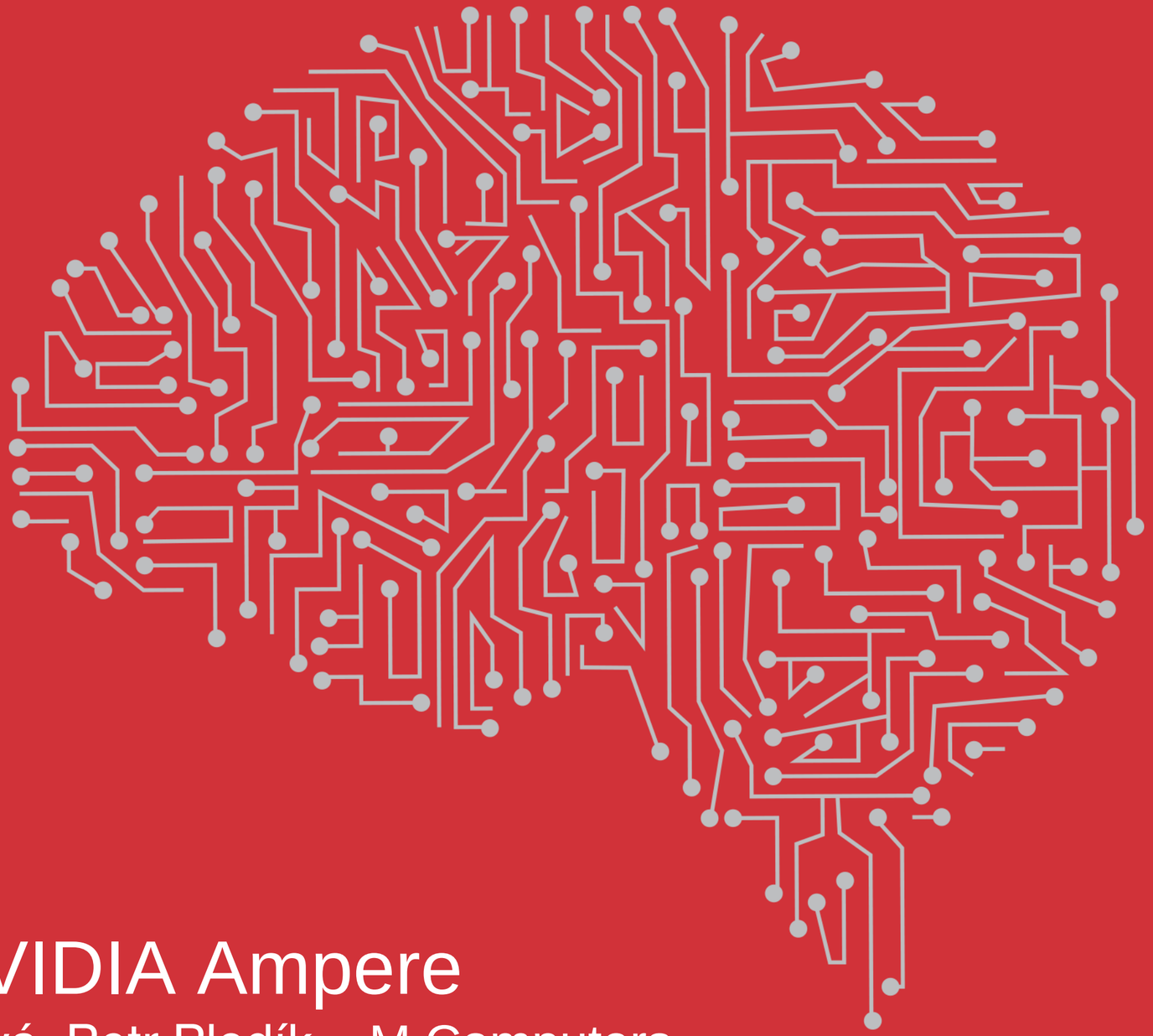


Do nitra NVIDIA Ampere

Kamila Jeřábková, Petr Plodík – M Computers

DL TRAINING: FROM SINGLE GPU TO MULTI-NODE

36000 Mins (25 Days)
1xK80 | 2015
CUDA

2015

1200 Mins (20 Hours)
DGX-1P | 2016
NVLink

2016

480 Mins (8 Hours)
DGX-1V | 2017
Tensor Core

2017

70 Minutes on MLPerf
DGX-2H | 2018
NVSwitch

6.3 Minutes on MLPerf
At Scale | 2018
DGX Cluster

2018

52.7 Minutes on MLPerf
DGX-2H | 2019
NVSwitch

1.33 Minutes on MLPerf
At Scale | 2019
DGX SuperPOD

2019

ResNet50 v1.5 training

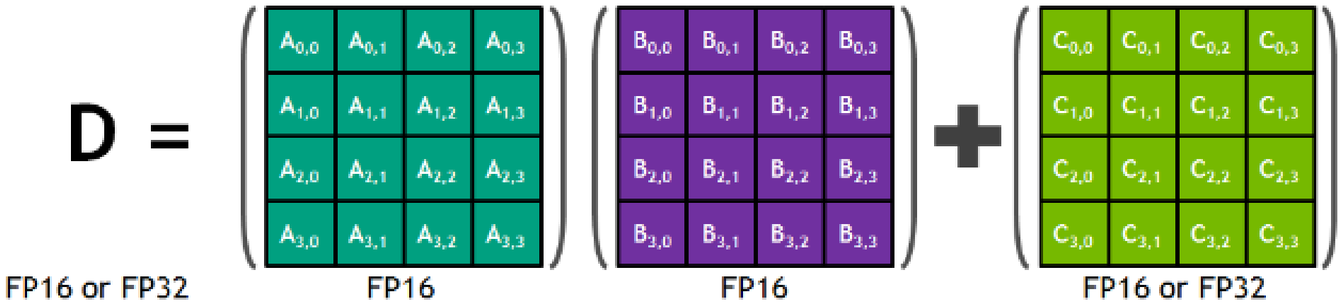
NVIDIA Tesla V100

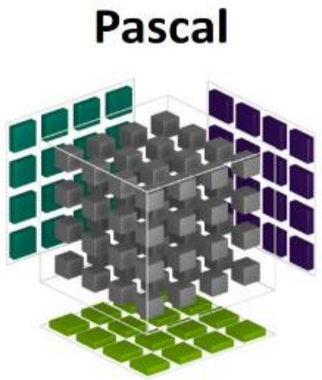
NVIDIA Tesla V100 s 80 SM jednotkami

Celkem 5 120× FP32, 2 560× FP64 jader, 640× tensor jader

Streaming Multiprocessor (SM) jednotka:
64× FP32 jader, 32× FP64 jader

Tensor jádra NVIDIA Tesla V100





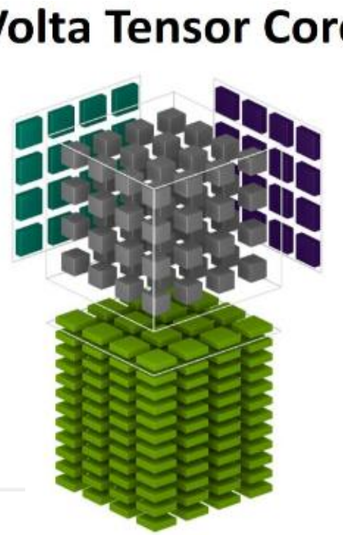
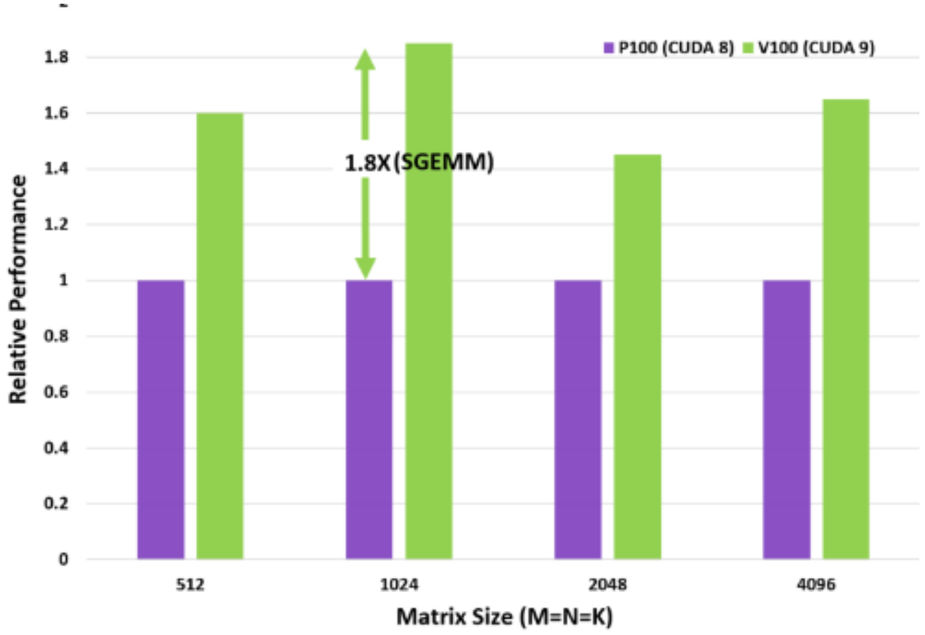
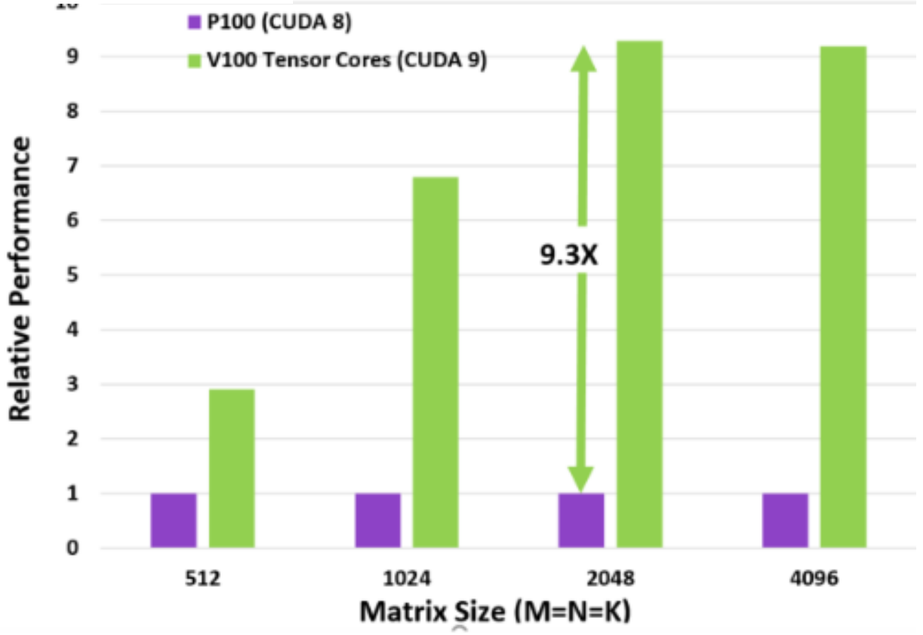


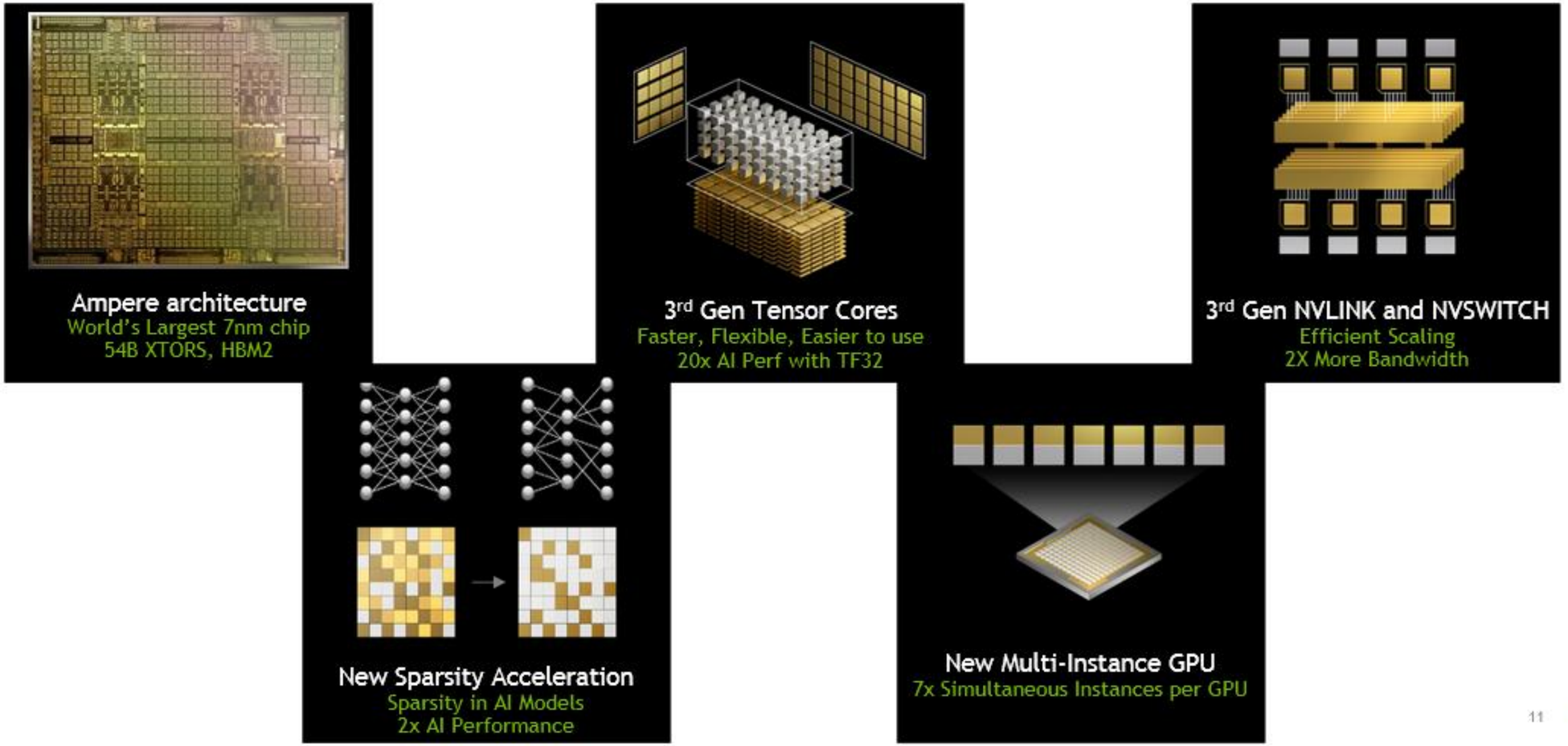
Figure 8. Tensor Core 4x4 Matrix Multiply and Accumulate





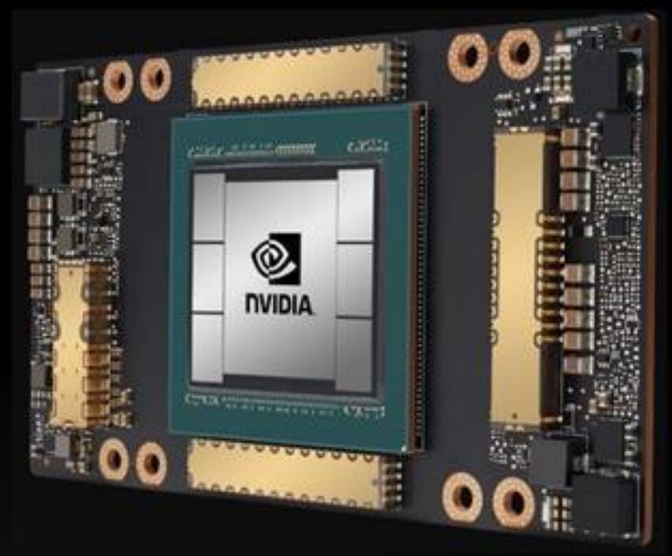
A NEW GENERATION OF GPUS

NVIDIA A100



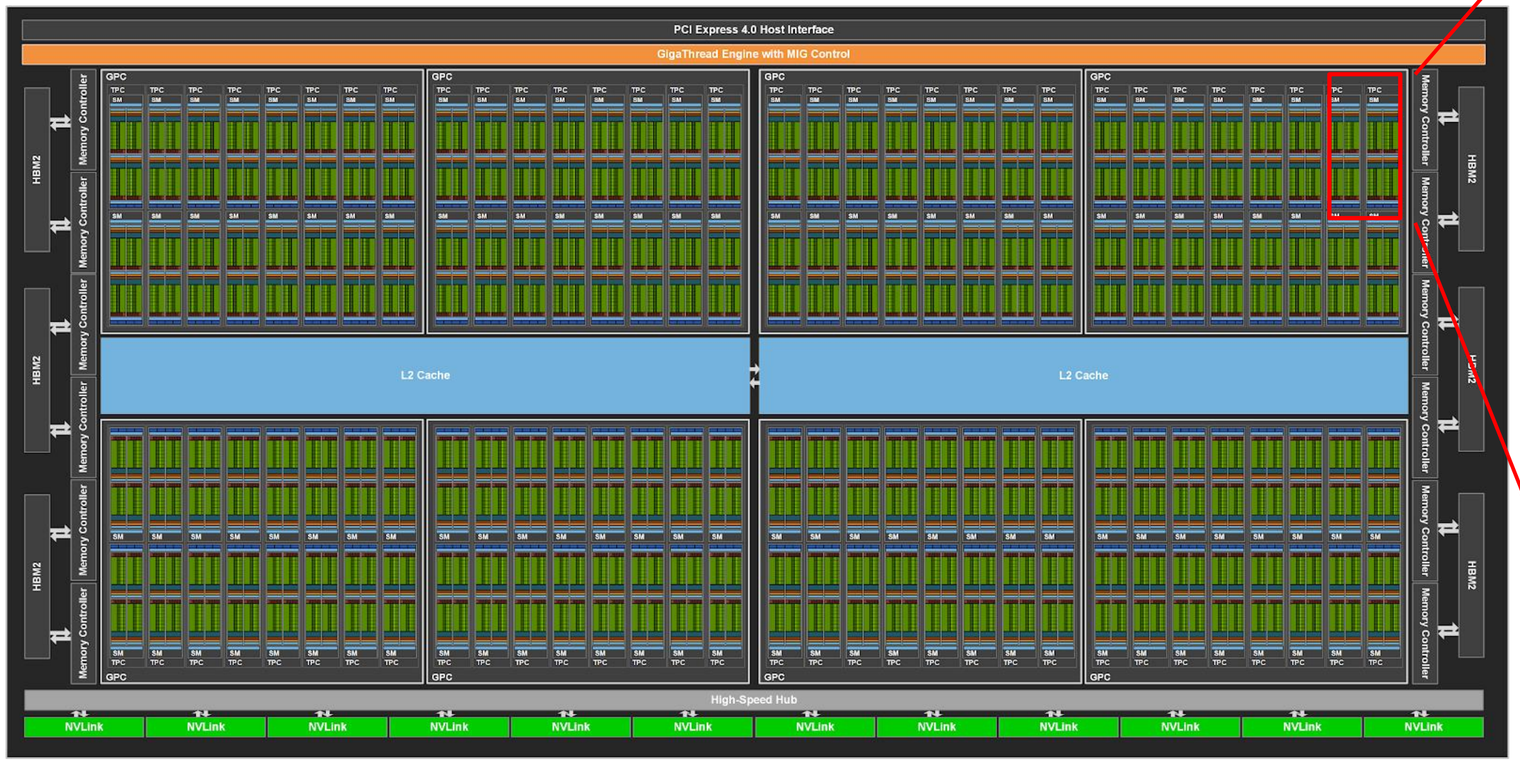
NVIDIA AMPERE GPU ARCHITECTURE

	V100	A100
SMs	80	108
Tensor Core Precision	FP16, I8, I4, B1	FP64, TF32, BF16, FP16, I8, I4, B1
Shared Memory per Block	96 kB	160 kB
L2 Cache Size	6144 kB	40960 kB
Memory Bandwidth	900 GB/sec	1600 GB/sec
NVLink Interconnect	300 GB/sec	600 GB/sec



NVIDIA A100

NVIDIA A100 se 108 SM jednotkami



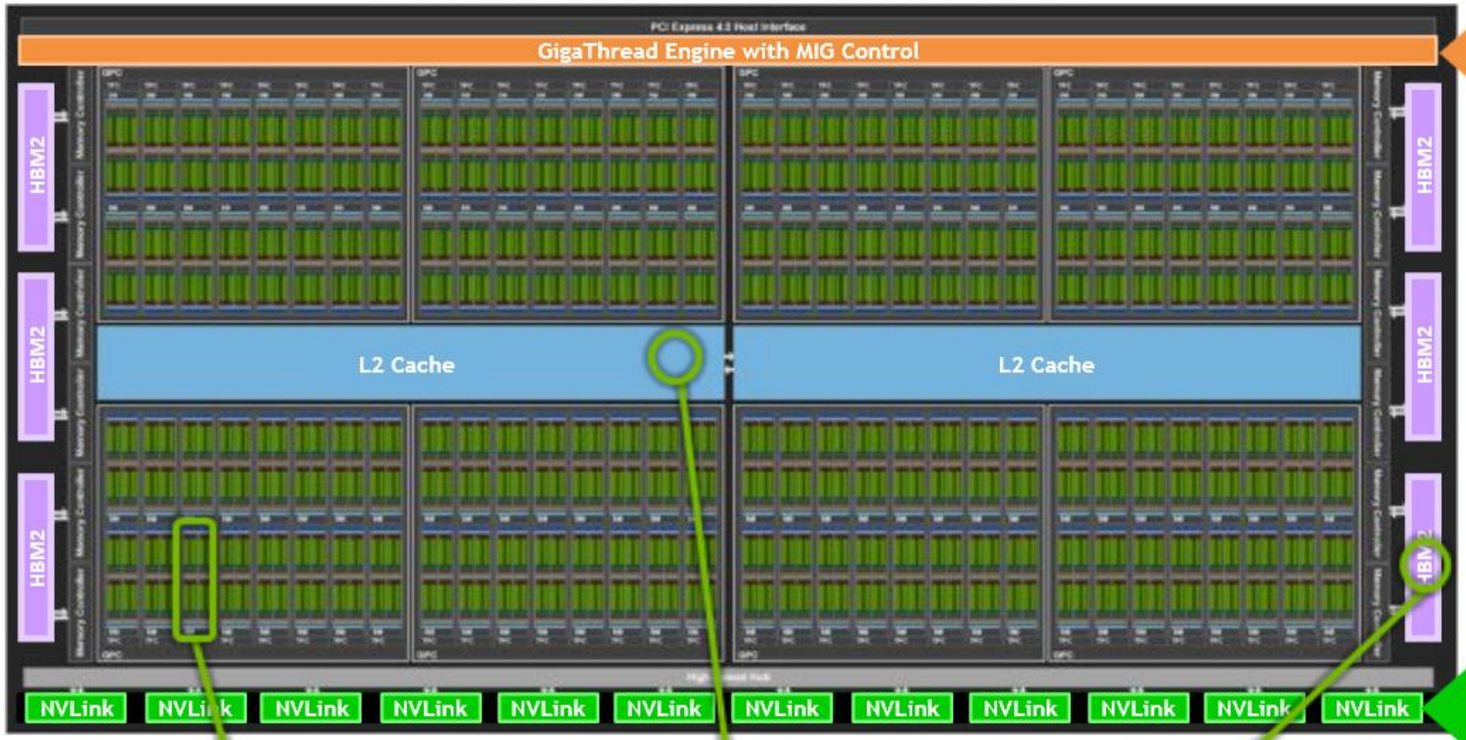
Celkem 6 912× FP32, 432× tensor jader

Streaming Multiprocessor (SM) jednotka:
64× FP32 jader, 4× tensor jadra

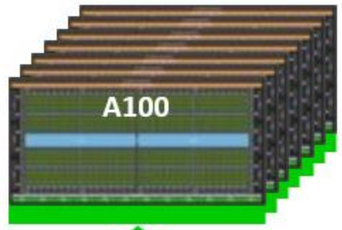
First

A100 TENSOR-CORE GPU

54 billion transistors in 7nm



7x
Scale OUT
Multi-Instance GPU



2x BW
Scale UP
3rd gen.
NVLINK

108 SMs
6912 CUDA Cores

40MB L2
6.7x capacity

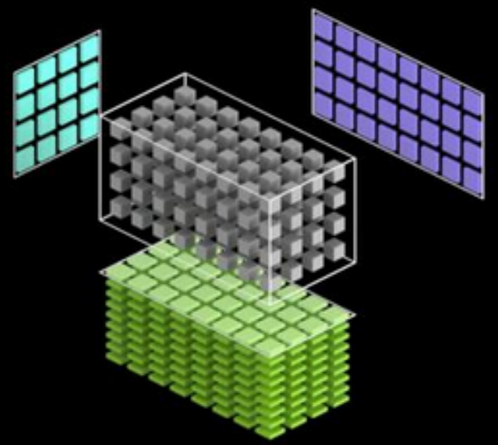
1.56 TB/s HBM2
1.7x bandwidth

NVIDIA

A100 SM

- Third-generation Tensor Core
  - *Faster and more efficient*
  - *Comprehensive data types*
  - *Sparsity acceleration*
- Asynchronous data movement and synchronization
- Increased L1/SMEM capacity

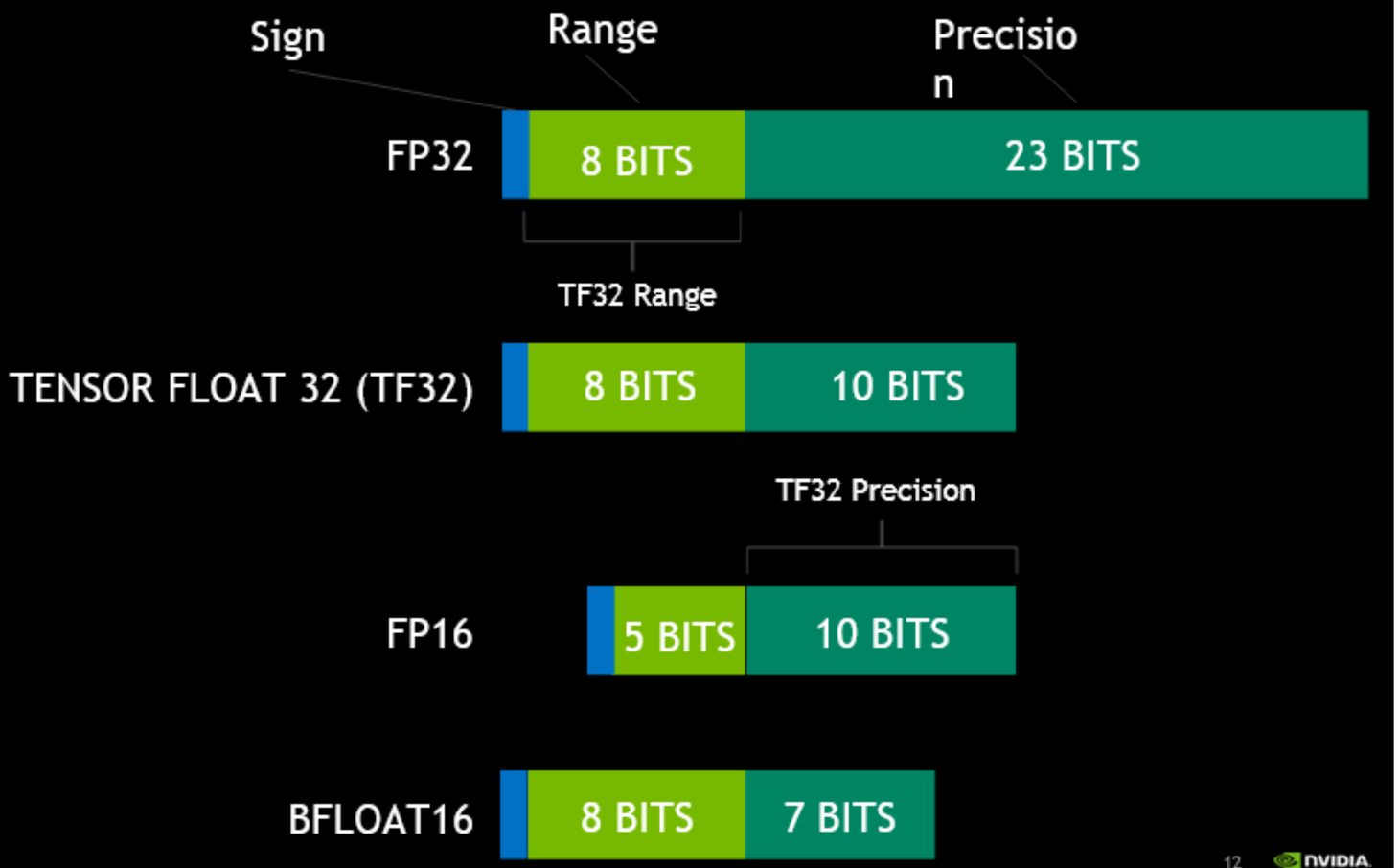
NEW TF32 TENSOR CORES



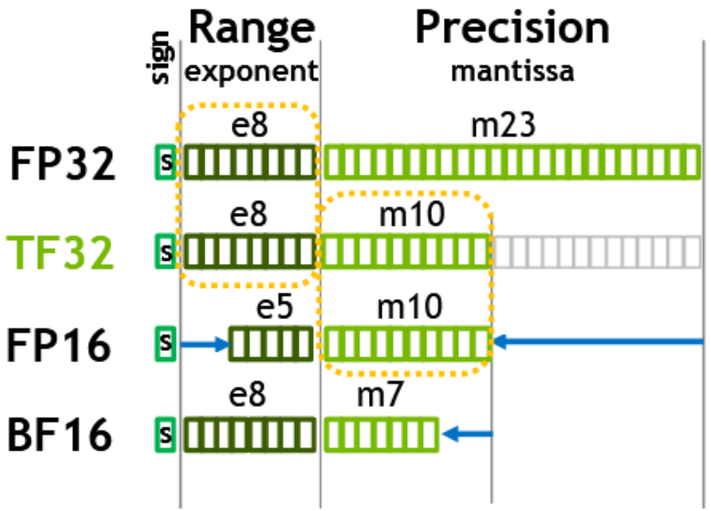
Range of FP32 and Precision of FP16

Input in FP32 and Accumulation in FP32

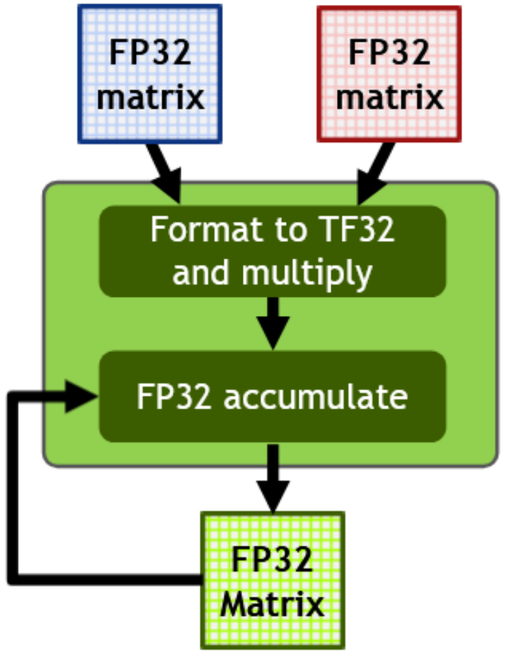
No Code Change Speed-up for Training



INSIDE A100 TensorFloat-32 (TF32)



Range of FP32 with precision of FP16



FP32 input/output

FP32 storage and math for all activations, gradients, …
everything outside tensor cores

Out-of-the-box tensor core acceleration for DL

Easy step towards maximizing tensor core performance with mixed-precision (FP16, BF16)

Up to 4x speedup on linear solvers for HPC

V100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x

V100

125 TOPS FF16/FP32 Mixed-precision

8x vs. FFMA

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA
V100	FP32	FP32	15.7	1x
	FP16	FP32	125	8x
A100	FP32	FP32	19.5	1x
	TF32	FP32	156	8x
	FP16	FP32	312	16x
	BF16	FP32	312	16x

TF32 accelerates FP32 in/out data → 10x vs. V100 FP32

BFloat16 (BF16) at same rate as FP16

A100 TENSOR CORE

	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x		
	IEEE FP64		19.5	1x		

V100→A100
2.5x FLOPS for HPC

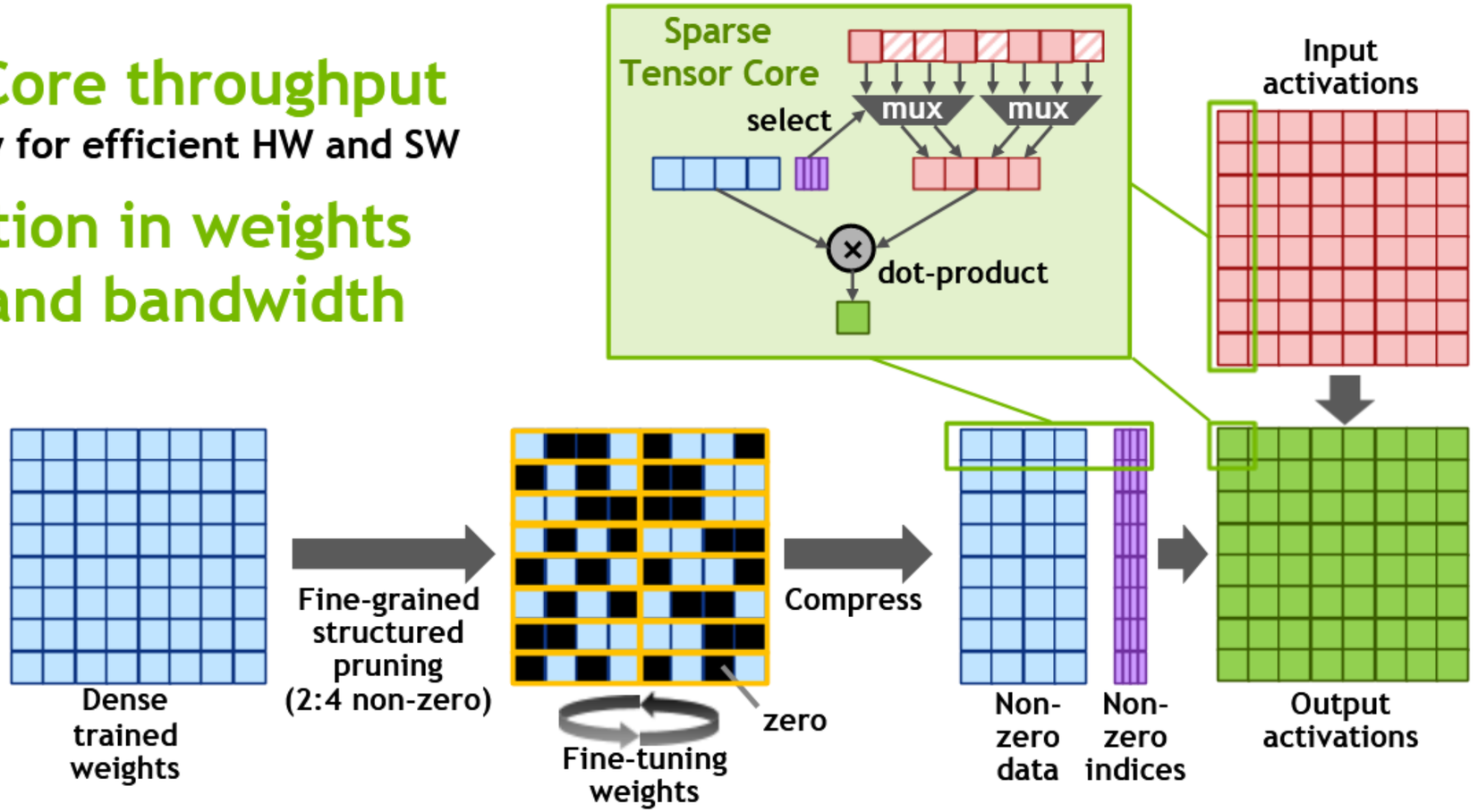
NVIDIA

INSIDE A100 SPARSE TENSOR CORE

2x Tensor Core throughput

Structured-sparsity for efficient HW and SW

~2x reduction in weights footprint and bandwidth

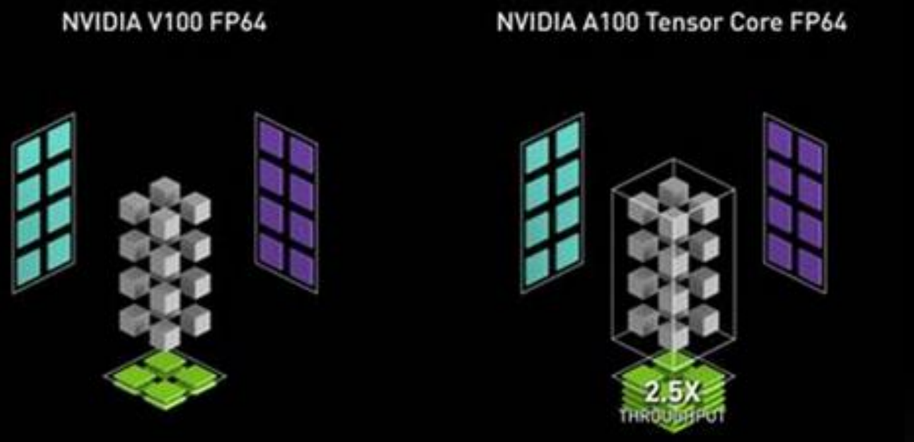


~No loss in inferencing accuracy

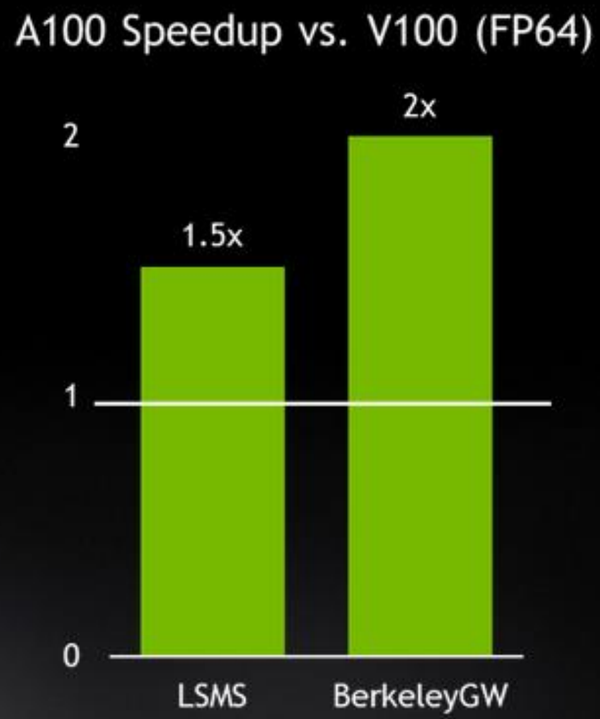
Evaluated across dozens of networks: vision, object detection, segmentation, natural language modeling, translation

A100 FP64 Tensor Cores

For even more Performance

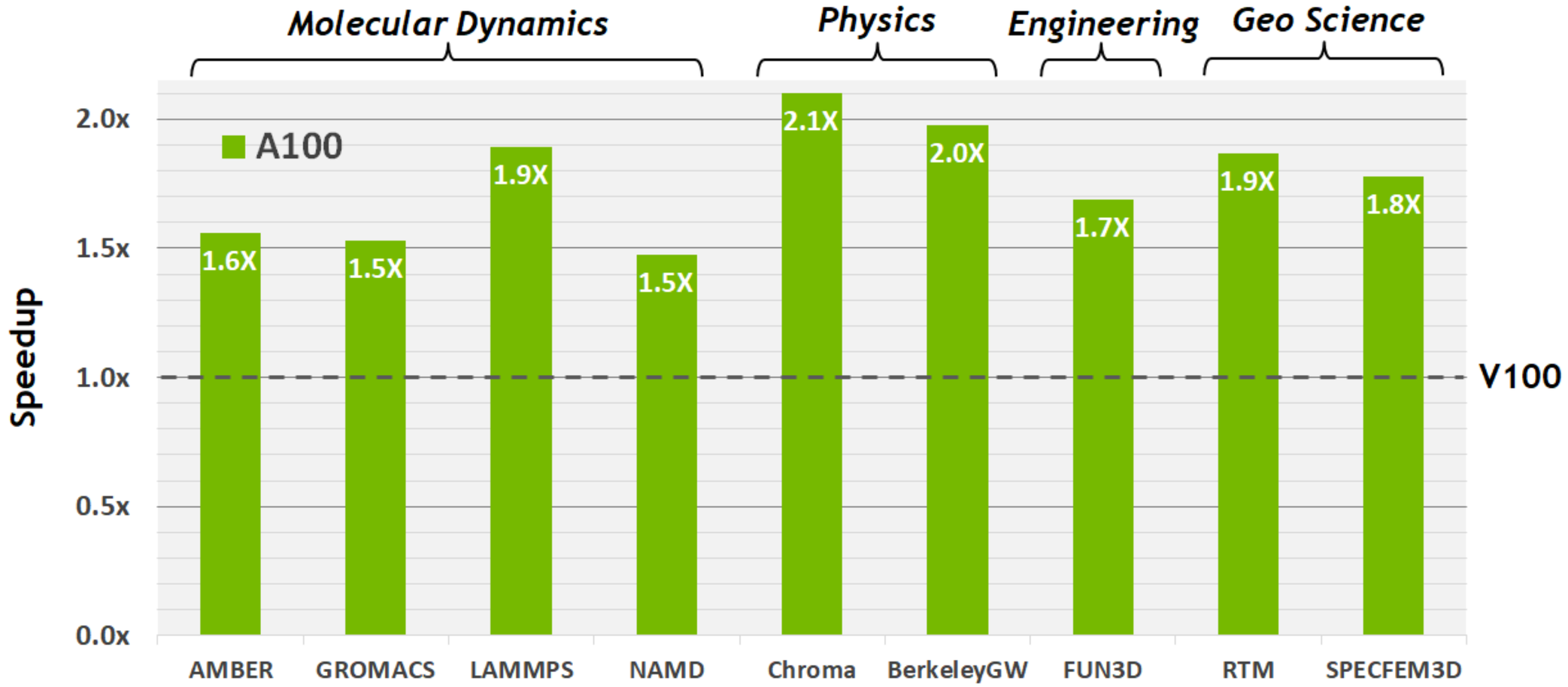


- IEEE FP64
- Drop-in acceleration via cuBLAS, cuTensor, and cuSolver



Application [Benchmarks]: BerkeleyGW [Chi Sum + MTXEL] using DGX-1V (8xV100) and DGX-A100 (8xA100) | LSMS [Fe128] single V100 SXM2 vs. A100 SXM4

ACCELERATING HPC



All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE
Chroma with szscl21_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model
BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 in DGX A100

Porovnání NVIDIA GPU akcelerátorů

PARAMETRY	GEFORCE RTX 2080TI	TITAN RTX	QUADRO RTX 5000	QUADRO RTX 6000 / 8000	TESLA T4	TESLA V100 / V100S PCIE	TESLA V100 SXM2	A100
Architektura	Turing	Turing	Turing	Turing	Turing	Volta	Volta	Ampere
# CUDA jader	4 352	4 608	3 072	4 608	2 560	5 120	5 120	6 912
# Tensor jader	544	576	384	576	320	640	640	432
GPU paměť	11 GB	24 GB	16 GB	24 / 48 GB	16 GB	32 GB	32 GB	40 GB
Paměti	GDDR6	GDDR6	GDDR6	GDDR6	GDDR6	HBM2	HBM2	HBM2
Propustnost pamětí	616 GB / s	672 GB / s	448 GB / s	624 GB / s	300 GB / s	900 GB/s / 1 134 GB/s	900 GB / s	1,5 TB/s
ECC paměti	není	není	ECC	ECC	ECC	ECC	ECC	ECC
Max. příkon	250 W	280 W	265 W	295 W	70 W	250 W	300 W	400 W
Provedení	PCIe gen3	PCIe gen3	PCIe gen3	PCIe gen3	PCIe gen3	PCIe gen3	SXM2	SXM4
Pro datacentra	Ne	Ne	Ano	Ano	Ano	Ano	Ano	Ano
Oznámení	2018	2018	2018	2018	2018	2017 / 2019	2017	2020
Listová cena	1 249 USD	2 499 USD	2 809 USD	5 949 / 8 269 USD	2 909 USD	11 839 USD	11 839 USD	součást systému

Porovnání NVIDIA GPU akcelerátorů

PARAMETRY	GEFORCE RTX 2080TI	TITAN RTX	QUADRO RTX 5000	QUADRO RTX 6000 / 8000	TESLA T4	TESLA V100 / V100S PCIE	TESLA V100 SXM2	A100
FP64 (TFlops)	0,4	0,5	0,4	0,5	0,25	7 / 8,2	7,8	9,7
FP64 tensor (Tflops)	-	-	-	--	-		-	19,5
FP32 (TFlops)	13,4	16,3	11,2	16,3	8,1	14 / 16,4	15,7	19,5
TF32 tensor (Tflops)	-	-	-	-	-		-	156 / 312*
FP16 (Tflops)							31,4	78
FP16 tensor (TFlops)	107,6	130	89,2	130	65	112 / 130	125	312 / 624*
BF16 tensor (Tflops)							125	312 / 624*
INT8 tensor (TFlops)					130		62	624 / 1 248*
INT4 tensor (TFlops)					260		-	1 248 / 2 496*
INT Binary (Tflops)							-	4 992
Max. příkon	250 W	280 W	265 W	295 W	70 W	250 W	300 W	400 W
Listová cena	1 249 USD	2 499 USD	2 809 USD	5 949 USD / 8 269 USD	2 909 USD	11 839 USD	11 839 USD	součást systému

~20×

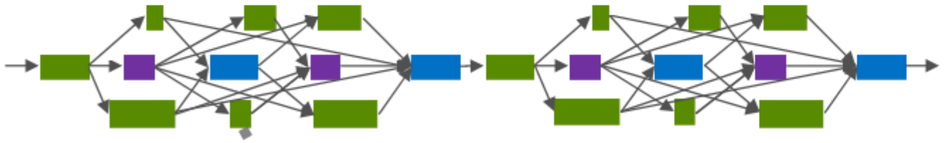
~23×

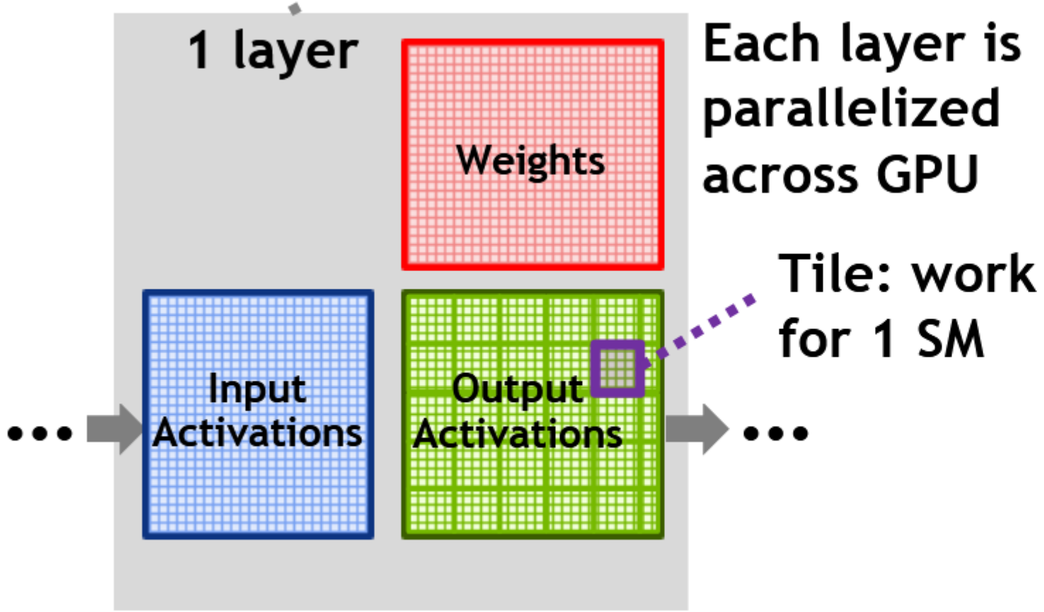
~6×

*** Sparcity – výpočty s řídkými maticemi**

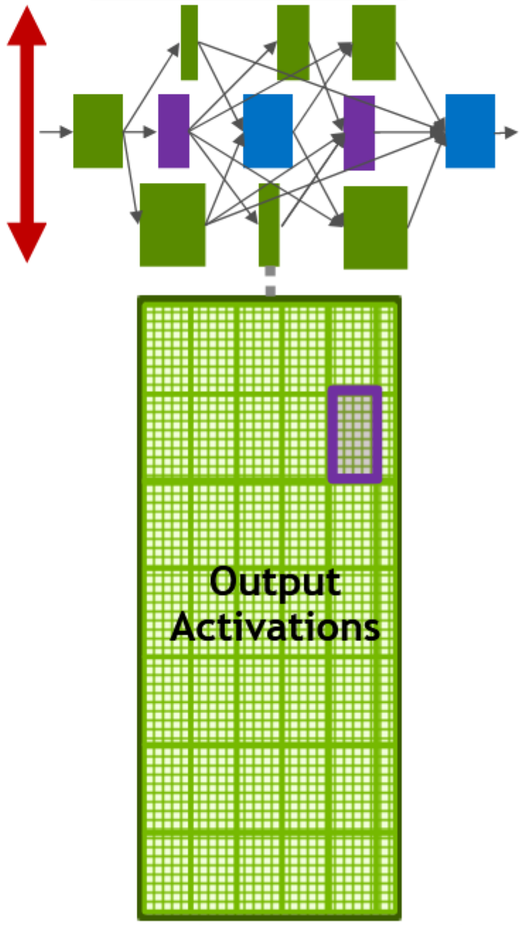
DL STRONG SCALING

DL networks:
Long chains of sequentially-dependent compute-intensive layers



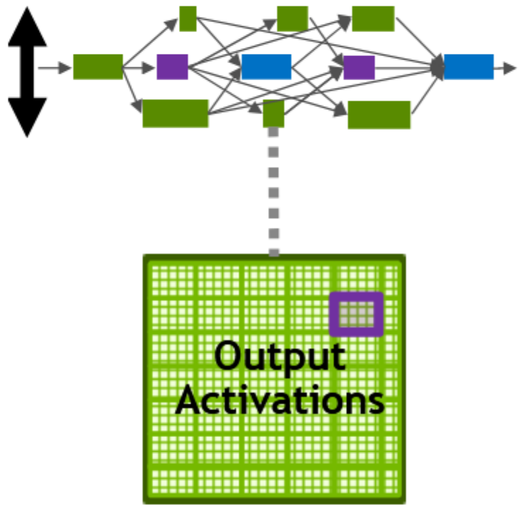


Weak scaling



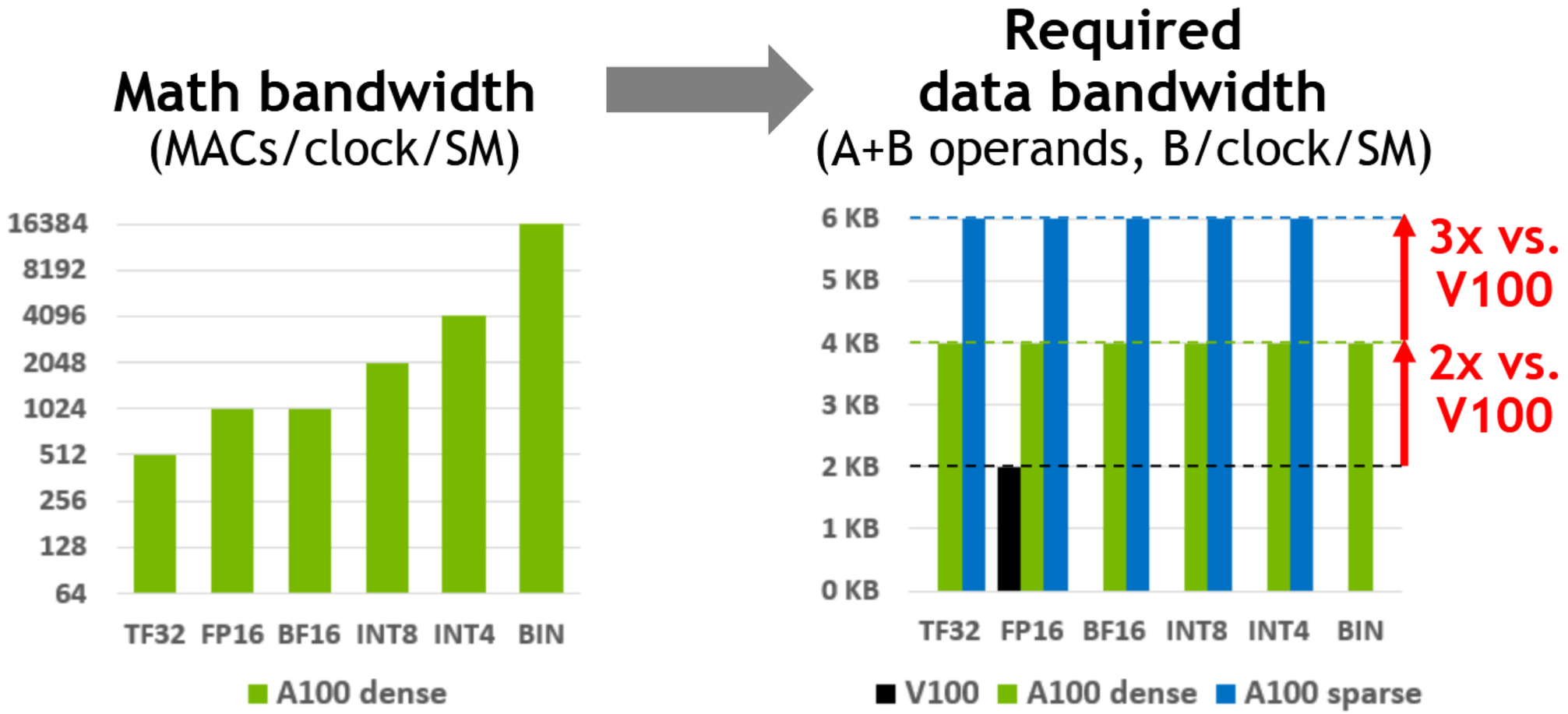
~2.5x larger network runs in same time

Strong scaling

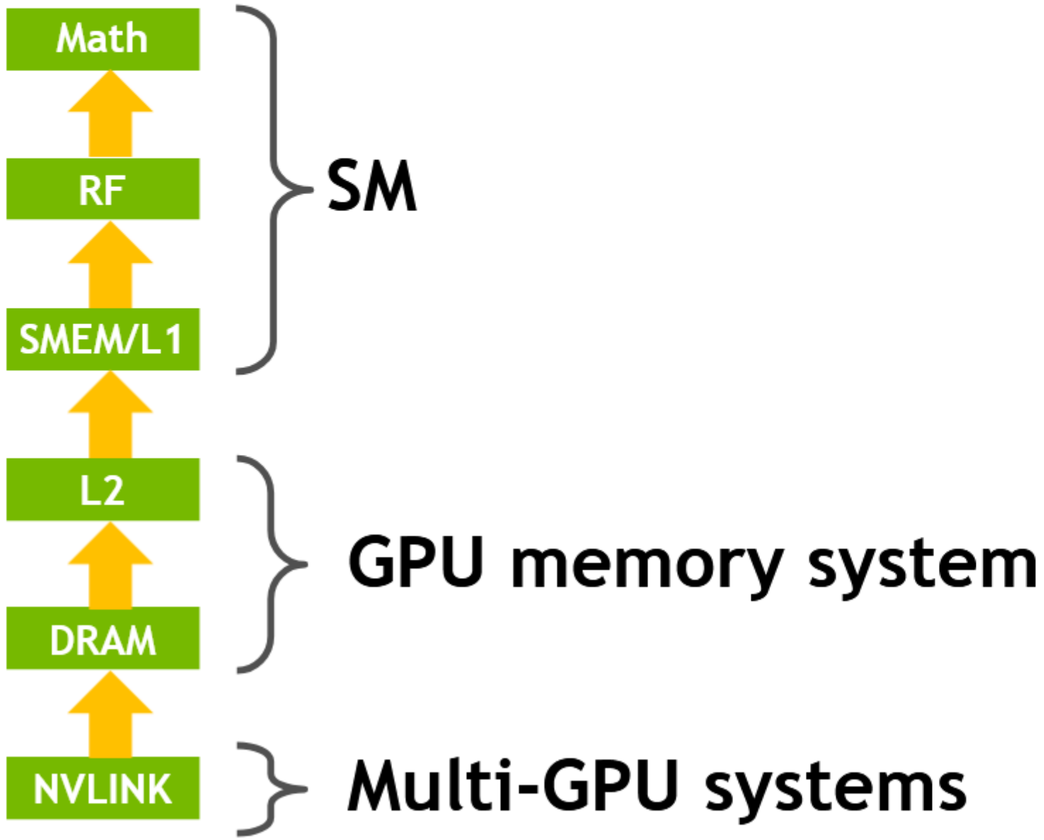


Fixed network runs ~2.5x faster

HOW TO KEEP TENSOR CORES FED?



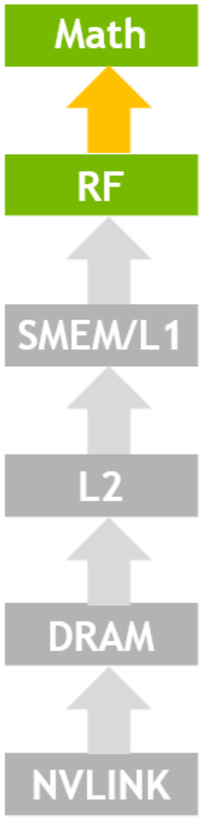
A100 STRONG SCALING INNOVATIONS

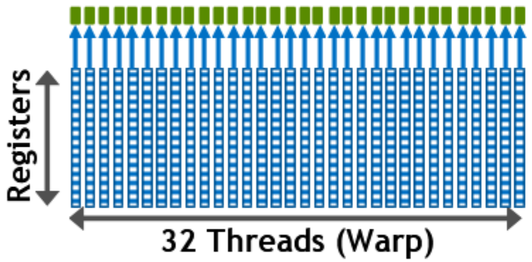


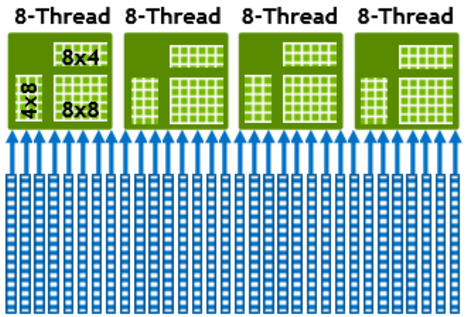
Improve speeds & feeds and efficiency across all levels of compute and memory hierarchy

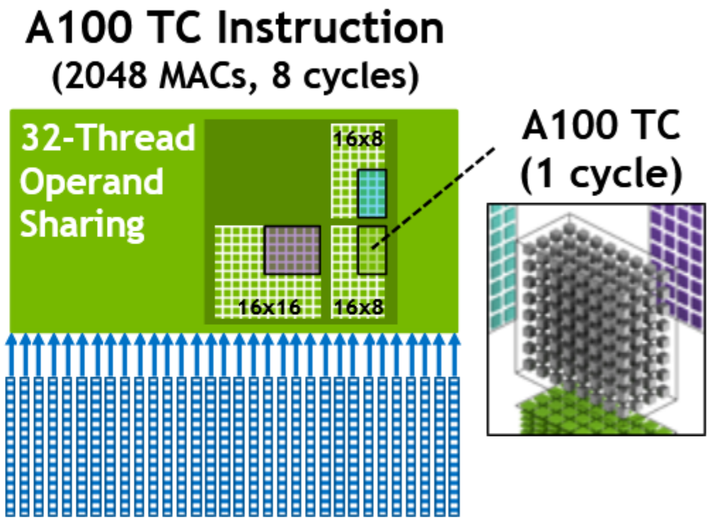
A100 TENSOR CORE

2x throughput vs. V100, >2x efficiency



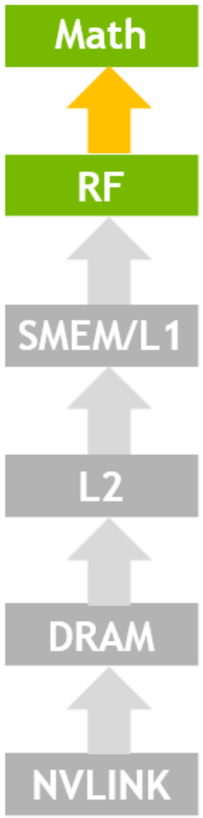


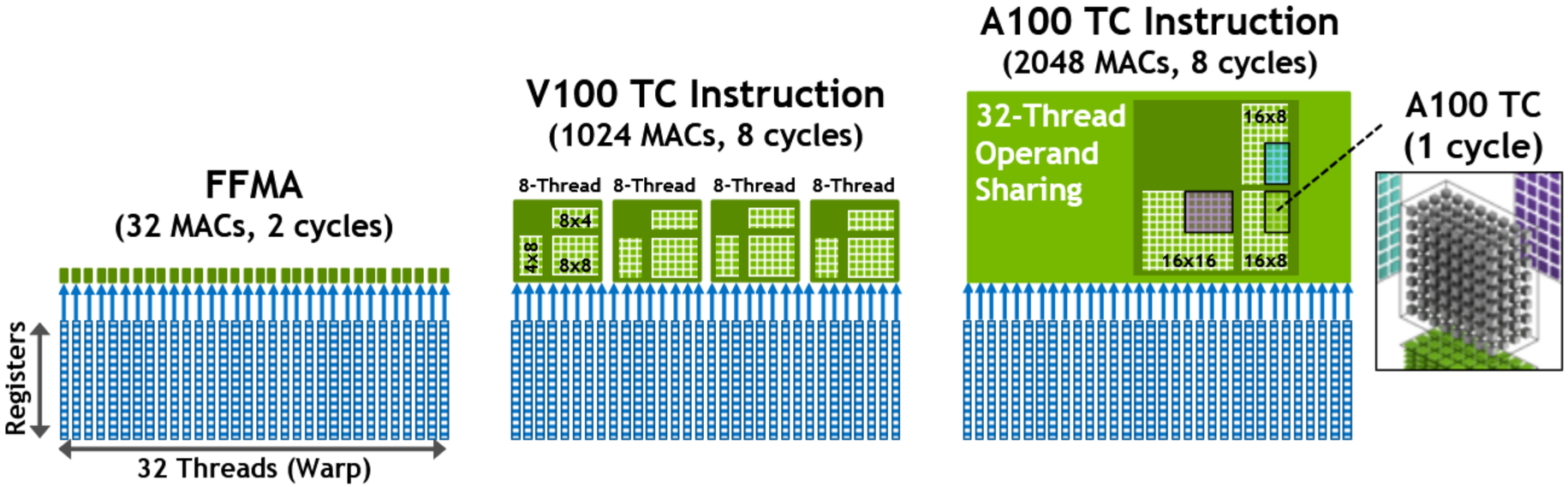




A100 TENSOR CORE

2x throughput vs. V100, >2x efficiency



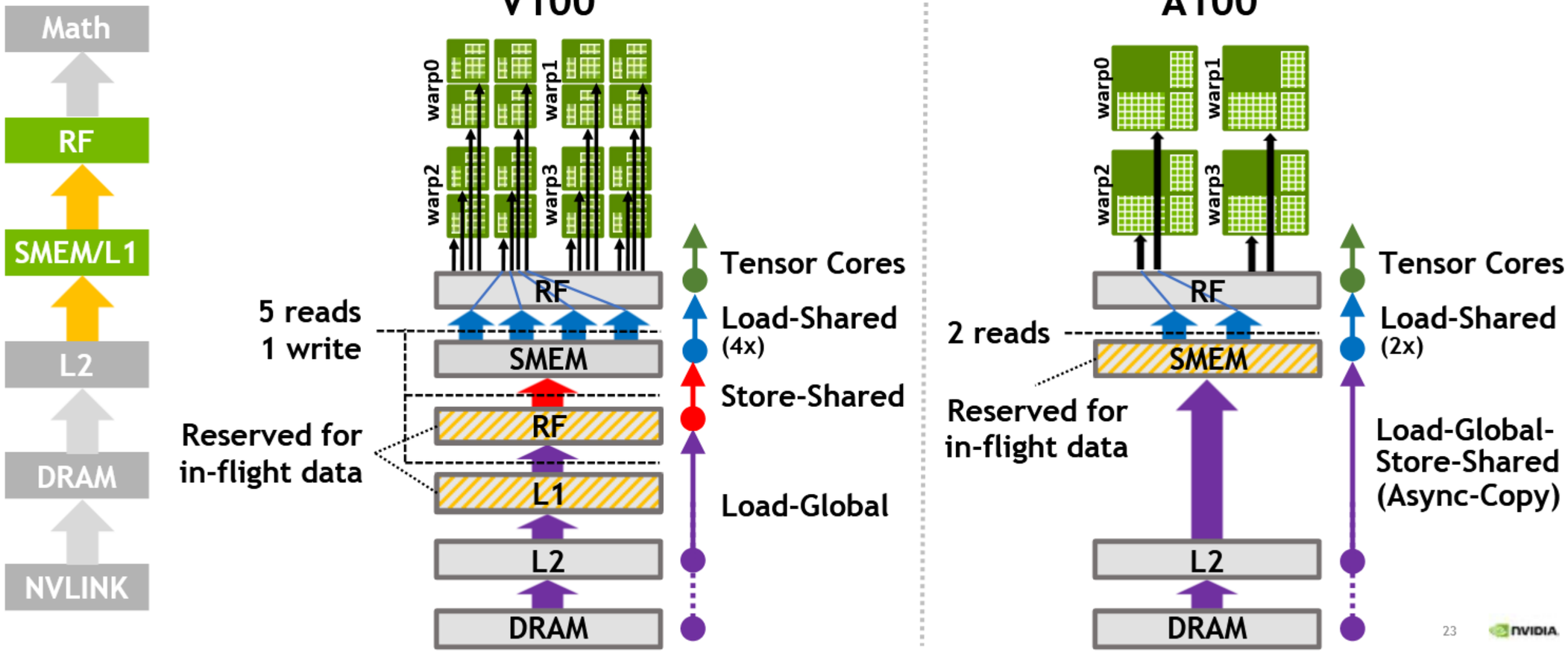


16x16x16 matrix multiply	FFMA	V100 TC	A100 TC	A100 vs. V100 (improvement)	A100 vs. FFMA (improvement)
Thread sharing	1	8	32	4x	32x
Hardware instructions	128	16	2	8x	64x
Register reads+writes (warp)	512	80	28	2.9x	18x
Cycles	256	32	16	2x	16x

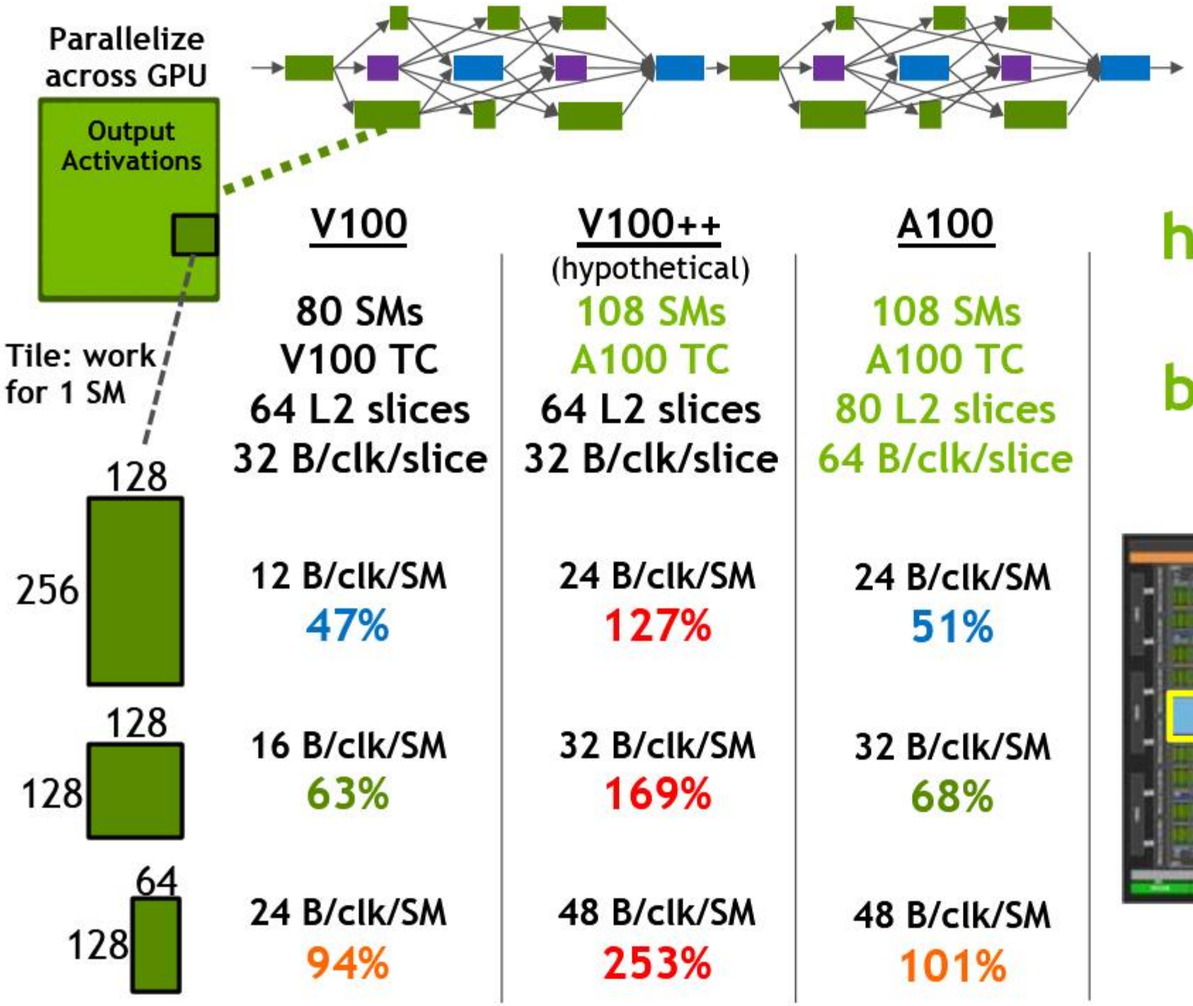
Tensor Cores assume FP16 inputs with FP32 accumulator, V100 Tensor Core instruction uses 4 hardware instructions

A100 SM DATA MOVEMENT EFFICIENCY

3x SMEM/L1 bandwidth, 2x in-flight capacity

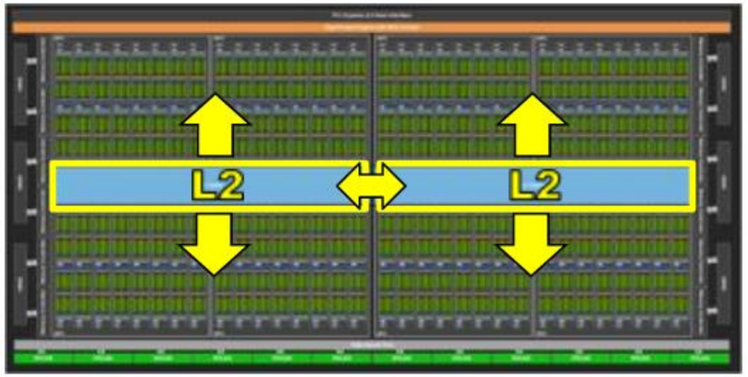


A100 L2 BANDWIDTH



	V100	V100++ (hypothetical)	A100
	80 SMs V100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 64 L2 slices 32 B/clk/slice	108 SMs A100 TC 80 L2 slices 64 B/clk/slice
256	12 B/clk/SM 47%	24 B/clk/SM 127%	24 B/clk/SM 51%
128	16 B/clk/SM 63%	32 B/clk/SM 169%	32 B/clk/SM 68%
64	24 B/clk/SM 94%	48 B/clk/SM 253%	48 B/clk/SM 101%

Split L2 with hierarchical crossbar - 2.3x increase in bandwidth over V100, lower latency



A100 DRAM BANDWIDTH

Faster HBM2

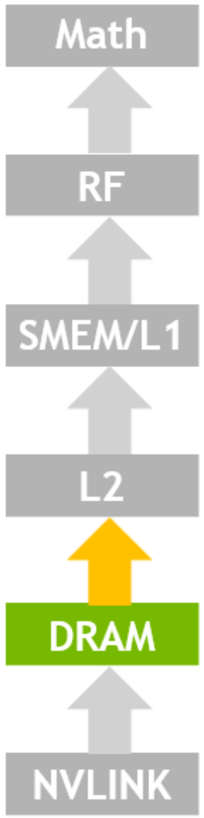
25% more pins, 38% faster clocks

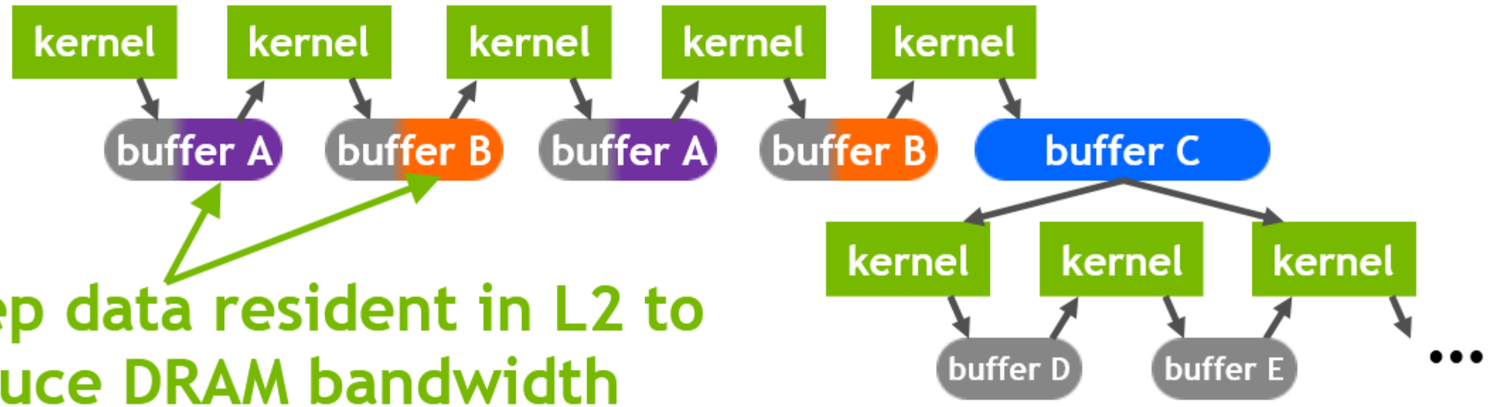
→ 1.6 TB/s, 1.7x vs. V100

Larger and smarter L2

40MB L2, 6.7x vs. V100

L2-Residency controls





Keep data resident in L2 to reduce DRAM bandwidth

→ S21819: Optimizing Applications for NVIDIA Ampere GPU Architecture, 5/21 10:15am PDT

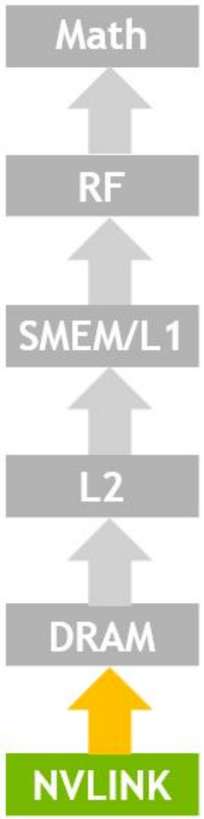
A100 NVLINK BANDWIDTH

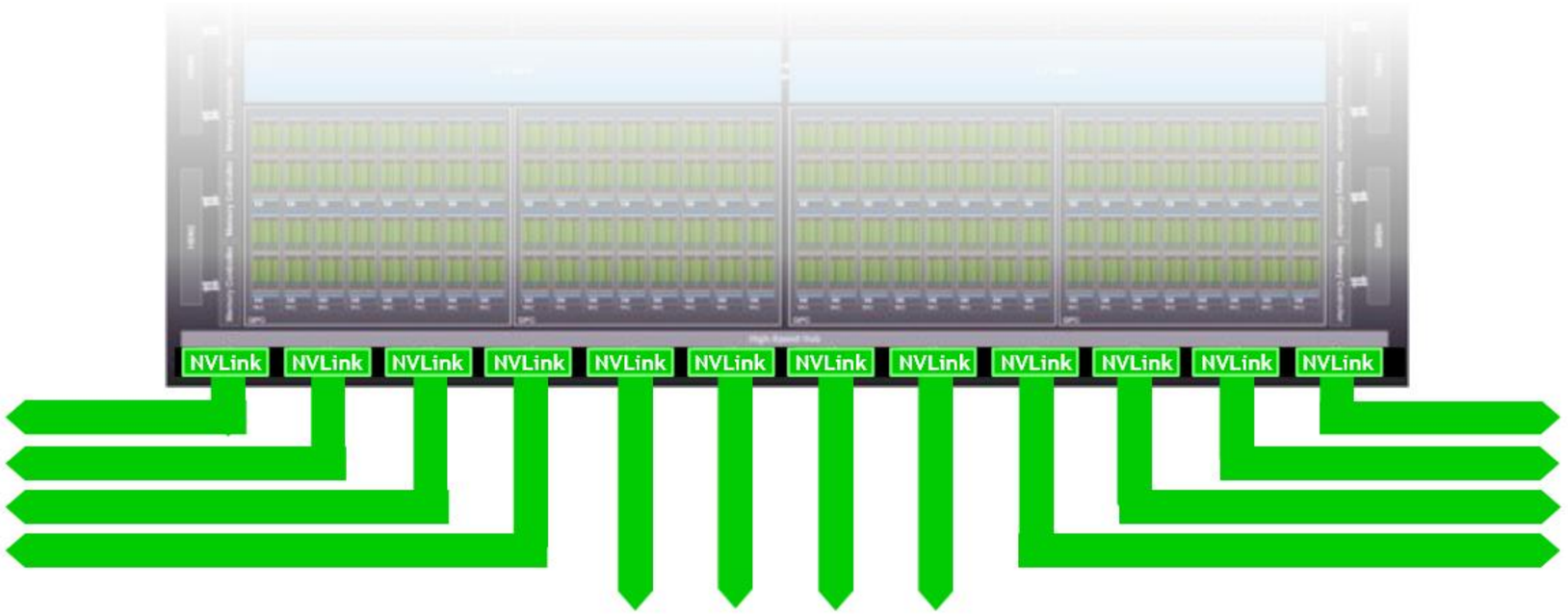
Third Generation NVLink

50 Gbit/sec per signal pair

12 links, 25 GB/s in/out, 600 GB/s total

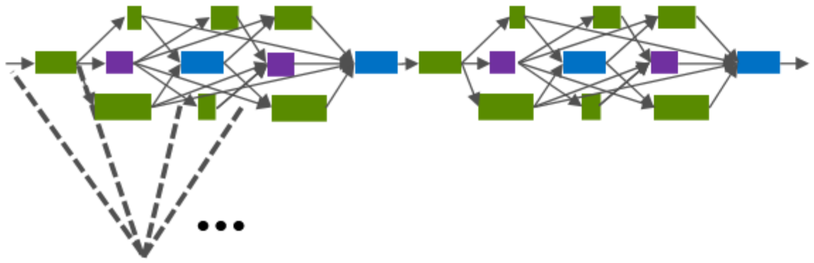
2x vs. V100





→S21884: Under the Hood of the new DGX A100 System Architecture (recording available soon)

A100 ACCELERATES CUDA GRAPHS

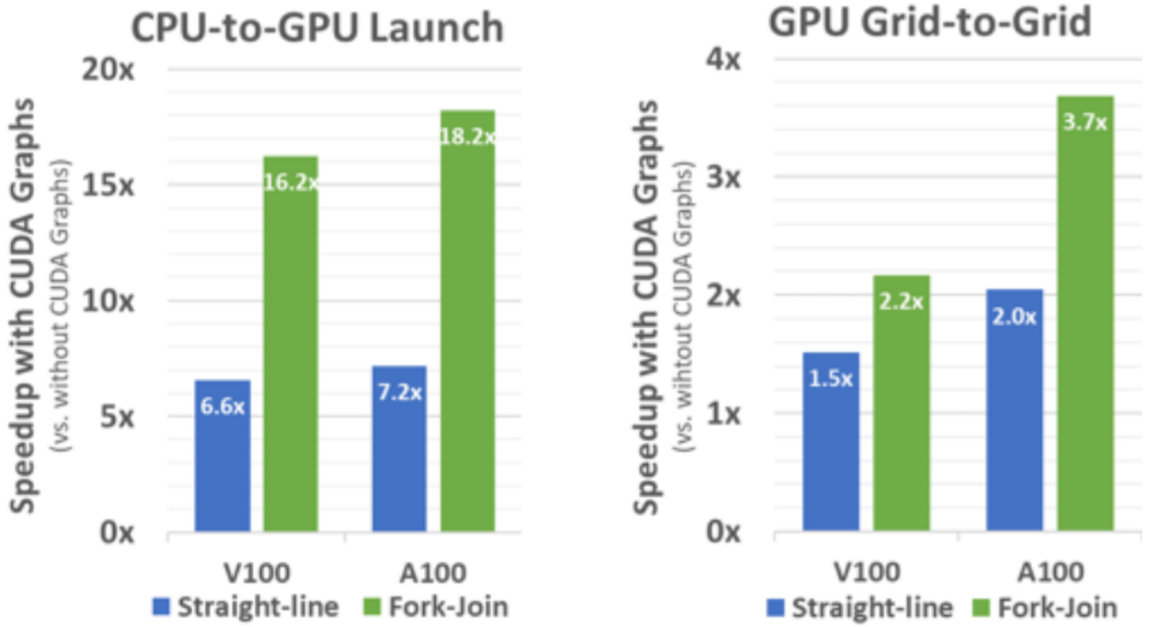


Grid launches:

- *CPU-to-GPU*
- *GPU grid-to-grid*

With strong scaling CPU and grid launch overheads become increasingly important

(Amdahl's law)



32-node graphs of empty grids, DGX1-V, DGX-A100

One-shot CPU-to-GPU graph submission and graph reuse

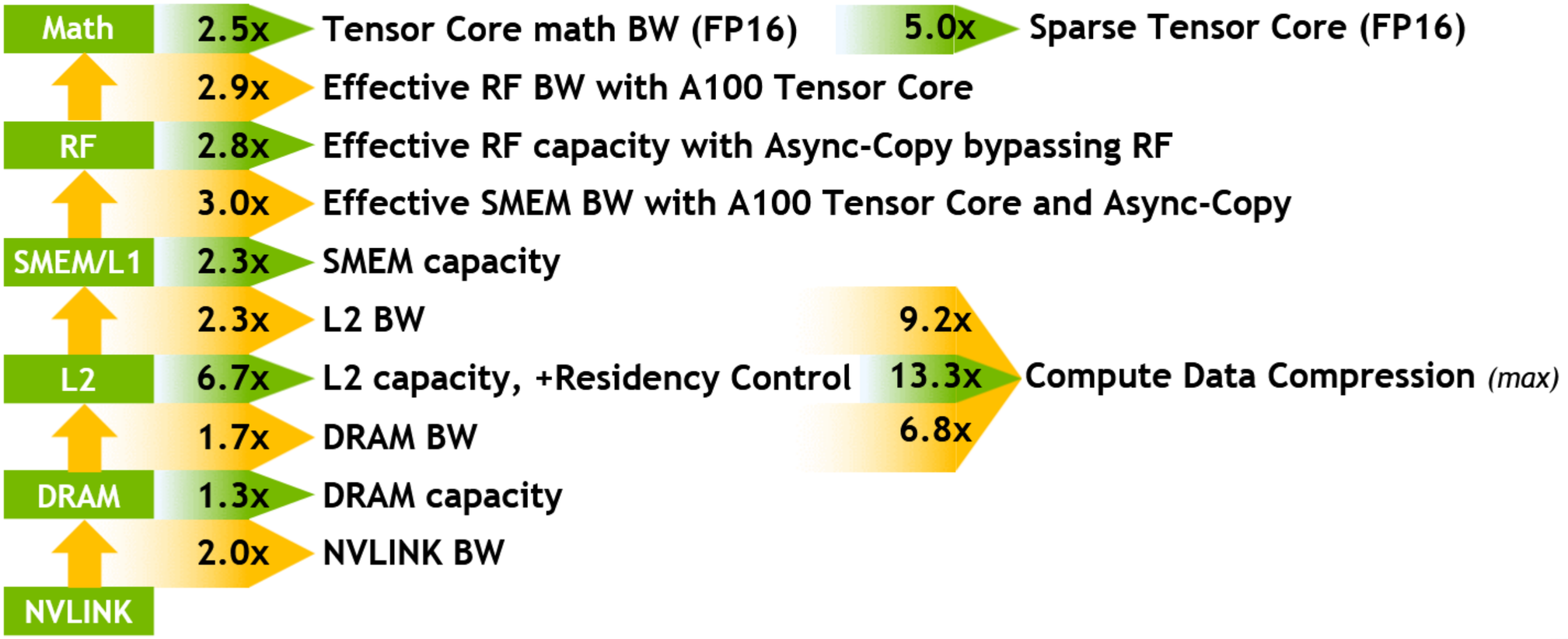
Microarchitecture improvements for grid-to-grid latencies

→S21760: CUDA New Features And Beyond, 5/19 10:15am PDT

A100 STRONG SCALING INNOVATIONS

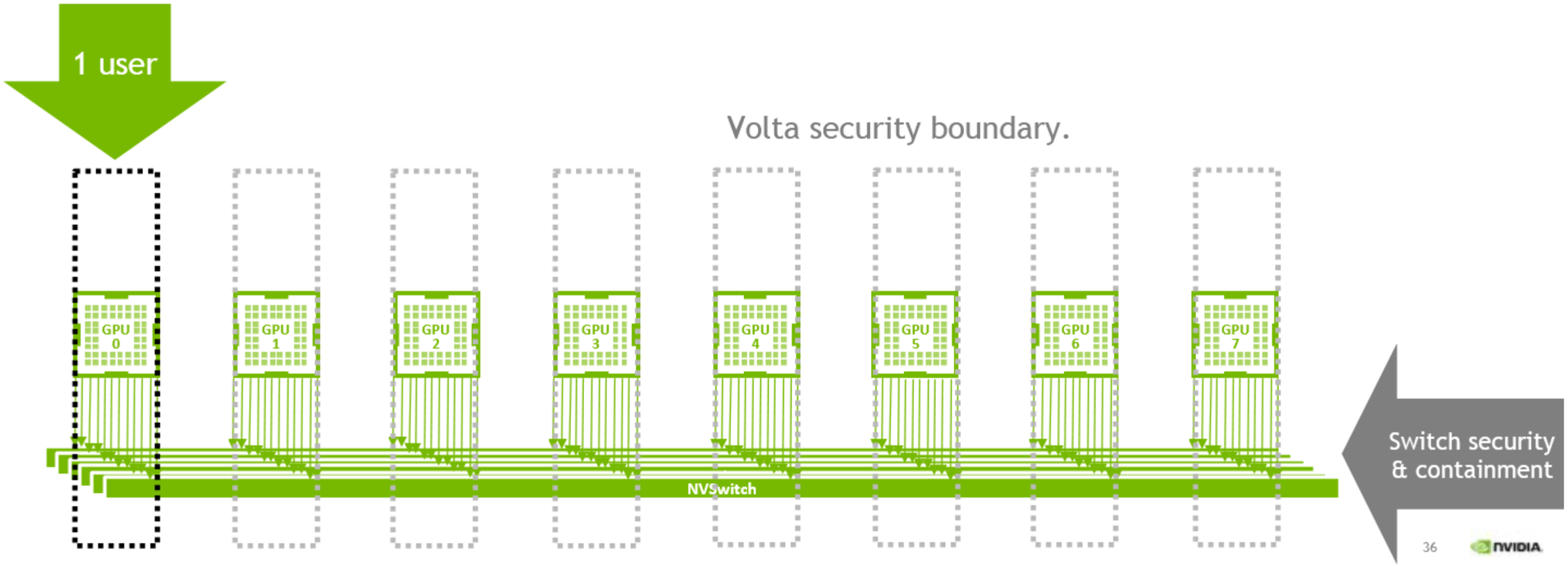
Delivering unprecedented levels of performance

A100 improvements over V100



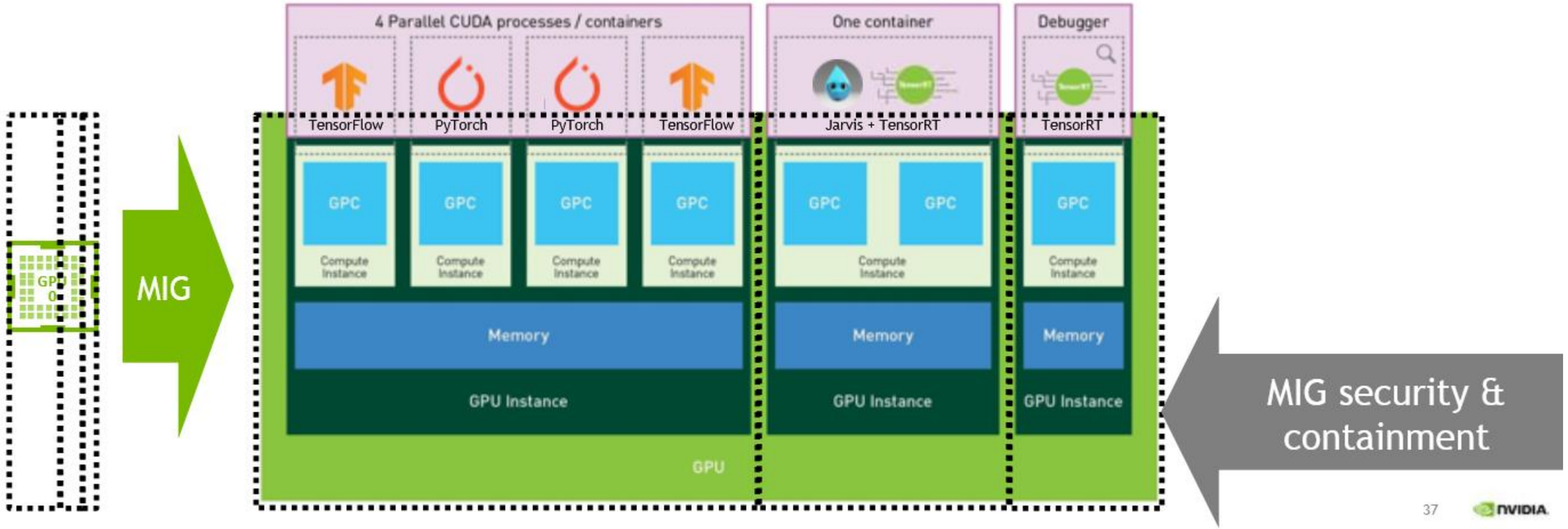
CLOUD SMALL INSTANCE USAGE

- Small workloads can under-utilize GPU cloud instances, provisioned at whole GPU level
- CSPs can't use MPS for GPU space-sharing, because it doesn't provide enough isolation



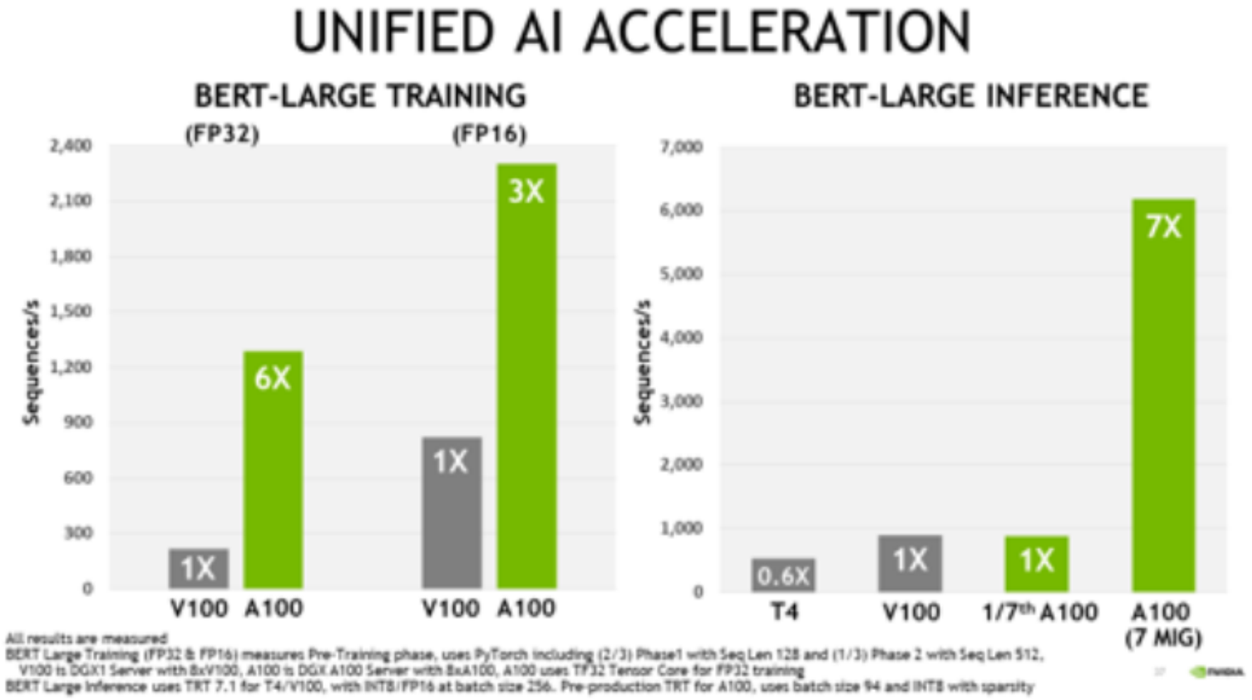
NEW: MULTI-INSTANCE GPU (MIG)

- Up to 7 instances total, dynamically reconfigurable
- **Compute instances:** compute/fault isolation, but share/compete for memory
- **GPU instances:** separate and isolated paths through the entire memory system

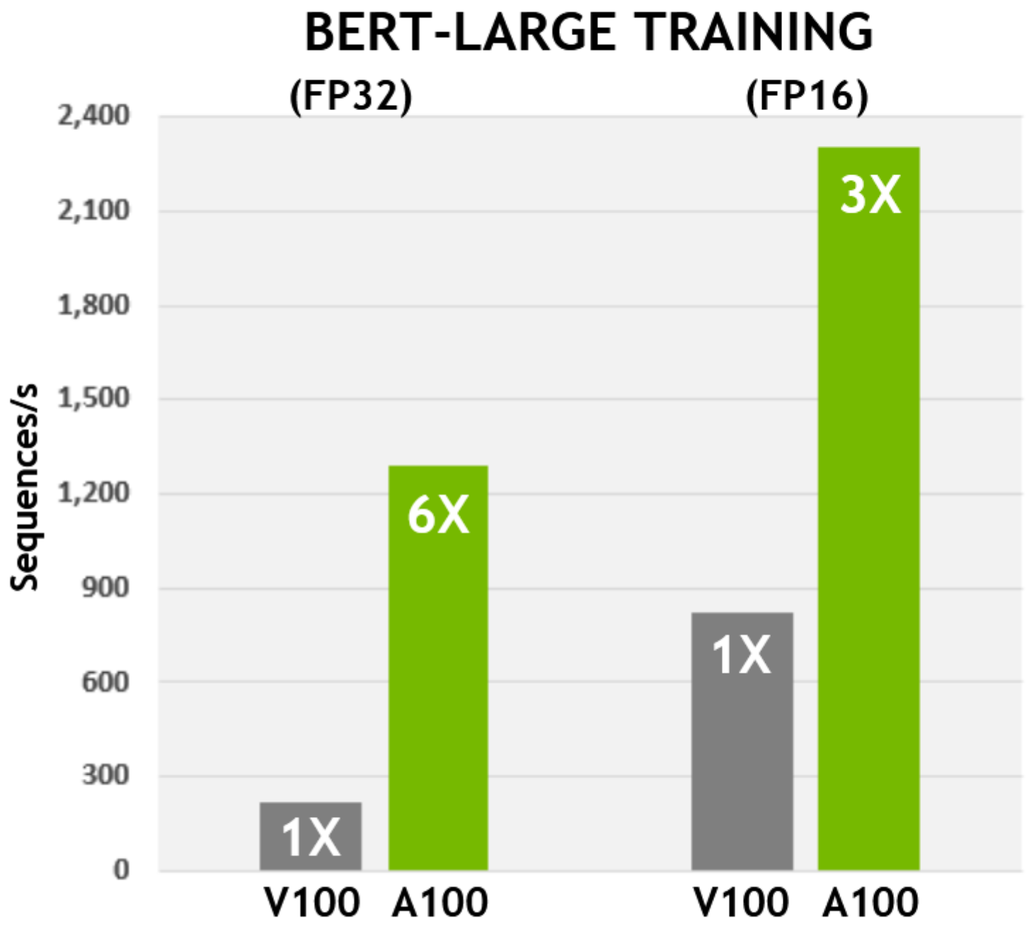


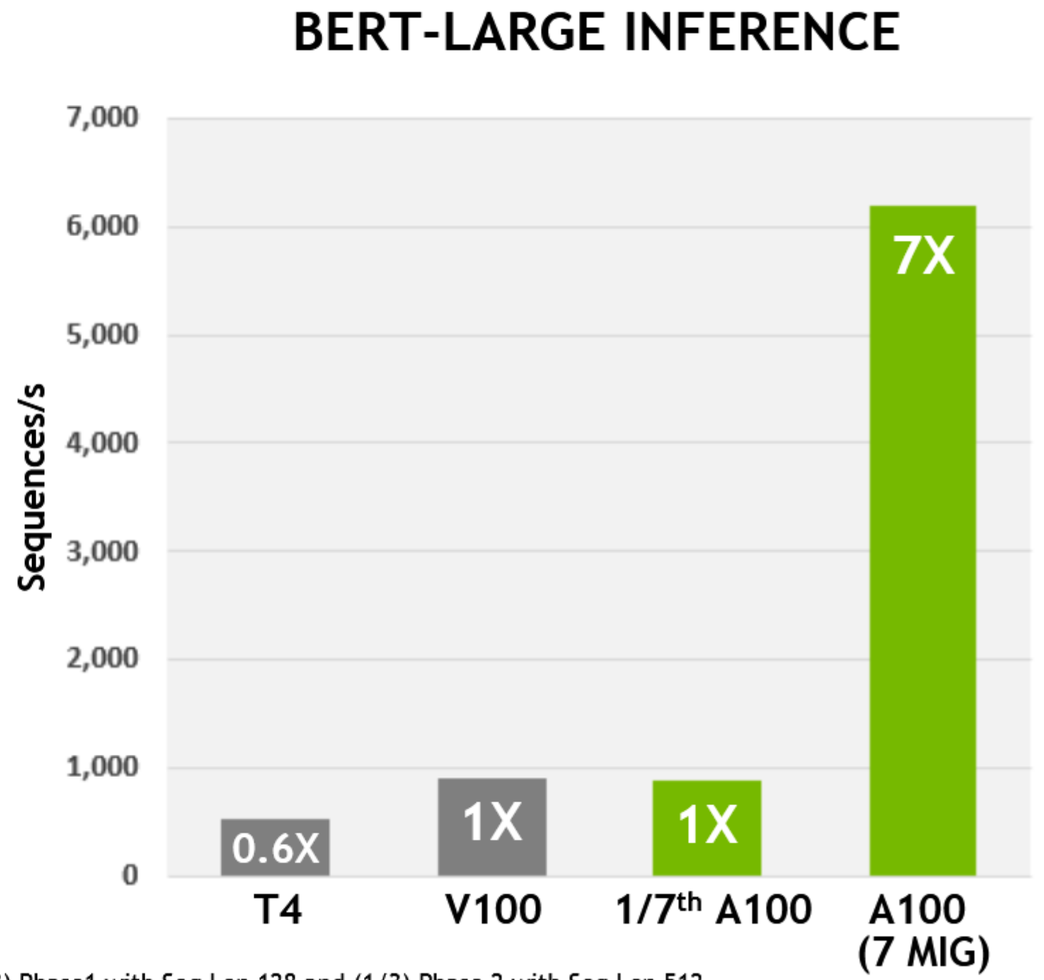
ELASTIC GPU COMPUTING

- Each A100 is 1 to 7 GPUs
- Each DGX A100 is 1 to 56 GPUs
- Each GPU can serve a different user, with full memory isolation and QoS



UNIFIED AI ACCELERATION

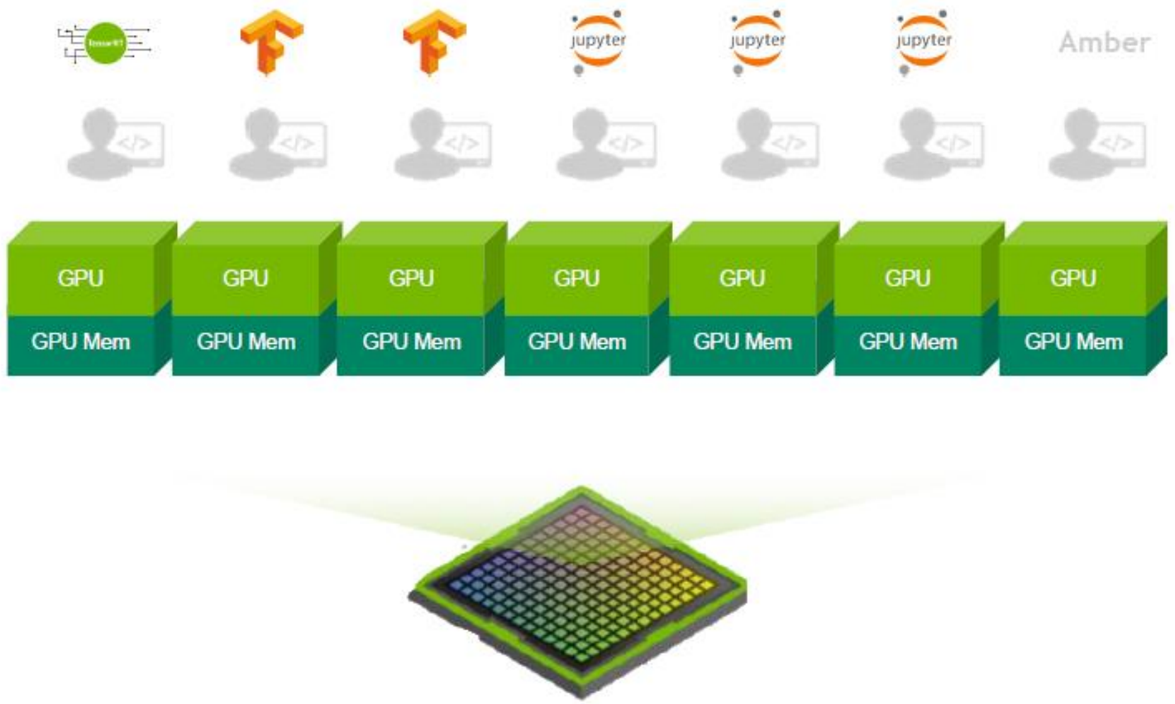




All results are measured
BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512, V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training
BERT Large Inference uses TRT 7.1 for T4/V100, with INT8/FP16 at batch size 256. Pre-production TRT for A100, uses batch size 94 and INT8 with sparsity

MOST FLEXIBLE AI PLATFORM WITH MULTI-INSTANCE GPU (MIG)

Optimize GPU Utilization, Expand Access to More Users with Guaranteed Quality of Service



- Up To 7 GPU Instances In a Single A100
- Simultaneous Workload Execution With Guaranteed Quality Of Service
  All MIG instances run in parallel with predictable throughput & latency
- Flexibility to run any type of workload on a MIG instance
- Right Sized GPU Allocation
  Different sized MIG instances based on target workloads

CUDA 11

CUDA TOOLKIT 11 - The most powerful SW development platform for building GPU-accelerated apps
● Develop for the NVIDIA Ampere GPU architecture including: o The new NVIDIA A100 GPU for accelerated scale-up and scale-out AI and HPC data centers
o Multi-GPU systems based on A100 such as DGX A100 and HGX A100

● New third generation Tensor Cores to accelerate mixed-precision matrix operations on different data types, including TF32 and Bfloat16
● Multi-Instance GPU virtualization and GPU partitioning capabilities for improved GPU utilization
● Library performance optimizations for linear algebra, FFTs, matrix multiplication, JPEG decoding, and more
● Programming and API improvements for task graphs, asynchronous data movement, fine grained synchronization, L2 cache residency control
● Enhancements to the Nsight developer tools family for tracing, profiling, debugging, and roofline analysis
● Support heterogeneous architectures with GPUs including X86_64, Arm64 server, and POWER architectures
● CUDA® C++ enhancements: o Compiler performance and usability improvements
o New link time optimization capabilities
o Support for new host compilers and language standards including C++17
o Introducing Parallel C++ STL support using libcu++ and integration of CUB as a CUDA C++ core library in the Toolkit

● Operating System support updates

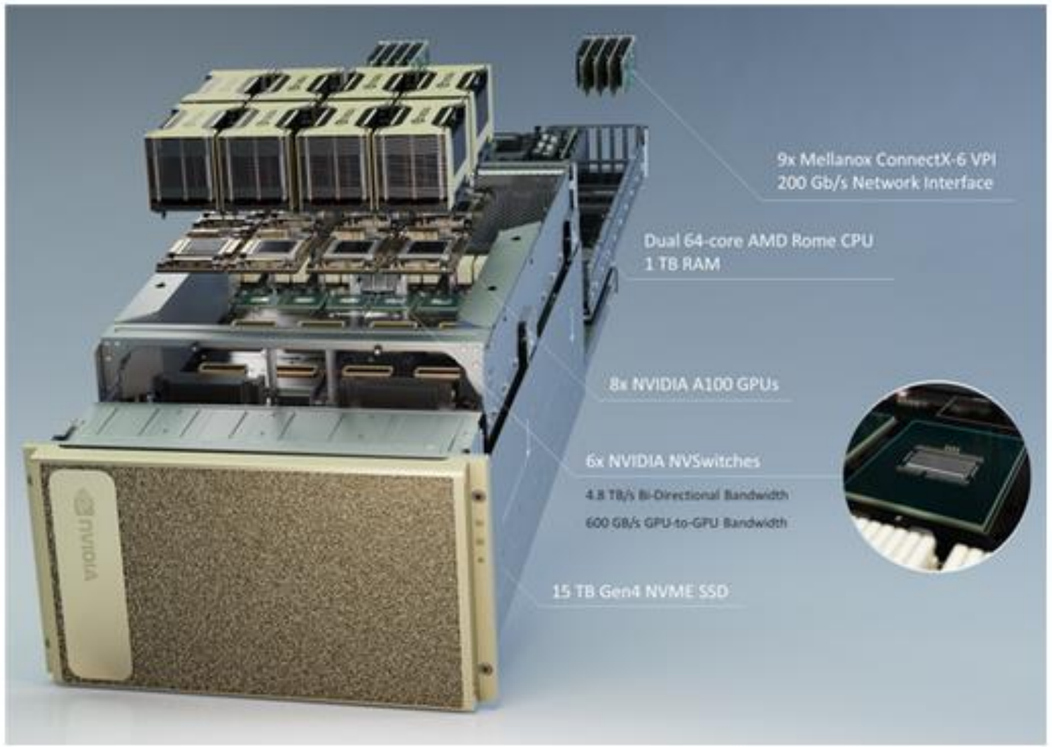
Pricing: Licenses free of charge
Availability: GA available ~May 28, 2020

A NEW GENERATION OF MACHINES

NVIDIA DGX A100

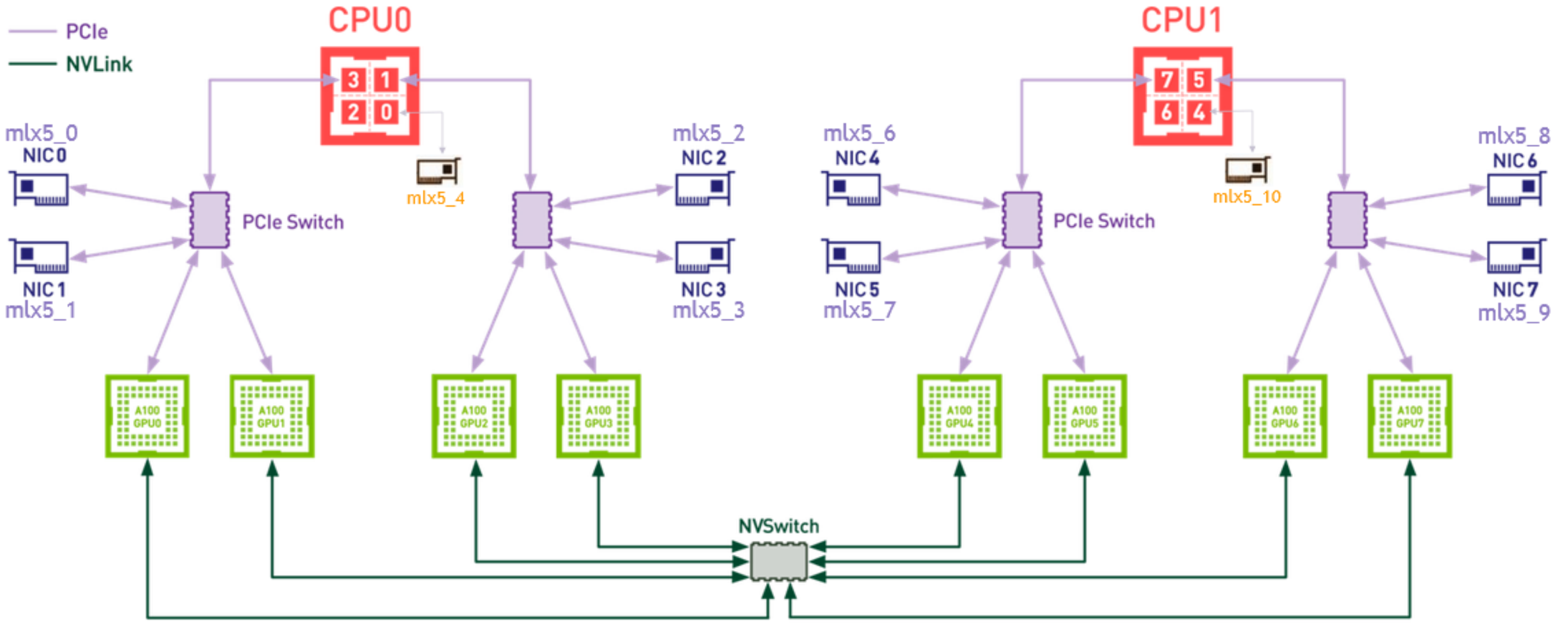
GPUs	**8x NVIDIA A100**
GPU Memory	320 GB total
Peak performance	5 petaFLOPS AI 10 petaOPS INT8
NVSwitches	6
System Power Usage	**6.5kW max**
CPU	**Dual AMD Rome 7742** 128 cores total, 2.25 GHz(base), 3.4GHz (max boost)
System Memory	**1TB**
Networking	**8x Single-Port Mellanox ConnectX-6** 200Gb/s HDR Infiniband (**Compute Network**) 1x (or **2x***) **Dual-Port Mellanox ConnectX-6** 200GB/s HDR Infiniband (**Storage Network** also used for Eth*)
Storage	OS: 2x 1.92TB M.2 NVME drives Internal Storage: 15TB (4x 3.86TB) U.2 NVME drives
Software	Ubuntu Linux OS (**5.3+ kernel**)
System Weight	271 lbs (123 kgs)
Packaged System Weight	315 lbs (143 kgs)
Height	**6U**
Operating temperature range	**5C to 30C** (41F to 86F)

* Optional upgrades



DGX A100

High-level Topology Overview (with options)



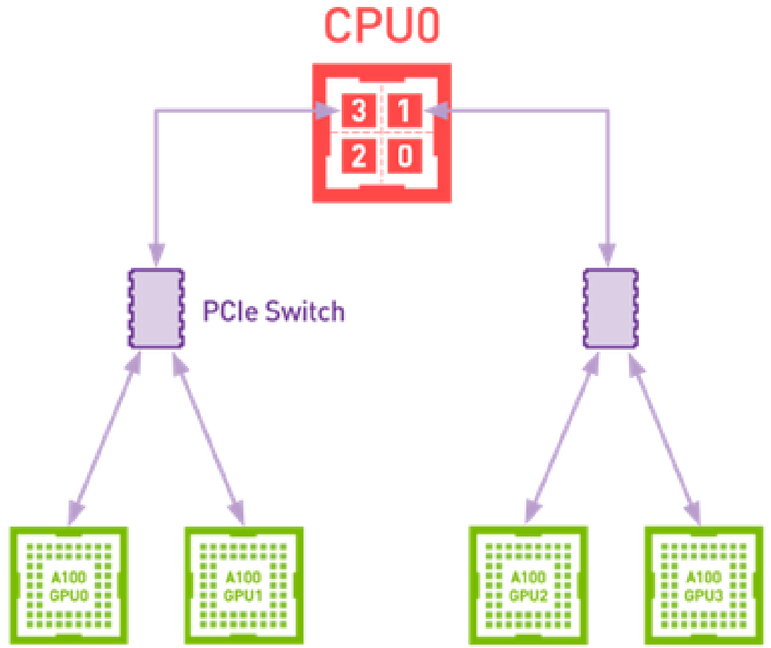
DGX A100

AMD Rome and NUMA

NPS = NUMA nodes per socket

- On DGX A100, **NPS = 4** (8 total NUMA nodes across two sockets).
- Each NUMA node has 16 physical cores and 2 memory channels providing total 50 GB/s DRAM BW (SOL).

```
$ lscpu
...
NUMA node0 CPU(s):   0-15,128-143
NUMA node1 CPU(s):   16-31,144-159
NUMA node2 CPU(s):   32-47,160-175
NUMA node3 CPU(s):   48-63,176-191
NUMA node4 CPU(s):   64-79,192-207
NUMA node5 CPU(s):   80-95,208-223
NUMA node6 CPU(s):   96-111,224-239
NUMA node7 CPU(s):   112-127,240-255
```



DGX A100 SuperPOD

Fast deployment ready
Cold aisle containment design

DGX A100 SuperPOD

A modular model

1K GPU SuperPOD Cluster

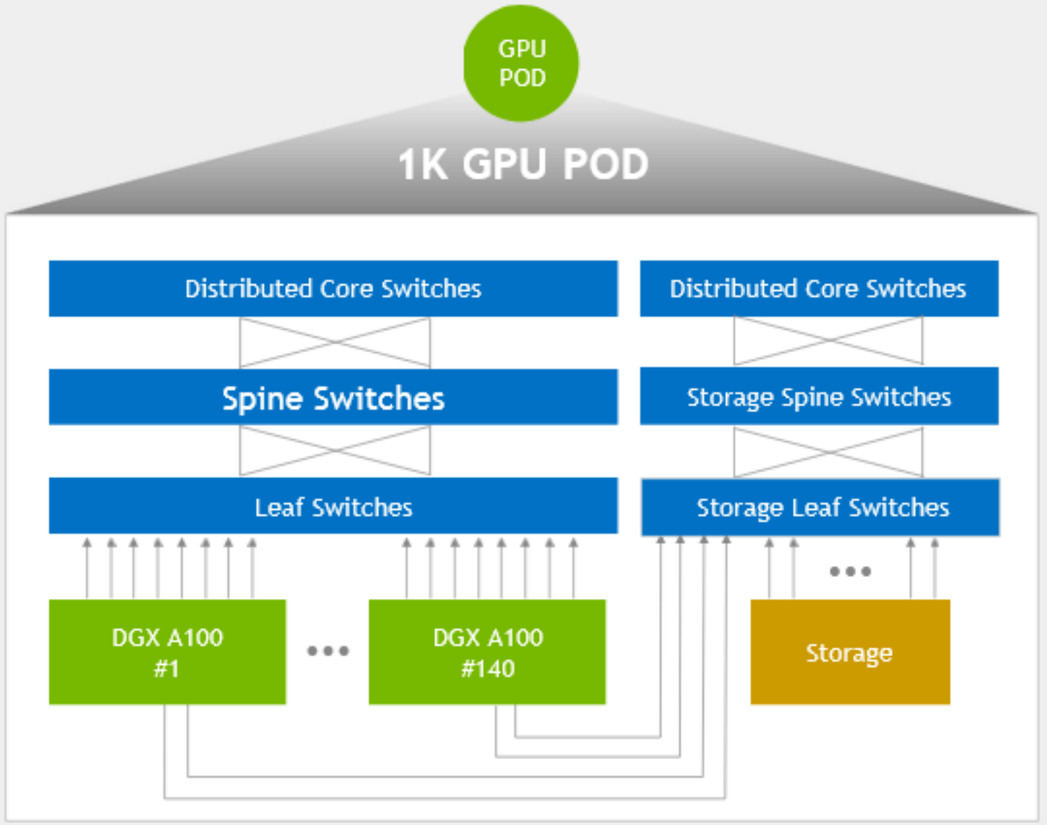
- 140 DGX A100 nodes (1120 GPUs) in a GPU POD
- 1st tier fast storage - DDN AI400x with Lustre
- Mellanox HDR 200Gb/s InfiniBand - Full Fat-tree
- Network optimized for AI and HPC

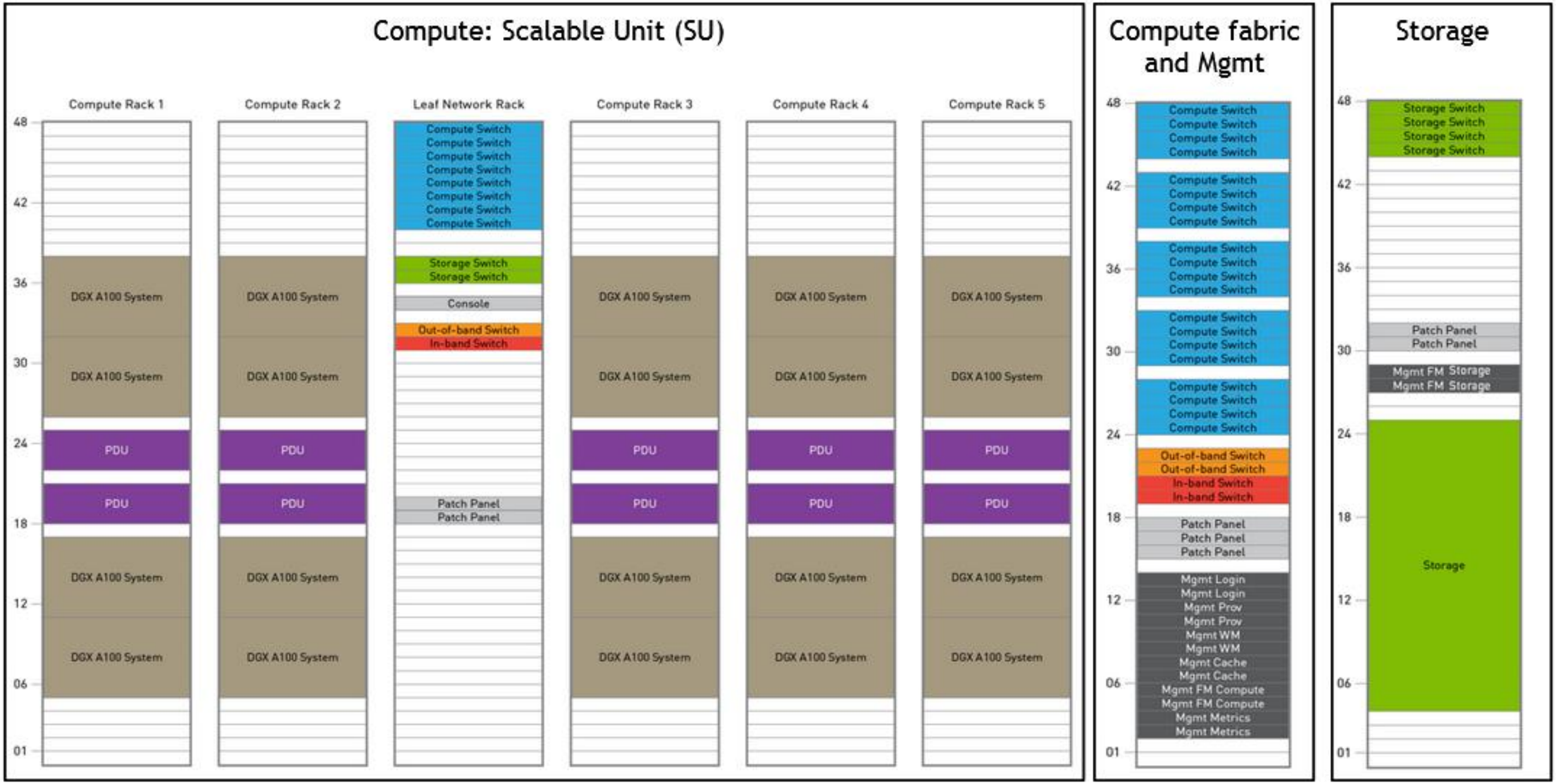
DGX A100 Nodes

- 2x AMD 7742 EPYC CPUs + 8x A100 GPUs
- NVLINK 3.0 Fully Connected Switch
- 8 Compute + 2 Storage HDR IB Ports

A fast interconnect

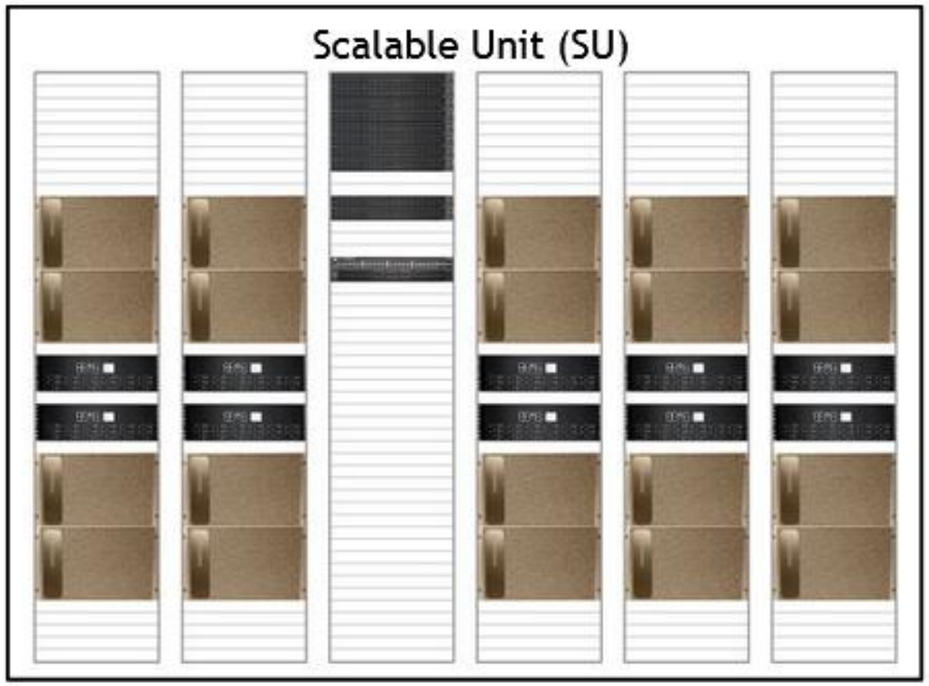
- Modular IB Fat-tree
- Separate network for Compute vs Storage
- Adaptive routing and SharpV2 support for offload

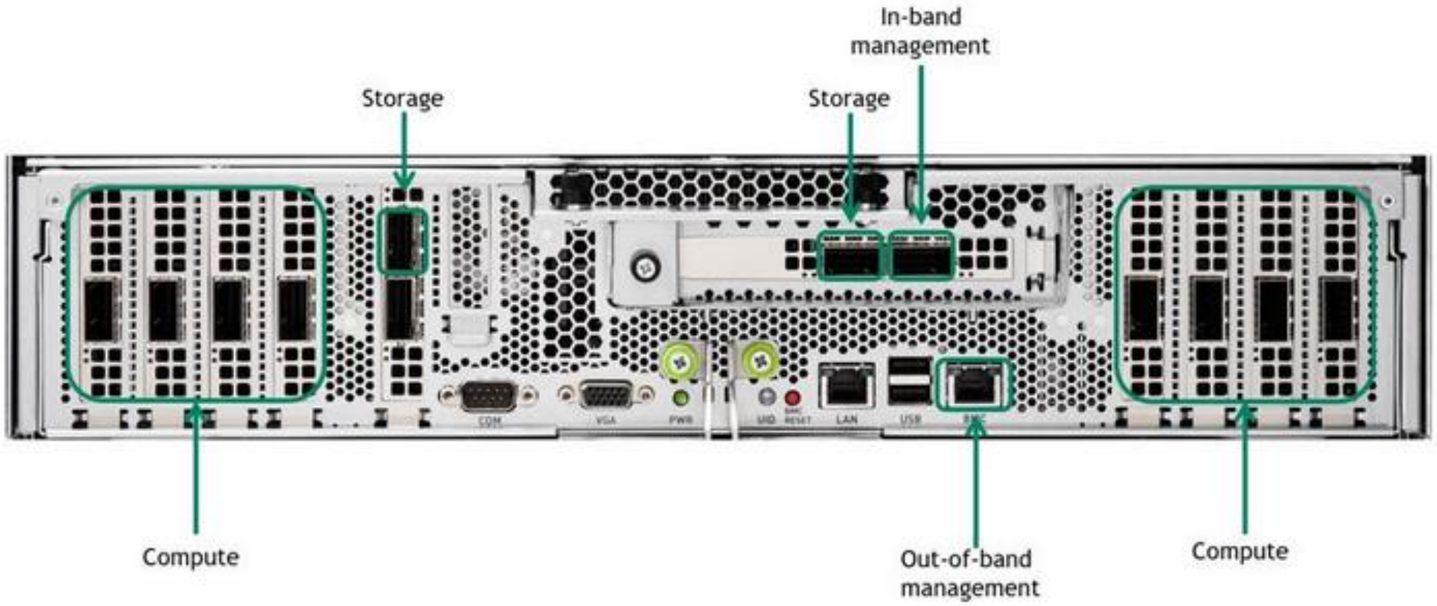



Compute: Scalable Unit (SU)
Compute Rack 1
Compute Rack 2
Leaf Network Rack
Compute Rack 3
Compute Rack 4
Compute Rack 5
48
42
36
30
24
18
12
06
01
DGX A100 System
PDU
Compute Switch
Storage Switch
Console
Out-of-band Switch
In-band Switch
Patch Panel
Compute fabric and Mgmt
Mgmt Login
Mgmt Prov
Mgmt WM
Mgmt Cache
Mgmt FM Compute
Mgmt Metrics
Storage
Storage Switch
Patch Panel
Mgmt FM Storage

DESIGNING FOR PERFORMANCE

In the datacenter





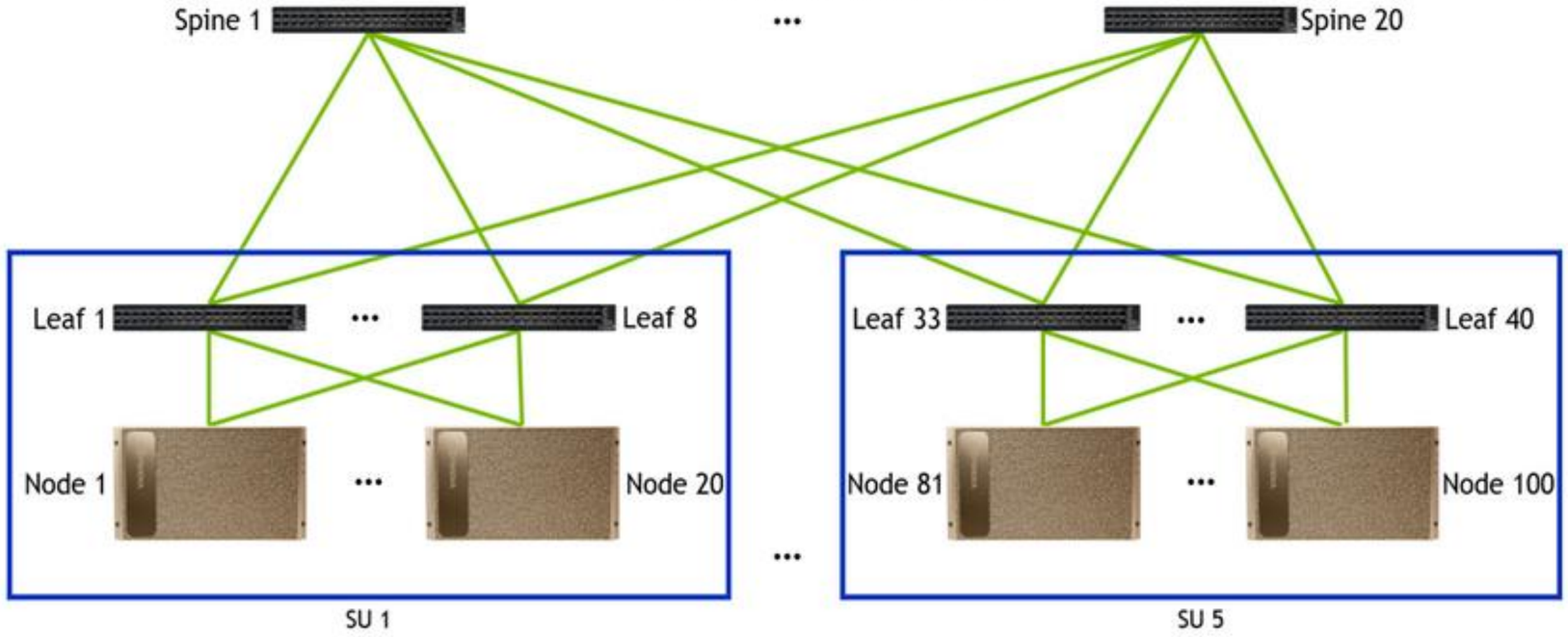
A POD at any scale

Growing with Scalable Units (SU)

Nodes	SUs	QM8790 Switches			Cables		
		Leaf	Spine	Core	Leaf	Spine	Core
10	½	8	2		80	80	
20 (Single SU)	1	8	4		160	160	
40	2	16	10		320	320	
80	4	32	20		640	640	
100	5	40	20		800	800	
140 (DGX A100 SuperPOD)	7	56	80	28	1120	1120	560

Full fat tree compute fabric

100 node example

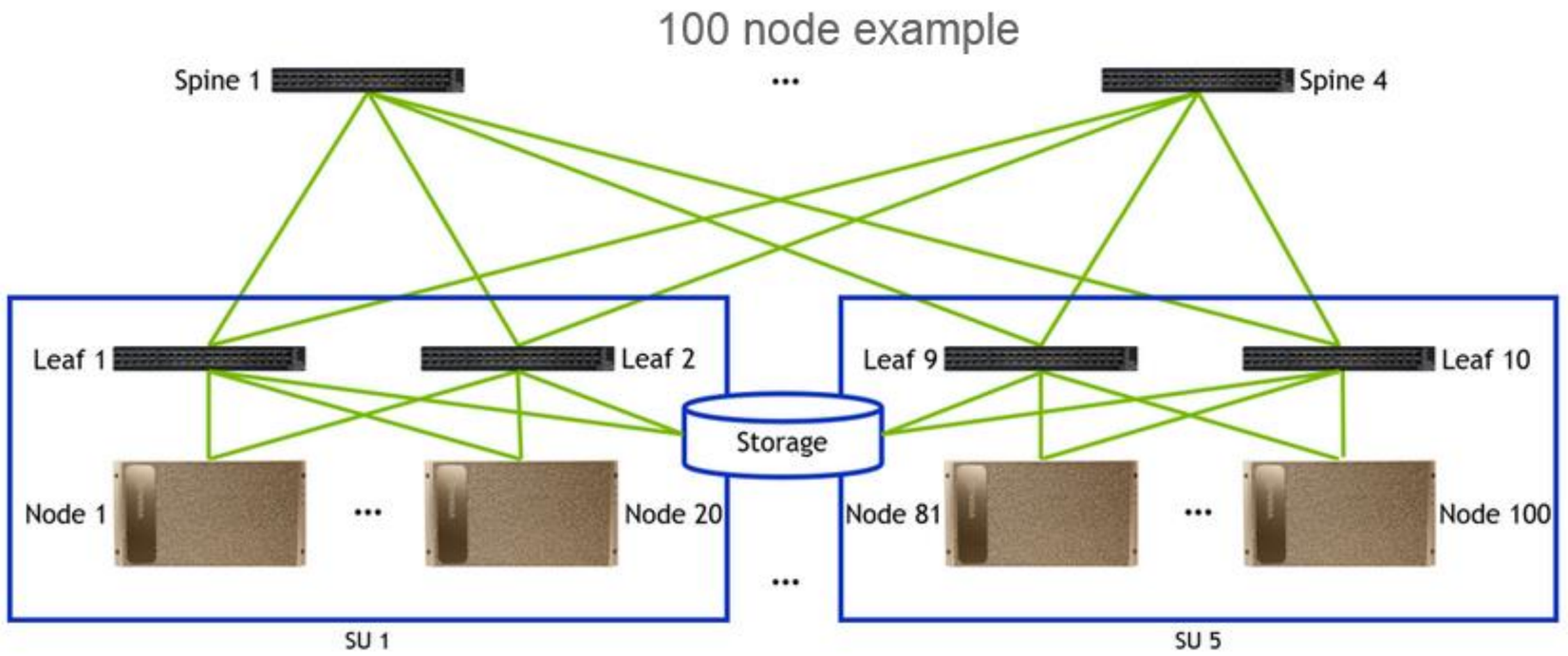


A POD at any scale

Growing with Scalable Units (SU)

Nodes	SUs	Storage Ports	QM8790 Switches		Cables			Subscription Ratio
			Leaf	Spine	Leaf	Spine	Storage	
10	1/2	4	2	1	20	20	4	1:1
20	1	8	2	1	40	32	8	3:2
40	2	16	4	2	80	64	16	3:2
80	4	32	8	4	160	128	32	3:2
100	5	40	10	4	200	160	40	3:2
140	7	56	14	8	280	224	56	5:4

Storage fabric with different ratios



Multi node IB compute

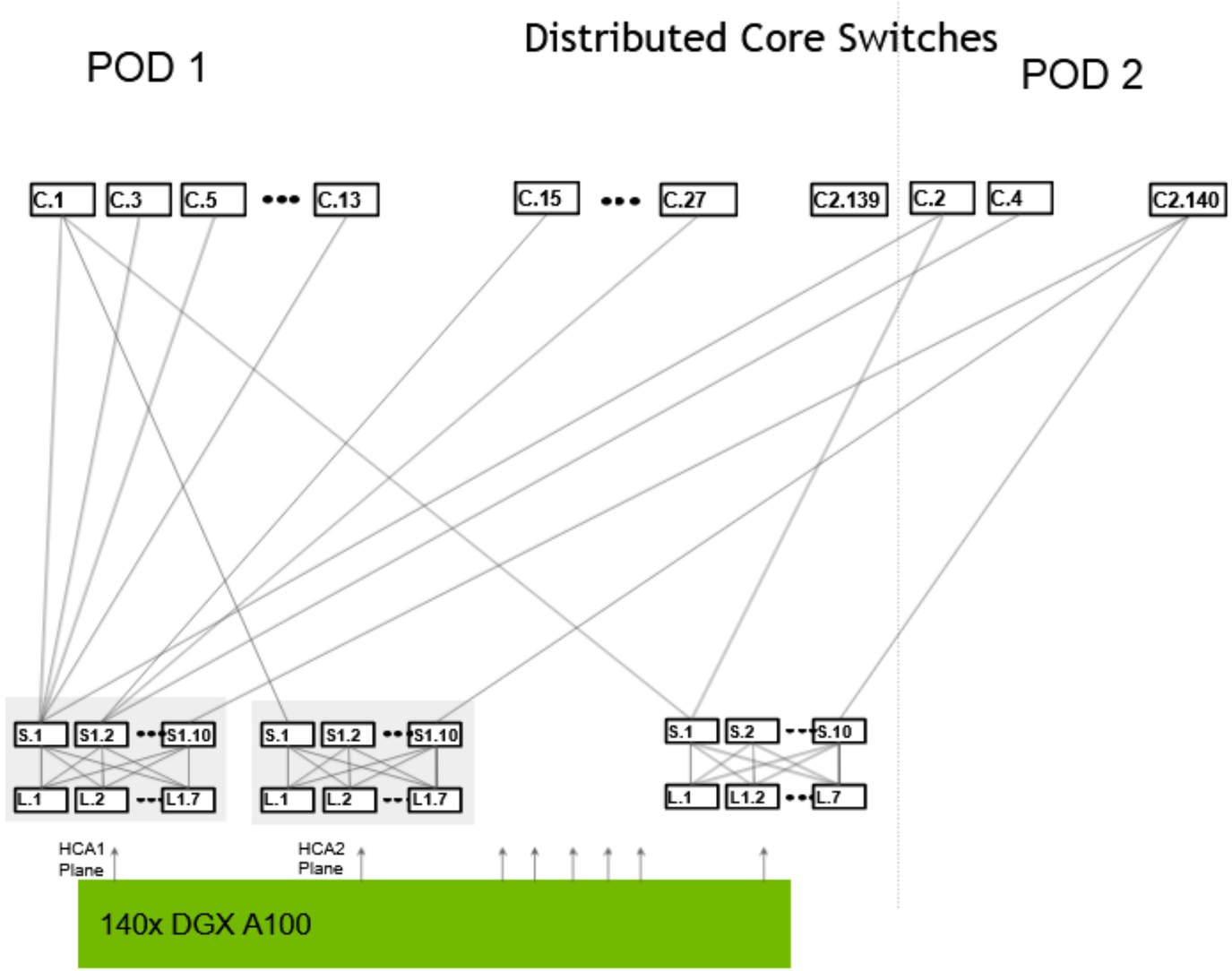
The details

Designed with Mellanox 200Gb HDR IB network

Separate compute and storage fabric
- 8 Links for compute
- 2 Links for storage (Lustre)
- Both networks share a similar fat-tree design

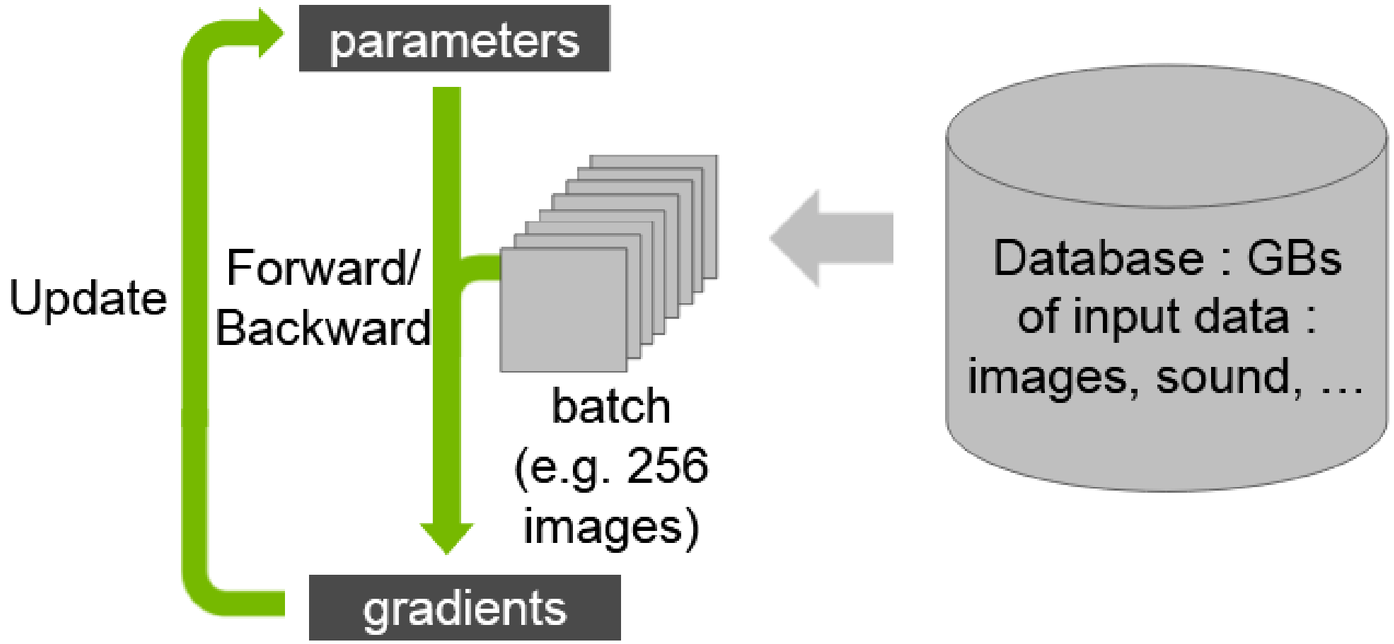
Modular POD design
- 140 DGX A100 nodes are fully connected in a POD
- POD contains compute nodes and storage
- All nodes and storage are usable between PODs

Sharp optimized design
- Leaf and Spines organized in HCA planes
- For a POD, all HCA1 from 140 DGX-2 connect to a HCA1 Plane fat-tree network
- Traffic from HCA1 to HCA1 between any two nodes in a POD stay either at the Leaf or Spine level
- Only use core switches when
  - 1. Moving data between HCA planes (e.g. mlx5_0 to mlx5_1 in another system)
  - 2. Moving any data between PODs



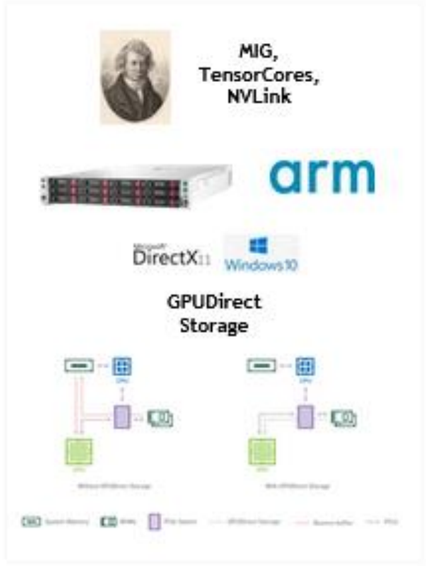
DL TRAINING

Single-GPU



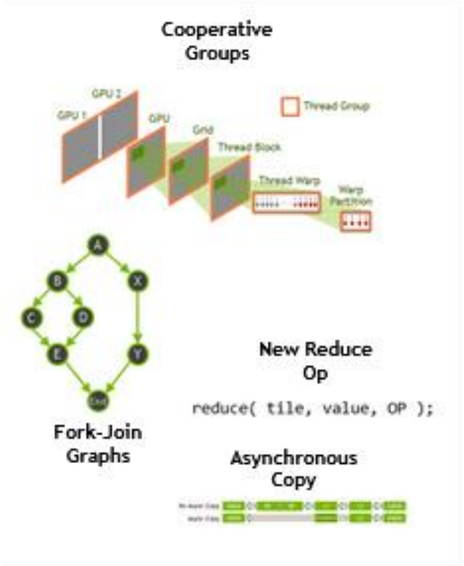
CUDA PLATFORM

Major Feature Areas for CUDA 11.0



New Platform Capabilities

- *Ampere Features*
- *CUDA on Arm Platforms*

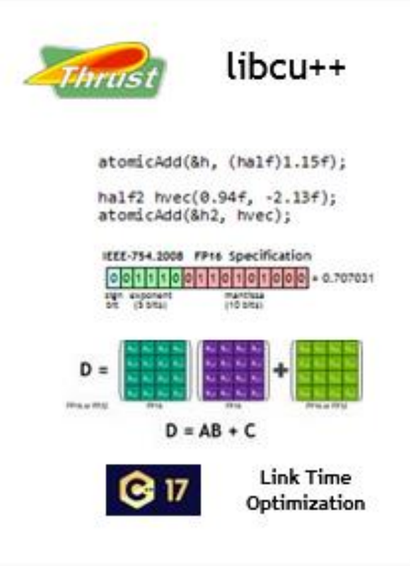


Programming Model Updates

- *Ampere Programming Model*
- *New APIs for CUDA Graphs*
- *Flexible Thread Programming*
- *Memory Management APIs*

Developer Tools

- *Support for Ampere*
- *Roofline plots with Nsight*
- *Next generation correctness tools*

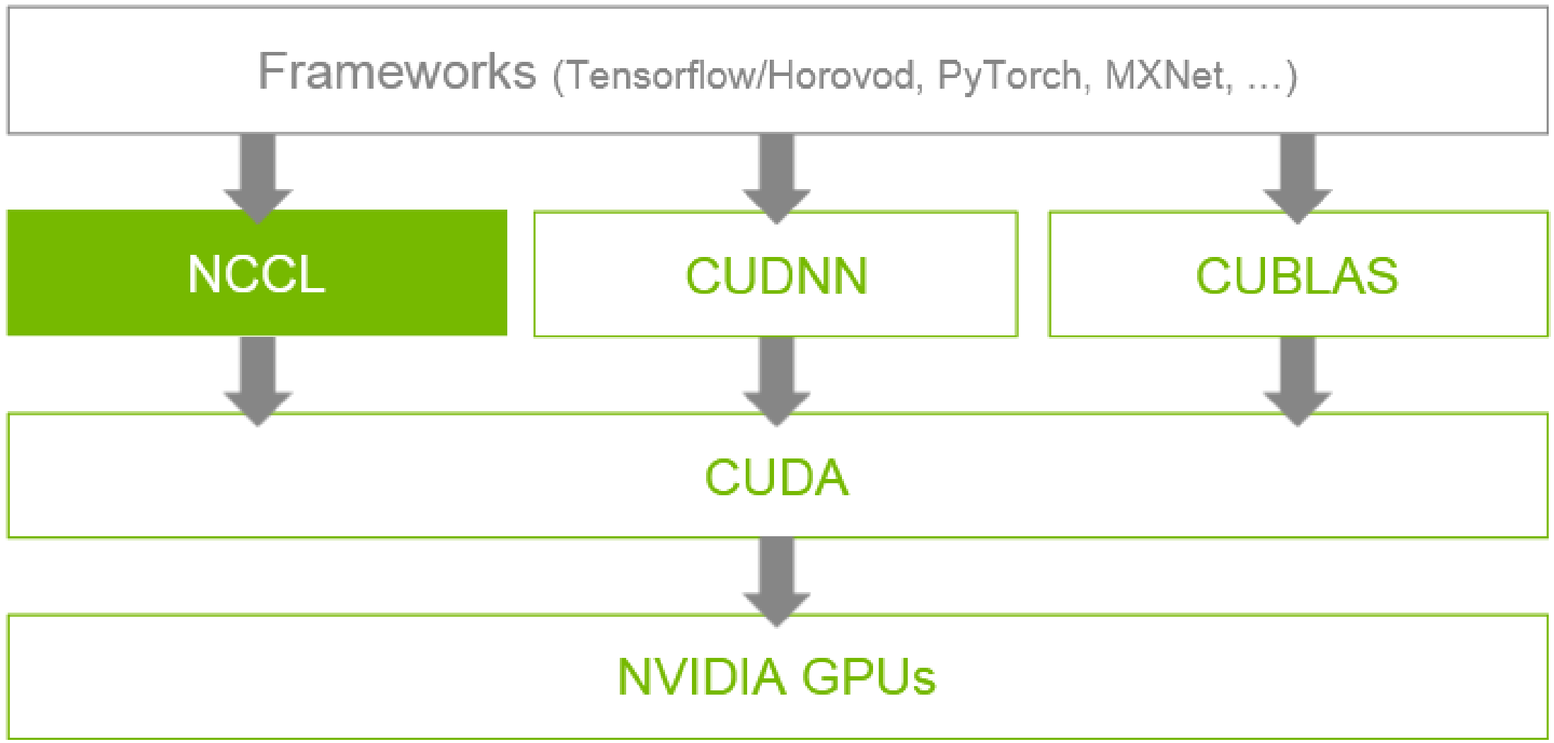


CUDA C++

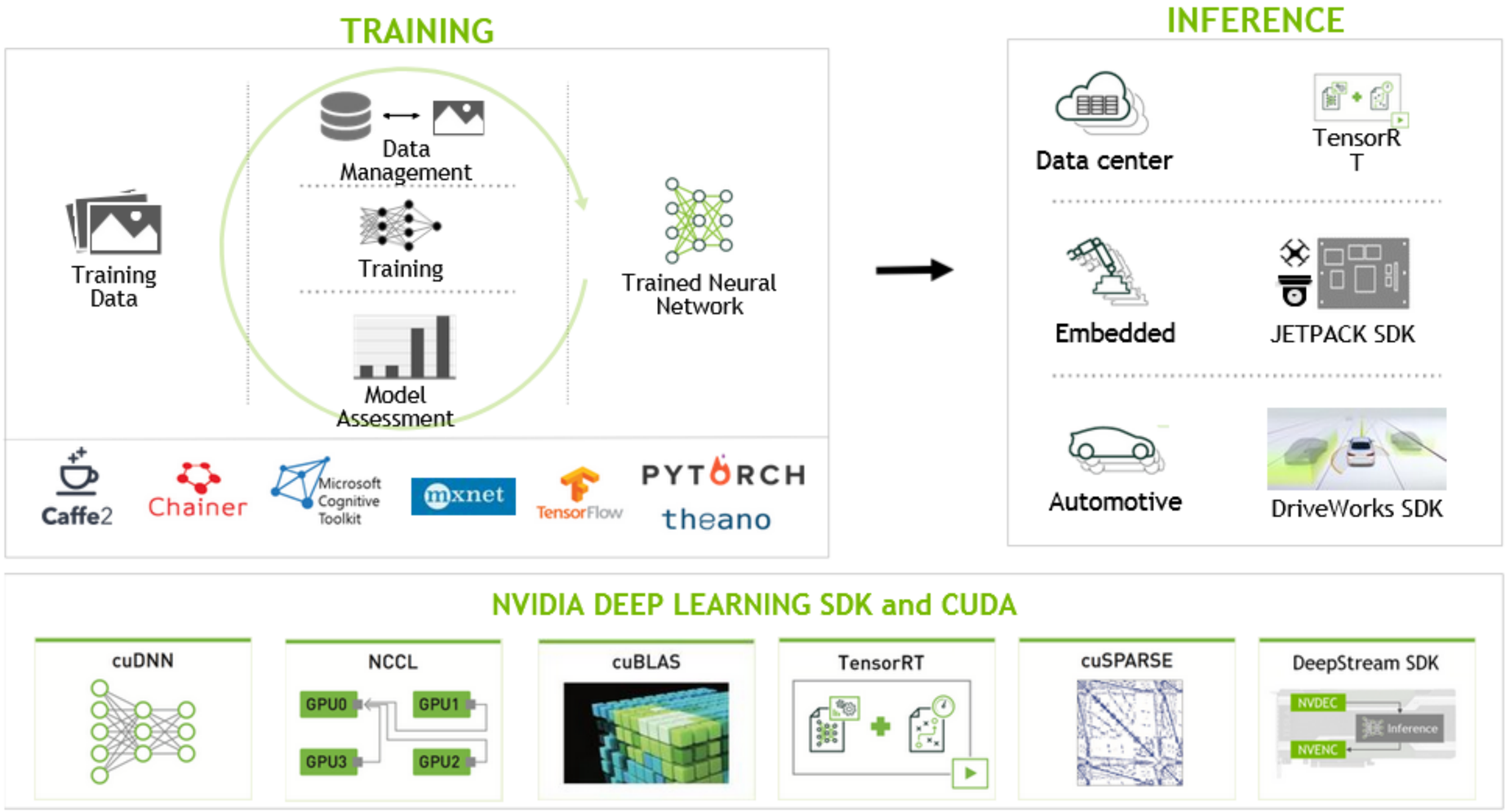
- *C++ Modernization*
- *Parallel standard C++ library*
- *Low precision datatypes and WMMA*

NCCL

DL stack



NVIDIA DEEP LEARNING SOFTWARE STACK



developer.nvidia.com/deep-learning-software

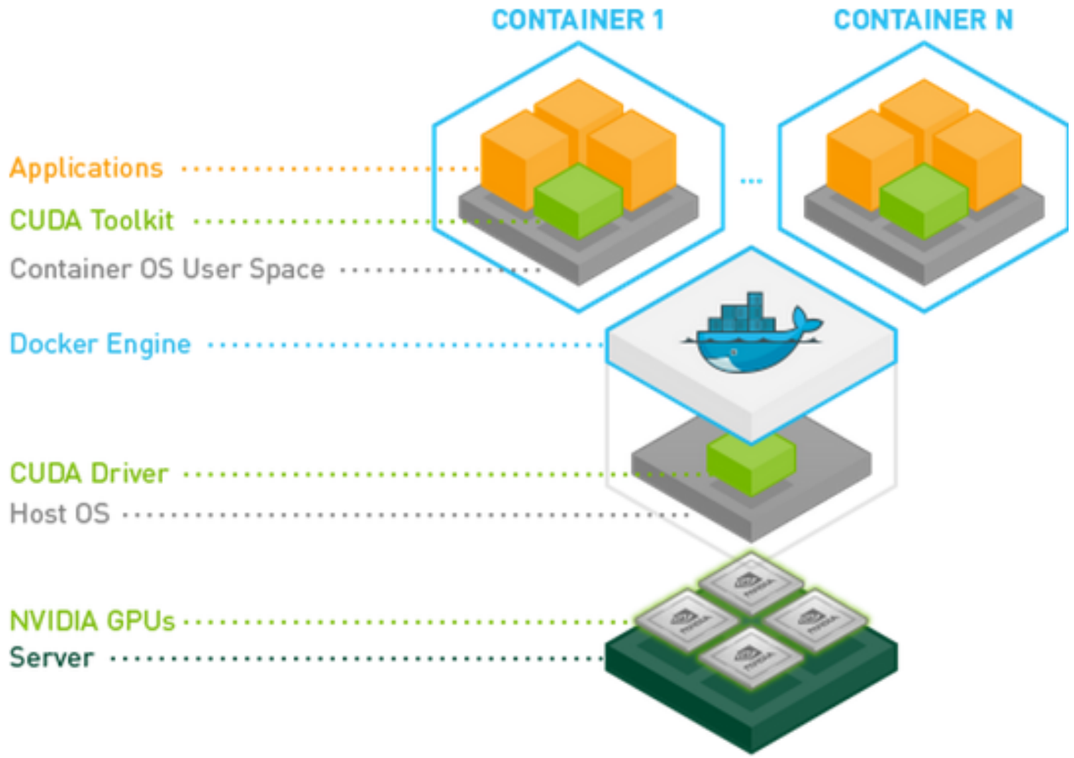
NGC CONTAINERS

We built libnvidia-container to make it easy to run CUDA applications inside containers.

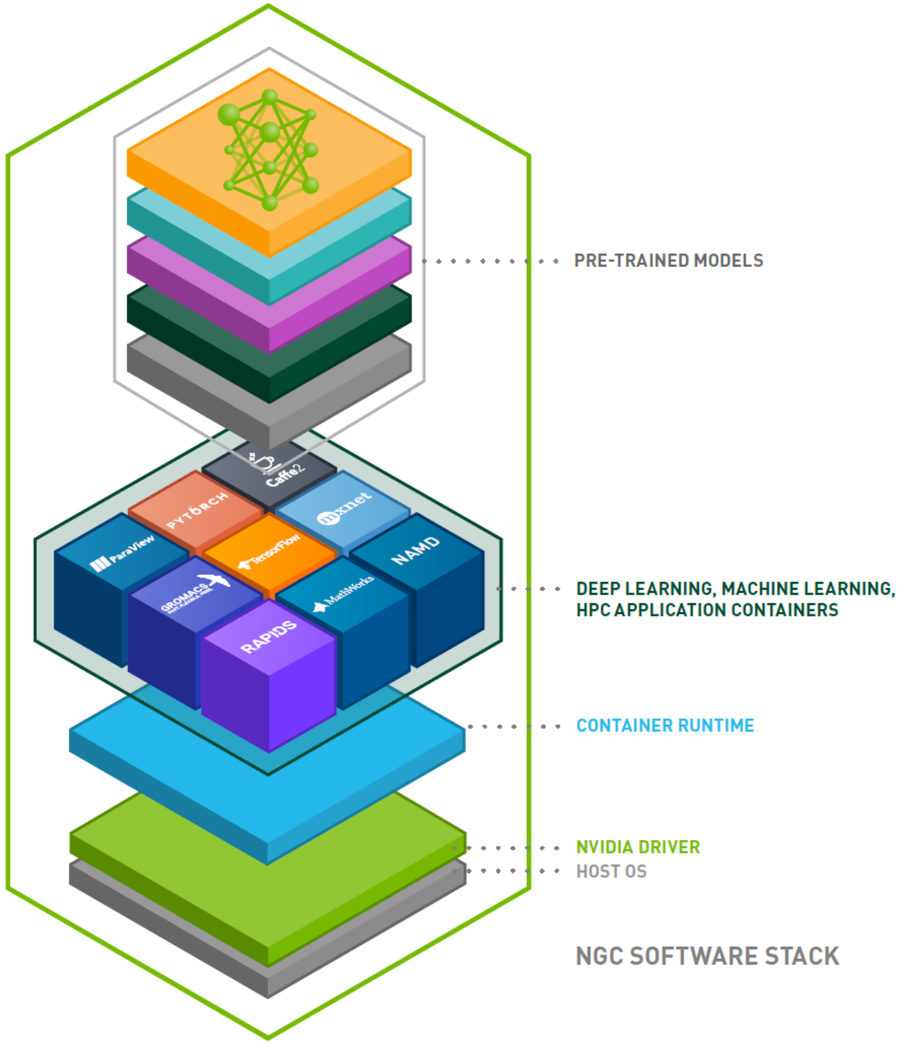
We release optimized container images for each of the major DL frameworks every month, and provide them for anyone to use.

We use containers for everything on our HPC clusters - R&D, official benchmarks, etc.

Containers give us portable software stacks without sacrificing performance.



NVIDIA GPU Cloud (NGC)



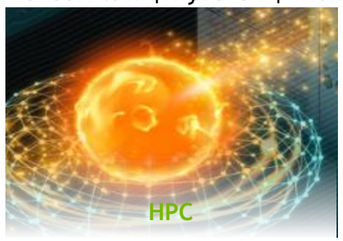
- AI a HPC kontejnery
- Předtrénované modely
- Skripty pro trénování modelů
- Worflow

volně dostupné, optimalizované pro NVIDIA GPU

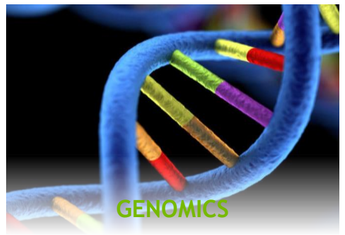
TensorFlow | PyTorch | more

RAPIDS | H2O | more

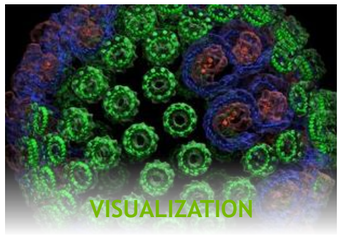
TensorRT | DeepStream | more



NAMD | GROMACS | more

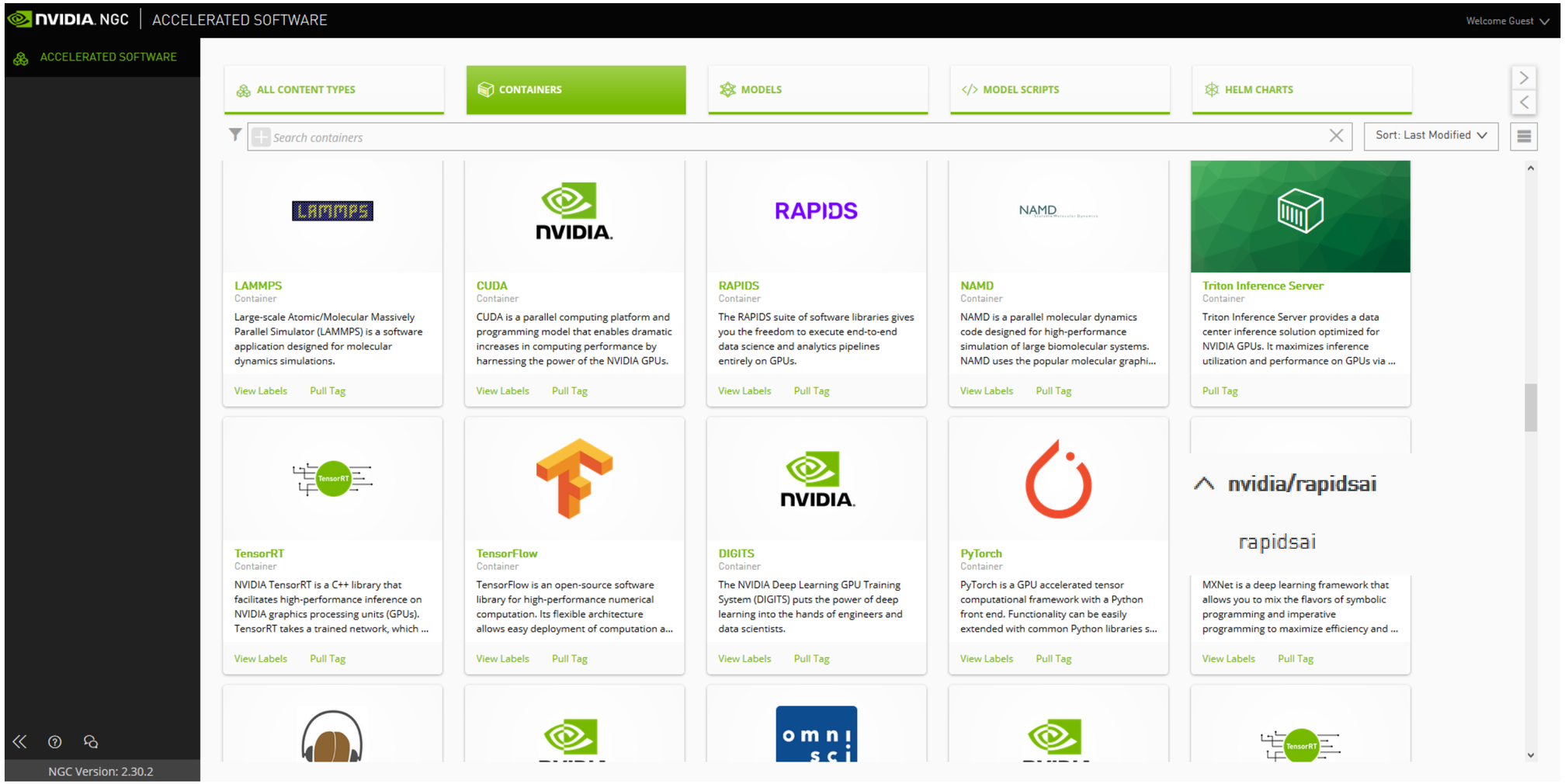


Parabricks



ParaView | IndeX | more

NVIDIA GPU Cloud (NGC)



NVIDIA GPU Cloud (NGC)

- Test provedený na DGX Station 3.6.2020:

TensorFlow:19.01-py3

Step	Epoch	Img/sec	Loss	LR	
1	1.0	127.1	7.708	8.679	2.00000
10	10.0	1088.4	5.749	6.720	1.62000
20	20.0	2977.2	2.247	3.222	1.24469
30	30.0	3303.3	0.021	1.002	0.91877
40	40.0	3311.5	0.006	0.983	0.64222
50	50.0	3298.8	0.017	0.988	0.41506
60	60.0	3295.8	0.003	0.970	0.23728
70	70.0	3305.3	0.001	0.965	0.10889
80	80.0	3314.2	0.000	0.963	0.02988
90	90.0	2735.0	0.000	0.963	0.00025

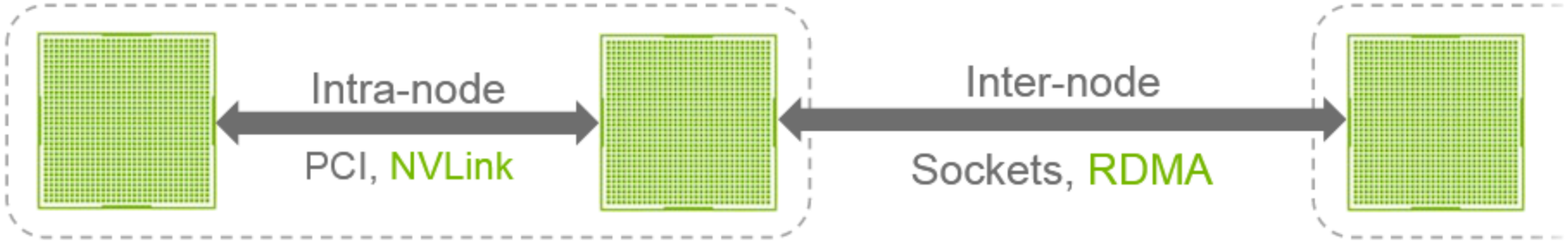
TensorFlow:20.02-tf1-py3

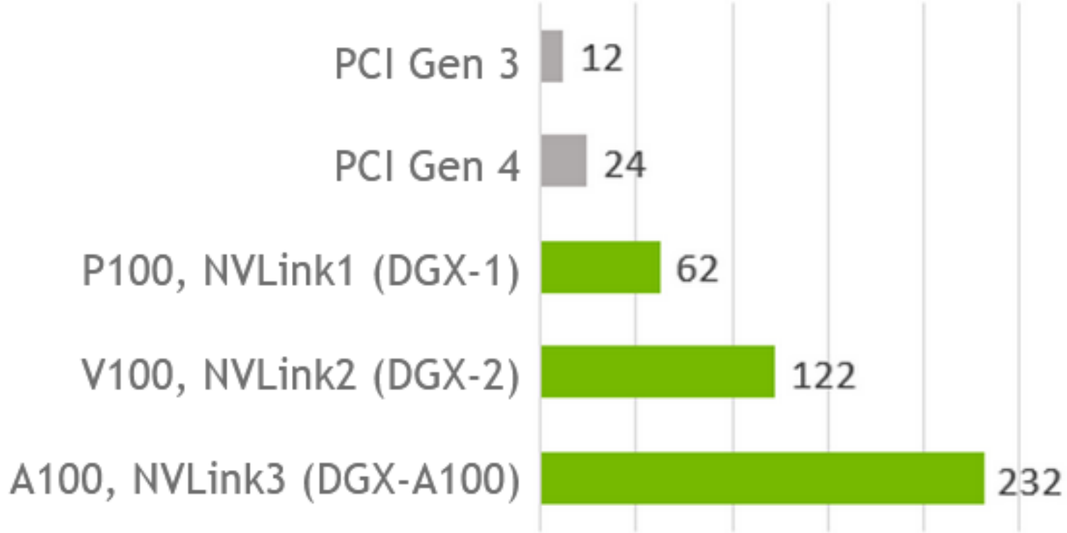
Step	Epoch	Img/sec	Loss	LR	
10	10.0	644.5	5.836	6.807	1.62000
20	20.0	5580.4	2.179	3.155	1.24469
30	30.0	5598.2	0.011	0.992	0.91877
40	40.0	5621.2	0.001	0.977	0.64222
50	50.0	5587.6	0.000	0.969	0.41506
60	60.0	5608.3	0.000	0.964	0.23728
70	70.0	5655.9	0.000	0.960	0.10889
80	80.0	5663.1	0.000	0.959	0.02988
90	90.0	4213.7	0.000	0.959	0.00025

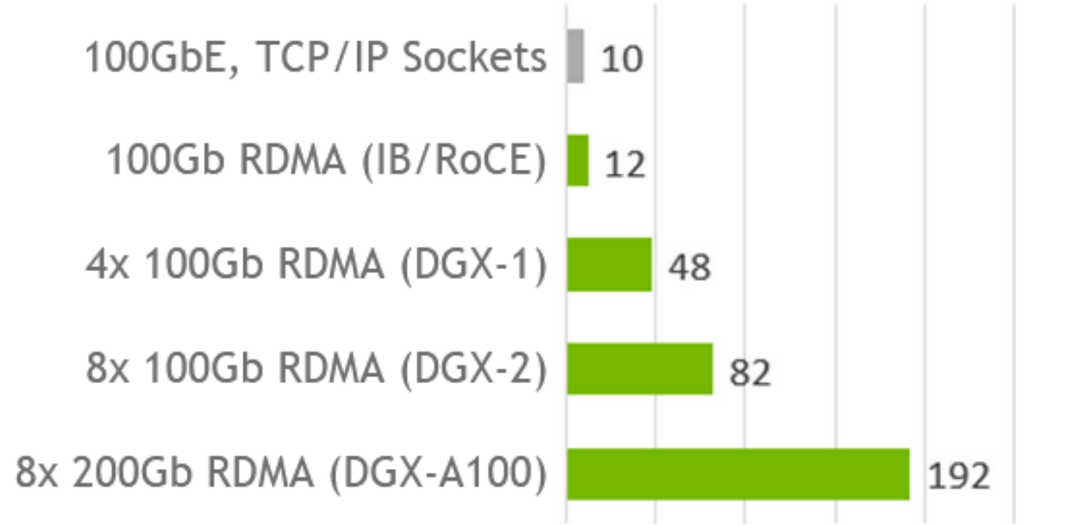
Výsledek dosažený v kontejneru o 13 měsíců novějším je vyšší o více než 50%.

INTER-GPU COMMUNICATION

Intra-node and Inter-node

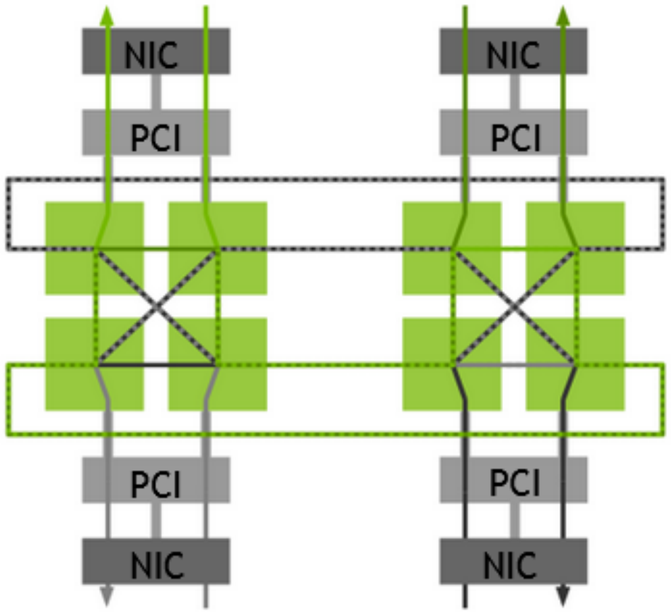




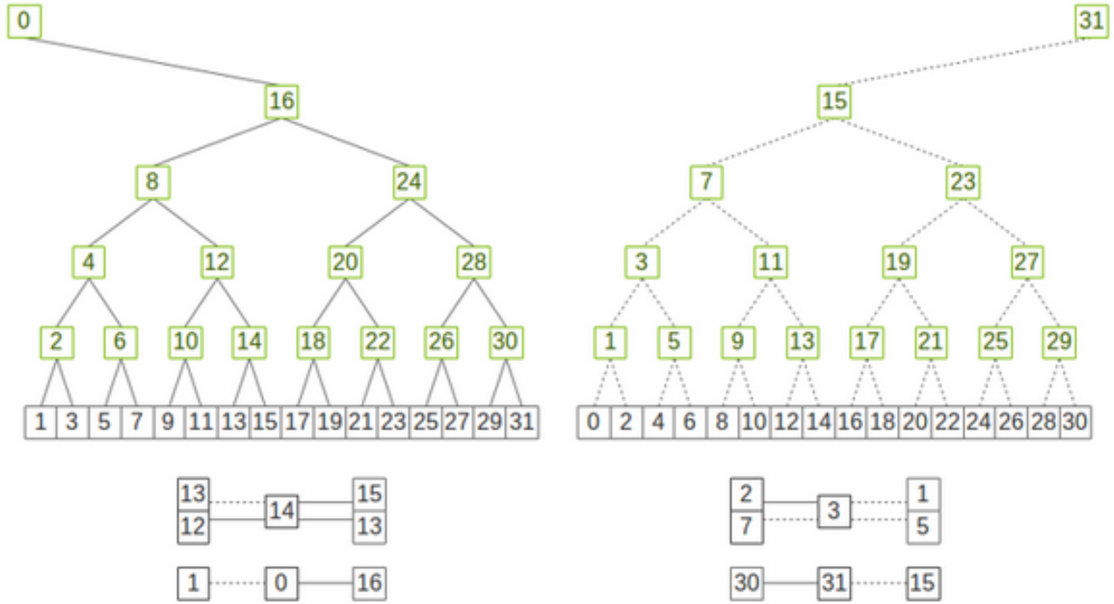


NCCL Communication Patterns

Map to Rings and Trees



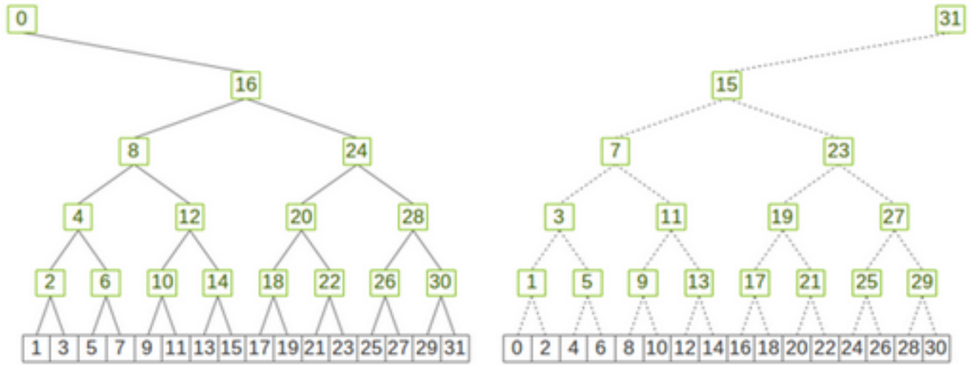
Adapt to most topologies
Full bandwidth
Linear latency



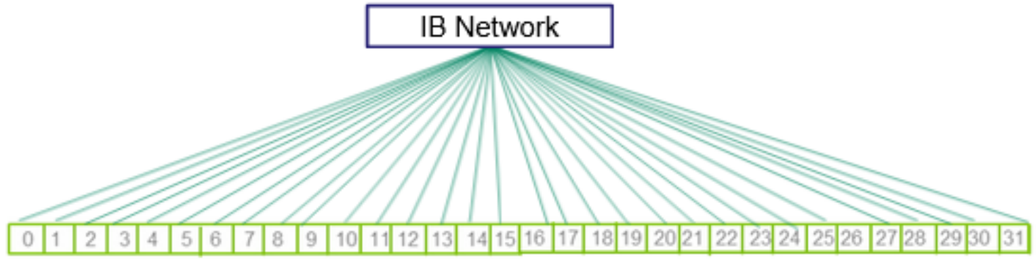
Use double binary tree for inter-node

DOUBLE BW + LOWER LATENCY

Switch-based Reductions with SHARP



Traditional Tree Reduction



SHARP Reduction

Sends data once, receives final result

- No intermediate results: effectively doubles bandwidth
- Fewer hops (1 per switch level): lower latency
- Offload GPU computation
- Less interference with other tenants

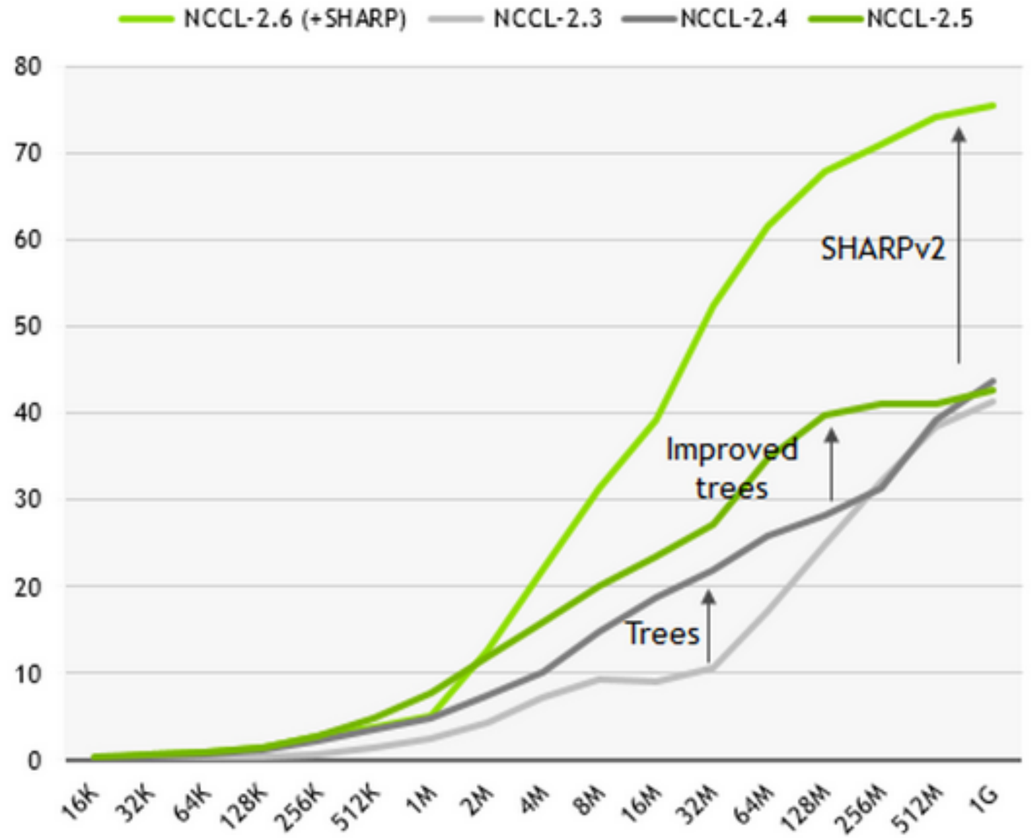
PERFORMANCE

Getting peak BW, at scale

Constantly improving performance

Trees (2.4)
Split trees (2.5)
Add support for adaptive routing (2.6)
In-network all-reduce operations (2.6)

Allow for large scale training, minimize communication time.



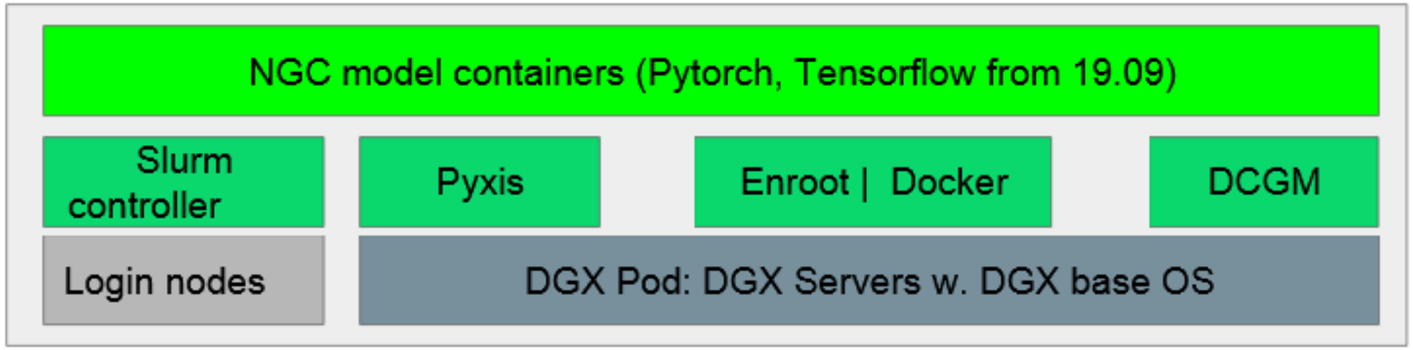
32xDGX1V + 4xMellanox CX-6, NCCL AllReduce BusBW in GB/s

NVIDIA

SCALE TO MULTIPLE NODES

Software stack - System

- Slurm: User job scheduling & management
- Enroot: NVIDIA open-source tool to convert traditional container/OS images into unprivileged sandboxes
- Pyxis: NVIDIA open-source plugin integrating Enroot with Slurm



- DeepOps: NVIDIA open-source toolbox for GPU cluster management w/Ansible playbooks

Multi node IB compute

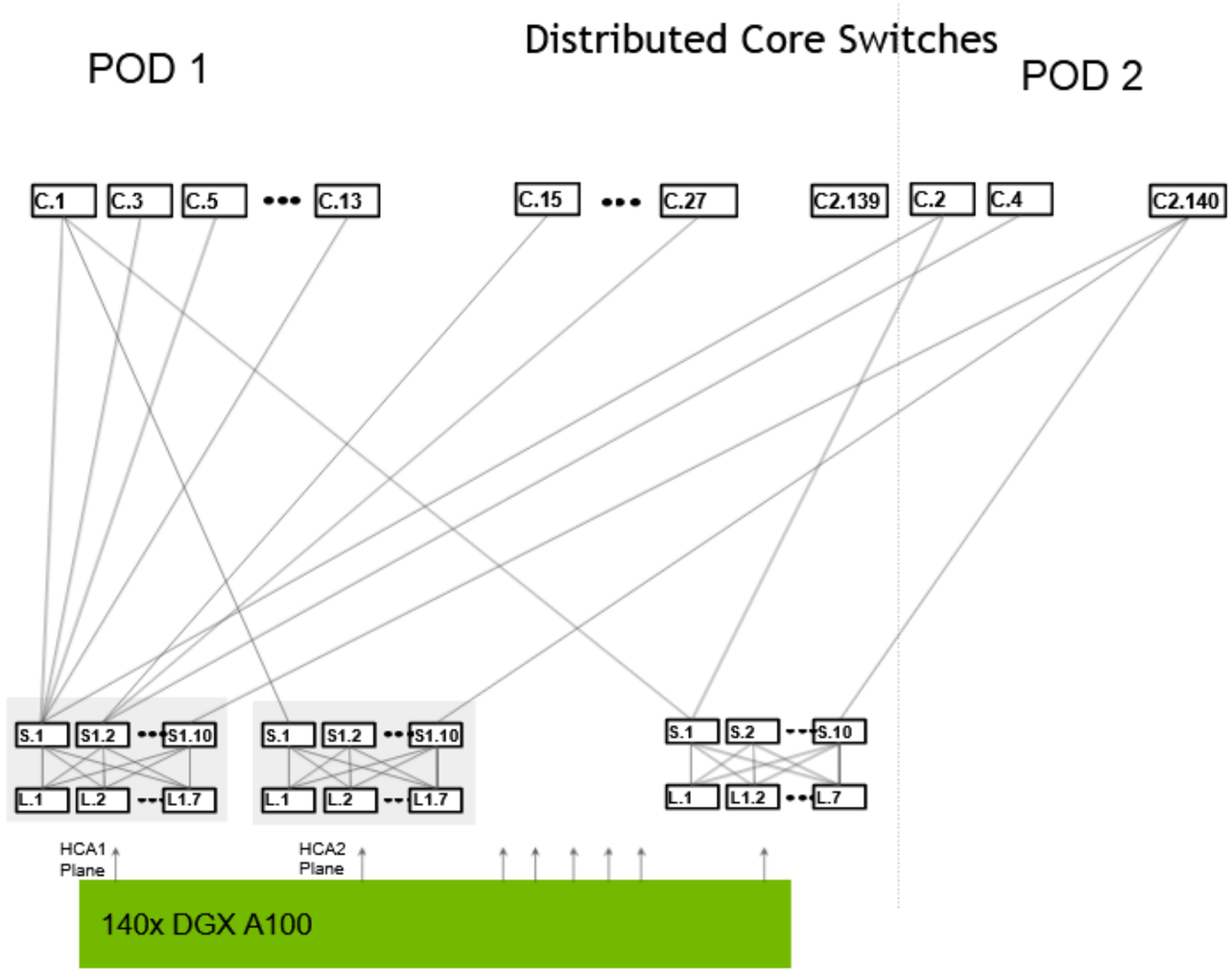
The details

Designed with Mellanox 200Gb HDR IB network

Separate compute and storage fabric
- 8 Links for compute
- 2 Links for storage (Lustre)
- Both networks share a similar fat-tree design

Modular POD design
- 140 DGX A100 nodes are fully connected in a POD
- POD contains compute nodes and storage
- All nodes and storage are usable between PODs

Sharp optimized design
- Leaf and Spines organized in HCA planes
- For a POD, all HCA1 from 140 DGX-2 connect to a HCA1 Plane fat-tree network
- Traffic from HCA1 to HCA1 between any two nodes in a POD stay either at the Leaf or Spine level
- Only use core switches when
  - 1. Moving data between HCA planes (e.g. mlx5_0 to mlx5_1 in another system)
  - 2. Moving any data between PODs



NVIDIA DGX systémy

Parametr	DGX A100	DGX-2	DGX-1	DGX Station
GPUs	8× NVIDIA A100 40GB	16× NVIDIA Tesla V100 32GB	8× NVIDIA Tesla V100 32GB	4× NVIDIA Tesla V100 32GB
Výkon (tensor operace)	5 PetaFLOPS	2 PetaFLOPS	1 PetaFLOPS	0,5 PetaFLOPS
GPU paměť	320 GB celkem	512 GB celkem	256 GB celkem	128 GB celkem
CPU	2× AMD 7742, 2.7 GHz 64c	2× Intel 8168, 2.7 GHz 24c	2× Intel E5-2698 v4 2.2GHz 20c	Intel E5-2698 v4 2.2GHz 20c
NVIDIA CUDA cores		81 920	40 960	20 480
NVIDIA Tensor cores		10 240	5 120	2 560
Propojení GPU karet	NVSwitch, non-blocking 2,4TB/s	NVSwitch, non-blocking, 2,4TB/s	NVLink, hypercube topologie	NVLink
RAM	1 TB	1,5 TB	512 GB	256 GB
HDD	2× 960GB NVME SSD 8× 3.84TB NVME SSD	2× 960GB NVME SSD, 8× 3.84TB NVME SSD	4× 1,92TB SSD	4× 1,92TB SSD
Network	1× 200GbE dual-port Eth. 8× 200Gb HDR Infiniband	2× 10/25GbE, 8× 100Gb EDR Infiniband/Ethernet	2× 10GbE, 4× 100Gb EDR Infiniband/Ethernet	2× 10GbE
Maximální příkon	6,6 kW	10 kW	3 500 W	1 500 W
Provedení	rack, 6U	rack, 10U	rack, 3U	tower, vodní chlazení CPU a GPU

Představení M Computers

Dodávky AI infrastruktury a superpočítačů

intel Technology Provider Platinum 2016
HPC Data Center Specialist

NVIDIA ELITE PARTNER

Business Partner Premium Gold Lenovo

Premier Business Partner IBM

AMD EPYC Partner Program ELITE

GIGABYTE

Hewlett Packard Enterprise Solution Provider Hybrid IT SILVER PARTNER

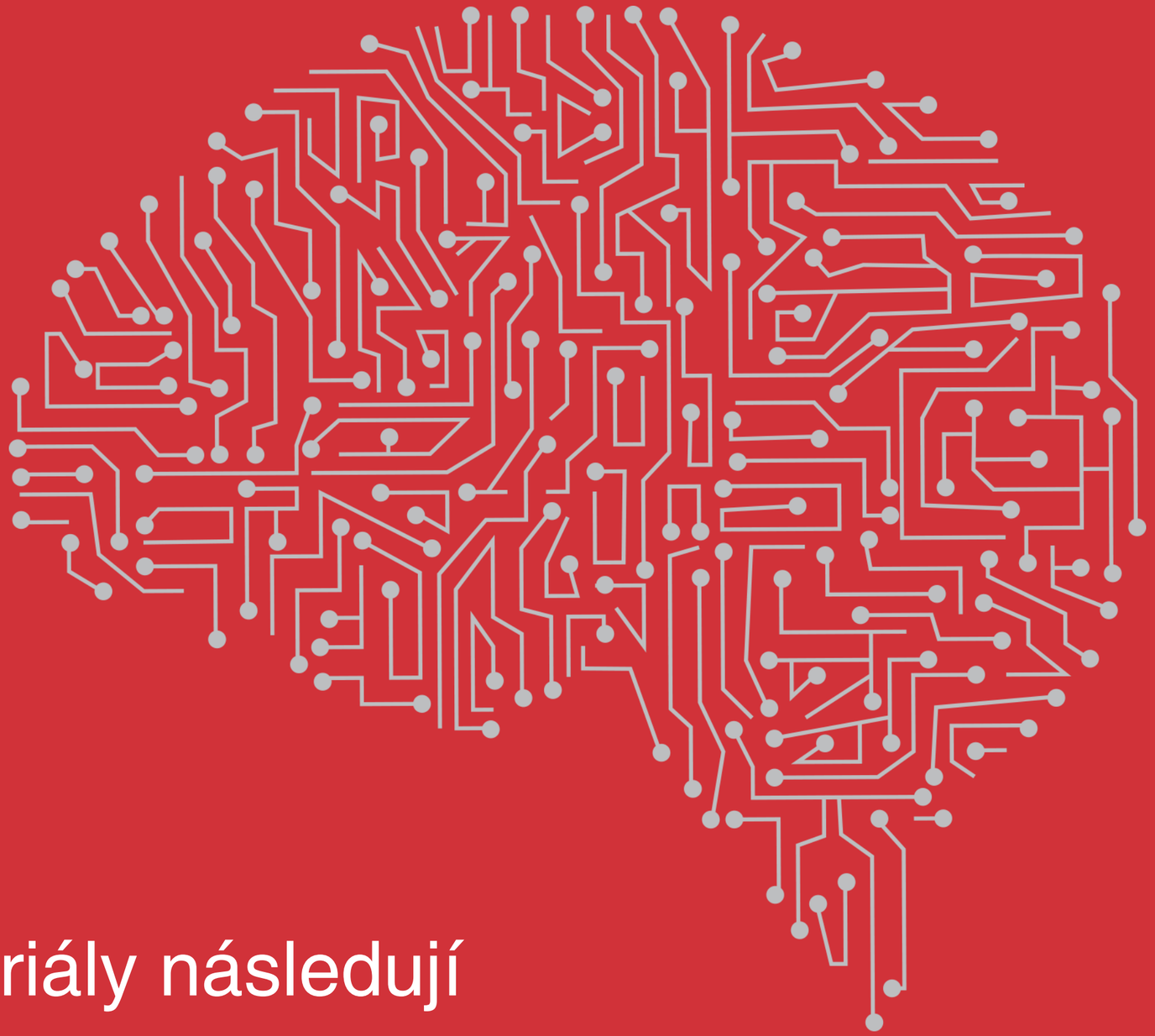
SUPERMICRO

vmware PARTNER PROFESSIONAL

NVIDIA ELITE PARTNER

Jsme jediný ELITE partner ve střední a východní Evropě
Nabízíve slevy pro vysoké školy, výzkumné instituce a start-upy
Demo pool nejnovějších NVIDIA technologií
- NVIDIA DGX Station
- NVIDIA Tesla akcelerátory (V100, T4)
- NVIDIA Jetson produkty

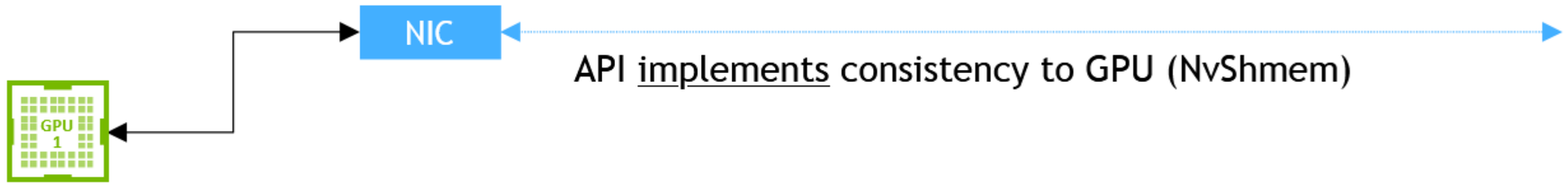
Máme tým certifikovaných specialistů



Další materiály následují

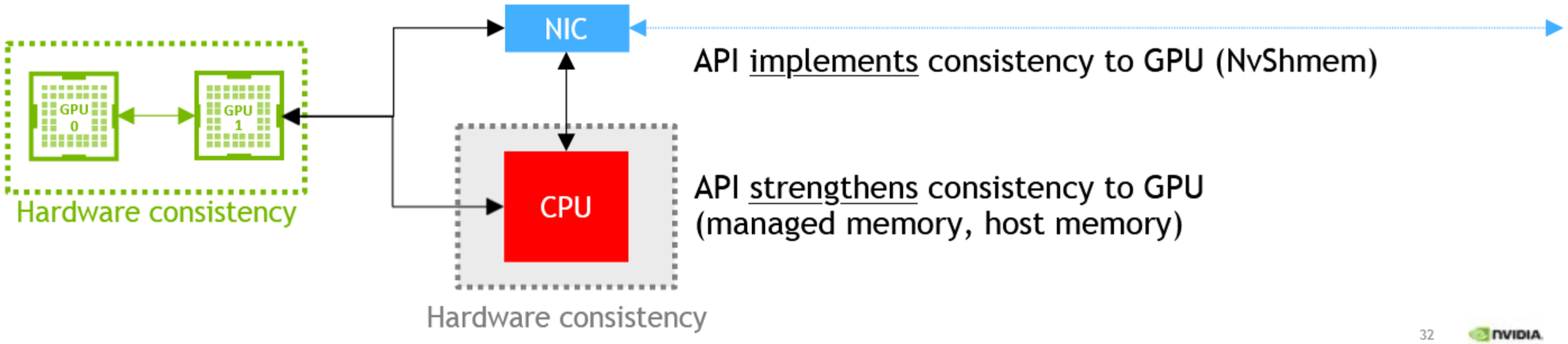
NVLINK: ONE BIG GPU

- **InfiniBand/Ethernet**: travels a long distance, consistency is the responsibility of software



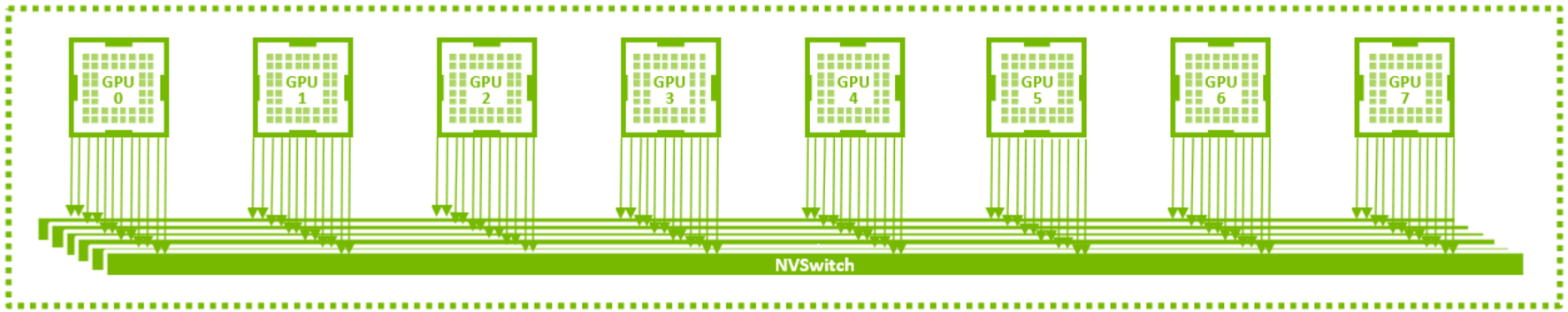
NVLINK: ONE BIG GPU

- **InfiniBand/Ethernet**: travels a long distance, consistency is the responsibility of software
- **PCI Express**: hardware consistency for I/O, not for programming language memory models
- **NVLINK**: hardware consistency for programming language memory models, like system bus



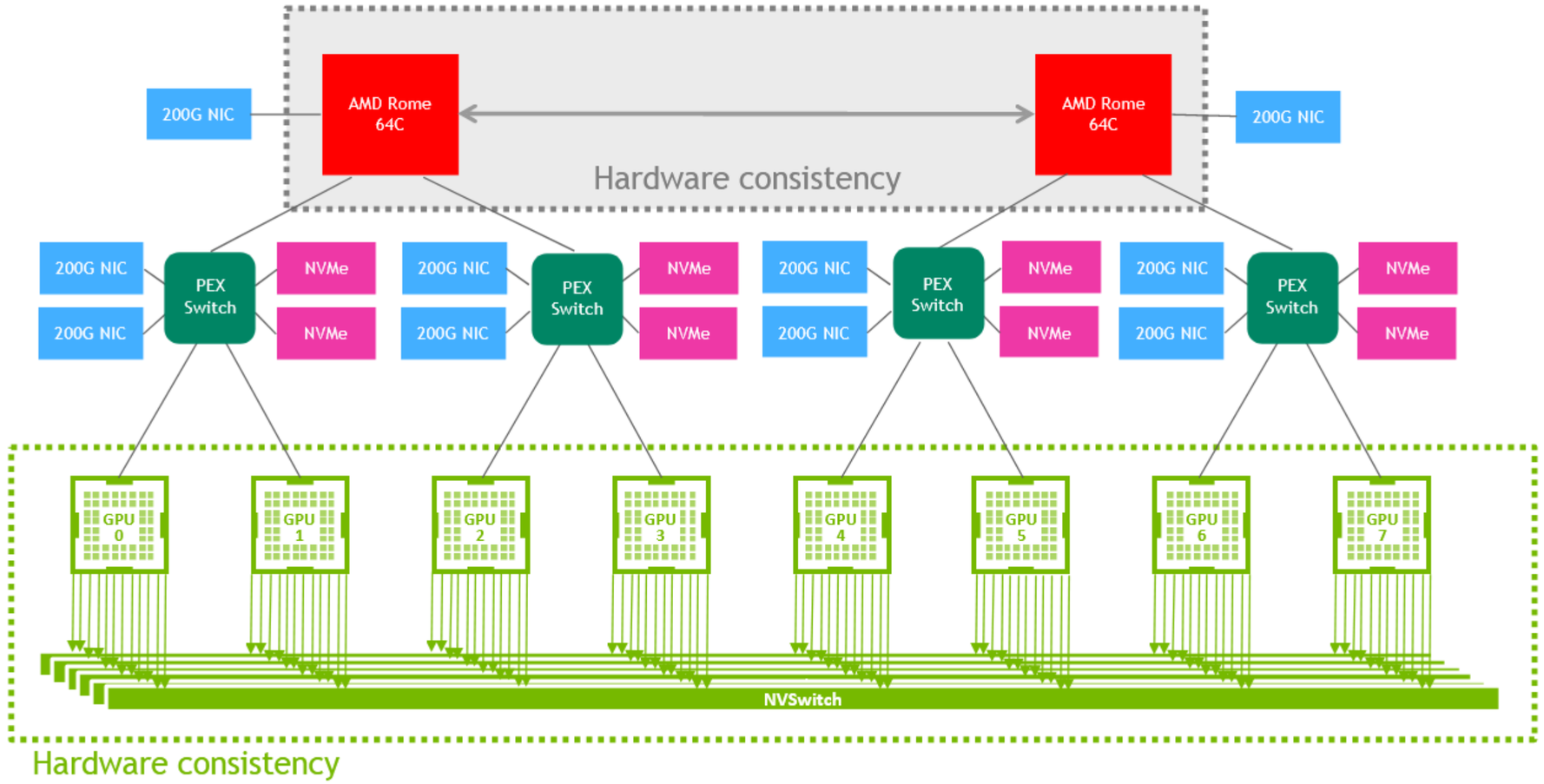
HGX A100: 3RD GEN NVLINK & SWITCH

- **HGX A100 4-GPU:** fully-connected system with 100GB/s all-to-all BW
- **New NVSwitch:** 6B transistors in TSMC 7FF, 36 ports, 25GB/s each, per direction
- **HGX A100 8-GPU:** 6x NVSwitch in a fat tree topology, 2.4TB/s full-duplex bandwidth



Hardware consistency

DGX A100: PCIE4 CONTROL & I/O



GPU PROGRAMMING IN 2020 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

GPU Accelerated
C++ and Fortran

```
#pragma acc data copy(x,y)
{

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});

...

}
```

Incremental Performance
Optimization with Directives

```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Math Libraries

→S21766 Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing, 5/19 1:30PM PST

PROGRAMMING MODEL WANTED

Software pipelining to hide latency is hard.

```
__device__ void exhibit_A1()
{
  memcpy(/* ... */); //< blocks here
  /* more work */

  compute();         //< needed here
  /* more work */
}
```

Data

```
__device__ void exhibit_B1()
{
  compute_head();
  __syncthreads(); //< blocks here
  /* more work */

  compute_tail();  //< needed here
  /* more work */
}
```

Compute

PROGRAMMING MODEL WANTED

Software pipelining to hide latency is hard.

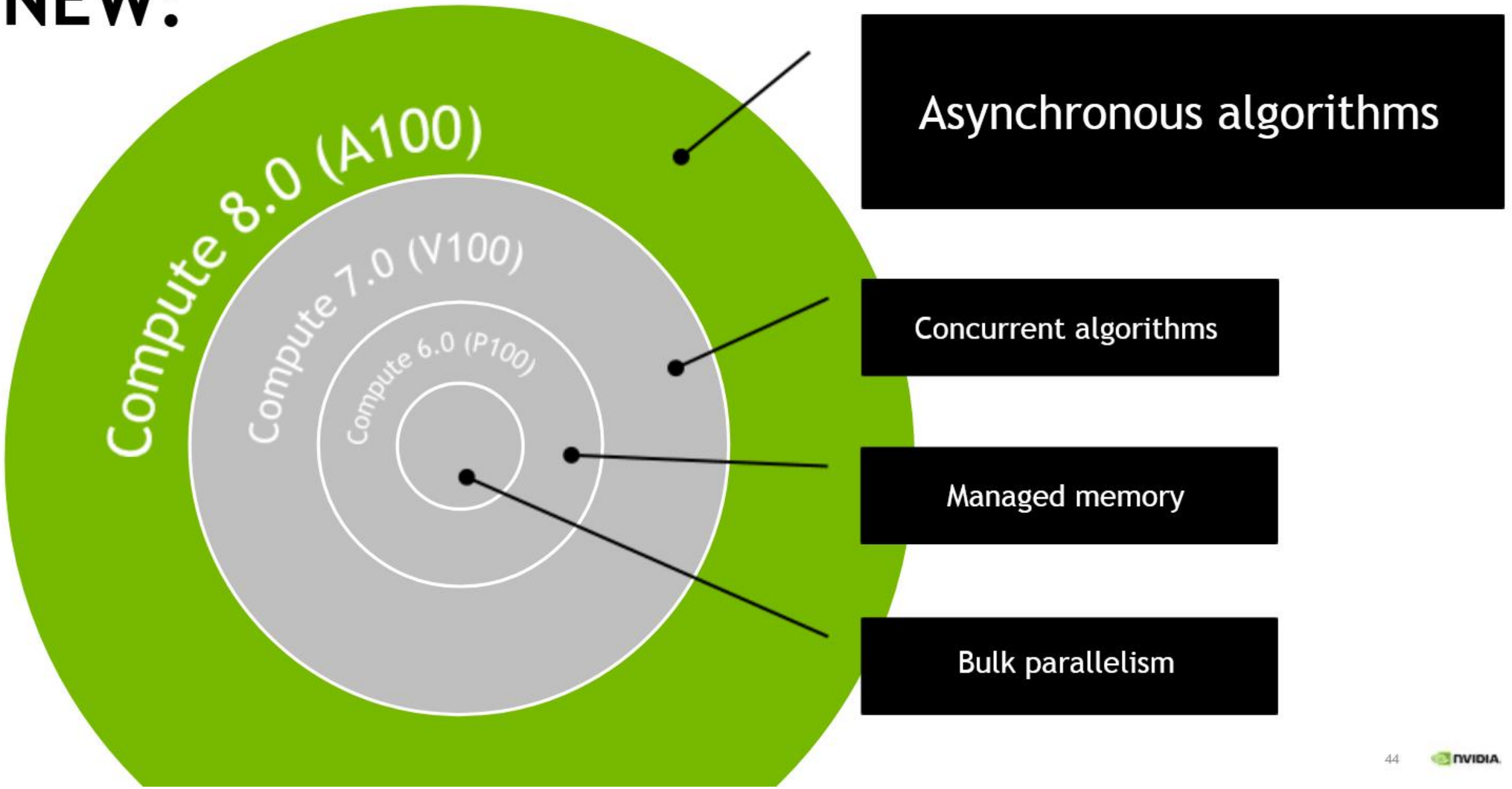
```
__device__ void exhibit_A2()
{
  memcpy(/* ... */); //< blocks here
  /* memcpy( ... ); */

  compute();         //< needed here
  /* compute(); */
}
```

Data

```
__device__ void exhibit_B2()
{
  compute_head();
  __syncthreads(); //< blocks here
  /* compute_head();
     __syncthreads(); */
  compute_tail();  //< needed here
  /* compute_tail(); */
}
```

Compute

NEW:


Compute 8.0 (A100)
Compute 7.0 (V100)
Compute 6.0 (P100)
Asynchronous algorithms
Concurrent algorithms
Managed memory
Bulk parallelism

CO-DESIGNED: A100 & C++20 barrier

Key to asynchronous programming in compute_80

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;
```

```
class barrier {  // synopsis
  //...
  void arrive_and_wait();
  arrival_token arrive(ptrdiff_t = 1);
  void wait(arrival_token &&) const;
  //...
};
```

ASYNCHRONOUS COPY + BARRIER

Capability	PTX ISA	CUDA C++ API
Asynchronous barrier	`mbarrier.{<basis functions>}`	`cuda::barrier<…>`
Asynchronous copy	`cp.async.ca +` `cp.async.mbarrier.arrive`	`cuda::memcpy_async(…)`
+Cache-bypass	`cp.async.cg`	CUDA 11 preview library in `experimental::` namespace
+Zero-fill ragged edge	`cp.async.* … wr-size, rd-size;`	
+User-level tracking	`cp.async.mbarrier.arrive.noinc`	
+Single-threaded mode	`cp.async.{commit_group, wait_group}`	

ASYNCHRONOUS PROGRAMMING MODEL

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;
```

```
__device__ void exhibit_A3()
{
  __shared__ barrier b1, b2;
  // ^^initialization omitted
   cuda::memcpy_async(/* ... */, b1);
    cuda::memcpy_async(/* ... */, b2);
  b1.arrive_and_wait();
  compute();
    b2.arrive_and_wait();
    compute();
}
```

Data

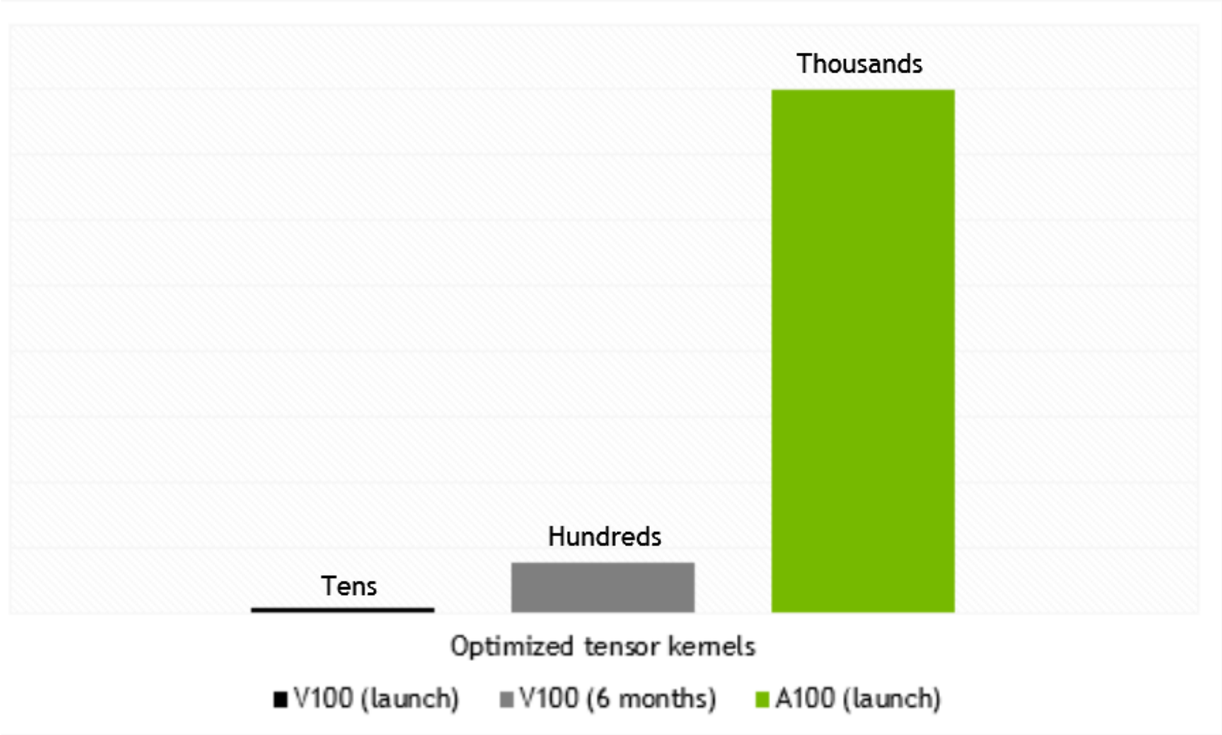
```
__device__ void exhibit_B3()
{
  __shared__ barrier b1, b2;
  // ^^initialization omitted
  compute_head();
  auto t1 = b1.arrive();
    compute_head();
    auto t2 = b2.arrive();
  b1.wait(t1);
  compute_tail();
    b2.wait(t2);
    compute_t
}
```

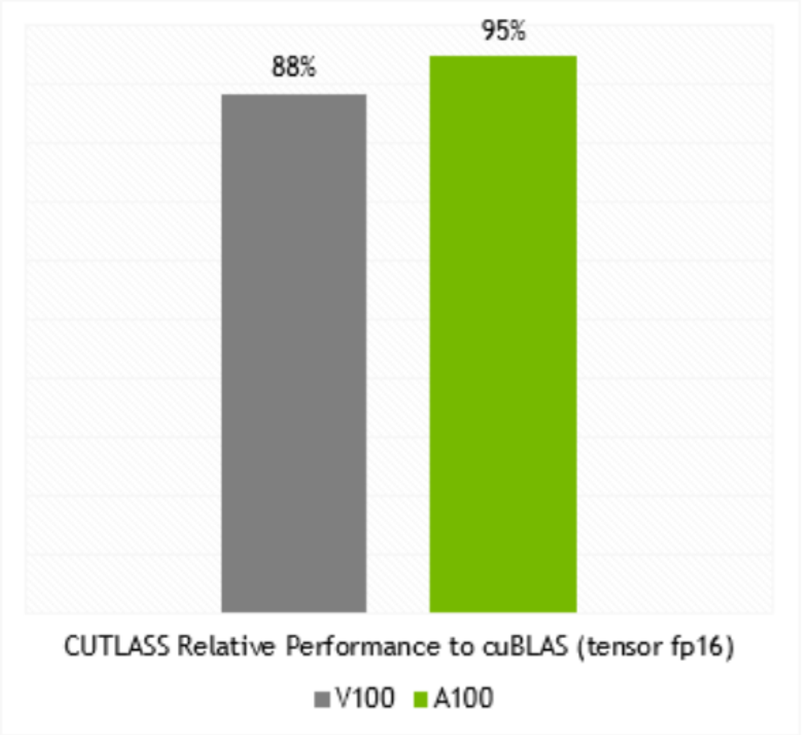
Compute

MULTI-BUFFERING PIPELINES IN C++

```
#include <cuda/barrier> // ISO C++20 conforming extension
using barrier = cuda::barrier<cuda::thread_scope_block>;

__global__ void exhibit_C(/* ... */) {
  __shared__ barrier b[2];
  // ^^initialization omitted
  barrier::arrival_token t[2];
  cuda::memcpy_async(/* ... */, b[0]);
  t[0] = b[0].arrive();
  for(int step = 0, next = 1; step < steps; ++step, ++next) {
    if(next < steps) {
      b[next & 1].wait(t[next & 1]);
      cuda::memcpy_async(/* ... */, b[next & 1]);
      t[next & 1] = b[next & 1].arrive();
    }
    b[step & 1].wait(t[step & 1]);
    compute();
    t[step & 1] = b[step & 1].arrive();
  }
}
```

OUR PRODUCTIVITY GAINS FROM A100

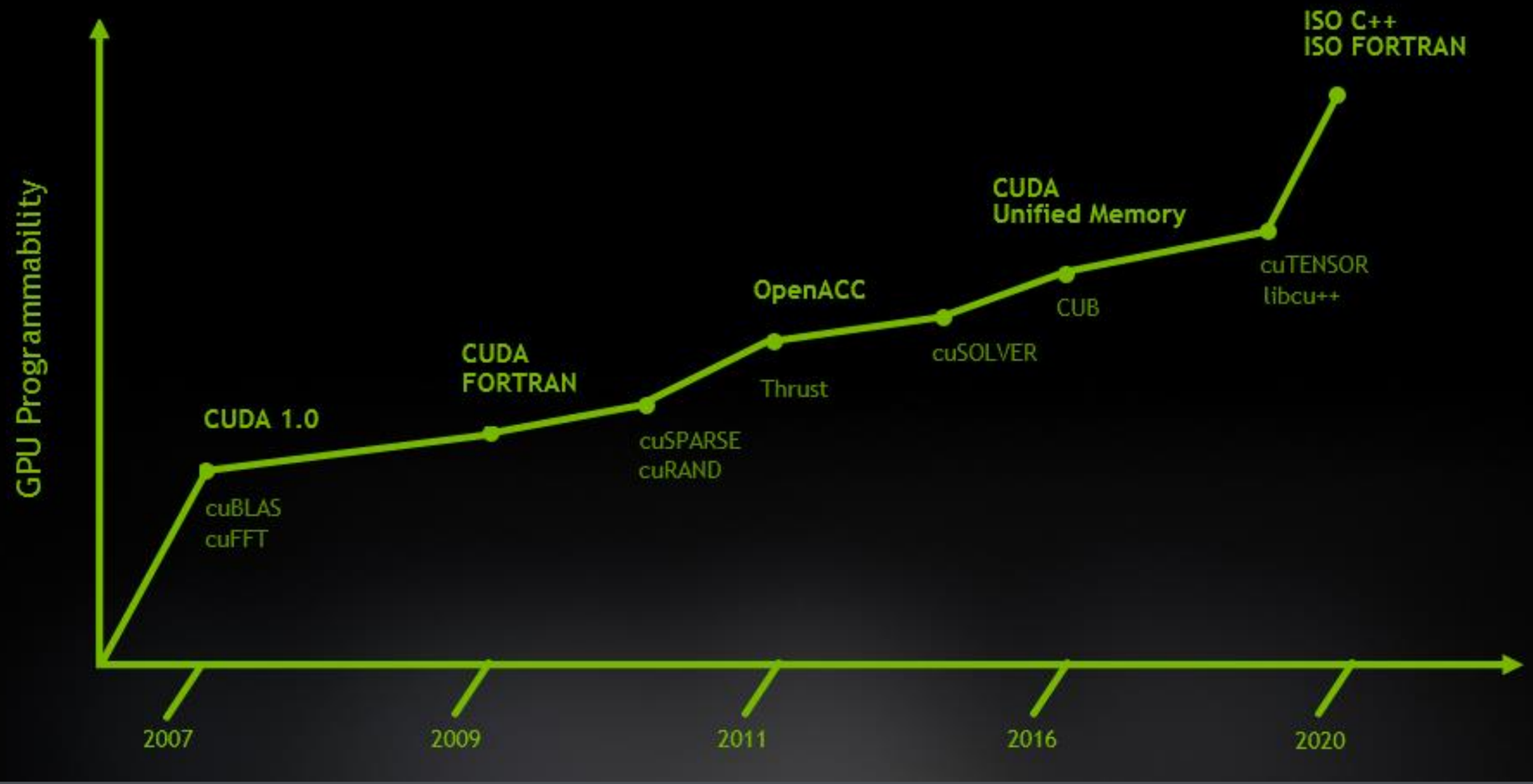




→S21745: Developing CUDA kernels to push Tensor Cores to the Absolute Limit on NVIDIA A100, 5/21 11:30AM PST

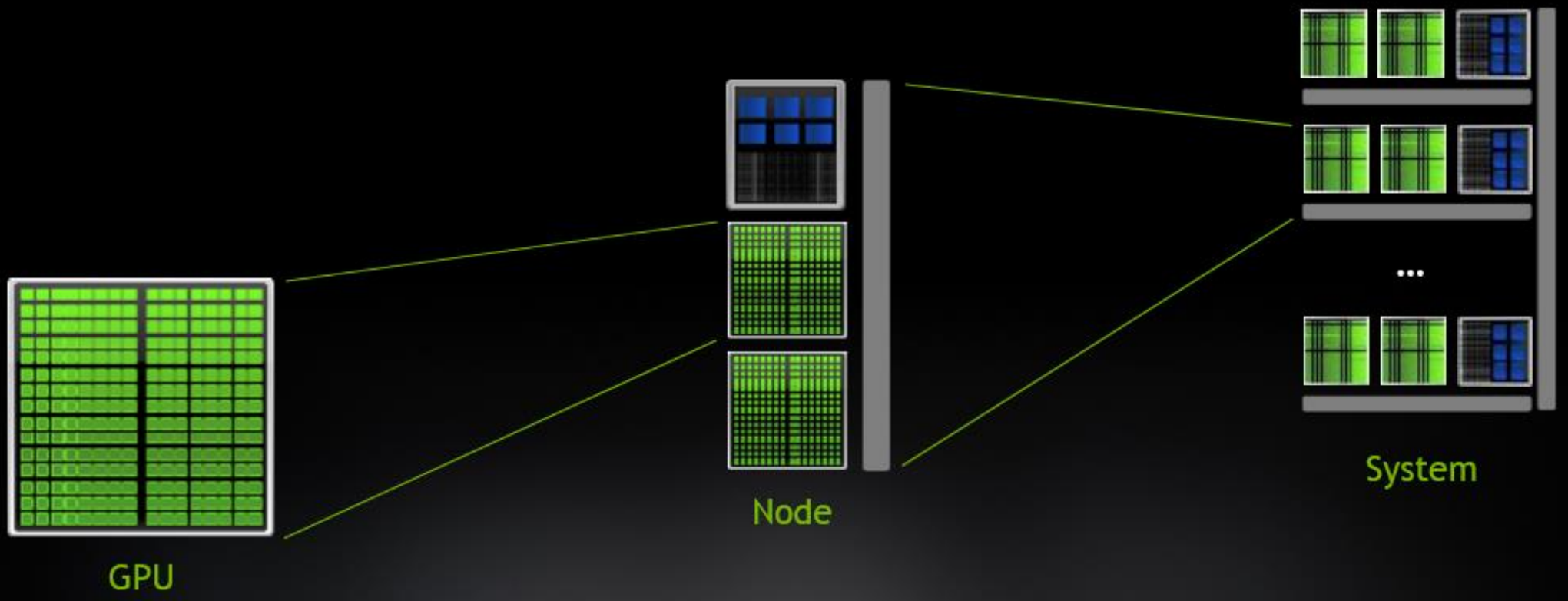
EVOLUTION OF HPC GPU PROGRAMMING

Libraries and Programming Models



PROGRAMMING GPU-ACCELERATED HPC SYSTEMS

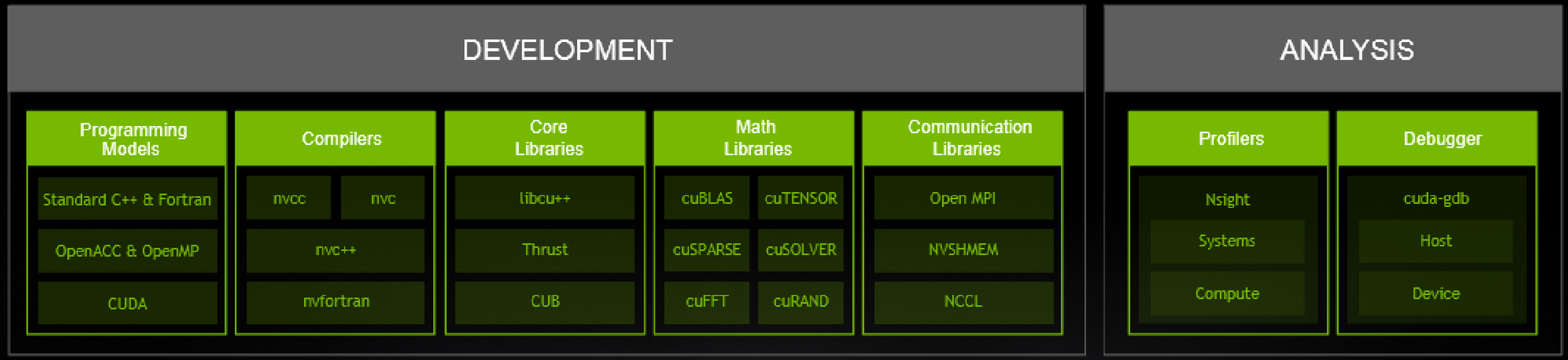
GPU | CPU | Interconnect



ANNOUNCING: THE NVIDIA HPC SDK

Apply now at developer.nvidia.com/hpc-sdk

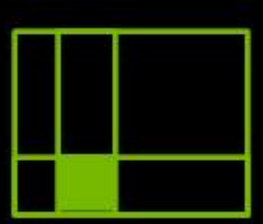
NVIDIA HPC SDK



Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect
HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA
Compatible with HPC Container Maker and 99% of Top500 Systems

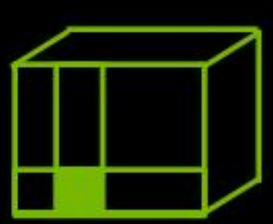
NVIDIA MATH LIBRARIES

Linear Algebra, FFT, RNG and Basic Math



cuBLAS

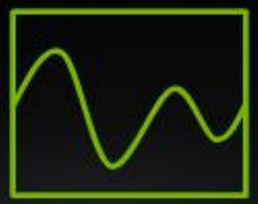
cuSPARSE



cuTENSOR

cuSOLVER

cuRAND

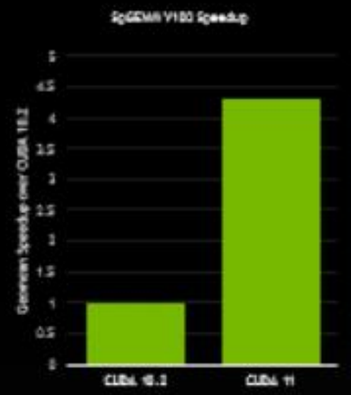


cuFFT

CUDA Math API

NVIDIA MATH LIBRARIES

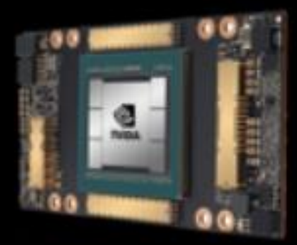
Major Initiatives



Performance
Tuning
New algorithms

Extended Features
New libraries
New APIs

Multi-GPU
Strong Scaling
Weak scaling



Single GPU
Tensor Cores
Device Functions

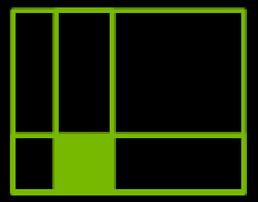
NVIDIA A100

At a glance

	NVIDIA V100	NVIDIA A100
SMs	80	108
Peak FP16 Tensor Core TFLOPS	125	312
Peak Bfloat16 Tensor Core TFLOPS	NA	312
Peak TF32 Tensor TFLOPS	NA	156
Peak FP64 Tensor TFLOPS	NA	19.5
Peak FP16 TFLOPS	31.4	78
Peak Bfloat16 TFLOPS	NA	39
Peak FP32 TFLOPS	15.7	19.5
Peak FP64 TFLOPS	7.8	9.7
Memory Size	32 GB / 16 GB	40 GB
Memory Bandwidth	900 GB/sec	1.6 TB/sec
L2 Cache Size	6144 KB	40960 KB
Shared Memory Size / SM	Configurable up to 96 KB	Configurable up to 164 KB

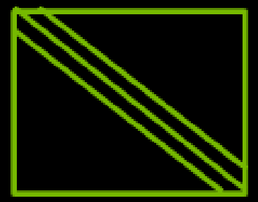
A100 FEATURES IN LIBRARIES

Automatic Acceleration of Critical Routines in HPC and AI



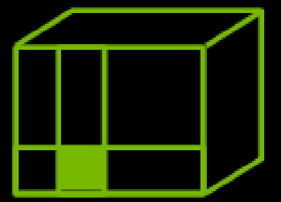
cuBLAS

BF16, TF32 and FP64 Tensor Cores



cuSPARSE

Increased memory BW, Shared Memory and L2

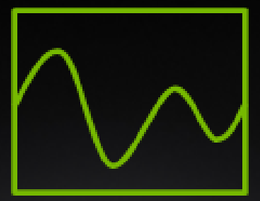


cuTENSOR

BF16, TF32 and FP64 Tensor Cores

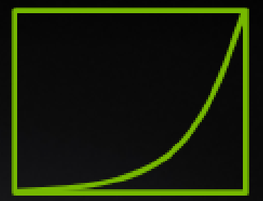
cuSOLVER

BF16, TF32 and FP64 Tensor Cores



cuFFT

Increased memory BW, Shared Memory and L2



CUDA Math API

BF16 Support

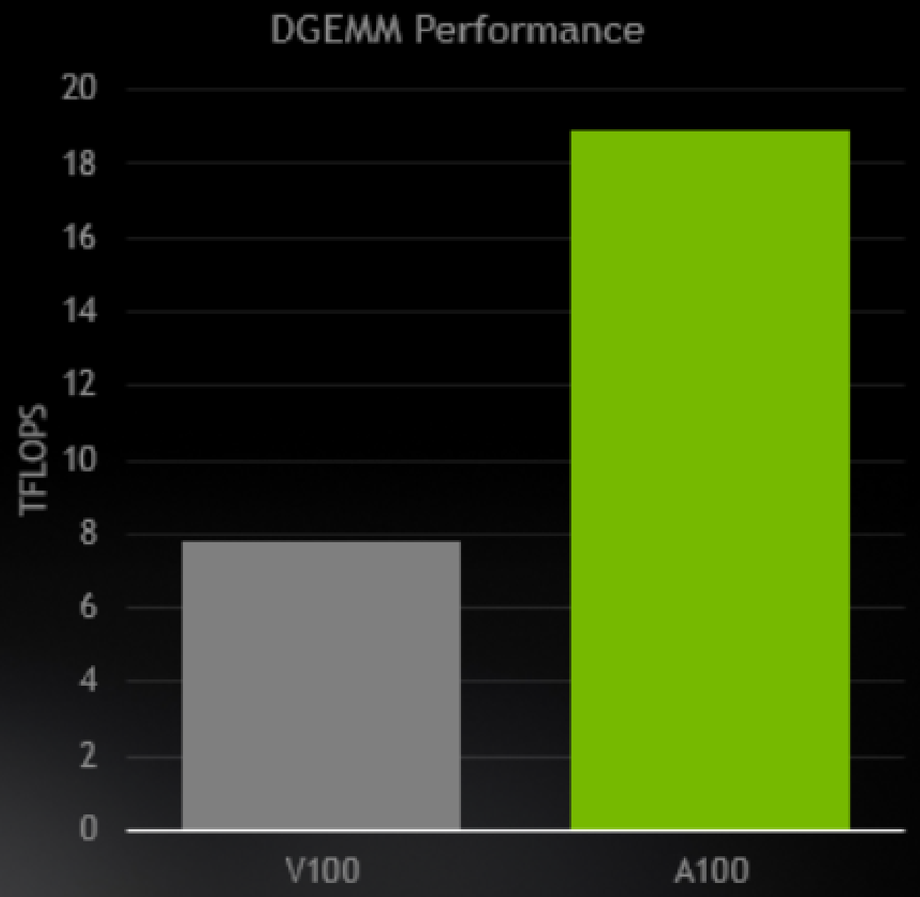
A100 TENSOR CORES IN LIBRARIES

cuBLAS

- Automatic Tensor Core acceleration
- Removed matrix size restrictions for Tensor Core acceleration

DGEMM on A100

- Up to 19 TFLOPs, 2.4x V100



A100 TENSOR CORES IN LIBRARIES

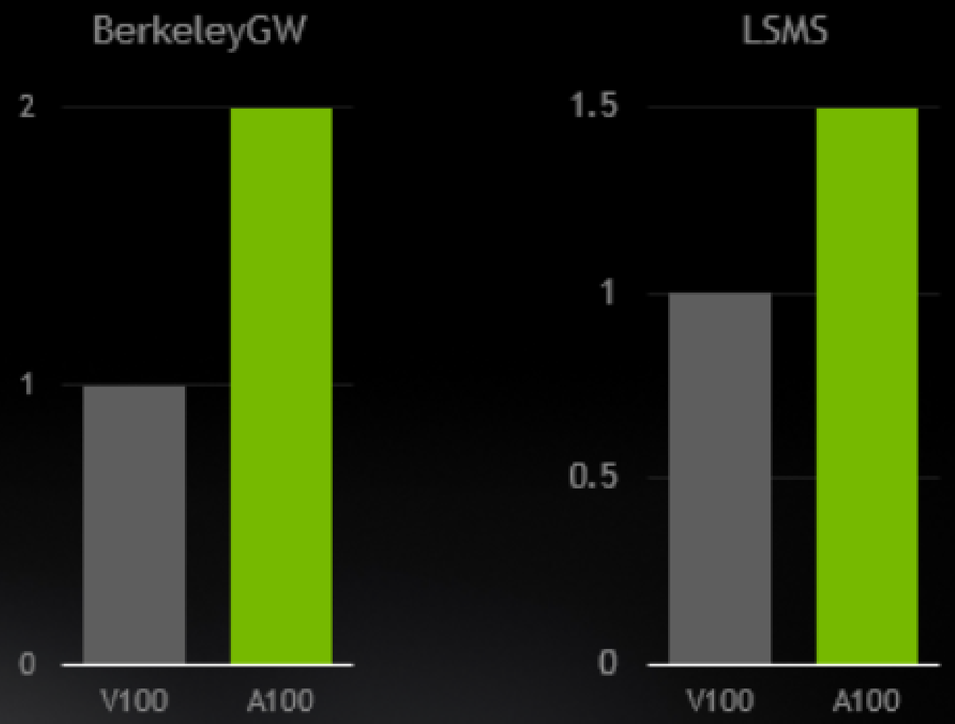
Application Speedup

BerkeleyGW: Materials Science

- Up to 70% ZGEMM on V100
- Automatic DMMA acceleration with cuBLAS
- 2X End-to-End Speedup V100 to A100

LSMS: Density Functional Theory

- Up to 50% ZGEMM on V100
- Automatic DMMA acceleration with cuBLAS
- 1.5X End-to-End Speedup V100 to A100



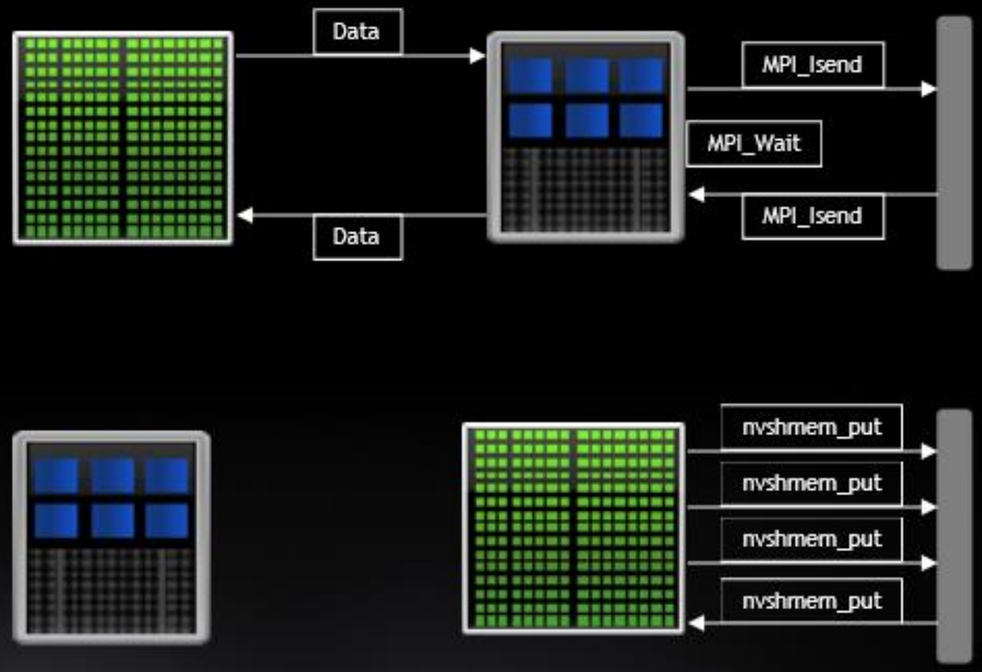
INTRODUCING NVSHMEM

GPU Optimized OpenSHMEM

- Initiate from CPU or GPU
- Initiate from within CUDA kernel
- Issue onto a CUDA stream
- Interoperable with MPI & OpenSHMEM
- Up to 5X better parallel efficiency

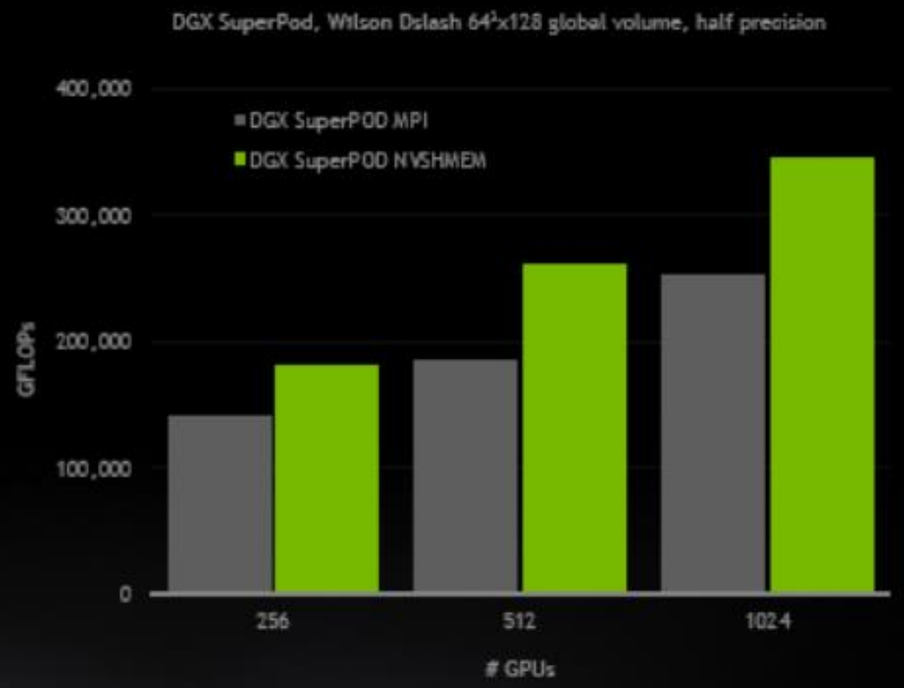
Pre-release Impact

- LBANN, Kokkos/CGSolve, QUDA



INTRODUCING NVSHMEM

Impact in HPC Applications



QUDA: Quantum Chromodynamics on CUDA

- Up to 1.7X Single Node Speedup
- Up to 1.4X Multi Node Speedup

NVIDIA

HPC COMPILERS

Programming the HPC Platform

NVC++

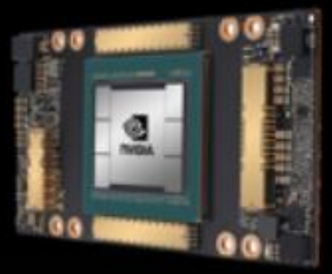
NVC

NVFORTRAN

NVCC

HPC COMPILERS

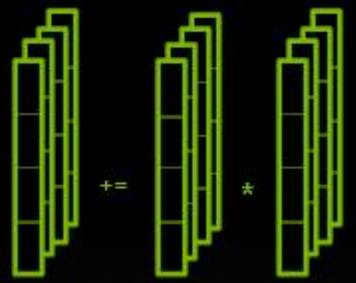
NVC | NVC++ | NVFORTRAN



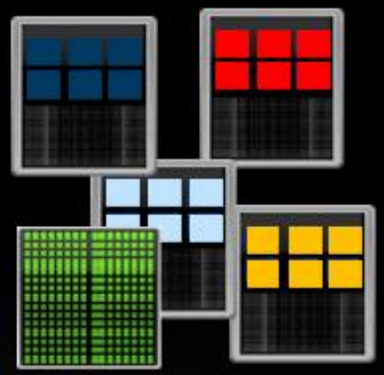
Accelerated
A100
Automatic

Fortran
NVIDIA CUDA
OpenACC
OpenMP

Programmable
Standard Languages
Directives
CUDA



Multicore
Directives
Vectorization



Multi-Platform
x86_64
Arm
OpenPOWER

HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

C++17

Parallel Algorithms

- In NVC++ 20.5
- Parallel and vector concurrency

Forward Progress Guarantees

- Extend the C++ execution model for accelerators.

Memory Model Clarifications

- Extend the C++ memory model for accelerators.

C++20

Scalable Synchronization Library

- Express thread synchronization that is portable and scalable across CPUs and accelerators.
- In libcu++ in CUDA 10.2:
  - `std::atomic<T>`
- In libcu++ in CUDA 11.0:
  - `std::barrier`
  - `std::counting_semaphore`
  - `std::atomic<T>::wait/notify_*`
- In libcu++ in the future:
  - `std::atomic_ref<T>`

C++23 and Beyond

Executors

- Simplify launching and managing parallel work across CPUs and accelerators.

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Linear Algebra

- Standard C++ API to vendor BLAS libraries.

Extended Floating Point Types

- First-class support for formats new and old: `std::float16_t/float64_t`

HPC PROGRAMMING IN ISO C++

C++ Parallel Algorithms

```
std::sort(std::execution::par, c.begin(), c.end());

std::unique(std::execution::par, c.begin(), c.end());
```

- Introduced in C++17
- Parallel and vector concurrency via execution policies

  `std::execution::par`, `std::execution::par_seq`, `std::execution::seq`
- Several new algorithms in C++17 including
  - `std::for_each_n(POLICY, first, size, func)`
- Insert `std::execution::par` as first parameter when calling algorithms
- NVC++ 20.5: automatic GPU acceleration of C++17 parallel algorithms
  - Leverages CUDA Unified Memory

```
static inline
void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                                 Index_t *regElemlist, Real_t dvovmax, Real_t& dthydro)
{
#if _OPENMP
   const Index_t threads = omp_get_max_threads();
   Index_t hydro_elem_per_thread[threads];
   Real_t dthydro_per_thread[threads];
#else
   Index_t threads = 1;
   Index_t hydro_elem_per_thread[1];
   Real_t  dthydro_per_thread[1];
#endif
#pragma omp parallel firstprivate(length, dvovmax)
   {
      Real_t dthydro_tmp = dthydro ;
      Index_t hydro_elem = -1 ;
#if _OPENMP
      Index_t thread_num = omp_get_thread_num();
#else
      Index_t thread_num = 0;
#endif
#pragma omp for
      for (Index_t i = 0 ; i < length ; ++i) {
         Index_t indx = regElemlist[i] ;

         if (domain.vdov(indx) != Real_t(0.)) {
            Real_t dtdvov = dvovmax / (FABS(domain.vdov(indx))+Real_t(1.e-20)) ;

            if ( dthydro_tmp > dtdvov ) {
                  dthydro_tmp = dtdvov ;
                  hydro_elem = indx ;
            }
         }
      }
      dthydro_per_thread[thread_num]    = dthydro_tmp ;
      hydro_elem_per_thread[thread_num] = hydro_elem ;
   }
   for (Index_t i = 1; i < threads; ++i) {
      if(dthydro_per_thread[i] < dthydro_per_thread[0]) {
         dthydro_per_thread[0]    = dthydro_per_thread[i];
         hydro_elem_per_thread[0] =  hydro_elem_per_thread[i];
      }
   }
   if (hydro_elem_per_thread[0] != -1) {
      dthydro =  dthydro_per_thread[0] ;
   }
   return ;
}
```

C++ with OpenMP

PARALLEL C++

- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable - nvc++, g++, icpc, MSVC, ...

```
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                                               Index_t *regElemlist,
                                               Real_t dvovmax,
                                               Real_t &dthydro) {
  dthydro = std::transform_reduce
      std::execution::par, counting_iterator(0), counting_iterator(length),
      dthydro, [](Real_t a, Real_t b) { return a < b ? a : b; },
      [=, &domain](Index_t i) {
        Index_t indx = regElemlist[i];
        if (domain.vdov(indx) == Real_t(0.0)) {
          return std::numeric_limits<Real_t>::max();
        } else {
          return dvovmax / (std::abs(domain.vdov(indx)) + Real_t(1.e-20));
        }
      });
}
```

Parallel C++17

LULESH PERFORMANCE

Speedup - Higher is Better