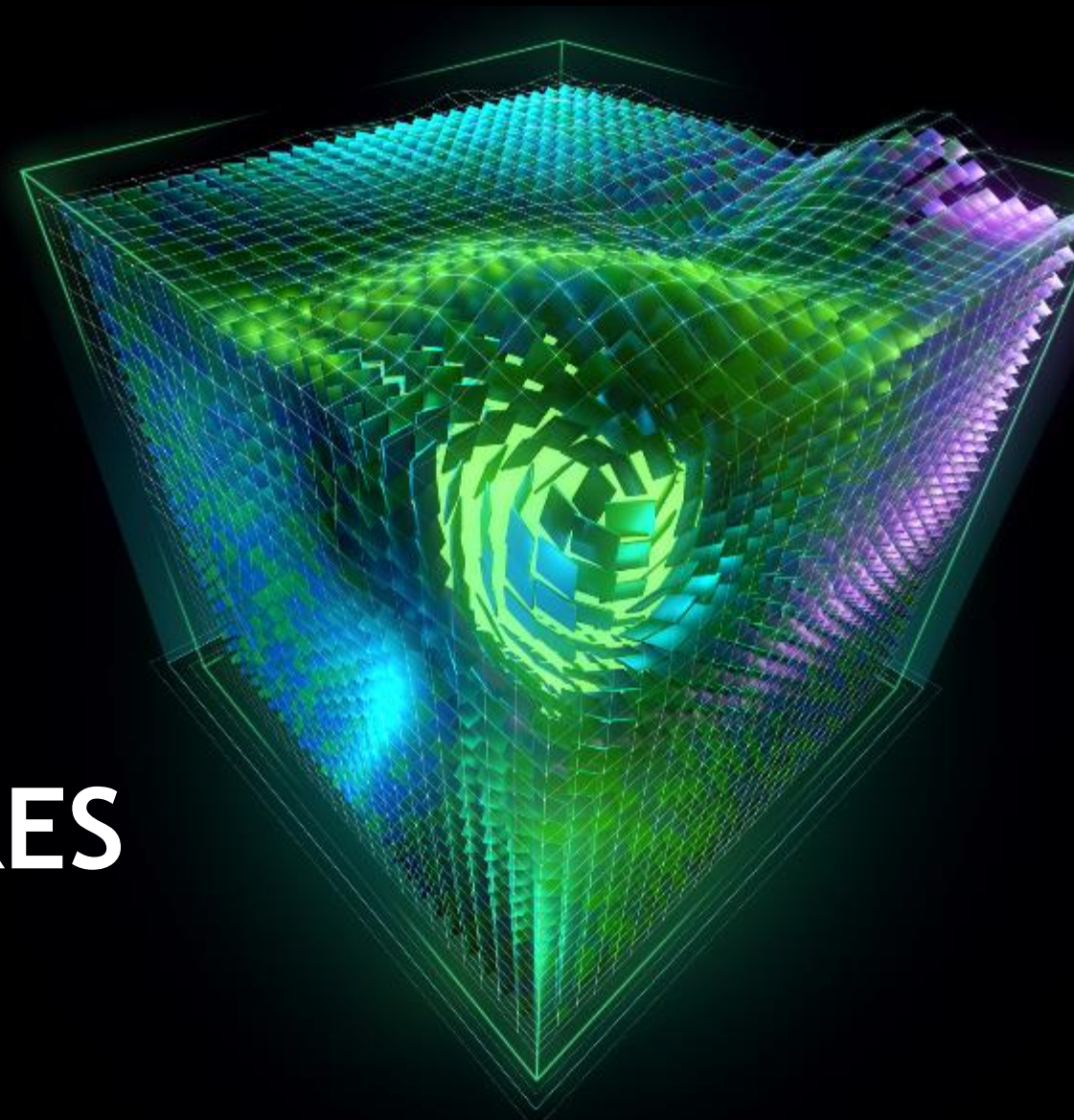
GPU TECHNOLOGY CONFERENCE

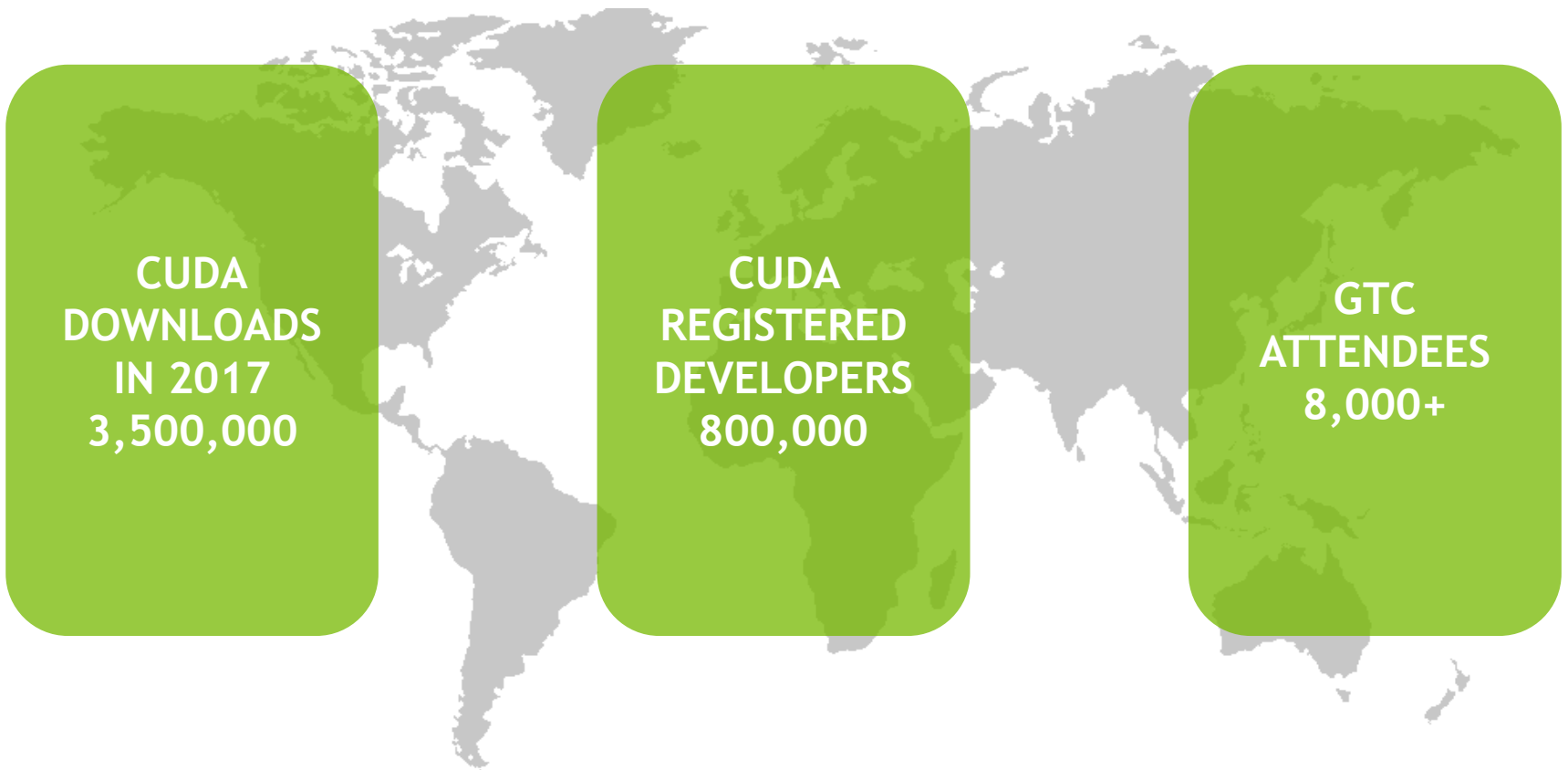


CUDA: NEW AND UPCOMING FEATURES

Stephen Jones, GTC 2018

PRESENTED BY NVIDIA.

CUDA ECOSYSTEM 2018



CUDA DEVELOPMENT ECOSYSTEM

From Ease of Use to Specialized Performance

GROMACS
FAST. FLEXIBLE. FREE.

b-initio
VASP
ienna
ackage
imulation

NAMD

ANSYS
FLUENT

Applications

Frameworks

cuDNN

TensorRT

cuBLAS

cuSPARSE

NCCL

Libraries

OpenACC
Directives for Accelerators

Thrust

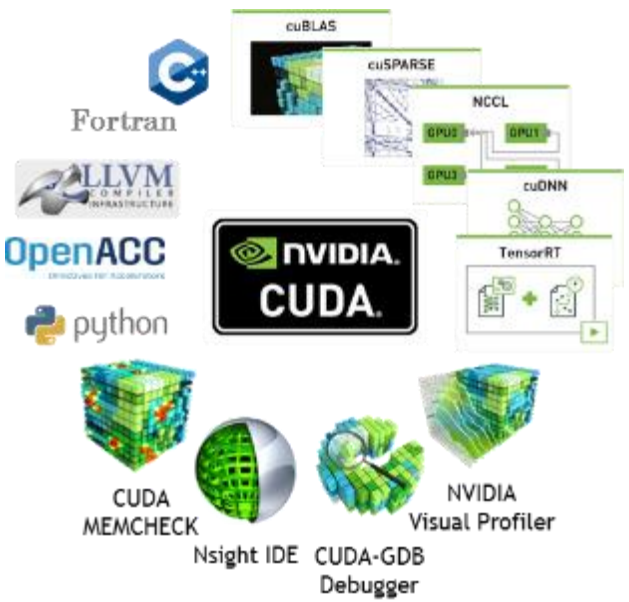
Directives and Standard Languages

CUDA-C++
CUDA Fortran

Specialized Languages

CUDA RELEASES

Accelerating the Pace



Four CUDA releases per year

Faster release cadence for new features and improved stability for existing users

Upcoming limited decoupling of display driver and CUDA release for ease of deployment

Monthly cuDNN & other library updates

Rapid innovation in library performance and functionality

Library Meta Packages independent of toolkit for easy deployment

CUDA 9.2 NEW FEATURES AT A GLANCE

SYSTEM & PERFORMANCE

Unified Memory + ATS on IBM-POWER9

Launch Latency Optimizations

DEVICE CODE IMPROVEMENTS

New WMMA sizes for Tensor Cores

Heterogeneous Half-Precision Datatypes

Volta Independent Thread Scheduling Control

MATH LIBRARIES

New CUDA Library Meta-Packages

Volta Architecture-Optimized Algorithms

TOOLS

Unified Nsight Product Family

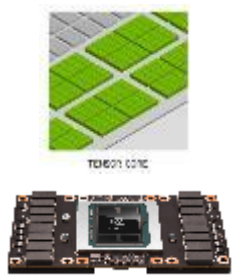
Single-Pass Tracing & Profiling

MATH LIBRARIES: WHAT'S NEW

VOLTA PLATFORM SUPPORT

Volta architecture optimized GEMMs, & GEMM extensions for Volta Tensor Cores (cuBLAS)

Out-of-box performance on Volta (all libraries)



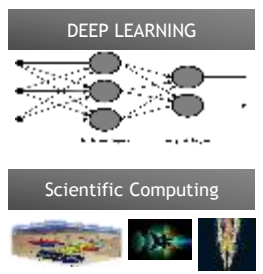
PERFORMANCE

GEMM optimizations for RNNs (cuBLAS)

Faster image processing (NPP)

Prime factor FFT performance (cuFFT)

SpMV performance (cuSPARSE)



NEW ALGORITHMS

Mixed-precision Batched GEMM for attention models (cuBLAS)

Image Augmentation and batched image processing routines (NPP)

Batched pentadiagonal solver (cuSPARSE)

MEMORY & FOOTPRINT OPTIMIZATION

Large FFT sizes on multi-GPU systems (cuFFT)

Modular functional blocks with small footprint (NPP)

TOOLS UPDATES FOR CUDA 9.2

NVPROF

New Metrics: Tensor Cores, L2, Memory Instructions Per Load/Store

PCIe Topology Display

Single-Pass Tracing & Profiling

CUPTI

New Activity Kind: PCIE

New Attribute: Profiling Scope (Device-Level, Context-Level)

Exposes New Metrics

VISUAL PROFILER

Summary View for Memory Hierarchy

Improved Handling of Segments for UVM Data on the Timeline

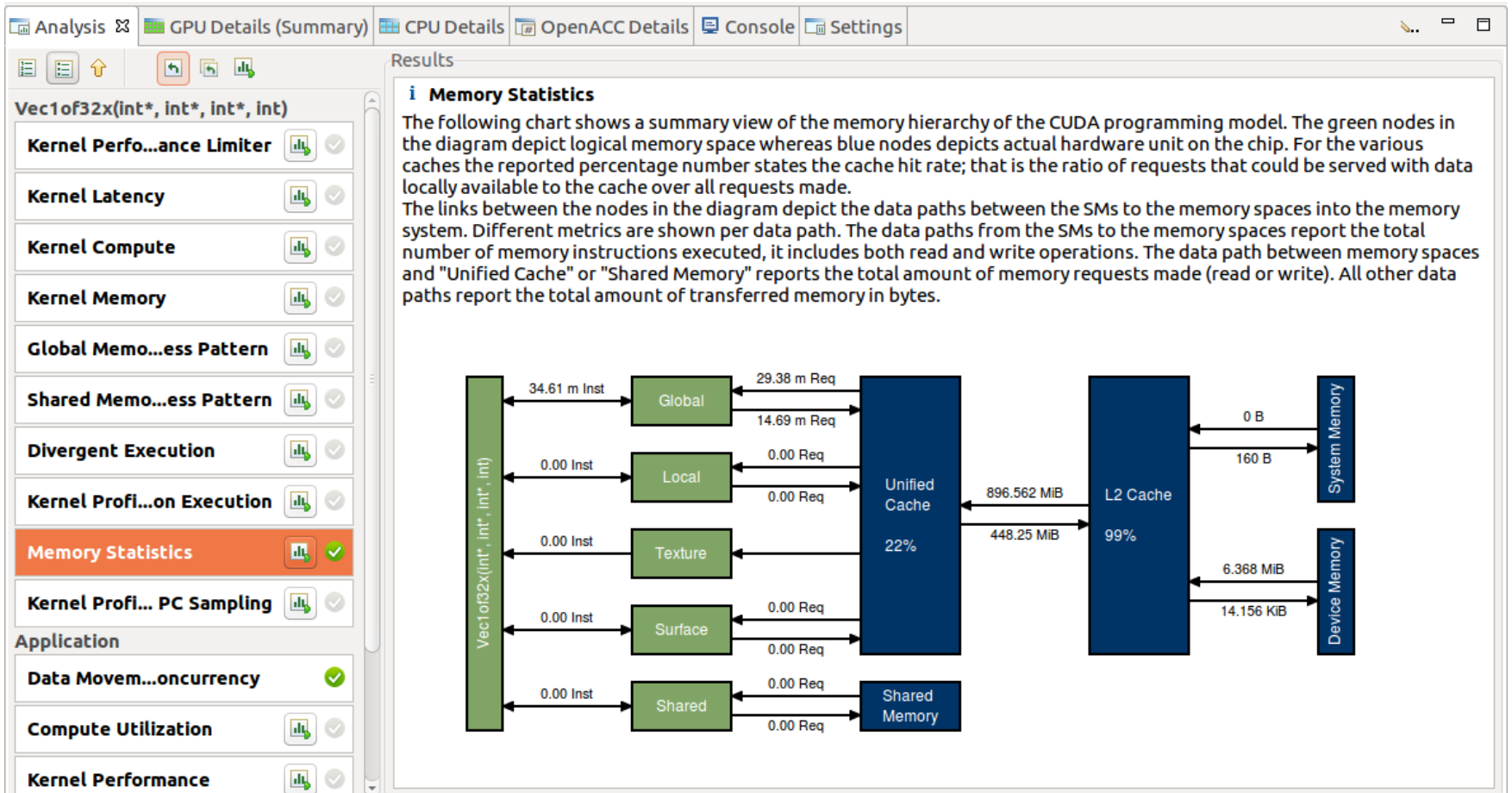
DEBUGGER

Lightweight Coredump Files

User-Induced Coredumps

Coredump Support on Volta-MPS

HIERARCHICAL MEMORY STATISTICS



NSIGHT PRODUCT FAMILY

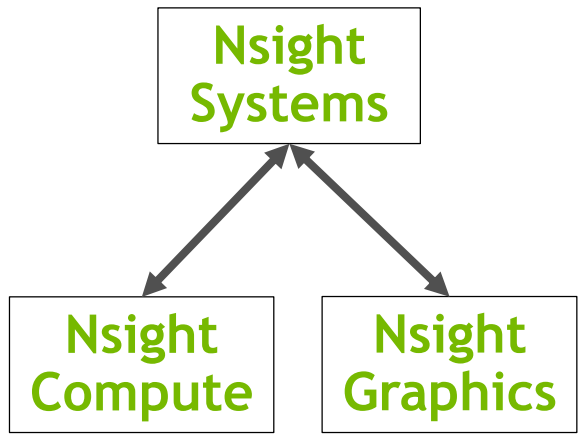
Standalone Performance Tools

Nsight Systems - System-wide application algorithm tuning

Nsight Compute - Debug/optimize specific CUDA kernel

Nsight Graphics - Debug/optimize specific graphics shader

Workflow



IDE Plugins

Nsight Visual Studio/Eclipse Edition – editor, debugger, some perf analysis

MORE INFORMATION: CUDA TOOLS

(S8726) *Debugging Updates and Details of Next-Gen Debugger*, Thurs 10:30am, Room 220C

(S8481) *Cuda Kernel Profiling*, Thursday 11:00am, Room 220C

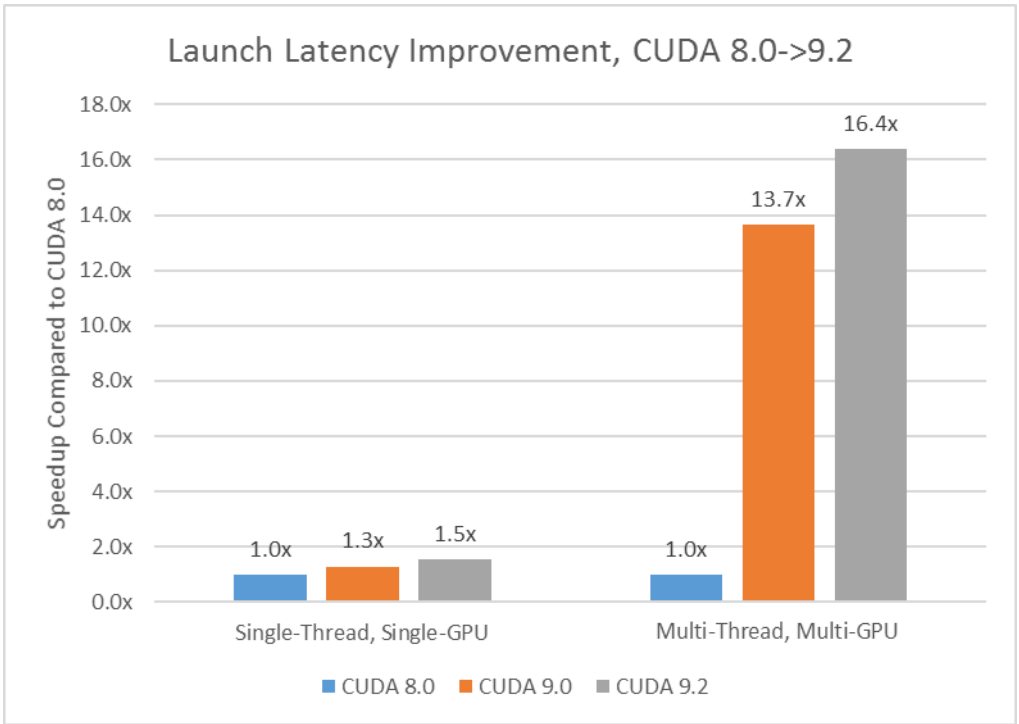
Tools Pod at Nvidia Booth on Showfloor

LAUNCH LATENCY IMPROVEMENTS

Multi-GPU Launches & Kernels With Many Arguments: Now Much Faster

Lower overhead for short kernels

- Significant factor for deep learning inference workloads
- Significant factor for small computational workloads (e.g. small FFT, small vector ops)



VOLTA NANOSLEEP TIMER

For Polling & Synchronization Operations

New *nanosleep()* instruction

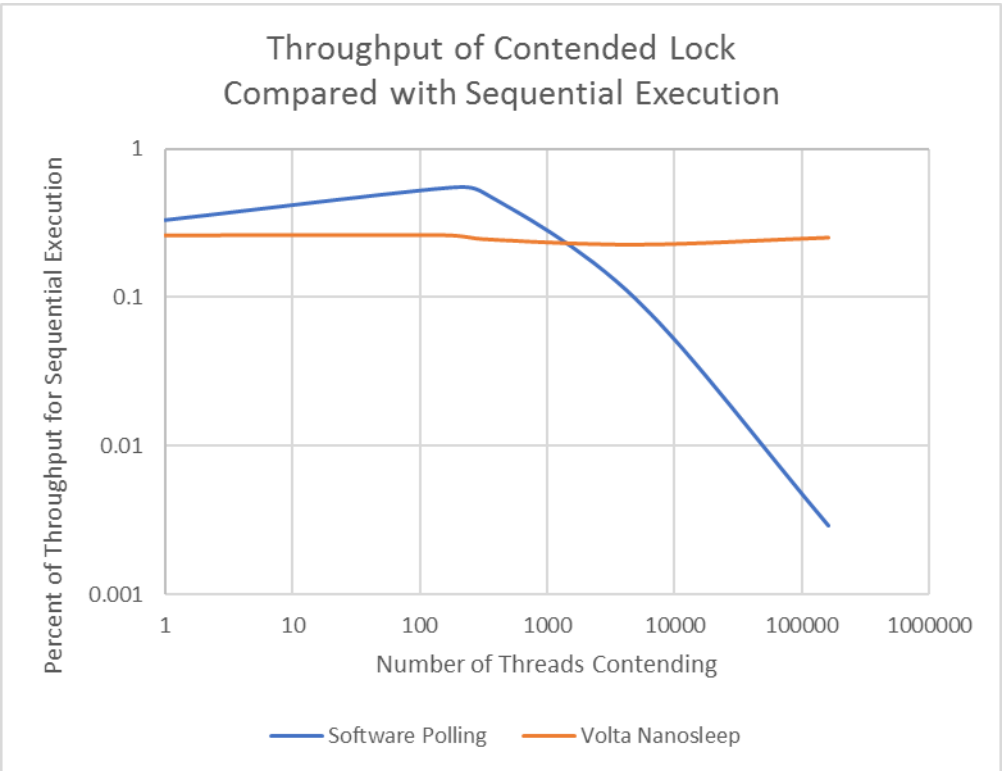
```
__device__ void __nanosleep(unsigned int ns);
```

Sleeps a thread for an amount of time

Sleeping thread yields execution to other active threads

Integrated into hardware thread scheduler

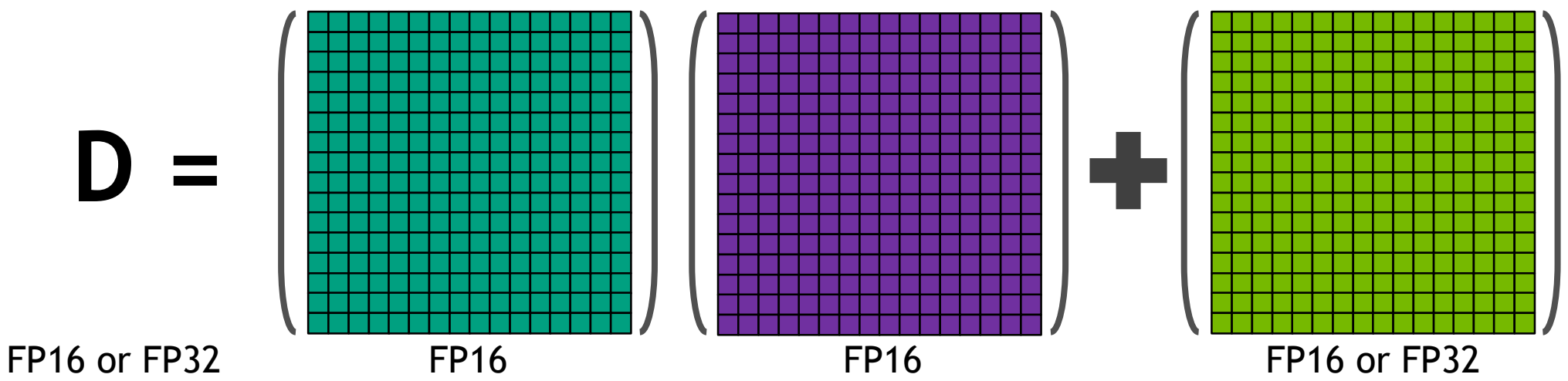
Ideal for timed-backoff polling



CUDA TENSOR CORE PROGRAMMING

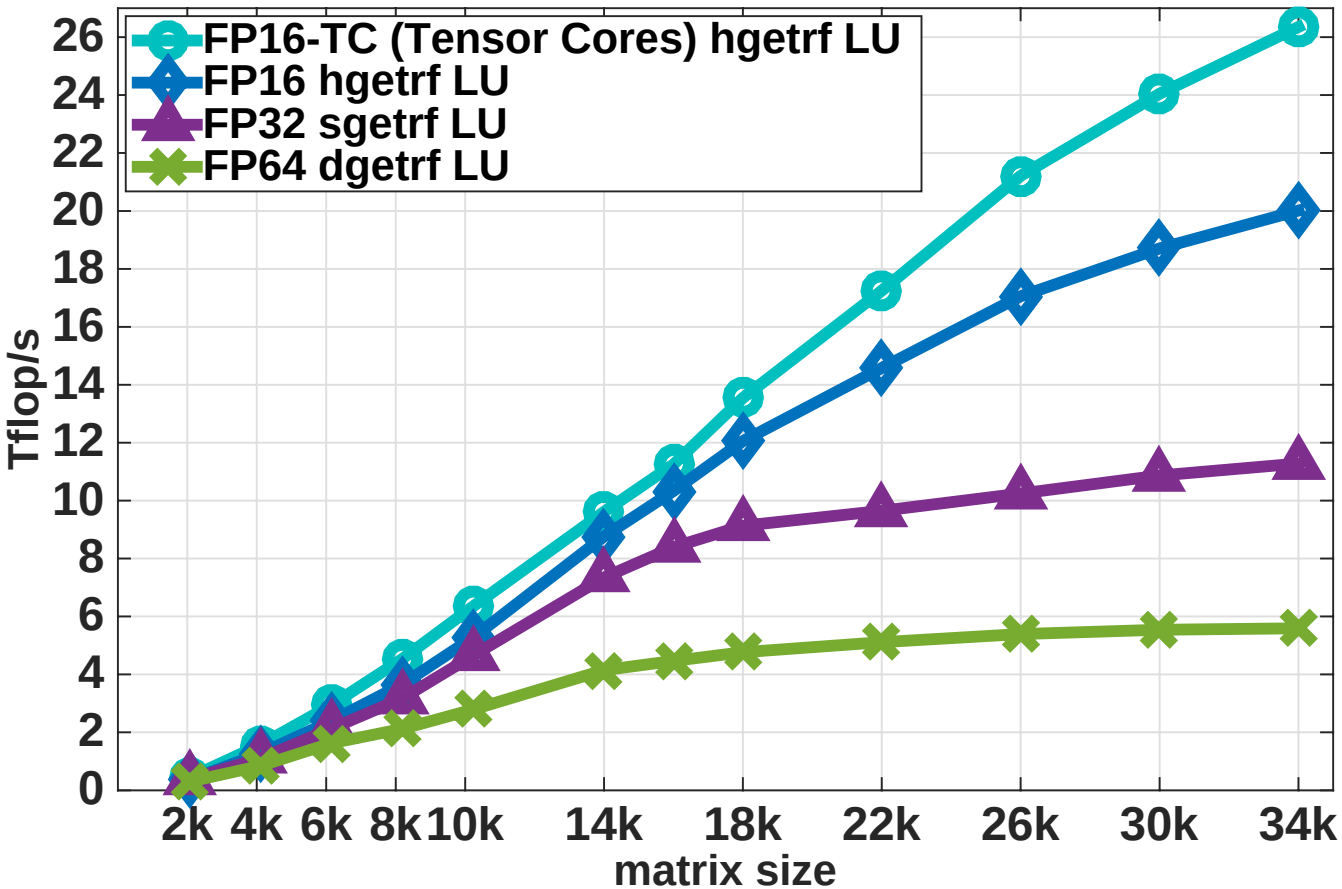
16x16x16 Warp Matrix Multiply and Accumulate (WMMA)

```
wmma::mma_sync(Dmat, Amat, Bmat, Cmat);
```



$$D = AB + C$$

LINEAR ALGEBRA + TENSOR CORES



Double Precision LU Decomposition

- Compute initial solution in FP16
- Iteratively refine to FP64

Achieved FP64 Tflops: 26

Device FP64 Tflops: 7.5

Data courtesy of: Azzam Haidar, Stan. Tomov & Jack Dongarra, Innovative Computing Laboratory, University of Tennessee

"*Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers*", A. Haidar, P. Wu, S. Tomov, J. Dongarra, SC'17

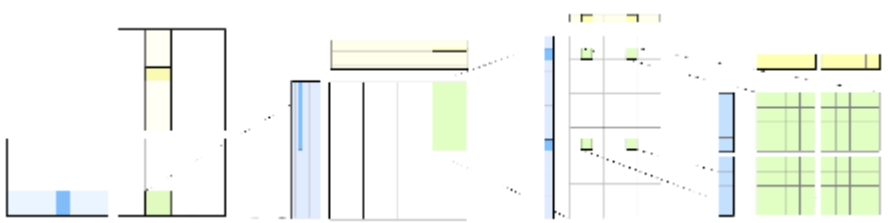
GTC 2018 Poster P8237: *Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solves*

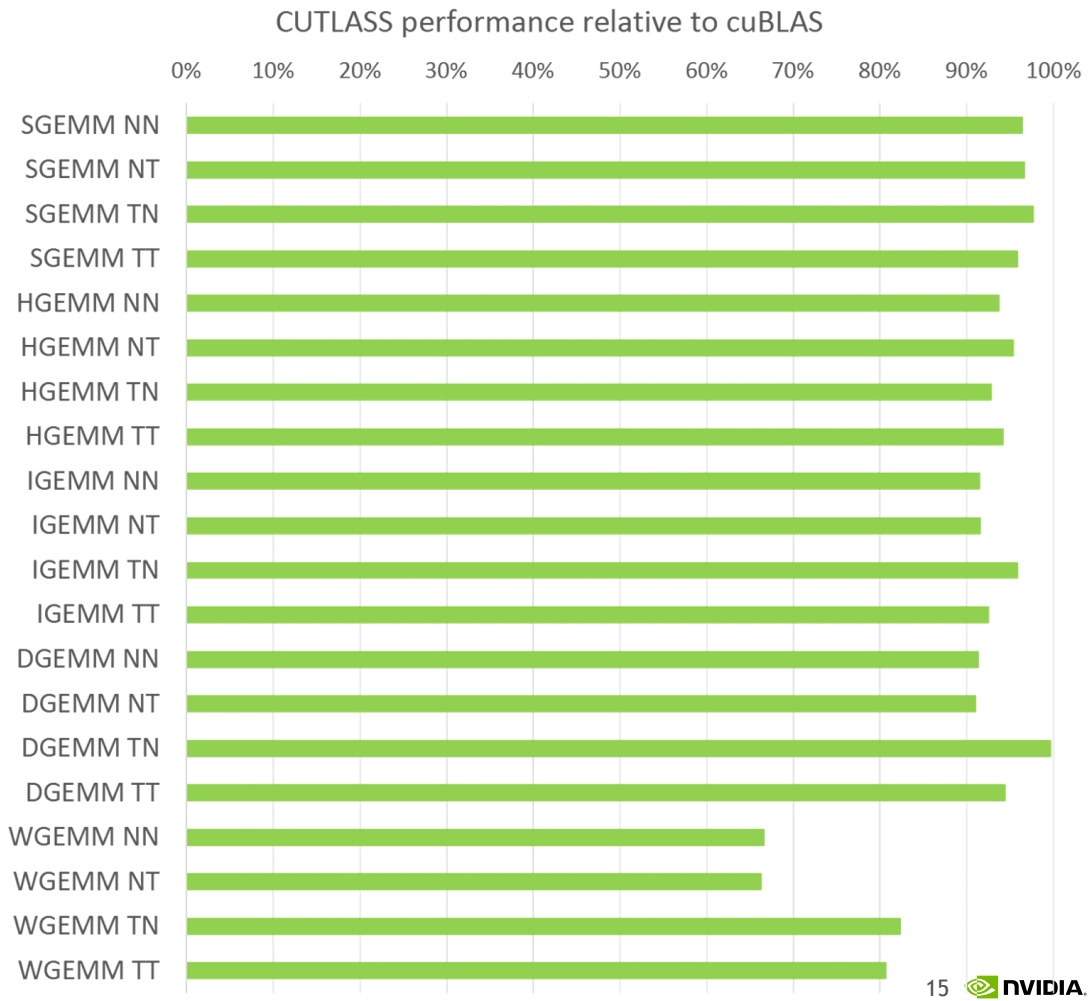
CUTLASS

Template library for linear algebra operations in CUDA C++

>90% CUBLAS performance

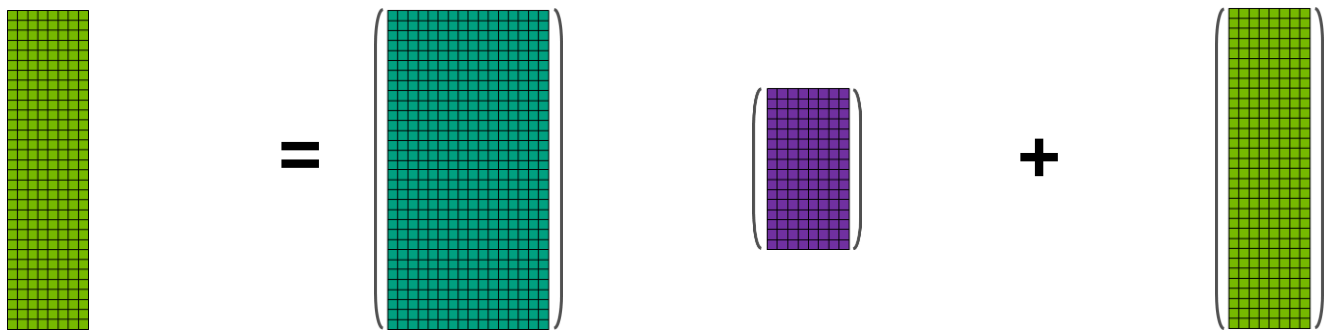
Open Source (3-clause BSD License)
https://github.com/NVIDIA/cutlass



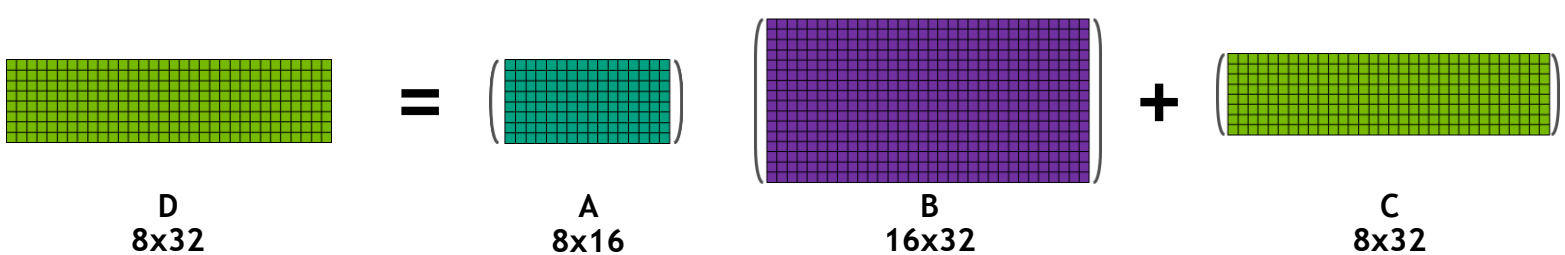


NEW WMMA MATRIX SIZES

WMMA 32x8x16



WMMA 8x32x16



MORE INFORMATION: TENSOR CORES

(S8478) *New Frontiers for Dense Linear Solvers: Towards Extreme Performance and Energy Efficiency*, Wednesday, 11:00 AM, Room 212B

(S8854) *CUTLASS: Software Primitives for Dense Linear Algebra at All Levels and Scales within CUDA*, Thursday, 9:00 AM, Room 220C

OpenACC DIRECTIVES

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          c[i] = a[i] + b[i];
          ...
      }
  }
  ...
}
```

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

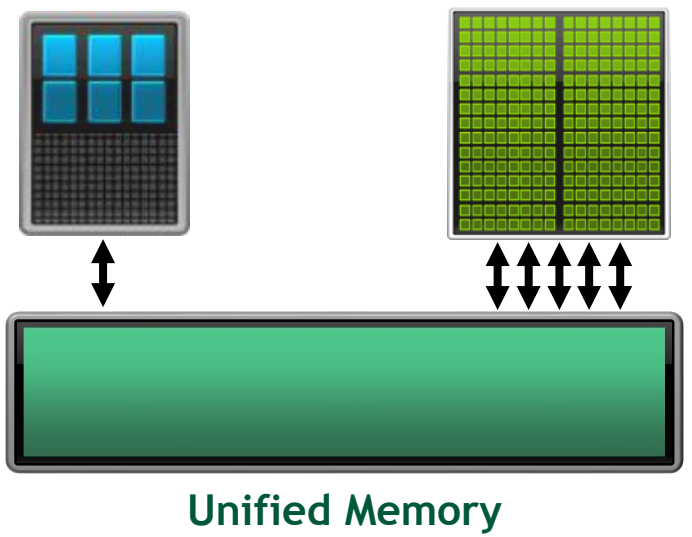
OpenACC
Directives for Accelerators

PGI OpenACC AND UNIFIED MEMORY

Compiling with the -ta=tesla:managed option

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          c[i] = a[i] + b[i];
          ...
      }
  }
  ...
}
```

GPU Developer View With CUDA Unified Memory



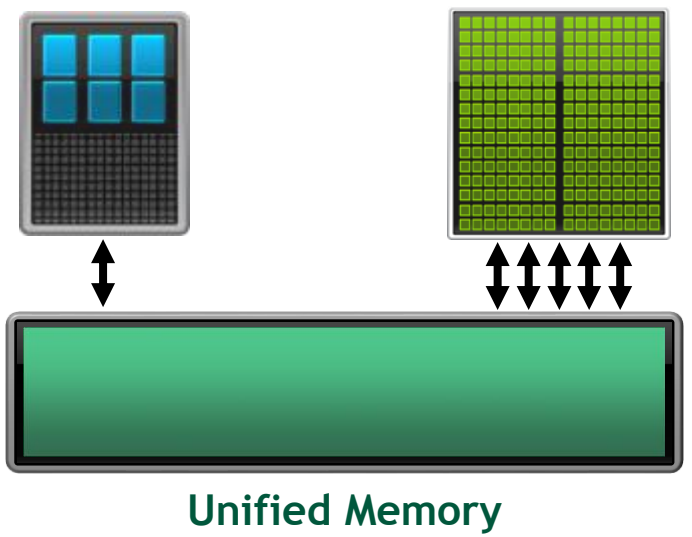
C `malloc`, C++ `new`, Fortran `allocate` all mapped to CUDA Unified Memory

PGI OpenACC AND UNIFIED MEMORY

Compiling with the -ta=tesla:managed option

```
...
#pragma acc parallel
{
#pragma acc loop gang vector
    for (i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
        ...
    }
}
...
```

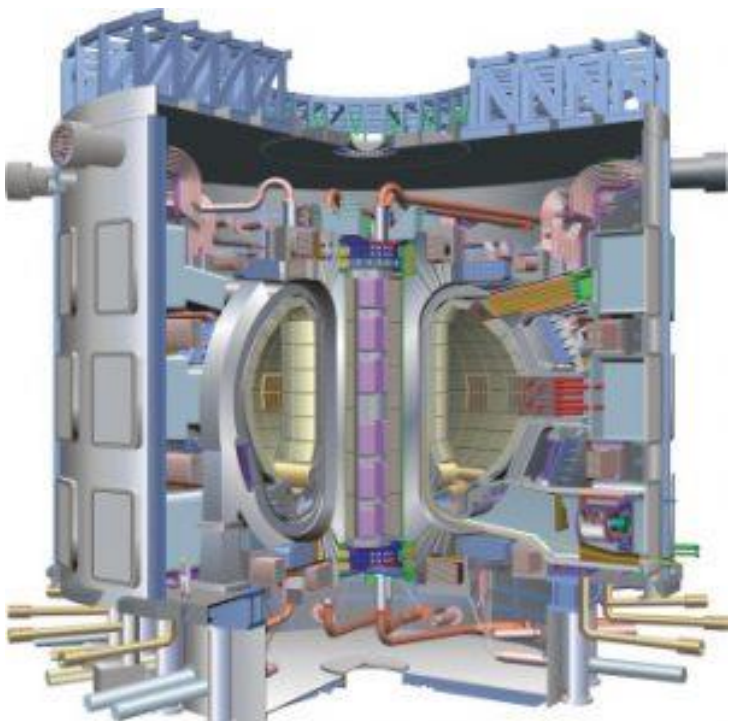
GPU Developer View With CUDA Unified Memory



C `malloc`, C++ `new`, Fortran `allocate` all mapped to CUDA Unified Memory

GYROKINETIC TOROIDAL CODE

Being ported for runs on the ORNL Summit supercomputer



http://phoenix.ps.uci.edu/gtc_group

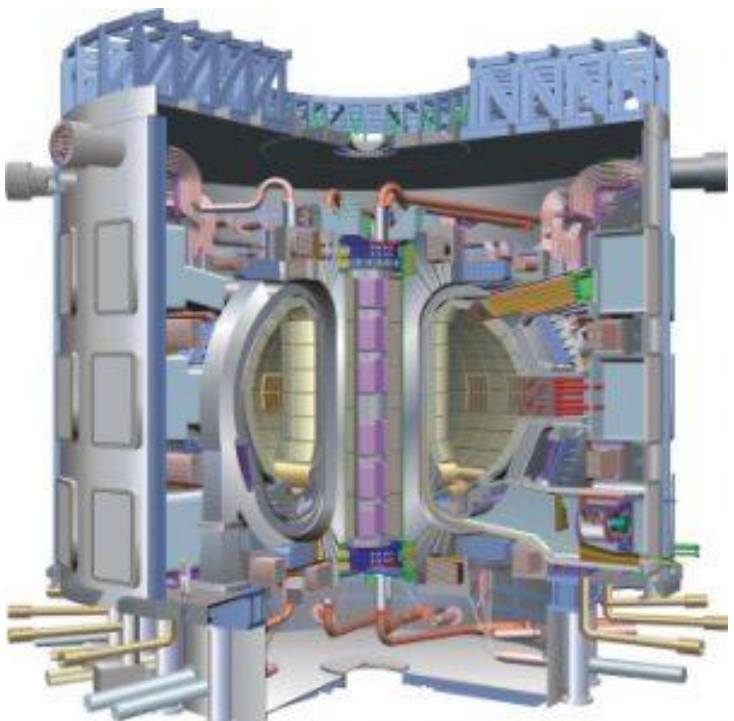
The Gyrokinetic Toroidal Code (GTC)

- Plasma turbulence simulation
- Supporting the ITER fusion experiment
- Massively parallel, particle-in-cell production code

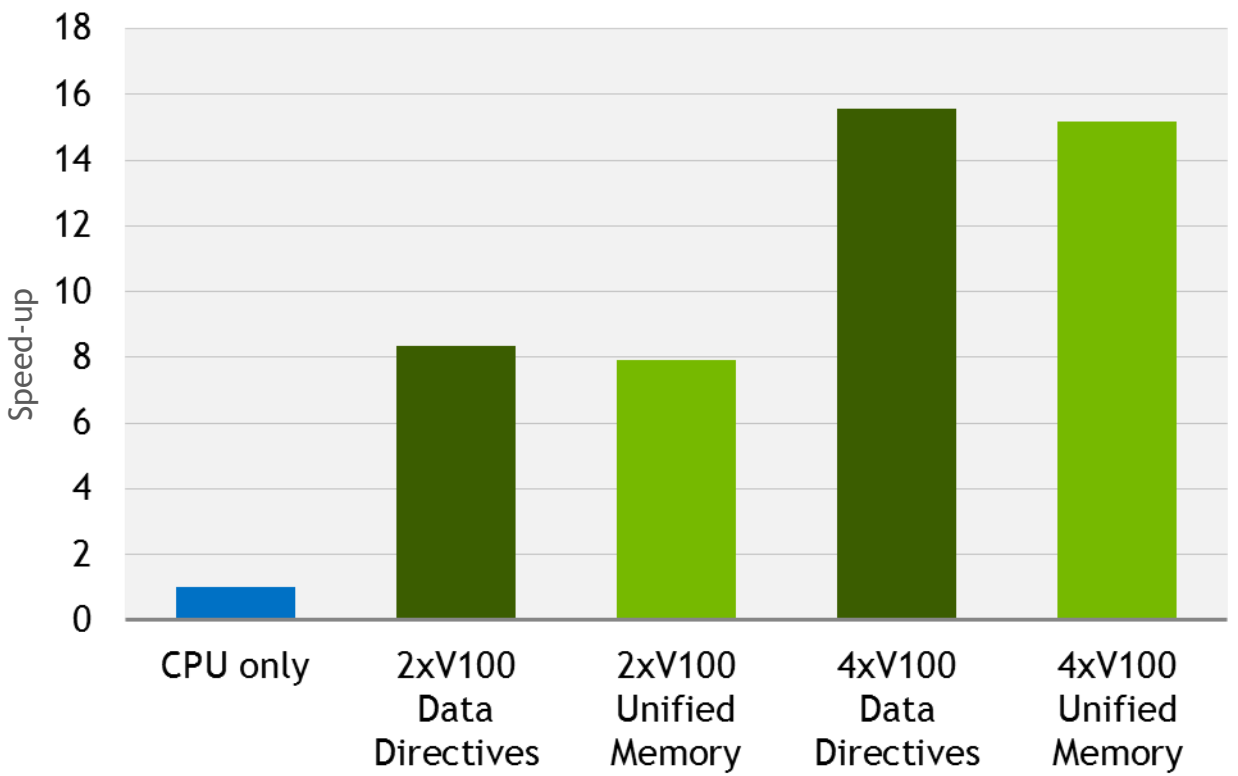
CPU: Haswell E5-2698 v3 @ 2.30GHz, dual socket 16-core

GYROKINETIC TOROIDAL CODE

Particle-In-Cell production code



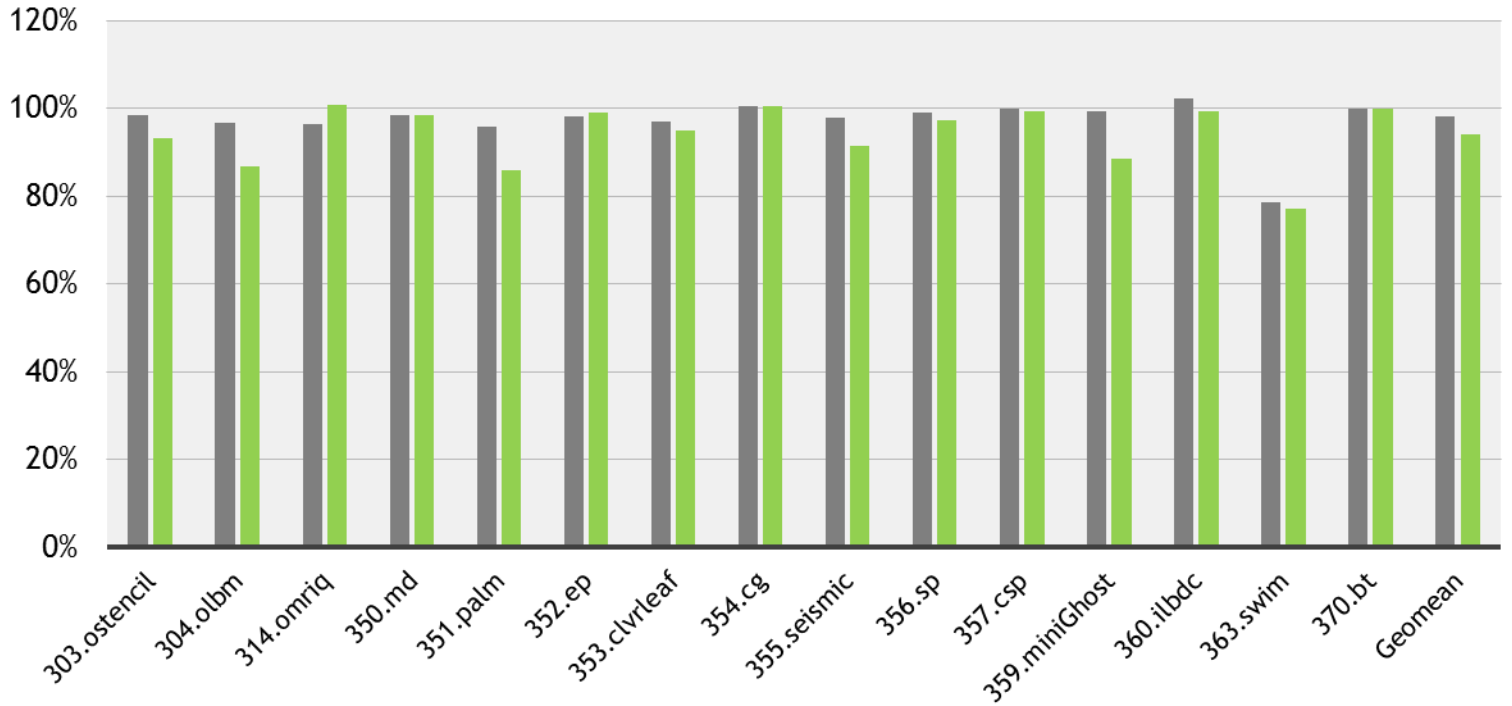
http://phoenix.ps.uci.edu/gtc_group



CPU: Haswell E5-2698 v3 @ 2.30GHz, dual socket 16-core

SPEC ACCEL 1.2 OpenACC Benchmarks

OpenACC with Unified Memory vs OpenACC Data Directives
Bigger is Better



100% = Pure Directive-based Data Movement

P8+P100 NVLink
HSW+P100 PCIe

Fortran allocate/deallocate,
C malloc/calloc/free calls,
C++ new/delete are all intercepted & mapped to CUDA Unified Memory

PGI 17.7 Compilers OpenACC SPEC ACCEL™ 1.2 performance measured August, 2017
SPEC® and the benchmark name SPEC ACCEL™ are registered trademarks of the Standard Performance Evaluation Corporation.

PGI COMPILERS FOR EVERYONE

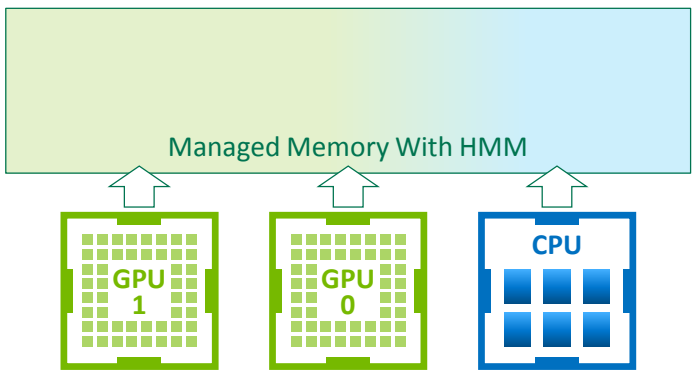
The PGI Community Edition

	FREE PGI Community Edition	PGI Professional Edition	PGI Enterprise Edition
PROGRAMMING MODELS OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools	✓	✓	✓
PLATFORMS X86, OpenPOWER, NVIDIA GPU	✓	✓	✓
UPDATES	1-2 times a year	6-9 times a year	6-9 times a year
SUPPORT	User Forums	PGI Support	PGI Premier Services
LICENSE	Annual	Perpetual	Volume/Site

BEYOND

HETEROGENEOUS MEMORY ON x86-LINUX

Feature Parity With POWER9 + ATS



ALLOCATION

Automatic access to all system memory: malloc, stack, file system

ACCESS

All data accessible concurrently from any processor, anytime

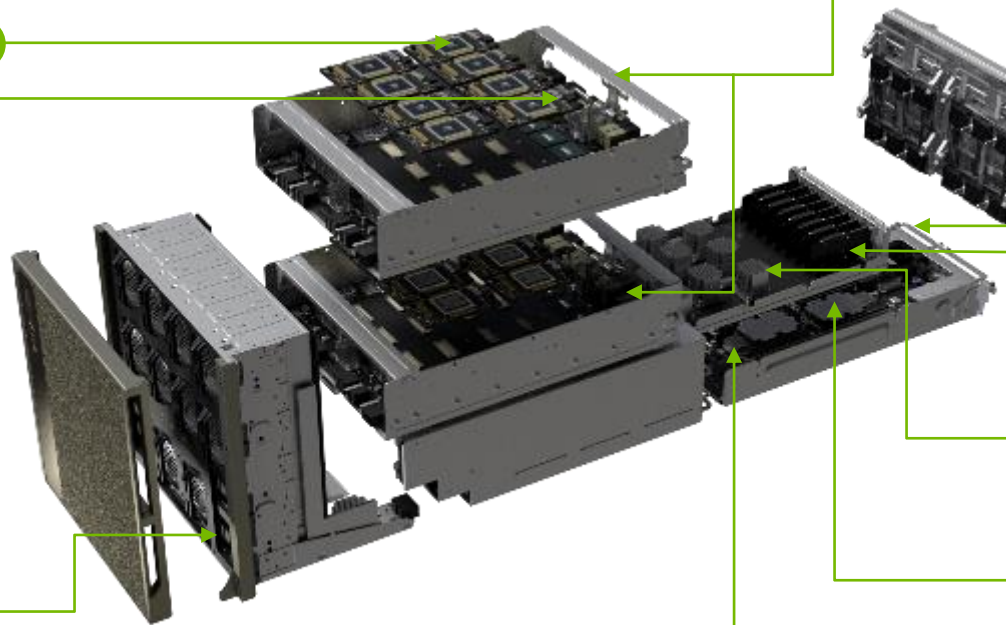
Concurrent atomic operations permitted, resolved via page fault

MORE INFORMATION: UNIFIED MEMORY

(S8430) *Everything You Need to Know About Unified Memory*, Tuesday 4:30pm, Room 211A

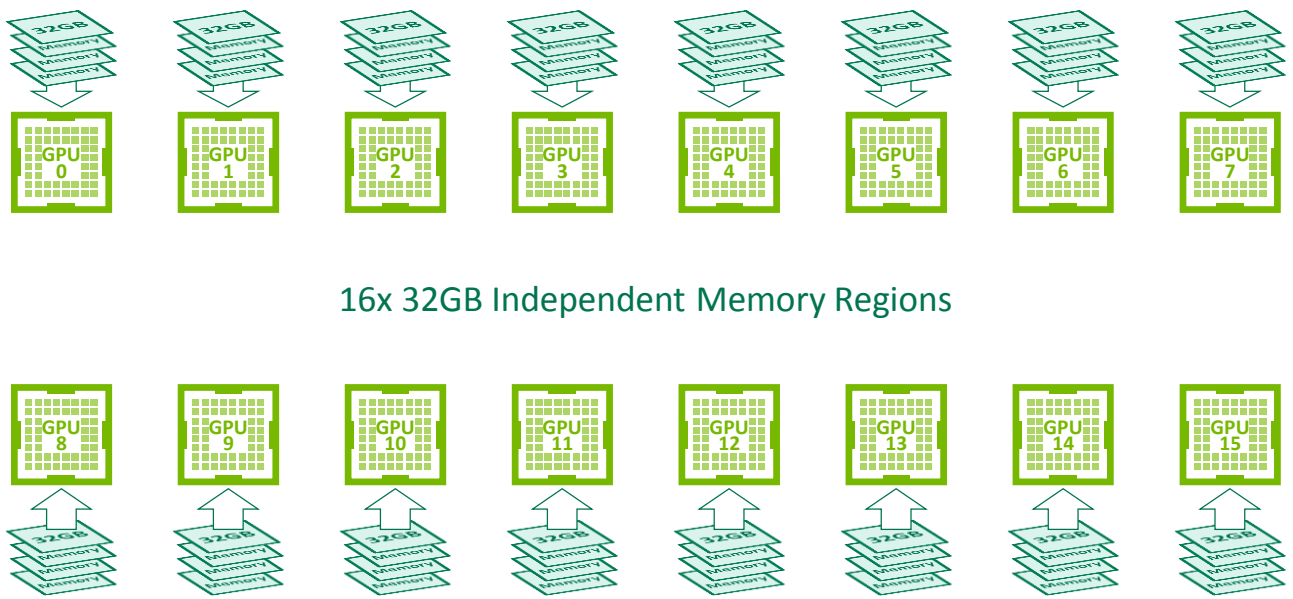
DESIGNED TO TRAIN THE PREVIOUSLY IMPOSSIBLE

Introducing NVIDIA DGX-2



1. **NVIDIA Tesla V100 32GB**
2. **Two GPU Boards**
   8 V100 32GB GPUs per board
   6 NVSwitches per board
   512GB Total HBM2 Memory
   interconnected by
   Plane Card
3. **Twelve NVSwitches**
   2.4 TB/sec bi-section bandwidth
4. **Eight EDR Infiniband/100 GigE**
   1600 Gb/sec Total
   Bi-directional Bandwidth
5. **PCIe Switch Complex**
6. **Two Intel Xeon Platinum CPUs**
7. **1.5 TB System Memory**
8. **30 TB NVME SSDs**
   Internal Storage
9. **Dual 10/25 GigE**

16 GPUs WITH 32GB MEMORY EACH

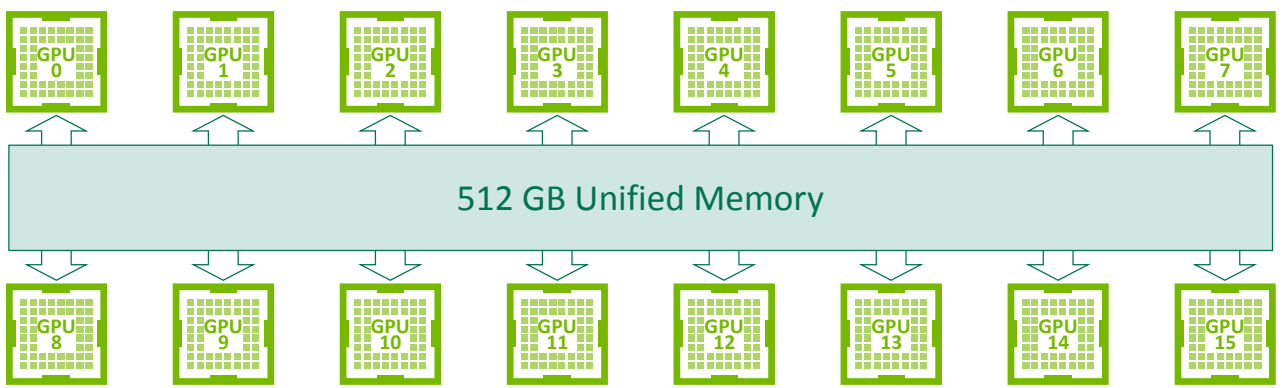


NVSWITCH PROVIDES

All-to-all high-bandwidth peer mapping between GPUs

Full inter-GPU memory interconnect (incl. Atomics)

UNIFIED MEMORY + DGX-2



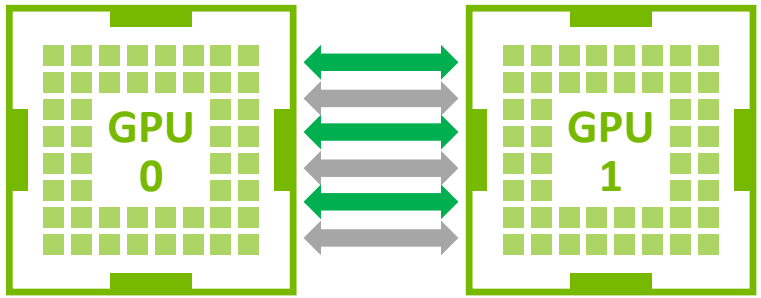
UNIFIED MEMORY PROVIDES

Single memory view shared by all GPUs

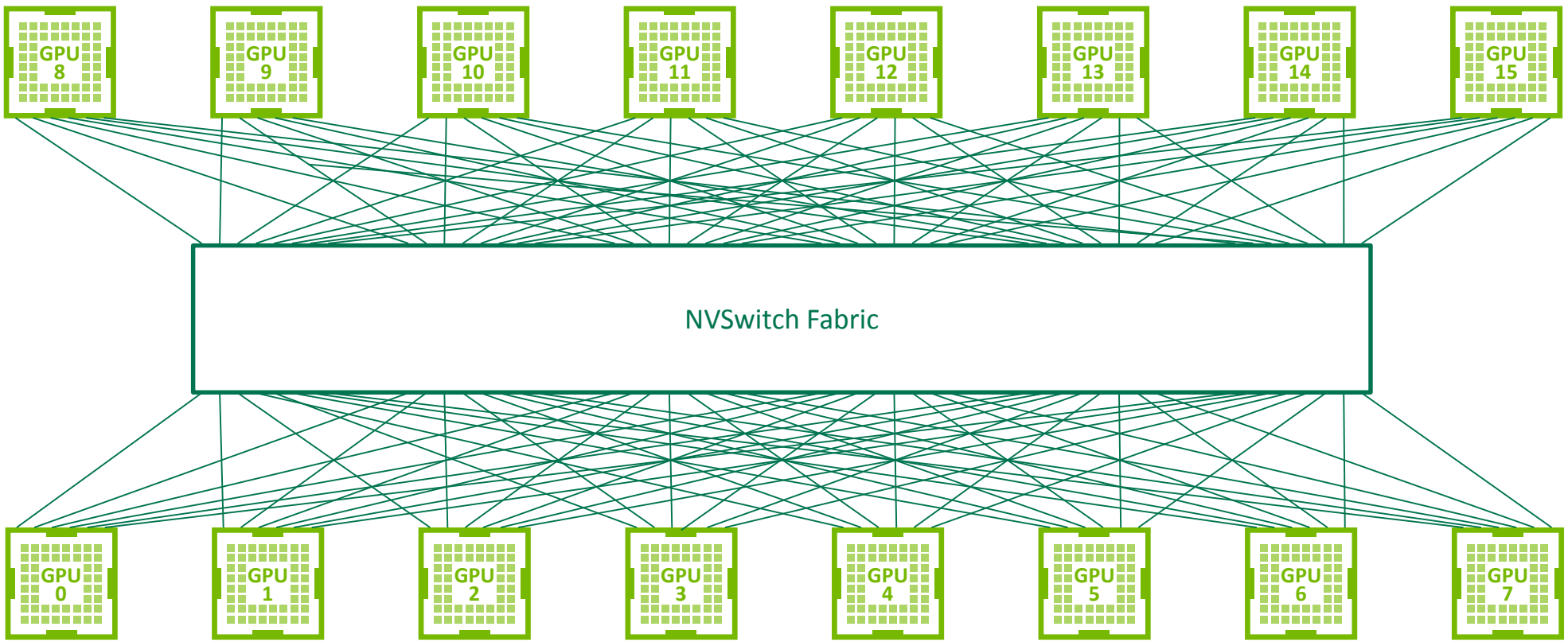
Automatic migration of data between GPUs

User control of data locality

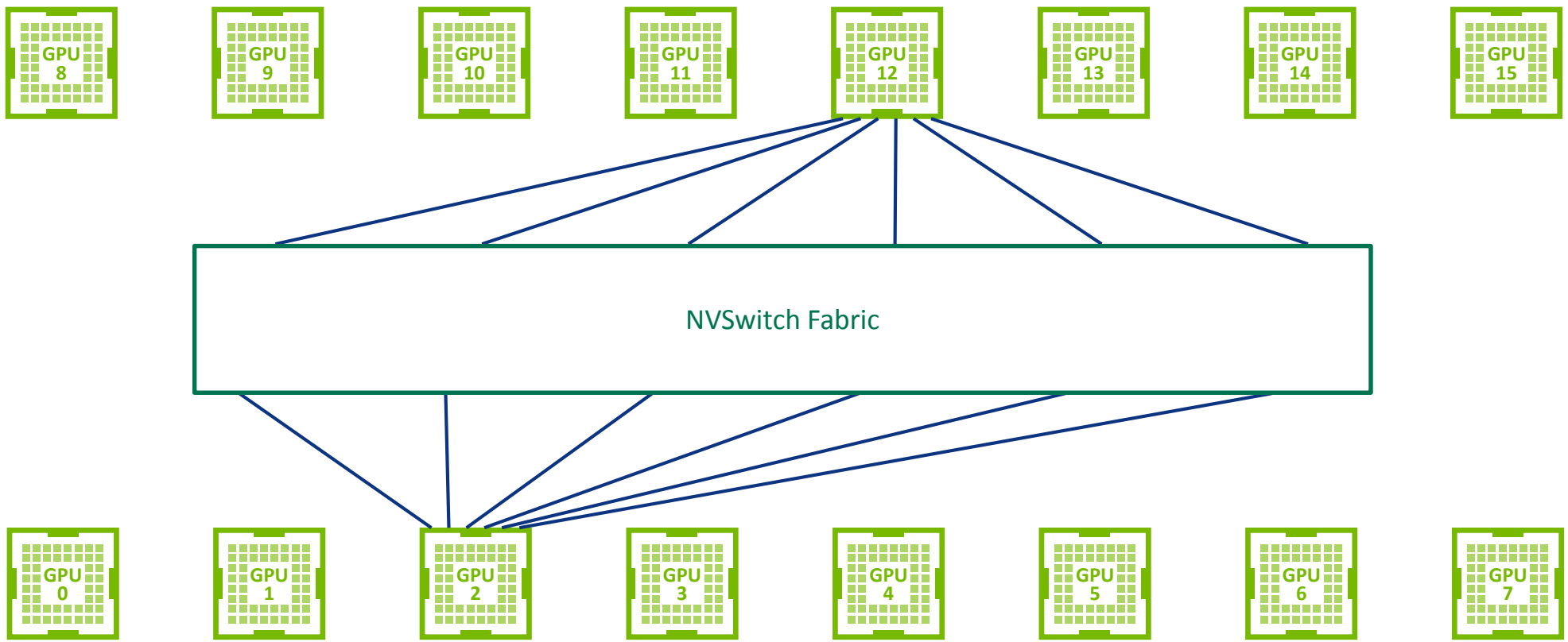
NVLINK: POINT-TO-POINT INTERCONNECT



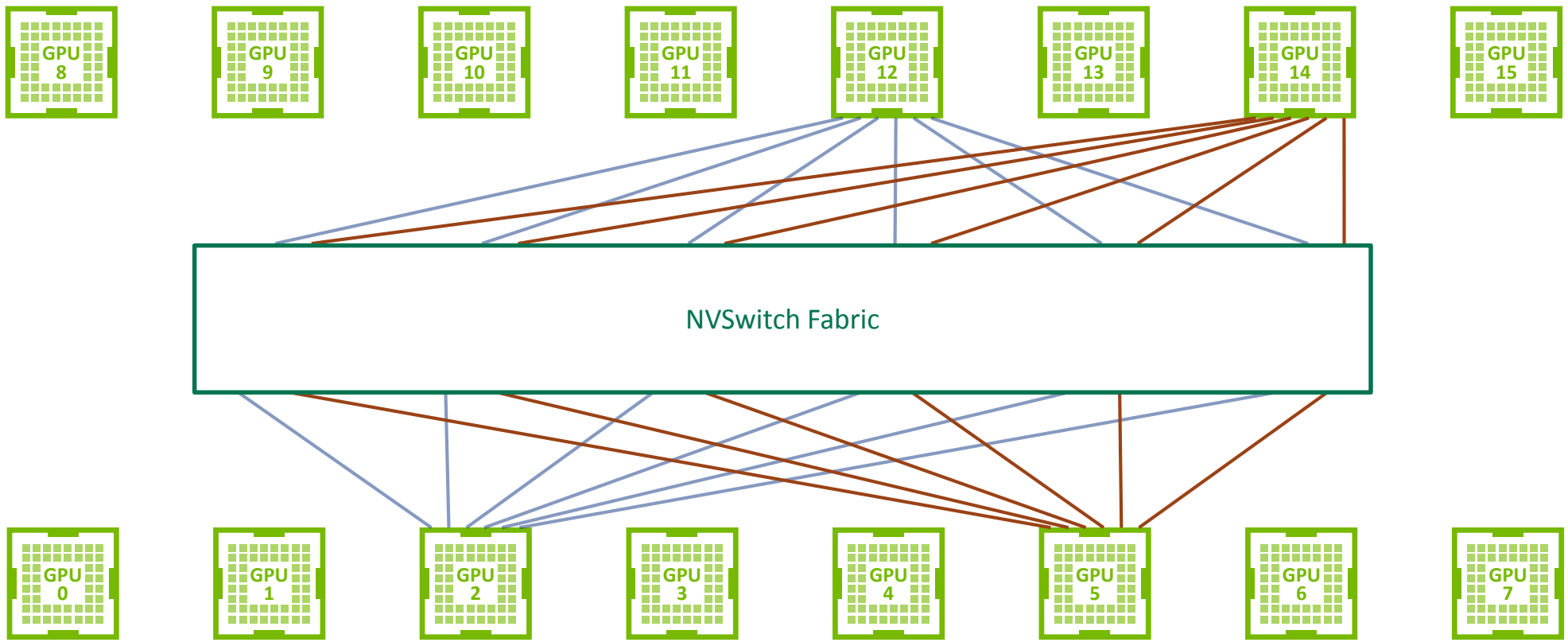
NVSWITCH: ALL-TO-ALL CONNECTIVITY



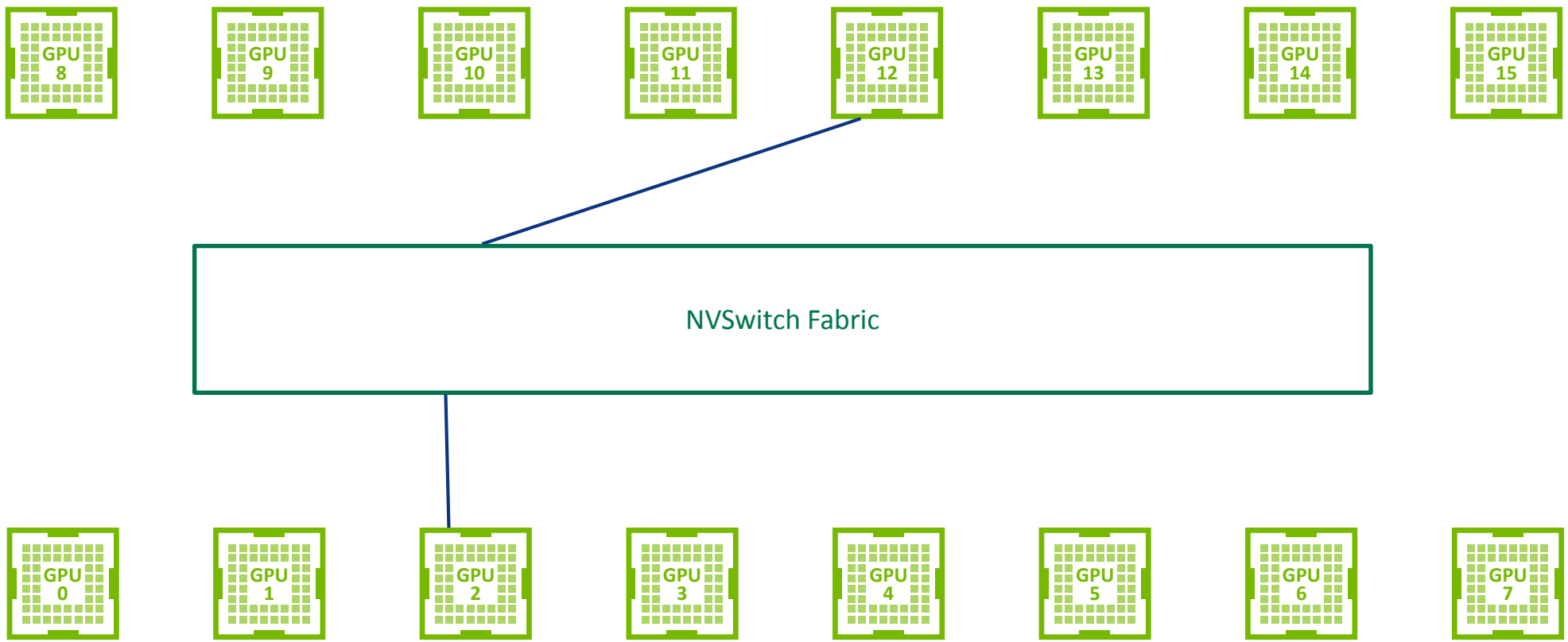
FULL 6-WAY POINT-TO-POINT



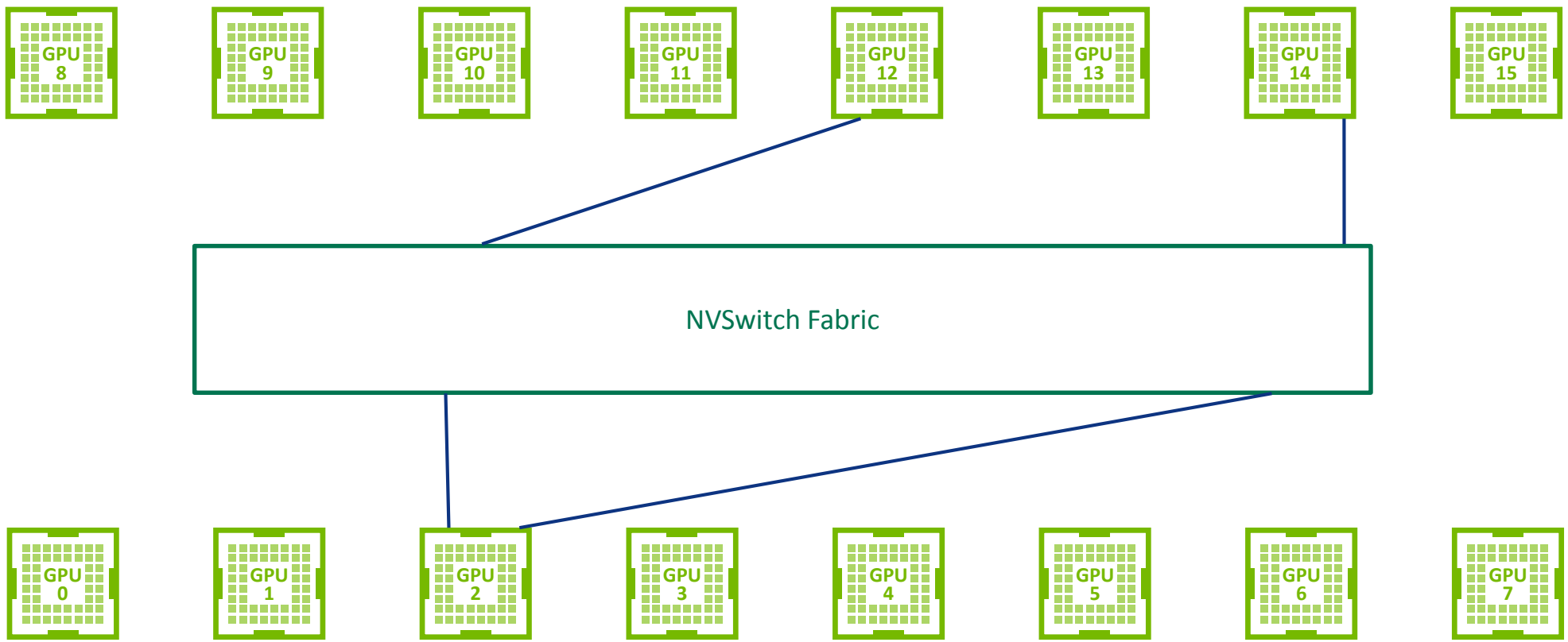
INDEPENDENT COMMUNICATION



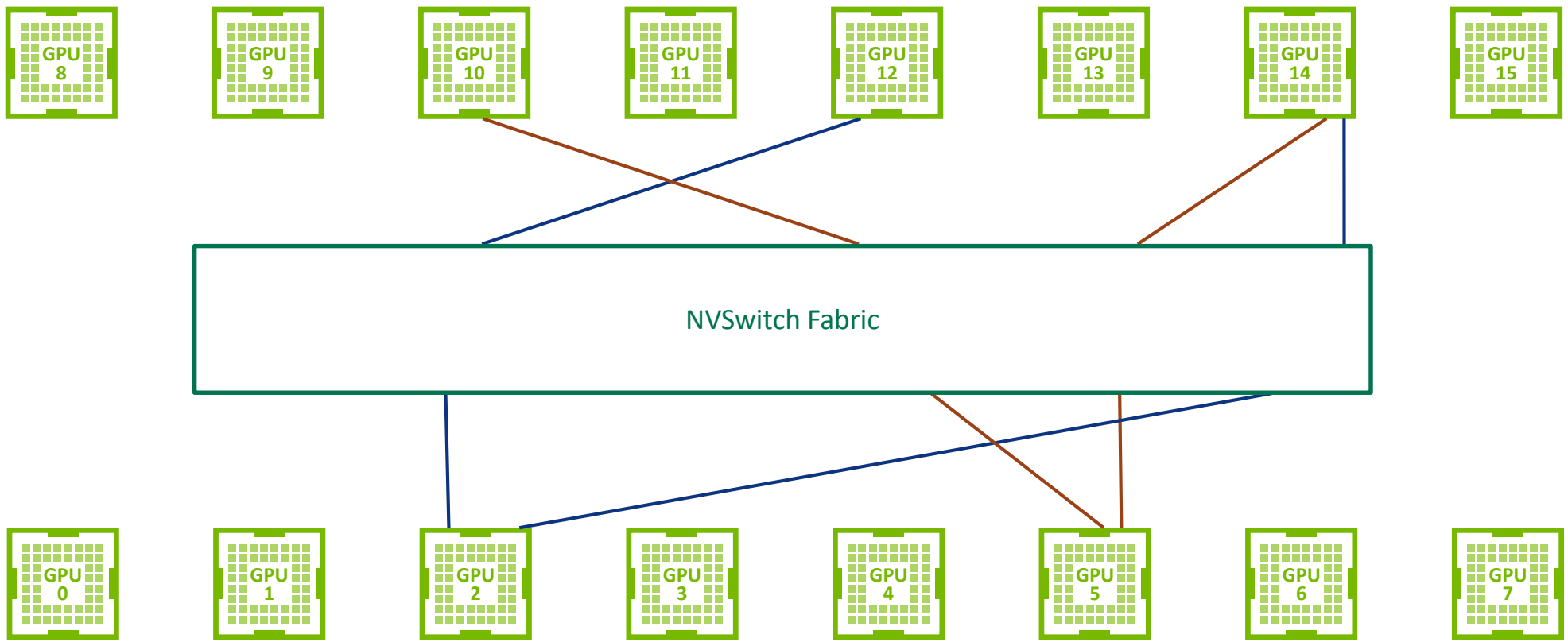
LOAD & STORE TO ANY GPU



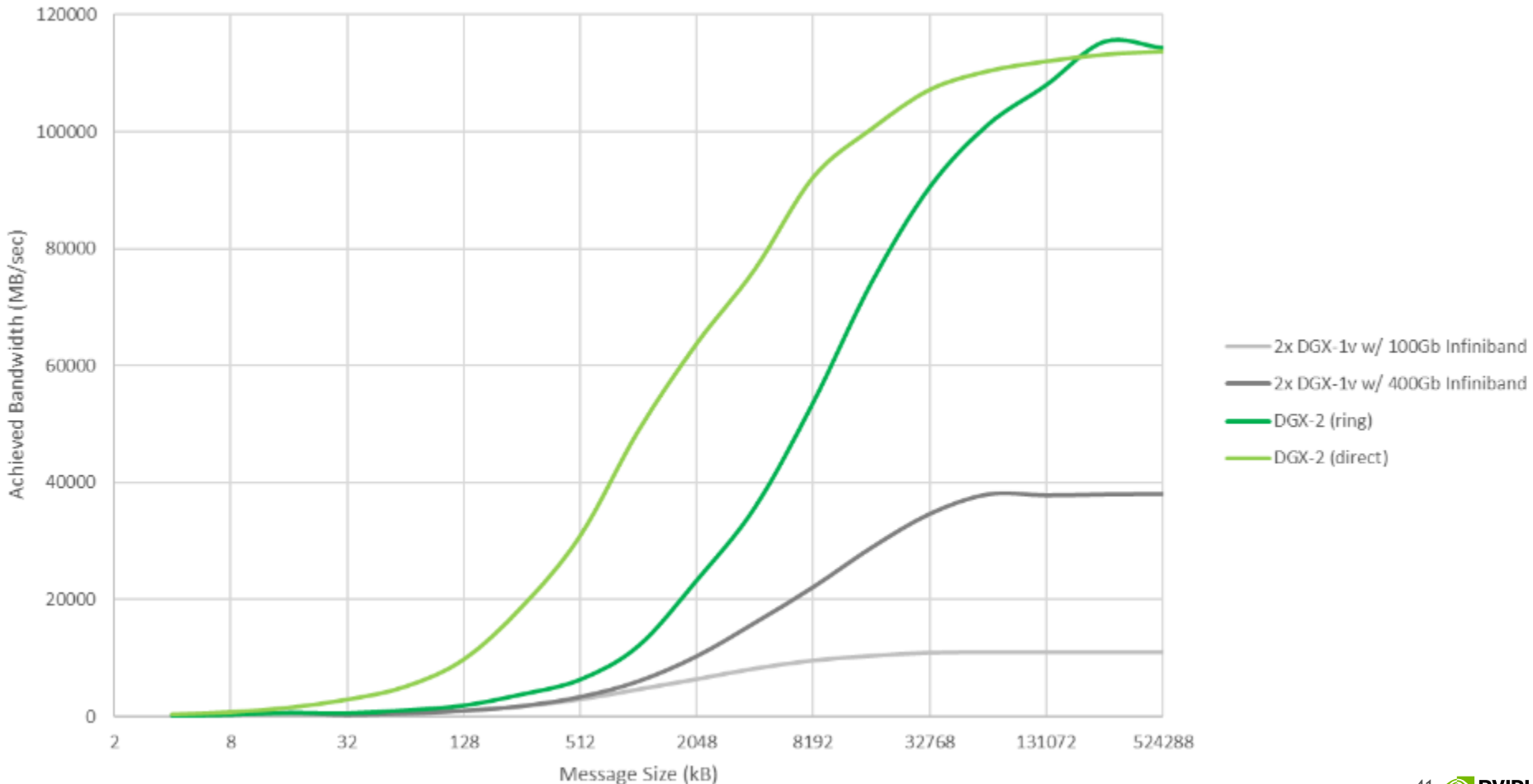
LOAD & STORE TO ANY GPU



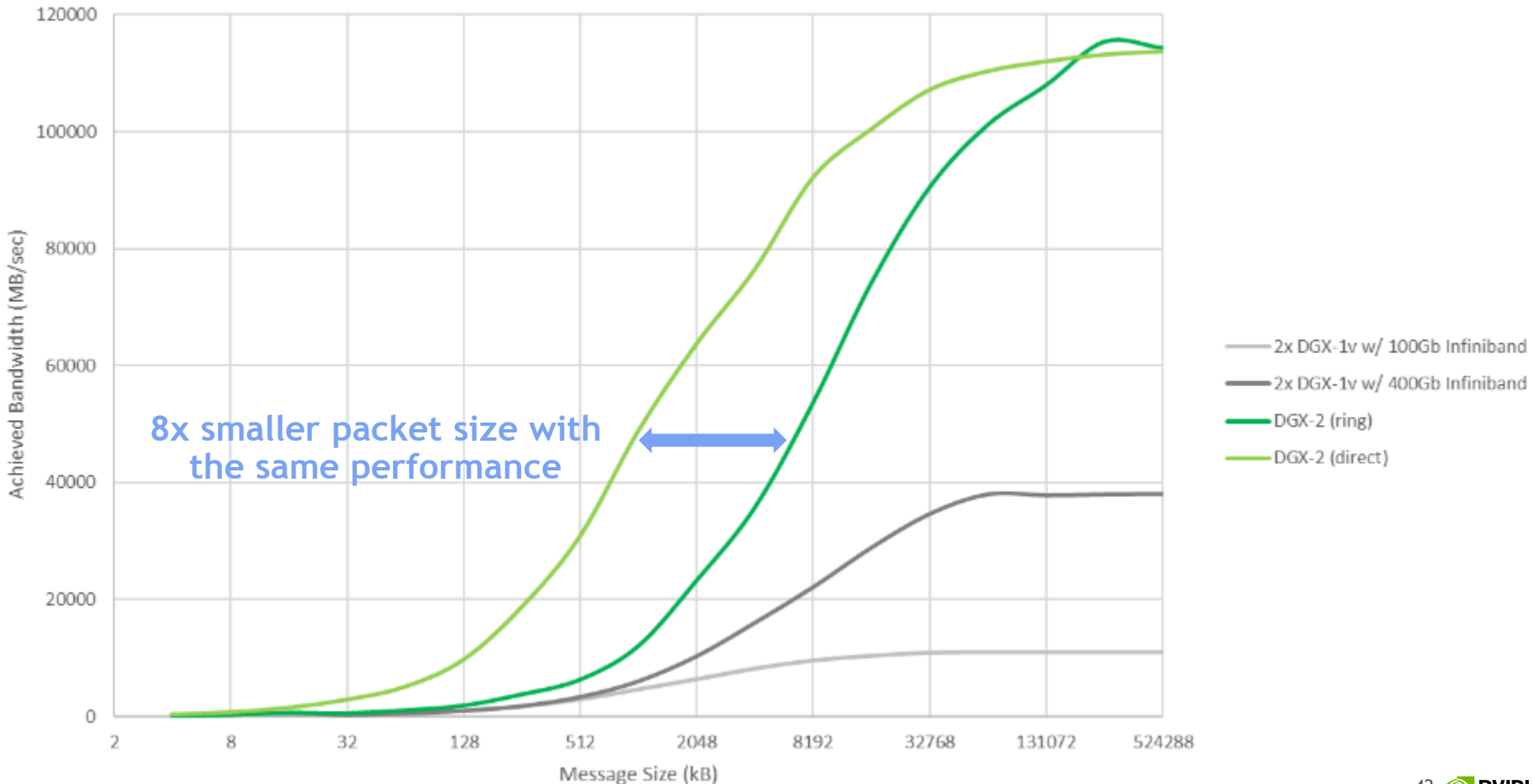
LOAD & STORE TO ANY GPU



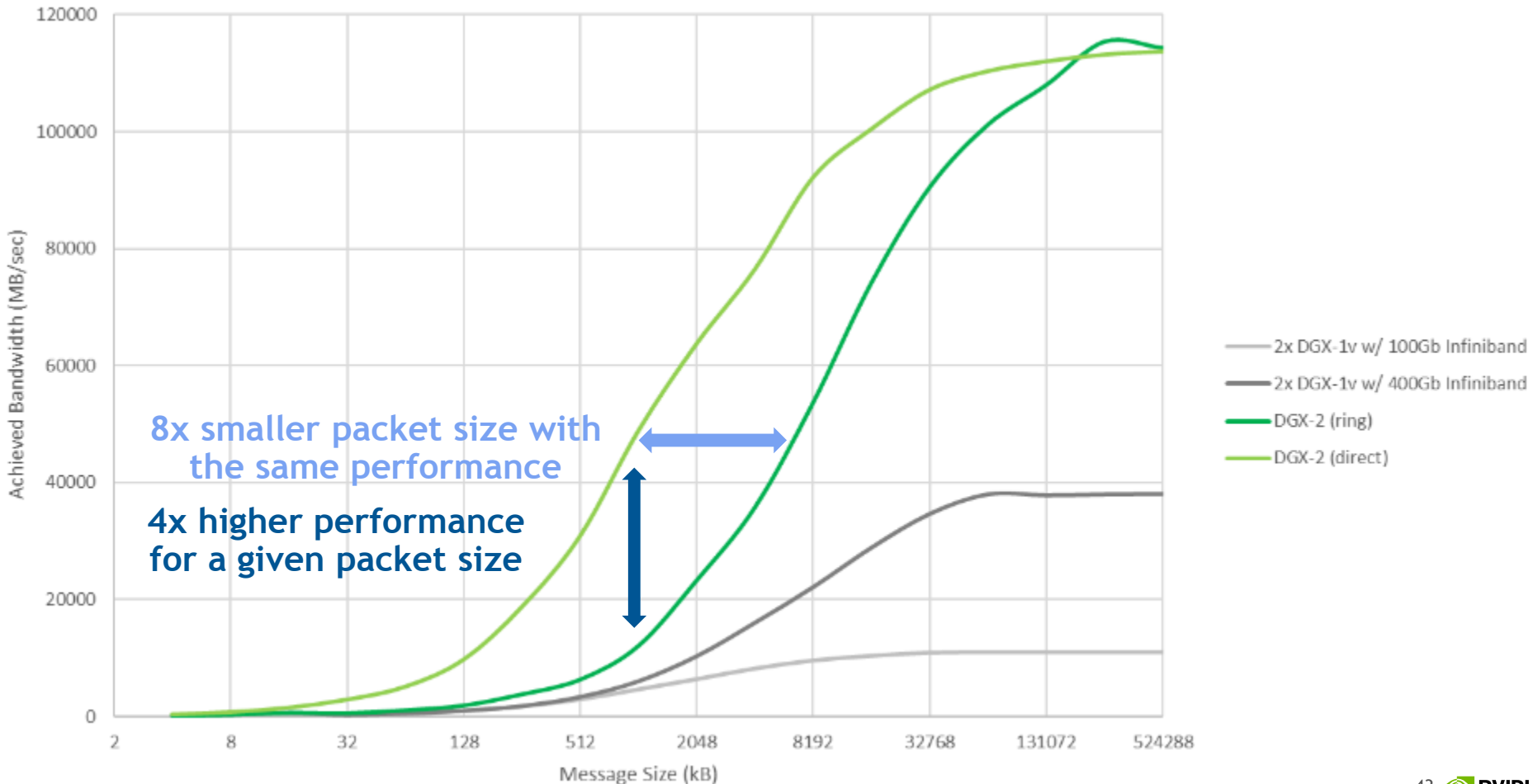
16-WAY ALL-REDUCE PERFORMANCE



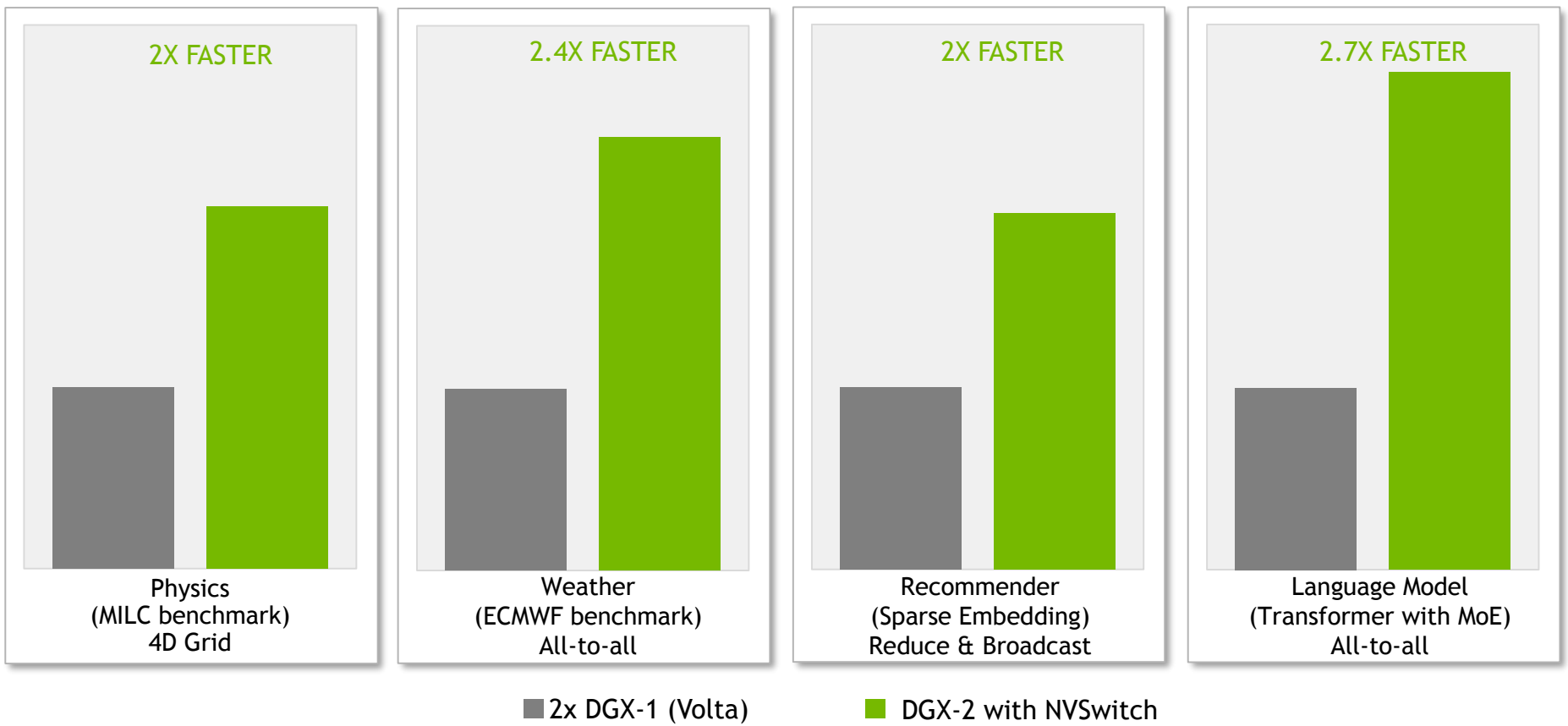
16-WAY ALL-REDUCE PERFORMANCE



16-WAY ALL-REDUCE PERFORMANCE



2X HIGHER PERFORMANCE WITH NVSWITCH



2 DGX-1V servers have dual socket Xeon E5 2698v4 Processor. 8 x V100 GPUs. Servers connected via 4X 100Gb IB ports | DGX-2 server has dual-socket Xeon Platinum 8168 Processor. 16 V100 GPUs

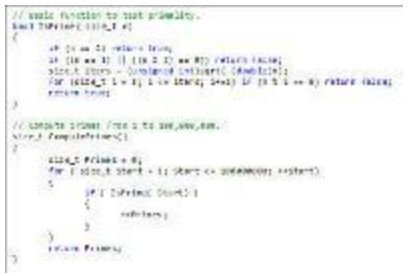
MORE INFORMATION: MULTI-GPU

(S8316) *Multi GPU Programming Models*, Tuesday 2pm, Room 211A

(S8670) *Multi-GPU Programming Techniques in CUDA*, Wednesday 2pm, Room 210B

ASYNCHRONOUS TASK GRAPHS

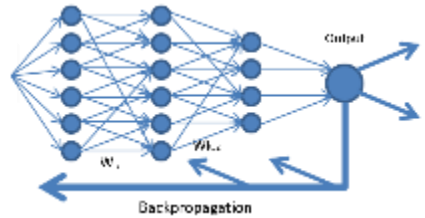
Increasingly Common Execution Paradigm



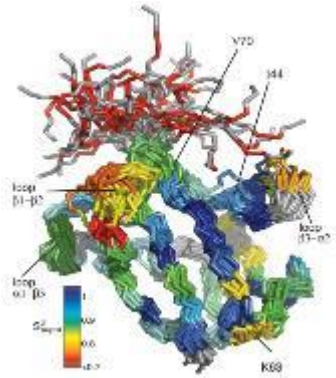
Loop & Function offload

DL Inference

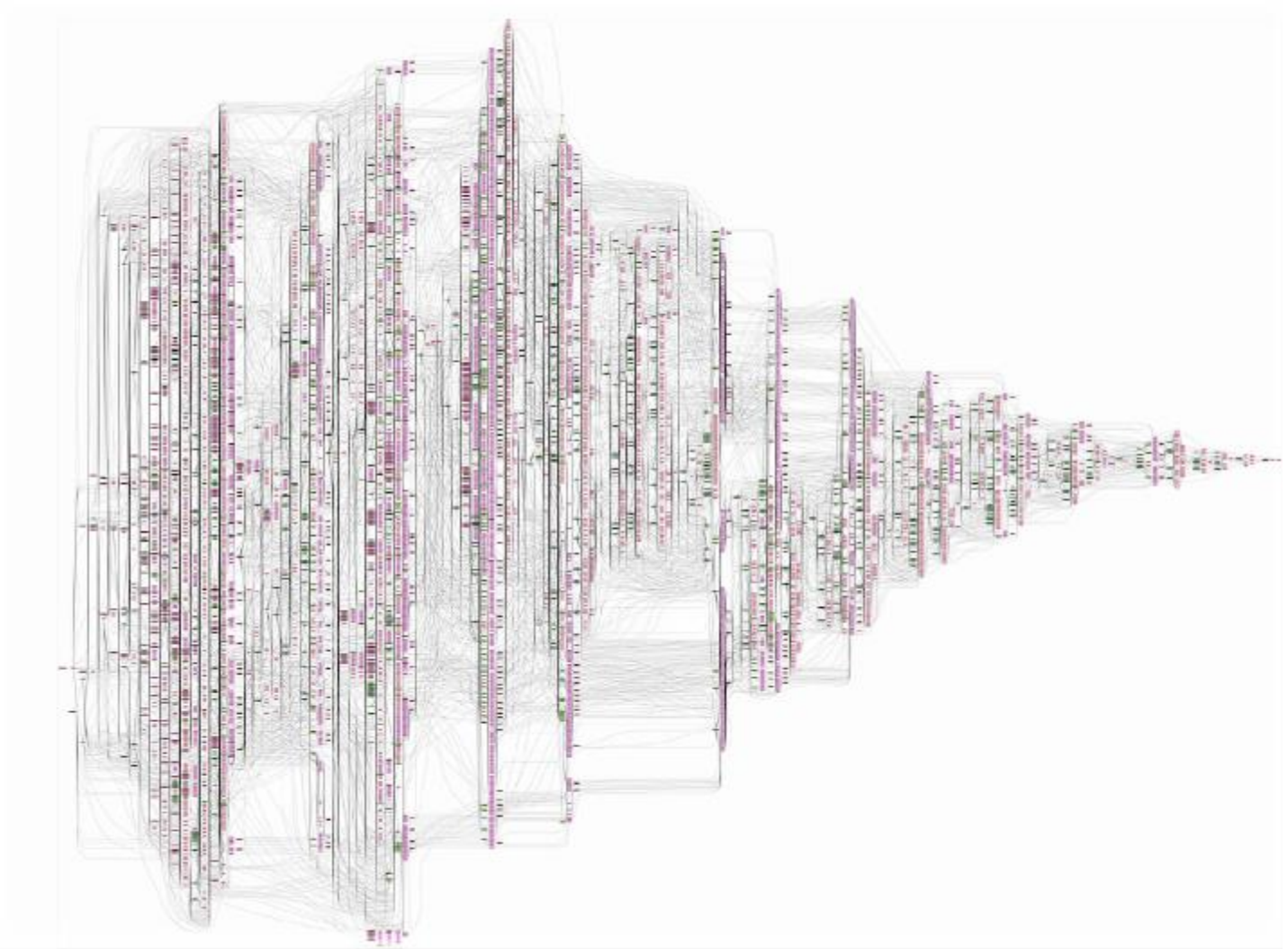
Linear Algebra



Deep Neural Network Training

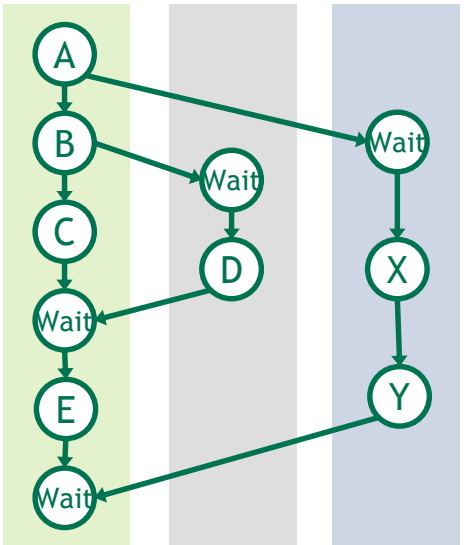


HPC Simulation



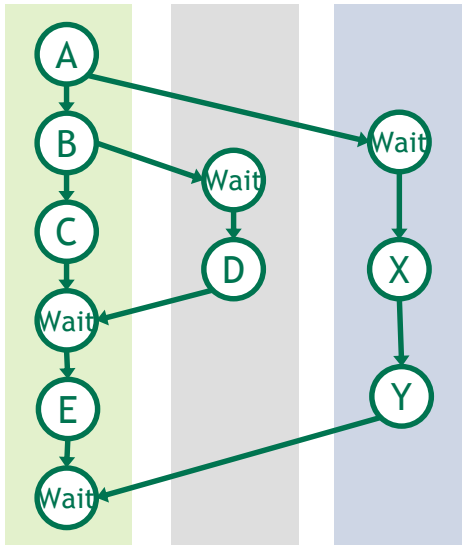
ALL CUDA WORK FORMS A GRAPH

CUDA Work in Streams



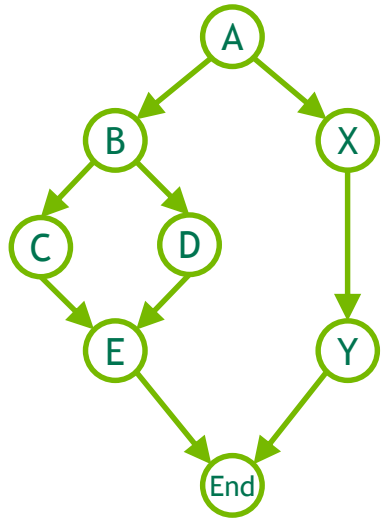
ALL CUDA WORK FORMS A GRAPH

CUDA Work in Streams



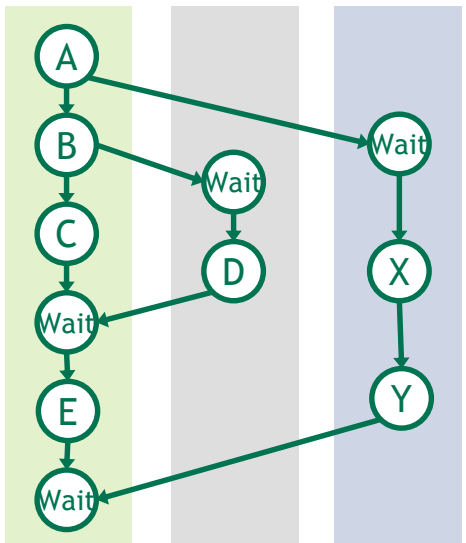
All CUDA streams can be mapped to a graph

Graph of Dependencies



GRAPHS CARRY RICHER INFORMATION

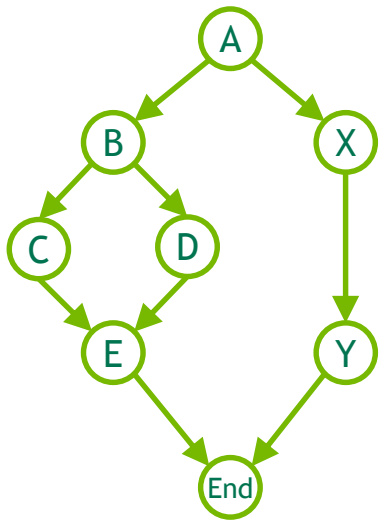
CUDA Work in Streams



Inline dependencies

Based on order that work is submitted

Graph of Dependencies



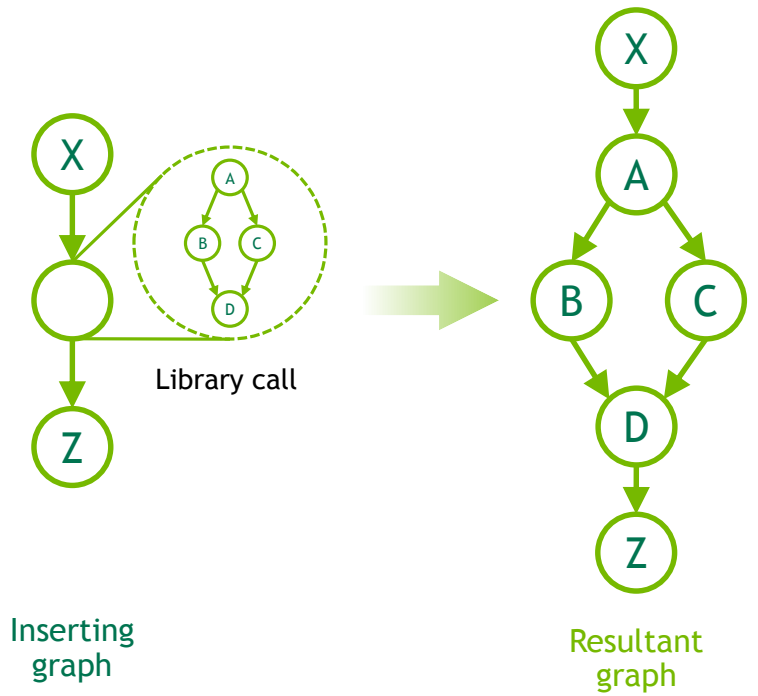
Explicit dependencies

Defined when graph is created

CAPTURE CUDA WORK INTO A GRAPH

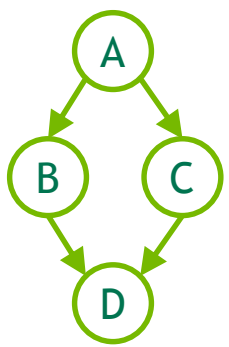
Apply Graph Optimizations to Existing CUDA Code

```
// Start by initating stream capture
cudaStreamBeginCapture(&stream);

// Captures my kernel launches and inside library calls
X<<< ..., stream >>>();
libraryCall(stream);        // Launches A, B, C, D
Z<<< ..., stream >>>();

// Now convert the stream to a graph
cudaStreamEndCapture(stream, &graph);
```



CREATE GRAPHS DIRECTLY

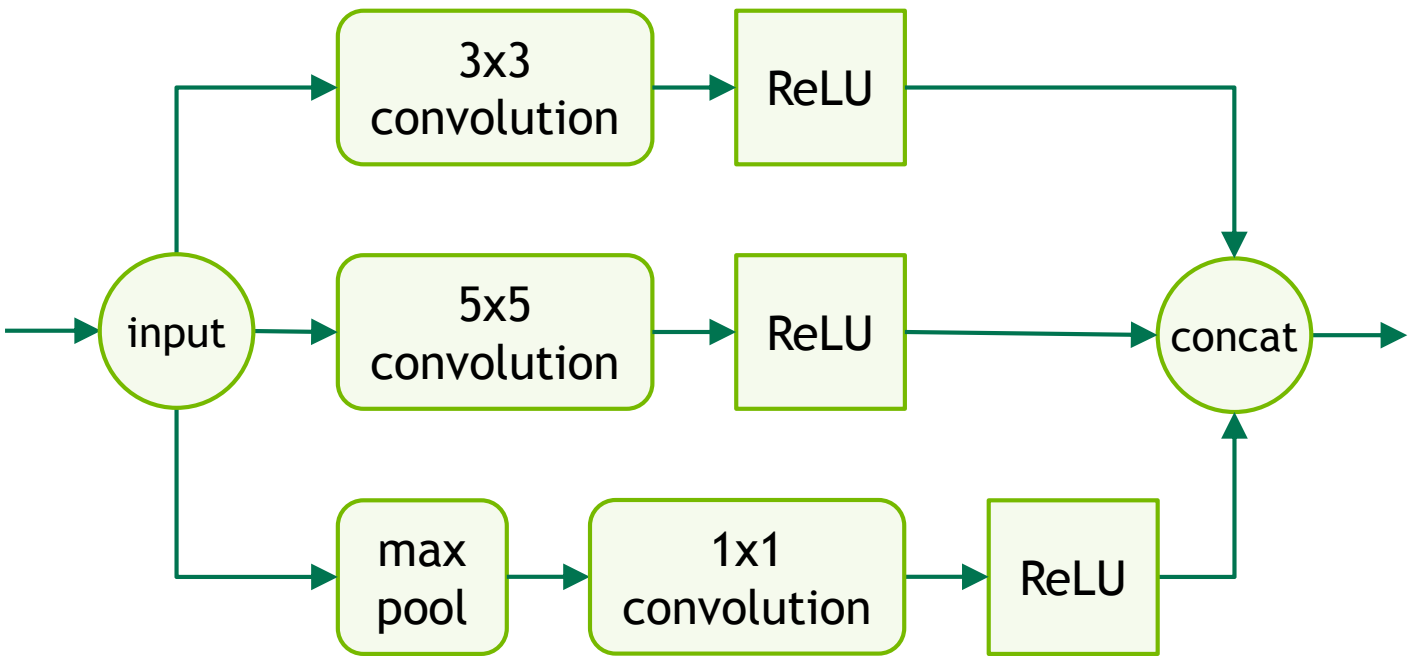
Map Graph-Based Workflows Directly Into CUDA



Graph from framework

```
// Define graph of work + dependencies
cudaGraphCreate(&graph);

cudaGraphAddNode(graph, kernel_a, {}, ...);
cudaGraphAddNode(graph, kernel_b, { a }, ...);
cudaGraphAddNode(graph, kernel_c, { a }, ...);
cudaGraphAddNode(graph, kernel_d, { a, b }, ...);

// Instantiate graph and apply optimizations
cudaGraphInstantiate(&instance, graph);

// Launch executable graph 100 times
for(int i=0; i<100; i++)
      cudaGraphLaunch(instance, stream);
```

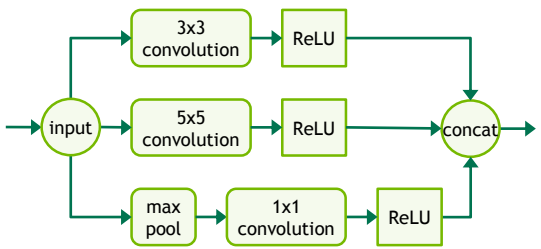
EXAMPLE INFERENCE WORK GRAPH



(representation only)

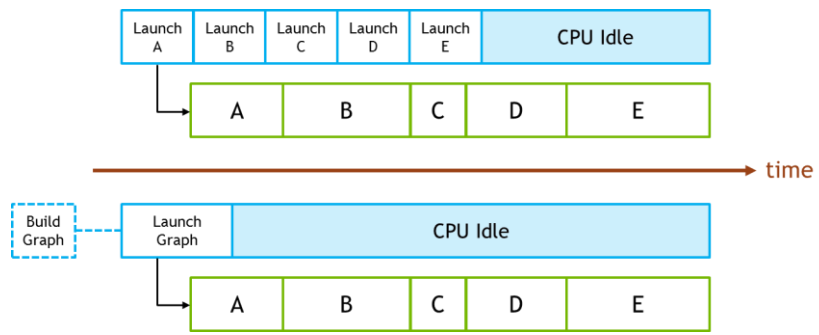
THE GRAPH ADVANTAGE

WHOLE WORKFLOW OPTIMIZATIONS



Seeing all work at once enables
new optimizations in hardware and software

EFFICIENT LAUNCH OF COMPLEX WORK

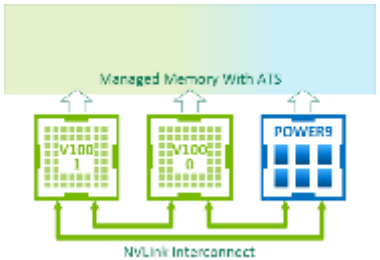


Launch potentially thousands of work items
with a single call

BEHIND THE CAMBRIAN EXPLOSION

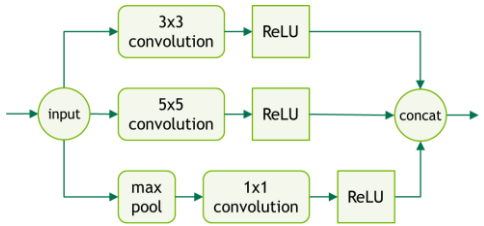
Enabling the State of the Art Through the CUDA Platform

CUDA Ecosystem



Unified Memory + POWER9-ATS

DGX-2 with full connectivity



Execution Models

CUDA 9.2

Beyond