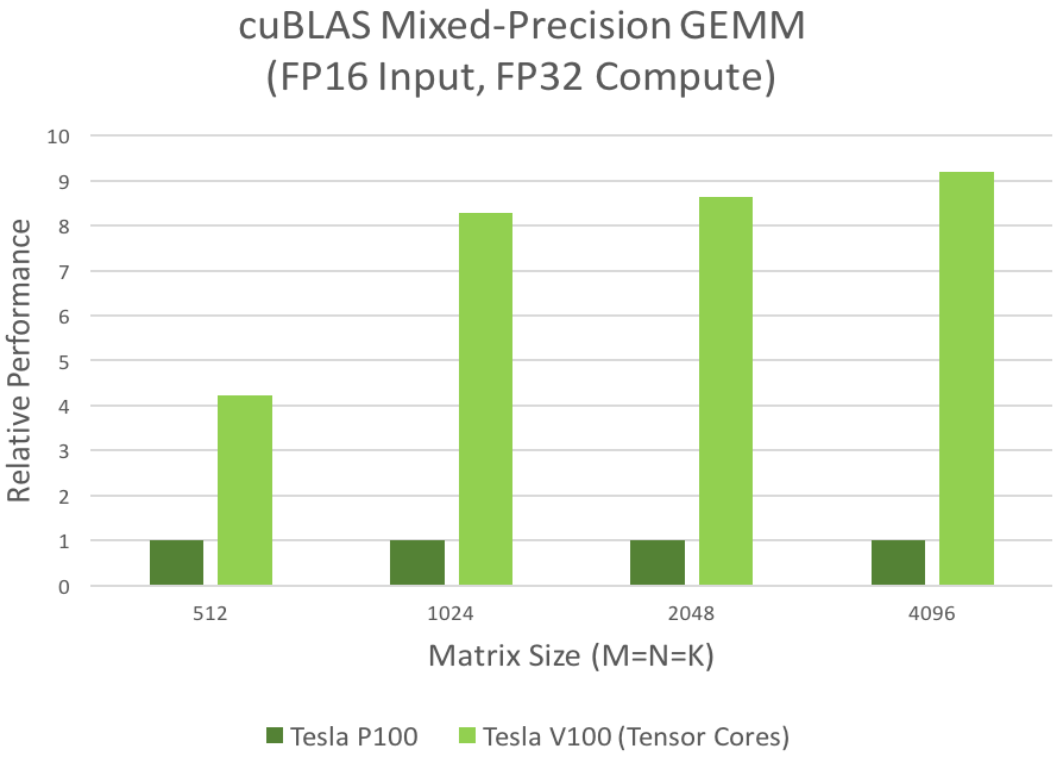
Programming Tensor Cores in CUDA

Tensor Cores in CUDA Libraries

*Two CUDA libraries that use Tensor Cores are **cuBLAS** and **cuDNN**. cuBLAS uses Tensor Cores to speed up GEMM computations (GEMM is the BLAS term for a matrix-matrix multiplication); cuDNN uses Tensor Cores to speed up both convolutions and recurrent neural networks (RNNs).*

How to Use Tensor Cores in cuBLAS

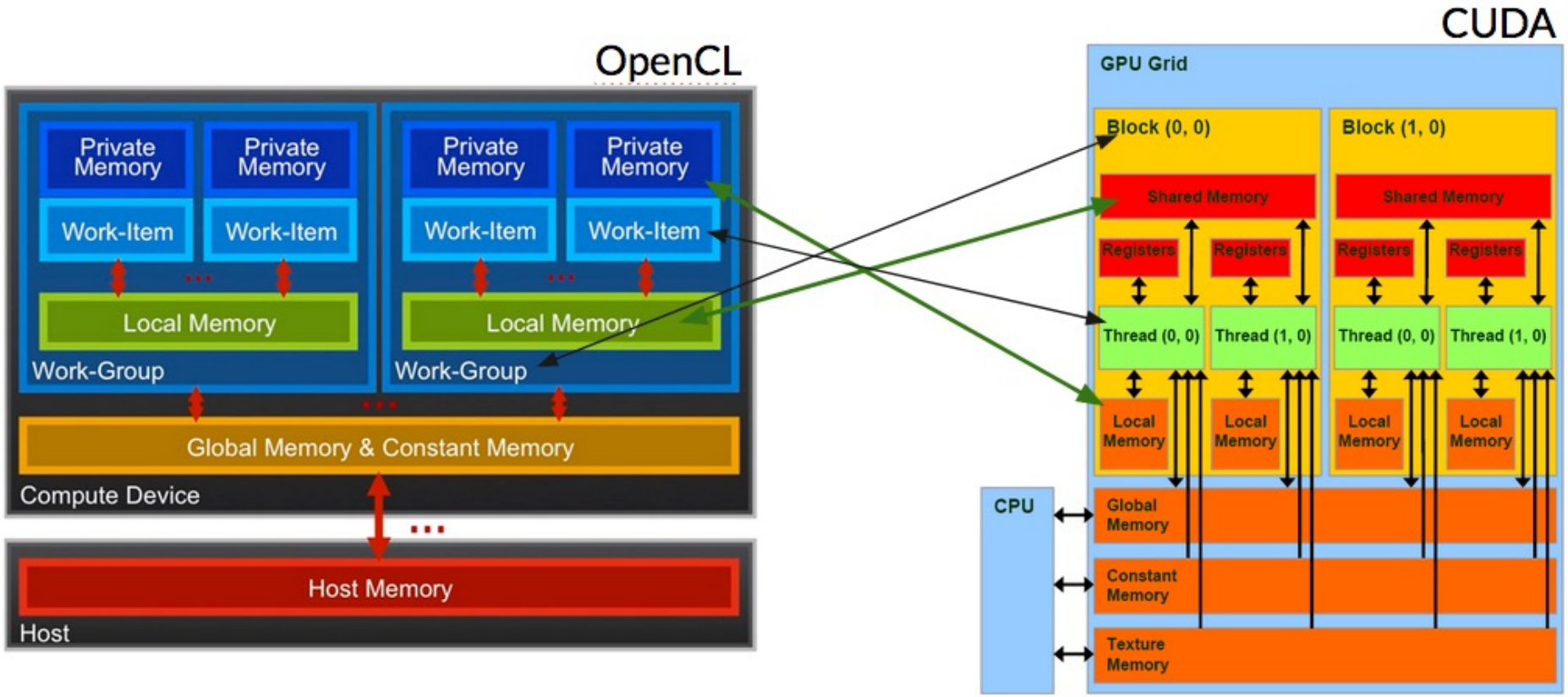
You can take advantage of Tensor Cores by making a few changes to your existing cuBLAS code. The changes are small changes in your use of the cuBLAS API.



Courtesy NVIDIA

https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

Terminology: CUDA and OpenCL



SYCL: an alternative to CUDA/OpenCL

WIKIPEDIA
The Free Encyclopedia

SYCL

From Wikipedia, the free encyclopedia

SYCL is a higher-level programming model to improve programming productivity on various hardware accelerators. It is a single-source embedded domain-specific language (eDSL) based on pure C++17. It is a standard developed by Khronos Group, announced in March 2014.

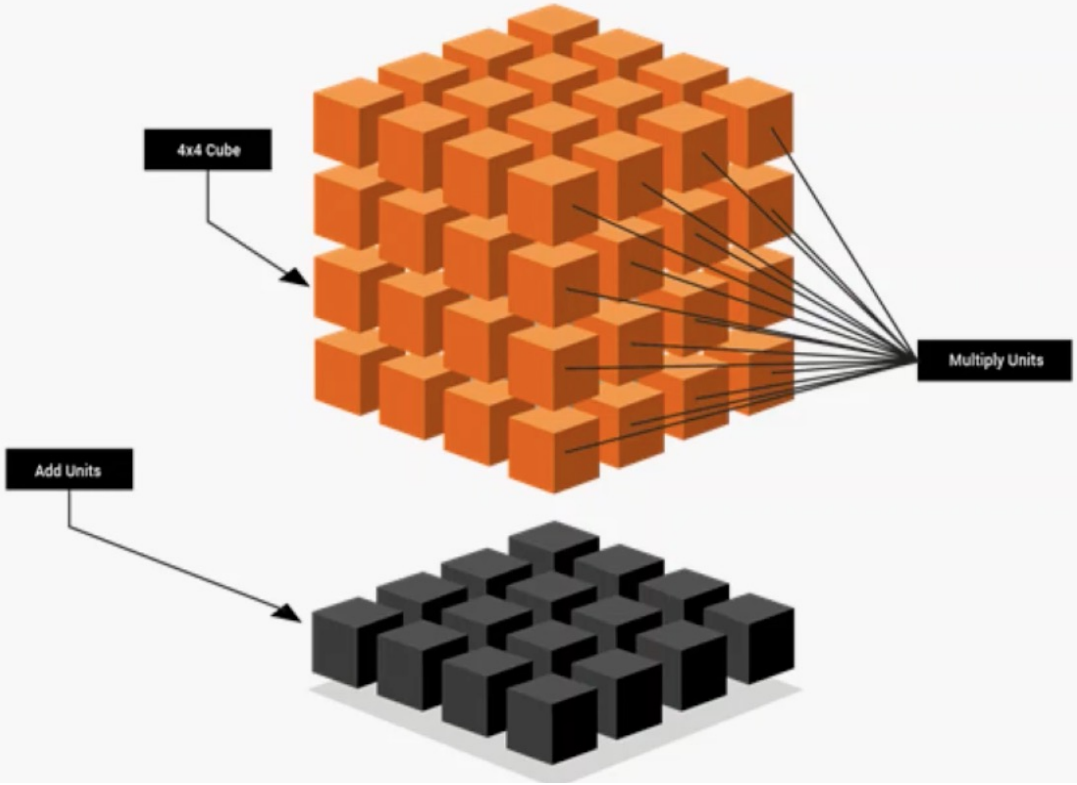
Contents [hide]

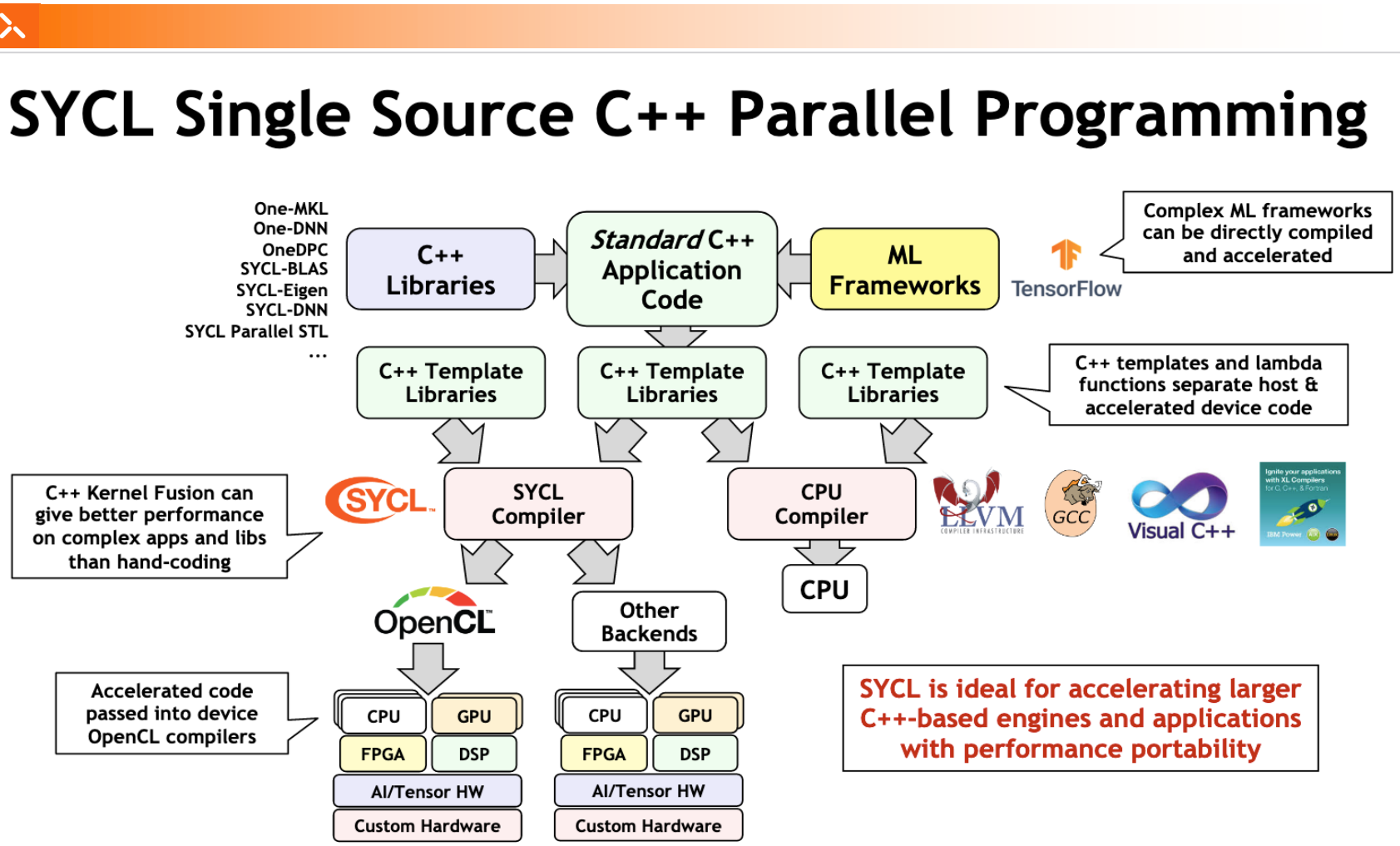
SYCL

Original author(s)	Khronos Group
Developer(s)	Khronos Group
Initial release	March 2014; 8 years ago
Stable release	2020 revision 5 / May 10, 2022; 5 months ago
Operating system	Cross-platform
Platform	Cross-platform
Type	High-level programming language
Website	www.khronos.org/sycl/ sycl.tech

SYCL 2020 and Beyond

- Ensuring SYCL 2020 feature support in DPC++ for CUDA, including:
  - Unified Shared Memory (USM.)
  - Reductions.
  - Group and sub-group operations
- Planned SYCL extensions:
  - Asynchronous barriers.
  - Tensor Core support.
- Better DPC++ for CUDA multi-device support.





Intel oneAPI: a SYCL implementation

WIKIPEDIA
The Free Encyclopedia

oneAPI (compute acceleration)

From Wikipedia, the free encyclopedia

oneAPI is an open standard for a unified application programming interface intended to be used across different compute accelerator (coprocessor) architectures, including GPUs, AI accelerators and field-programmable gate arrays. It is intended to eliminate the need for developers to maintain separate code bases, multiple programming languages, and different tools and workflows for each architecture.

Data Parallel C++ [edit]

DPC++[7][8] is an open, cross-architecture language built upon the ISO C++ and Khronos Group SYCL standards.[9] DPC++ is an implementation of SYCL with extensions that are proposed for inclusion in future revisions of the SYCL standard. An example of this is the contribution of unified shared memory, group algorithms and sub-groups to SYCL 2020.[10][11][12]

oneAPI libraries [edit]

The set of APIs[5] spans several domains that benefit from acceleration, including libraries for linear algebra math, deep learning, machine learning, video processing, and others.

Library Name	Short Name	Description
oneAPI DPC++ Library	oneDPL	Algorithms and functions to speed DPC++ kernel programming
oneAPI Math Kernel Library	oneMKL	Math routines including matrix algebra, FFT, and vector math
oneAPI Data Analytics Library	oneDAL	Machine learning and data analytics functions
oneAPI Deep Neural Network Library	oneDNN	Neural networks functions for deep learning training and inference
oneAPI Collective Communications Library	oneCCL	Communication patterns for distributed deep learning
oneAPI Threading Building Blocks	oneTBB	Threading and memory management template library
oneAPI Video Processing Library	oneVPL	Real-time video encode, decode, transcode, and processing