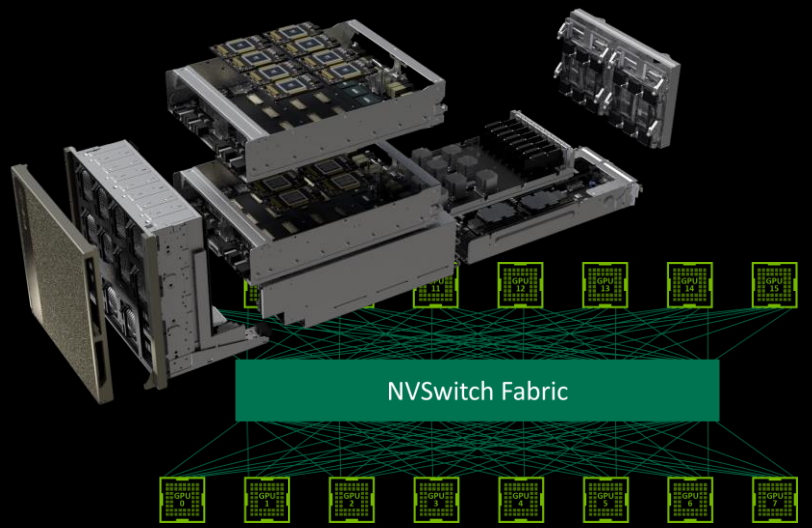
CUDA NEW FEATURES AND BEYOND

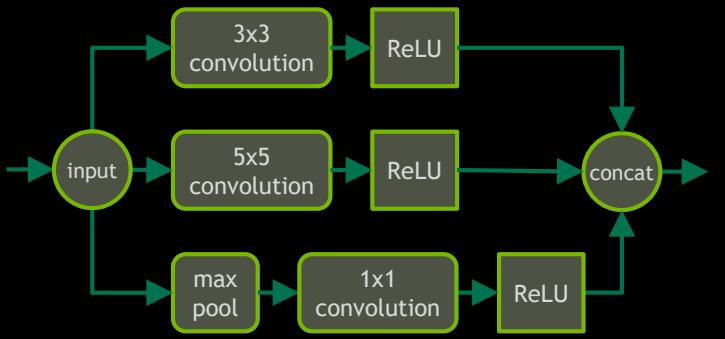
Stephen Jones, GTC 2019

A QUICK LOOK BACK

This Time Last Year...



DGX-2 + Unified Memory

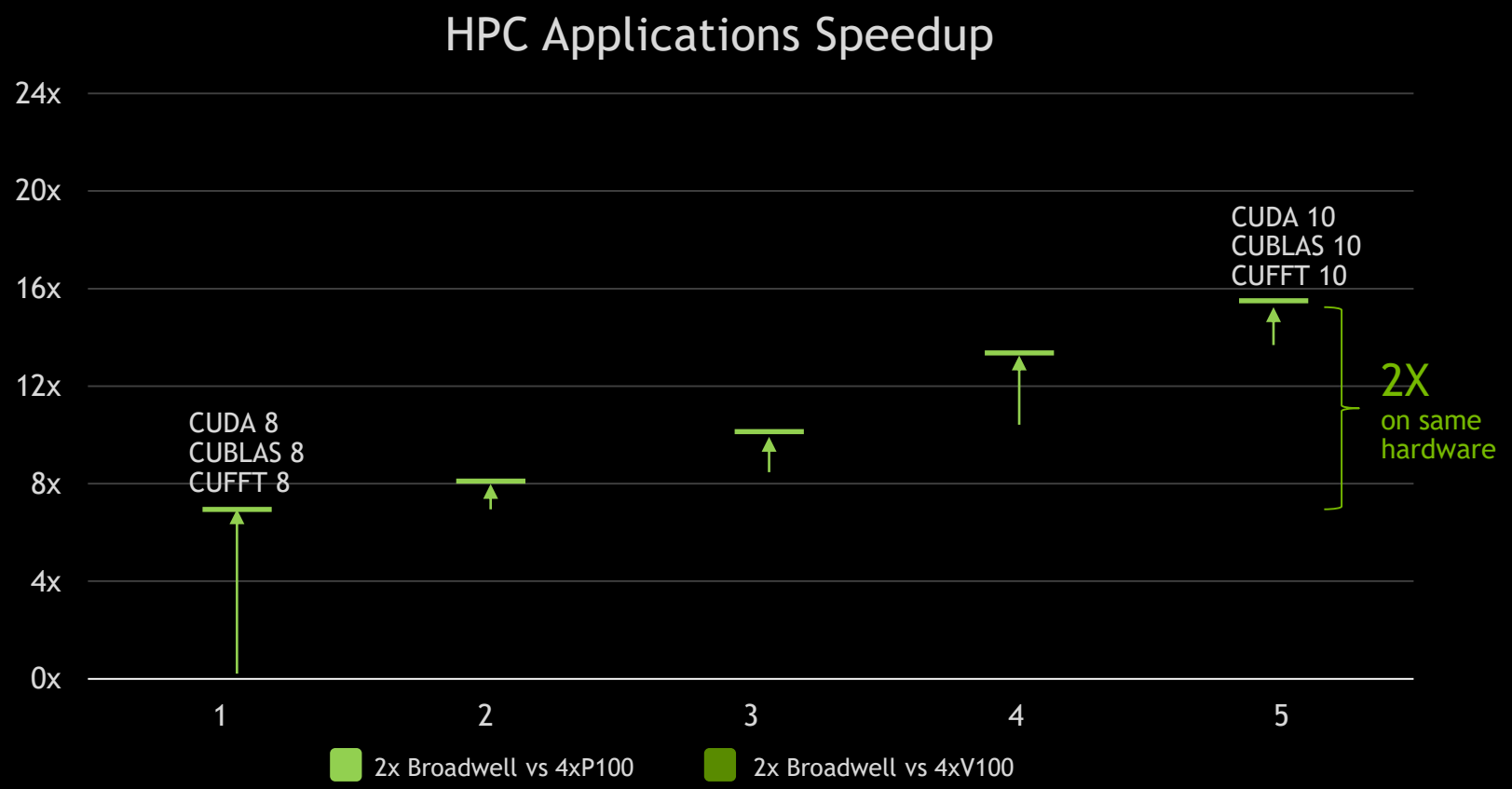


Asynchronous Task Graphs

S9241 – All You Need To Know About Programming NVIDIA's DGX-2, Wednesday March 20, 1-2PM

ACCELERATED COMPUTING IS FULL-STACK OPTIMIZATION

2X More Performance With Software Optimizations Alone



HPC Apps: AMBER, Chroma, GROMACS, GTC, LAMMPS, MILC, NAMD, QE, RTM, SPECFEM3D, VASP

NVIDIA

TESLA UNIVERSAL ACCELERATION PLATFORM

Single Platform To Drive Utilization and Productivity

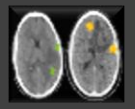
CUSTOMER USECASES


Speech


Translate


Recommender

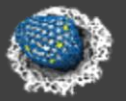
CONSUMER INTERNET

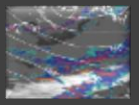

Healthcare

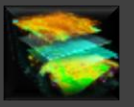

Manufacturing


Finance

INDUSTRIAL APPLICATIONS


Molecular Simulations


Weather Forecasting


Seismic Mapping

SUPERCOMPUTING

APPS & FRAMEWORKS

python

TensorFlow mxnet ONNX
PYTORCH RAPIDS KALDI Chainer

Amber ANSYS +550 Applications
NAMD SIMULIA

NVIDIA SDK & LIBRARIES

MACHINE LEARNING | RAPIDS
cuDF | cuML | cuGRAPH

DEEP LEARNING
cuDNN | cuBLAS | CUTLASS | NCCL | TensorRT

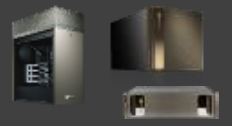
SUPERCOMPUTING
CuBLAS | CuFFT | OpenACC

CUDA

TESLA GPUs & SYSTEMS


TESLA GPU


VIRTUAL GPU

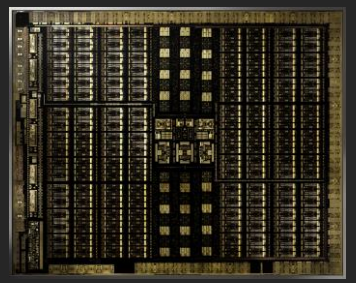

NVIDIA DGX FAMILY


NVIDIA HGX

DELL Hewlett Packard Enterprise IBM
SYSTEM OEM

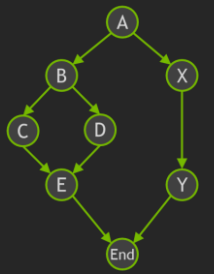
aws Google Cloud Platform Microsoft Azure
CLOUD

TECHNOLOGY

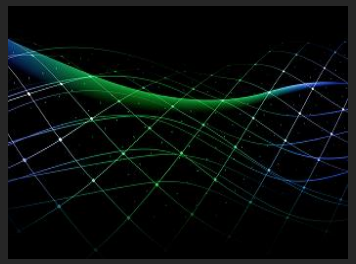


PLATFORM

DEVELOPMENT

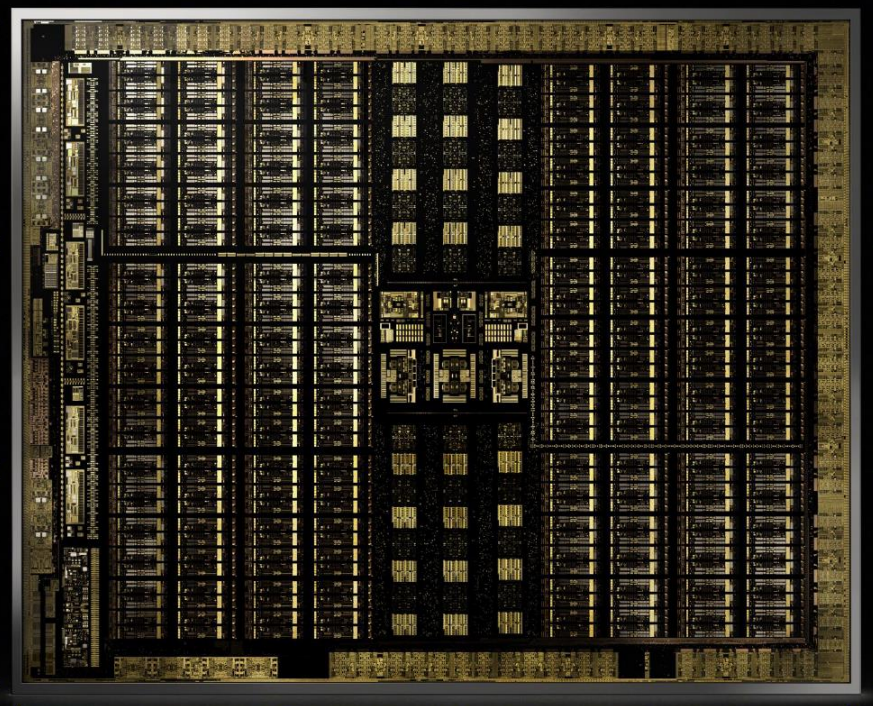


TOOLKIT



NEW TURING GPU

GREATEST LEAP SINCE 2006 CUDA GPU



Turing SM
16 TFLOPS + 16 TIPS
Concurrent FP & INT Execution
Unified L1 Cache
Variable Rate Shading

RT Core
10 Giga Rays/sec
Ray Triangle Intersection
BVH Traversal

Tensor Core
125 TFLOPS FP16
250 TOPS INT8
500 TOPS INT4

Display
Native HDR
8K DisplayPort
VirtualLink

NVLINK
100 GB/sec
GPU-GPU Memory Access

Video
HEVC 8K Real Time Encode
25% Improved Bitrate

Memory
6MB L2 Cache
384-bit G6 @ 14Gbps
672 GB/sec

TESLA T4

WORLD'S MOST ADVANCED SCALE-OUT GPU

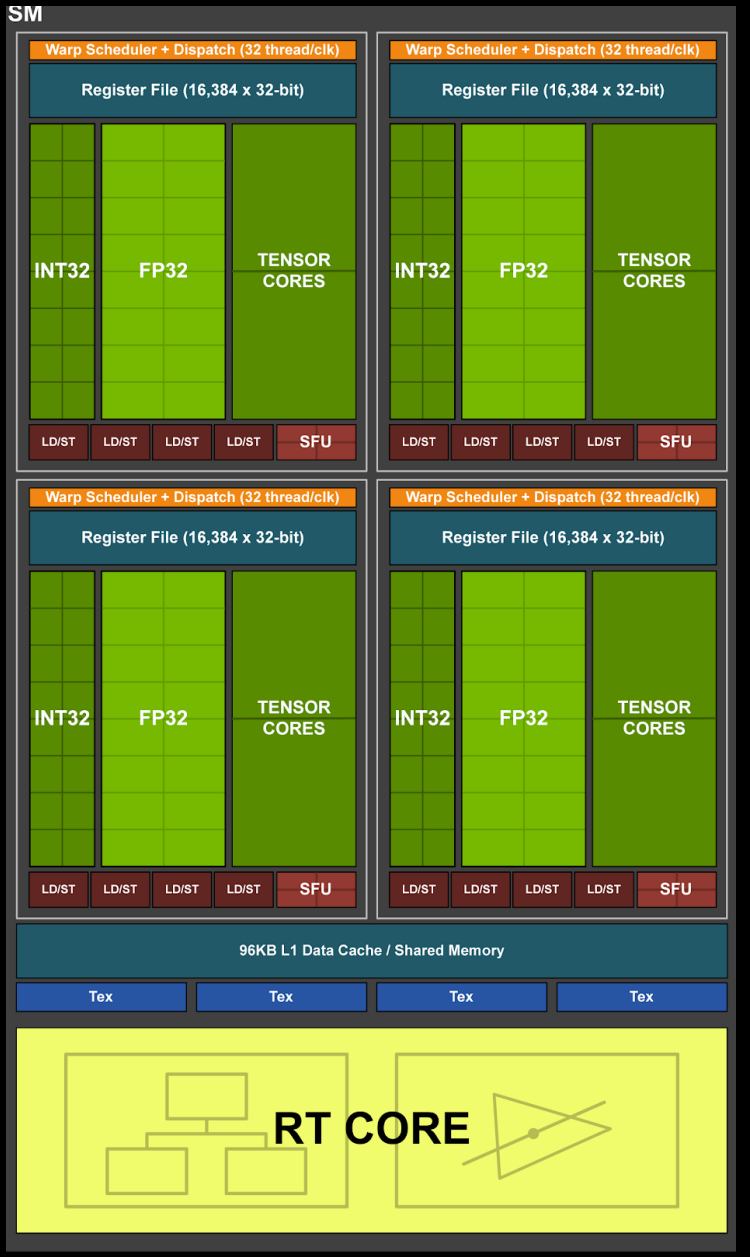
320 Turing Tensor Cores
2,560 CUDA Cores
65 FP16 TFLOPS | 130 INT8 TOPS | 260 INT4 TOPS
16GB | 320GB/s
70 W

Deep Learning Training & Inference
HPC Workloads
Video Transcode
Remote Graphics

TURING SM

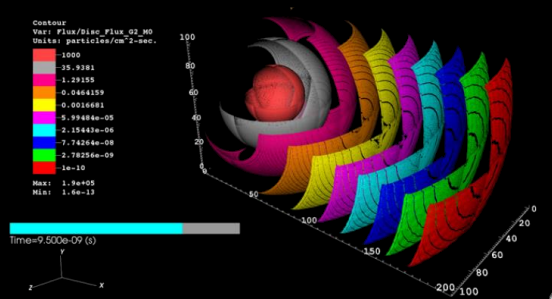
	TU102
INT32	64
FP32	64
Tensor Cores	8
RT Core	1
Register File	256 KB
L1 and shmem	96 KB
Max threads	1024
Compute Capability	75*

**Volta (cc70) code runs on Turing without JIT or recompile!*

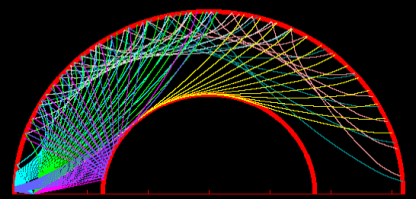


RT CORE POTENTIAL FOR ACCELERATION OF NUMERICAL ALGORITHMS

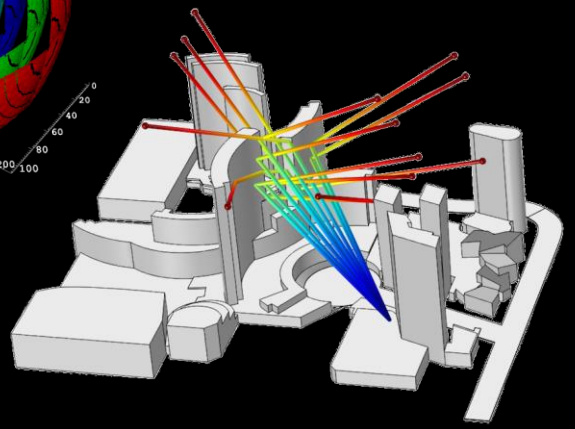
Geometry-Heavy Compute Applications



Neutron Transport
Credit: CERT, Texas A&M

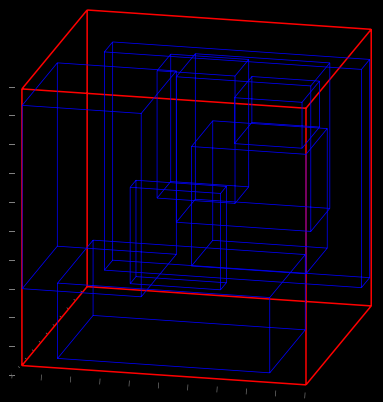


Seismic Shear Wave Tracing
Credit: SERC, Carleton College

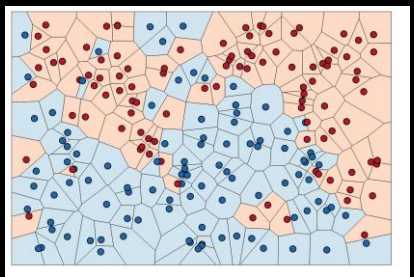


RF Wave Propagation
Credit: COMSOL

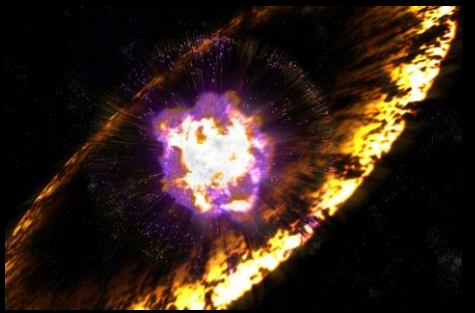
Unstructured Algorithms



R-Trees, Decision Trees
Credit: Wikimedia



Nearest Neighbor Search
Credit: Fortmann-Roe



Radiaton Transport
Credit: Greg Stewart / SLAC

LOCATING NEIGHBORS WITHIN A RANGE

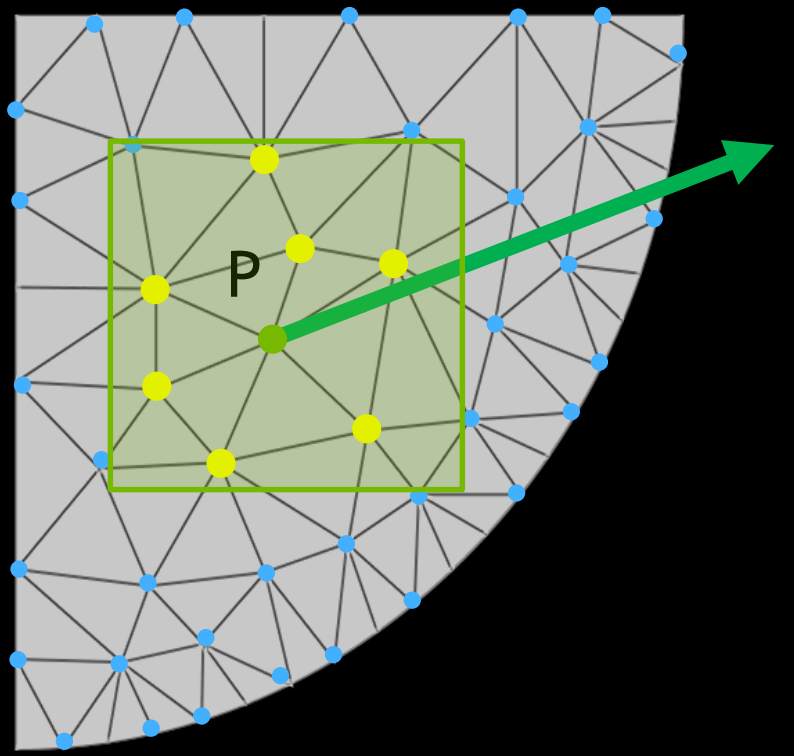
Intersect Rays With Bounding Box Around Points Of Interest

For any arbitrary set of points

For a point P, find neighbors within a shape enclosed in a Bounding Box

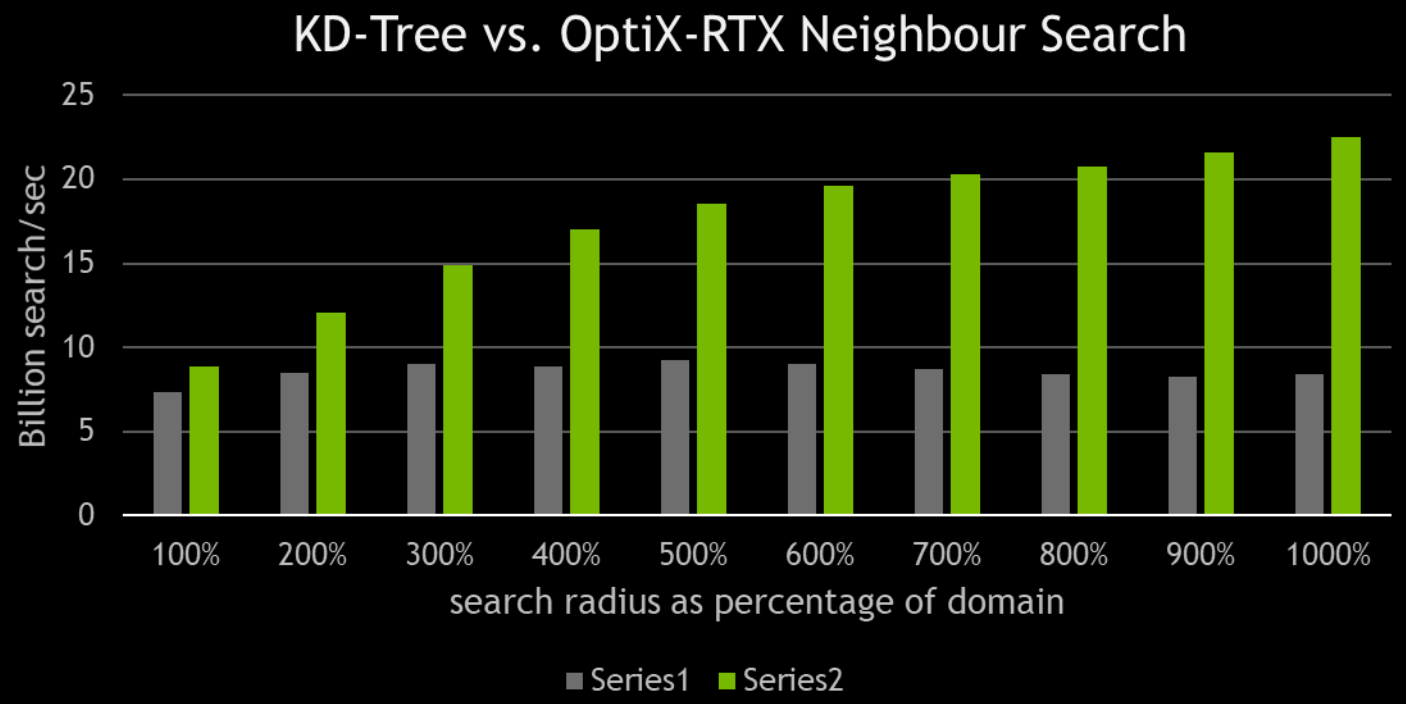
Ray-based solution

1. Attach a box of width R to each point
2. Shoot one ray from P in arbitrary direction, t_max = 2*R
3. Neighbors boxes will have either entry/exit intersection but never both.
4. Refine result points to any shape within the box in SM.



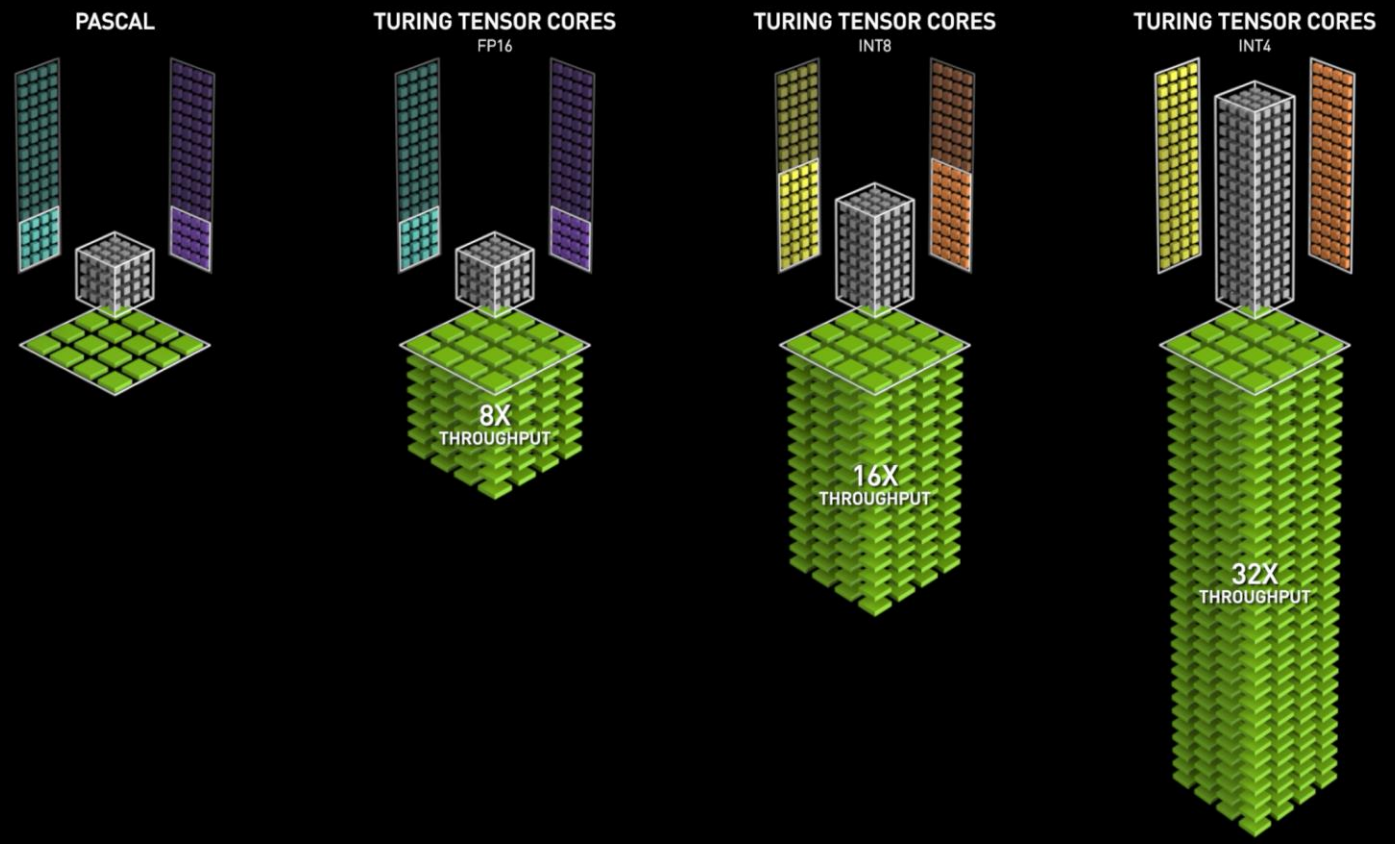
RAY TRACED NEAREST NEIGHBOUR SEARCH

Using RT-Cores Through OptiX RTX



NEW TURING TENSOR CORE

MULTI-PRECISION FOR AI INFERENCE & SCALE-OUT TRAINING
65 TFLOPS FP16 | 130 TeraOPS INT8 | 260 TeraOPS INT4

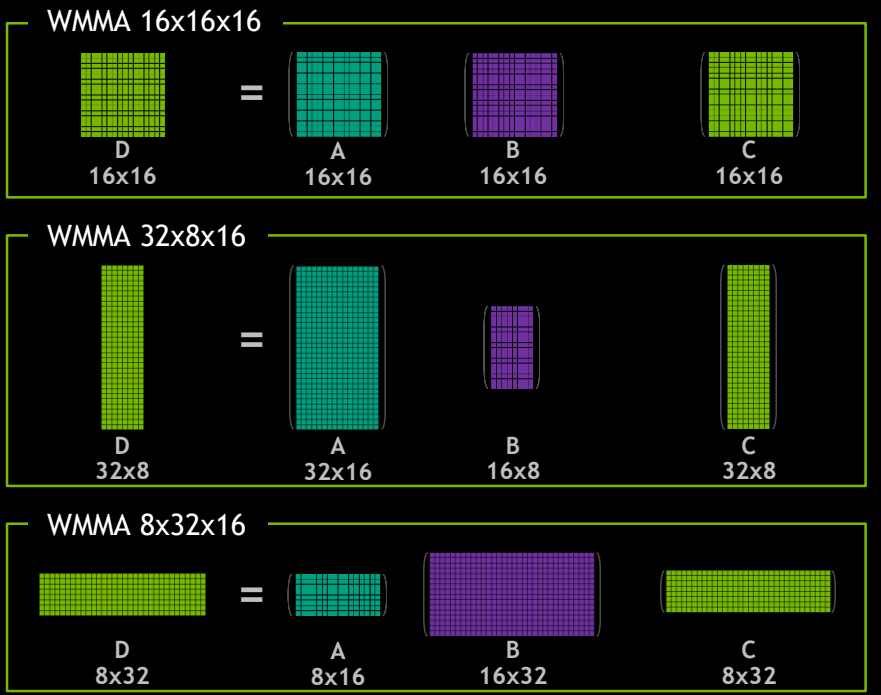


TURING TENSOR CORE

New 8-Bit & Sub-Byte Warp Matrix Functions In CUDA

8-bit integer WMMA operations

- Turing (sm_75) only
- Signed & unsigned 8-bit input
- 32-bit integer accumulator
- Match input/output dimensions with *half*
- **2048 ops per cycle, per SM**



EXPERIMENTAL WARP MATRIX FUNCTIONS

Turing Enables Experimental Sub-Byte Tensor Core Operations

Experimental Sub-Byte Operations

- 4-bit signed & unsigned input
- 1-bit input with custom matrix operations
- 32-bit accumulator output

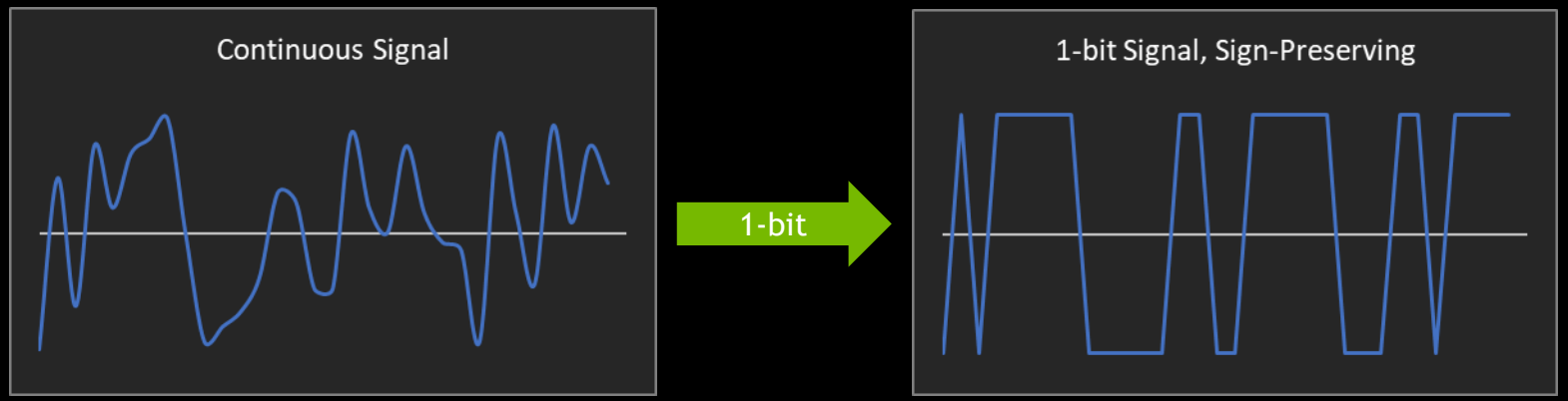
Access via special namespace
nvcuda::wmma::experimental

```
namespace experimental {
    namespace precision {
        struct u4; // 4-bit unsigned
        struct s4; // 4-bit signed
        struct b1; // 1-bit
     }
    enum bmmaBitOp { bmmaBitOpXOR = 1 };
    enum bmmaAccumulateOp { bmmaAccumulateOpPOPC = 1 };
}
```

Enables researchers to experiment with ultra low precision

BINARY TENSOR CORES

Example: Binarized Neural Networks

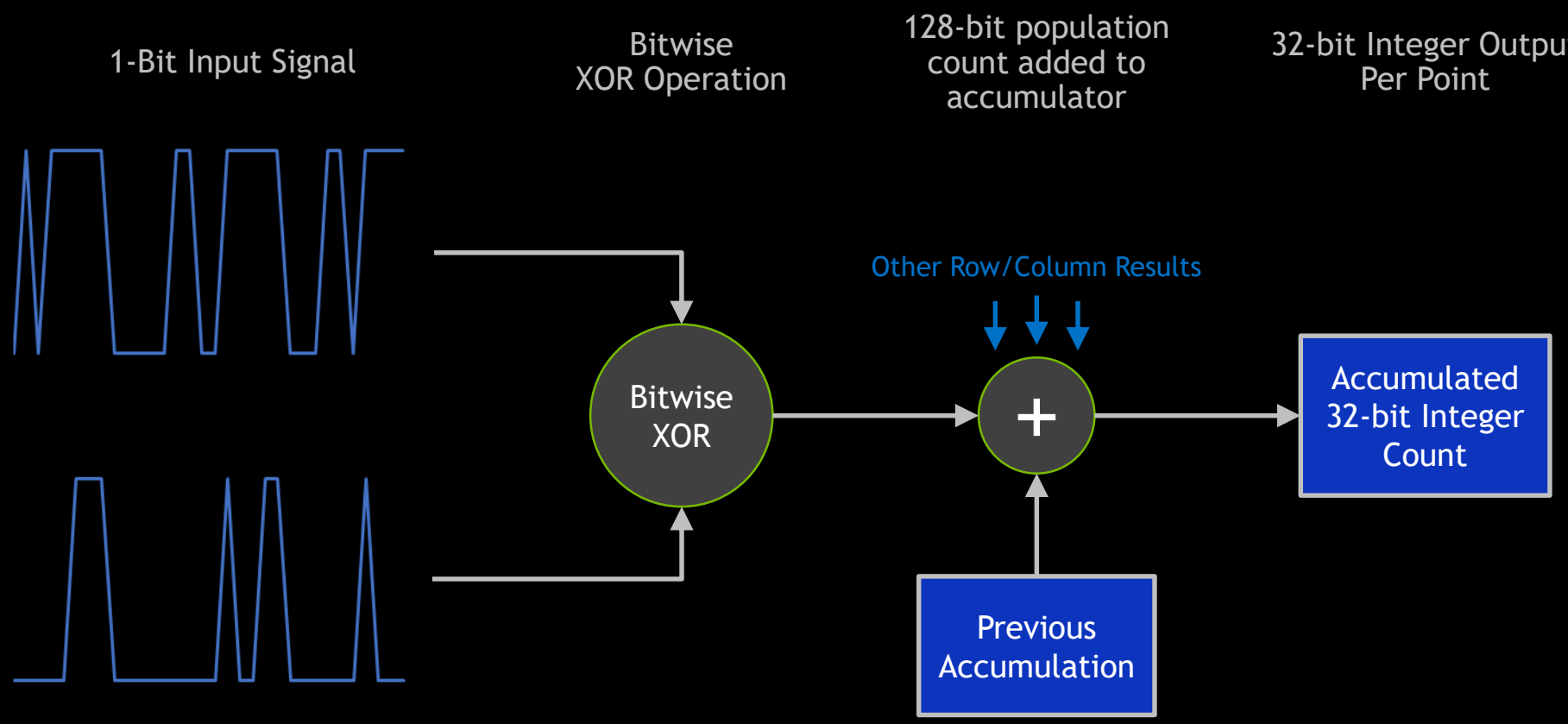


Concept

- Train neural networks on lower-precision data: faster compute, lower memory size
- Reduce data to positive / negative sign value – can fit in single bit (1 = +ve, 0 = -ve)
- 1-bit weight & activation calculations based only on sign of data

https://arxiv.org/pdf/1602.02830.pdf

BINARY TENSOR CORE OPERATION



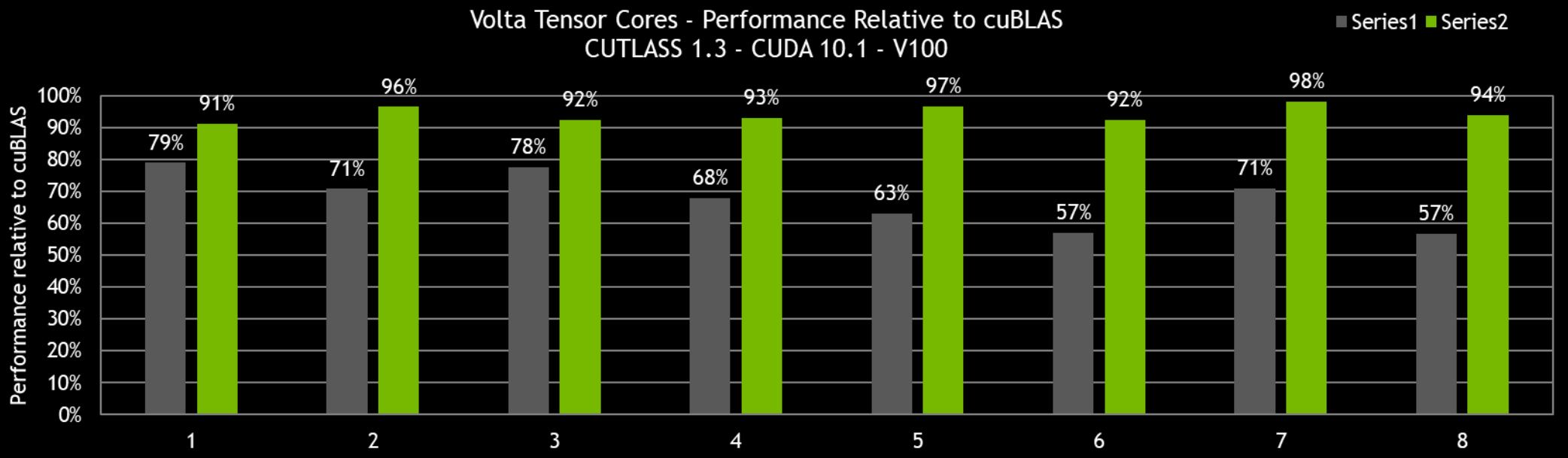
NEW TURING WARP MATRIX FUNCTIONS

	Input Precision	Output	Supported Sizes	Max Ops/Clock/SM
Native Types	half *	half or float	16 x 16 x 16 32 x 8 x 16 8 x 32 x 16	1024
	char	integer (int32)		2048
	unsigned char			
Experimental	precision::u4 (4-bit unsigned)	integer (int32)	8 x 8 x 32	4096
	precision::s4 (4-bit signed)			
	precision::b1 (1-bit)		8 x 8 x 128	16384

* Also available on Volta sm_70. Note: WMMA requires recompilation for Turing sm_75 for peak performance

CUTLASS 1.3

GEMM kernels targeting Volta Tensor Cores natively with **mma.sync**

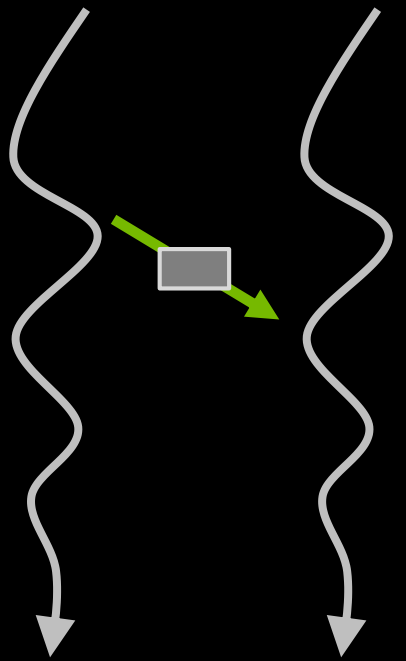


New in CUDA 10.1 & CUTLASS 1.3: **mma.sync**

PTX assembly instruction enables maximum efficiency of Volta Tensor Cores operation

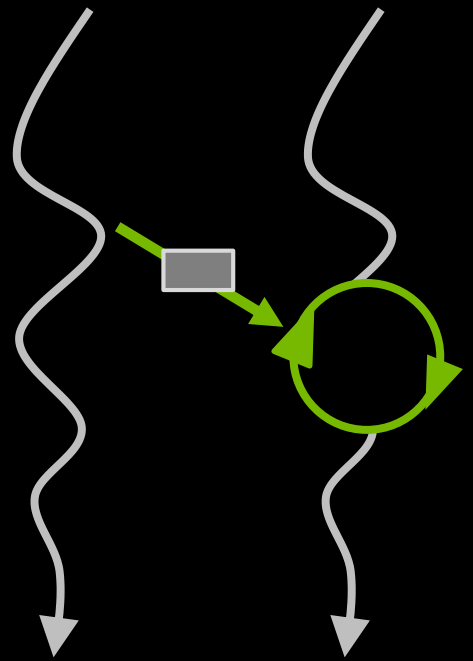
INDEPENDENT THREAD SCHEDULING

Communicating Algorithms



Pascal: Lock-Free Algorithms

Threads cannot wait for messages

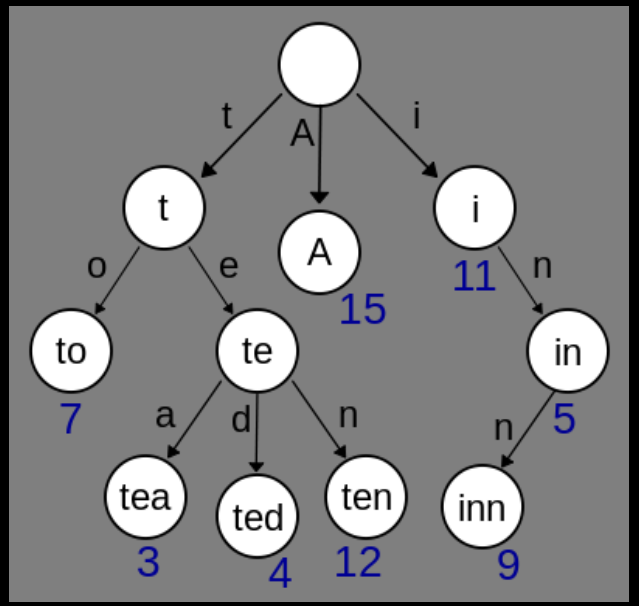


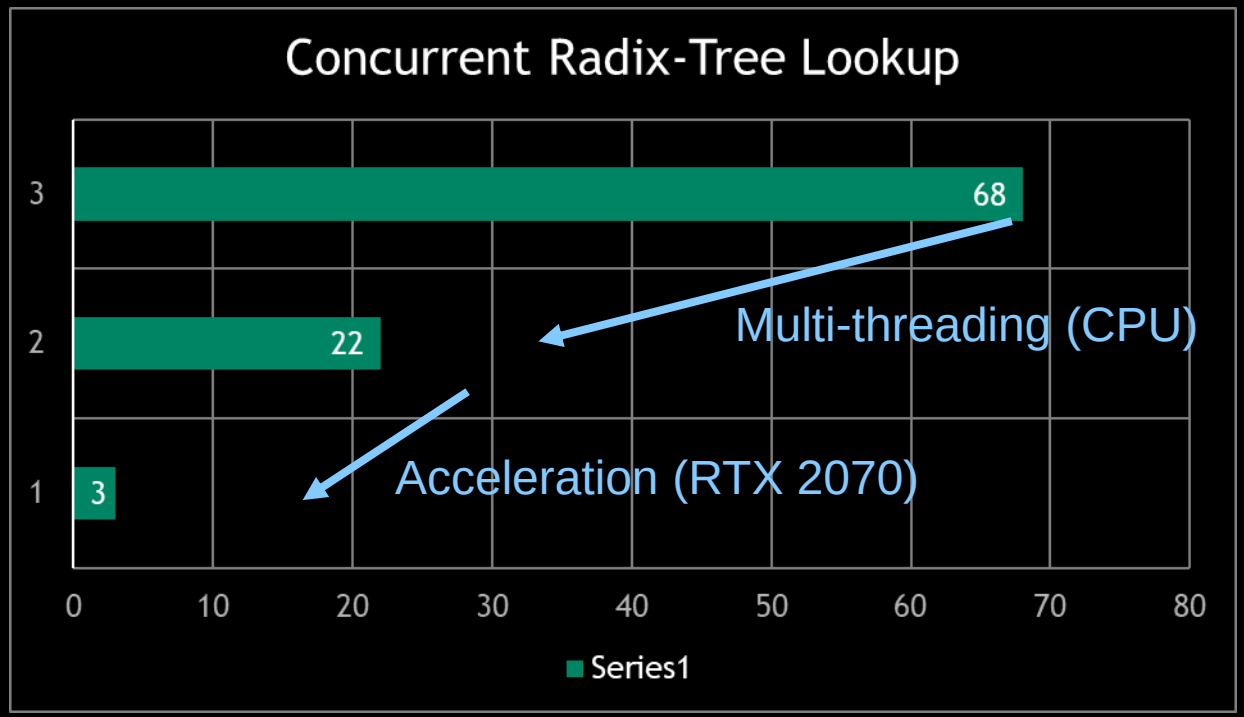
Volta/Turing: Starvation Free Algorithms

Threads may wait for messages

INDEPENDENT THREAD SCHEDULING

Enable Fast Mutexes For Concurrent Data Structures, Replace Complex Lock-Free Algorithms



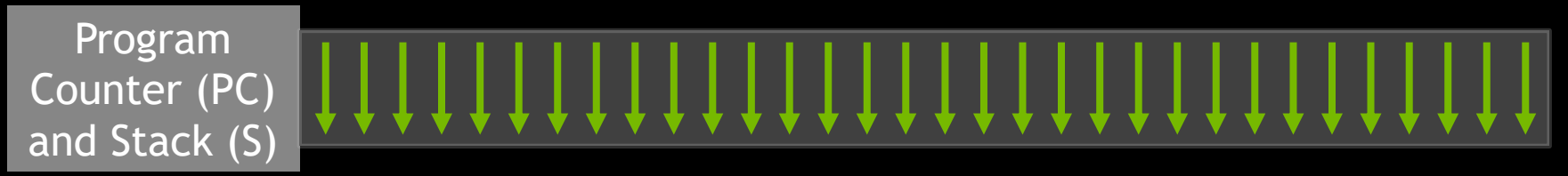


Ref: *High Radix Concurrent C++*, Olivier Giroux, CppCon 2018 - https://www.youtube.com/watch?v=75LcDvlEIYw

See Also: https://devblogs.nvidia.com/cuda-turing-new-gpu-compute-possibilities/

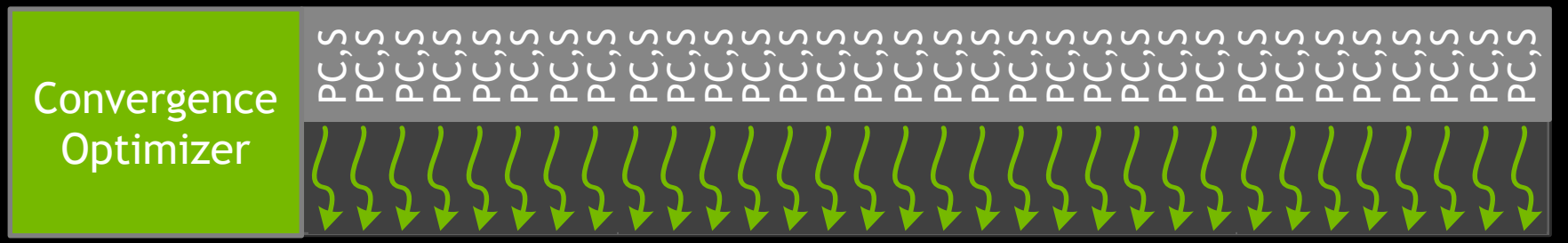
WARP IMPLEMENTATIONS

Pre-Volta



32 thread warp

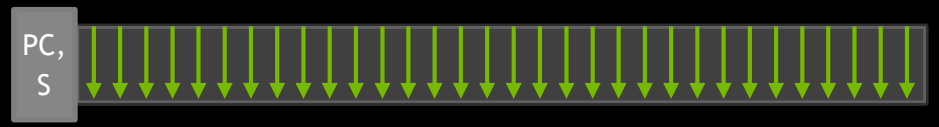
Volta/Turing



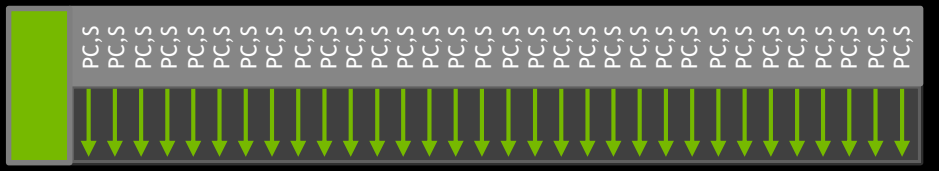
32 thread warp with independent scheduling

SYNCHRONIZING WARP FUNCTIONS

Pre-Volta



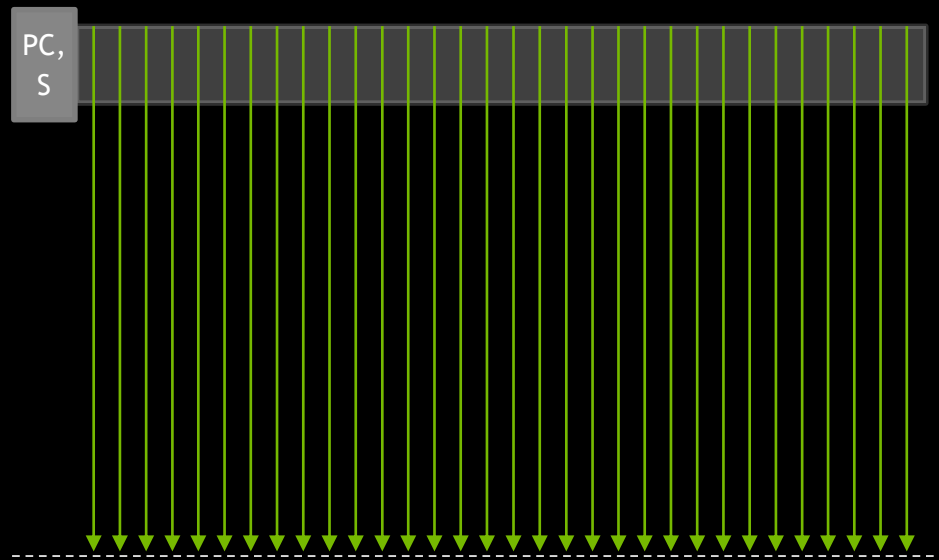
Volta & Turing



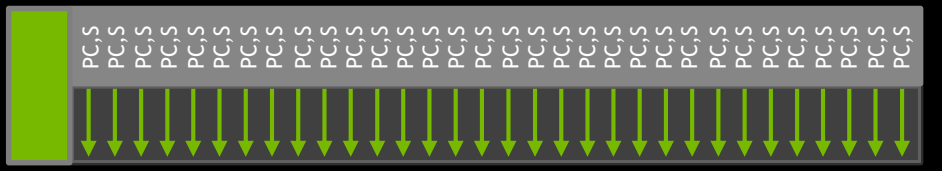

```
my_value = __shfl(thread, their_value)
```

SYNCHRONIZING WARP FUNCTIONS

Pre-Volta



Volta & Turing

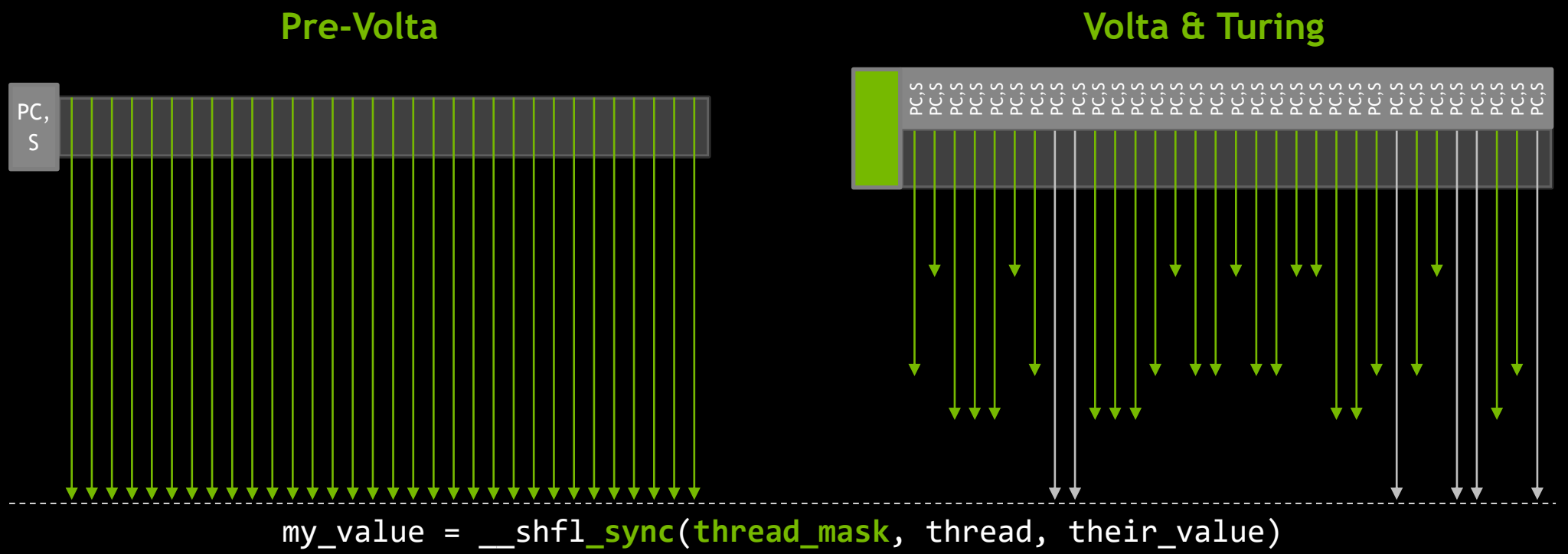



```
my_value = __shfl(thread, their_value)
```

SYNCHRONIZING WARP FUNCTIONS

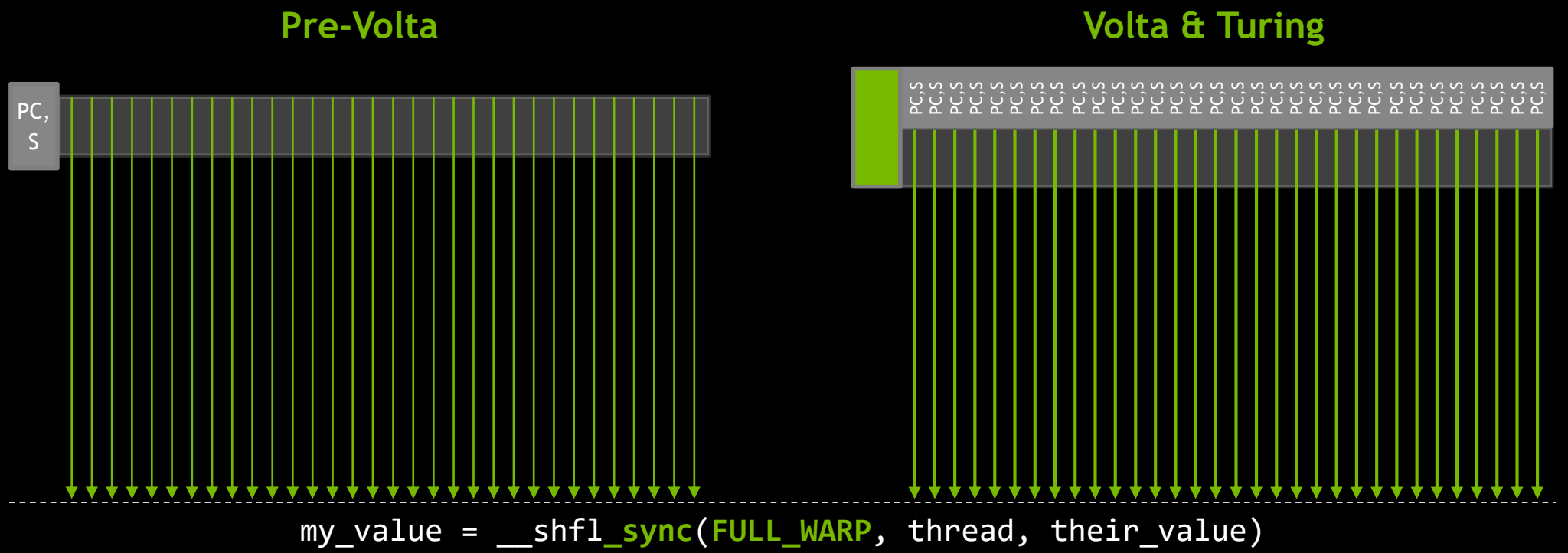

```
my_value = __shfl(thread, their_value)
```

SYNCHRONIZING WARP FUNCTIONS




```
my_value = __shfl_sync(thread_mask, thread, their_value)
```

SYNCHRONIZING WARP FUNCTIONS




```
my_value = __shfl_sync(FULL_WARP, thread, their_value)
```

__shfl_sync() and all other *_sync collective operations work on all GPU architectures

REMOVAL OF NON-SYNC WARP FUNCTIONS

Functions Deprecated In CUDA 9.0: Now **Removed** In CUDA 10.1

Removed Function	Replacement Function
`__ballot()`	`__ballot_sync()`
`__any()`	`__any_sync()`
`__all()`	`__all_sync()`
`__shfl()`	`__shfl_sync()`
`__shfl_up()`	`__shfl_up_sync()`
`__shfl_down()`	`__shfl_down_sync()`
`__shfl_xor()`	`__shfl_xor_sync()`

Programs using old functions:

- **Will no longer compile** for sm_70 (Volta), or sm_75 (Turing)
- Will still compile as older *compute_60* (Pascal) architecture, but without support for any Volta or Turing features

To compile as *compute_60*, add the following arguments to your compile line:

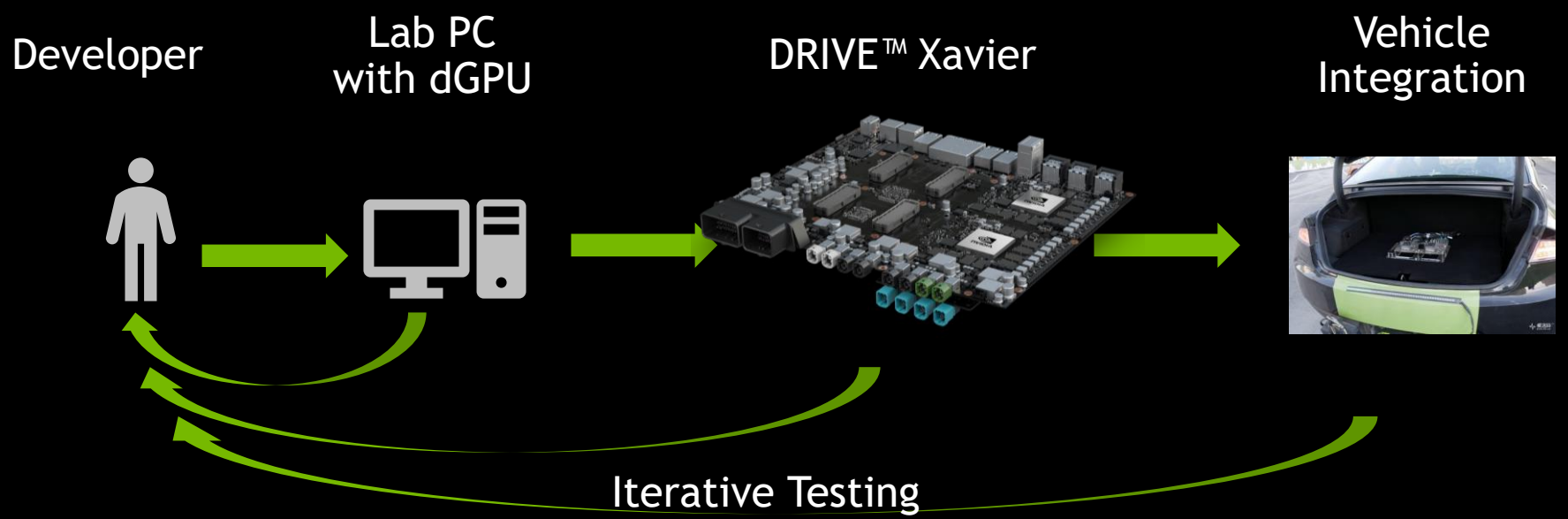
```
-arch=compute_60 -code=sm_70
```

CUDA 10.1 FOR TEGRA SYSTEMS

Platform	Host OS	Version	Target OS	Version	Compiler Support
L4T	ubuntu	16.04 LTS 18.04 LTS	ubuntu	18.04 LTS	GCC 7.3
Android		16.04 LTS	android	P (Pie)	Clang 6.0
Auto		16.04 LTS	ubuntu	18.04 LTS	GCC 7.3
			QNX	QNX SDP 7.0.2	GCC 5.4
			yocto PROJECT	Yocto 2.5	GCC 7.3

DRIVE DEVELOPER WORKFLOW

Iterative Workflow



Fast iteration loop with PC, same CUDA code used across PC, DRIVE Dev Platform, and vehicle

CUDA 10.1 TEGRA SYSTEMS ENHANCEMENTS

NVIDIA-Direct™ RDMA

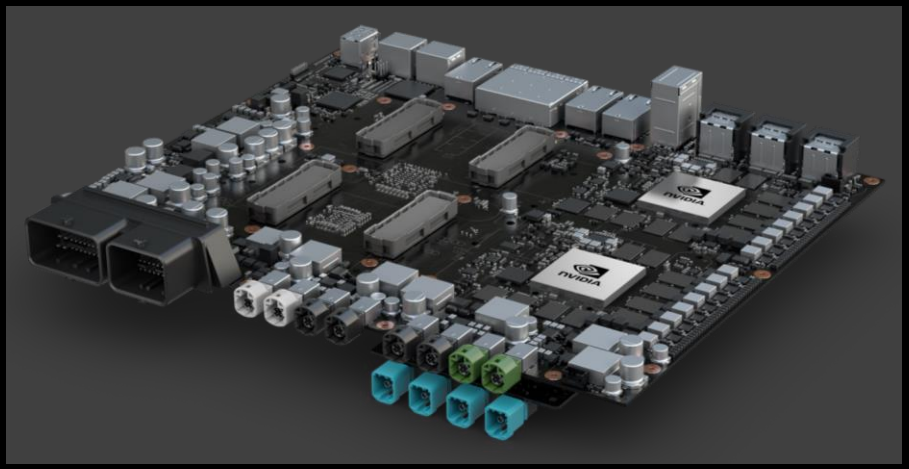
Third-party PCIe devices can communicate directly with the integrated GPU

User-Mode Submission on Linux-4-Tegra

Faster and more predictable work submission latency

Rich Error Reporting

Detailed error reporting from GPU execution faults (MMU, alignment, etc)



CUDA 10.1 PLATFORM SUPPORT

New OS and Host Compilers

PLATFORM	OS	VERSION	COMPILERS
Linux		18.04.2 LTS 16.04.5 LTS 14.04.5 LTS	GCC 8.x PGI 19.x Clang 7.0.x ICC 19 XLC 16.1.x (POWER)
		7.6 7.6 POWER LE	
		SLES 15	
		29	
		Leap 15	
Windows	 Windows Server	2019 2016 2012 R2	Microsoft Visual Studio 2017 (15.x) Microsoft Visual Studio 2019 (Previews)
Mac	macOS	10.13.6	Xcode 10.1

TESLA DRIVERS AND COMPATIBILITY

Run New Versions Of CUDA **Without** Upgrading Kernel Drivers

Long Term Service Branch (LTSB)

- One per GPU architecture (i.e. major CUDA release such as CUDA 10.0)
- Supported for up to 3 years
- R418 is the first LTSB
- CUDA compatibility will be supported for the lifetime of the LTSB

Driver Branch	CUDA 10 Compatible	CUDA 10.1 Compatible
CUDA 9.0	Yes	Yes
CUDA 9.1	No	No
CUDA 9.2	No	Coming soon
CUDA 10.0	-	Yes

CUDA CONTAINERS ON NVIDIA GPU CLOUD

CUDA containers available from NGC Registry at **nvcr.io/nvidia/cuda**

Three different flavors:

Base

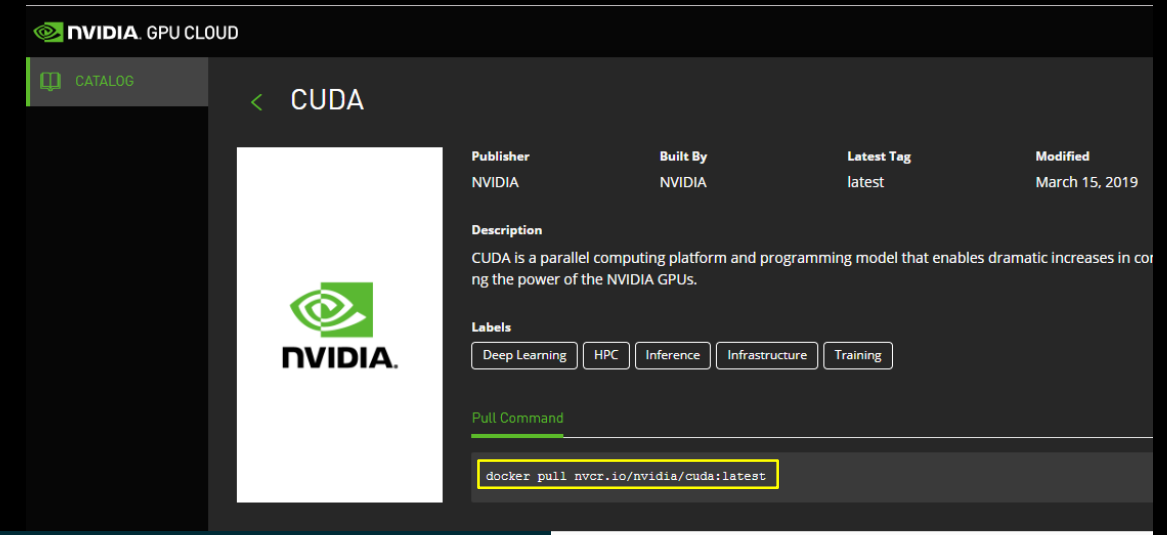
Contains the minimum components required to run CUDA applications

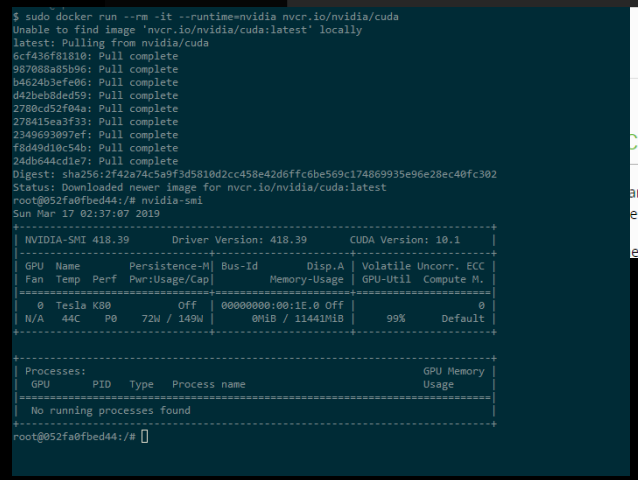
Runtime

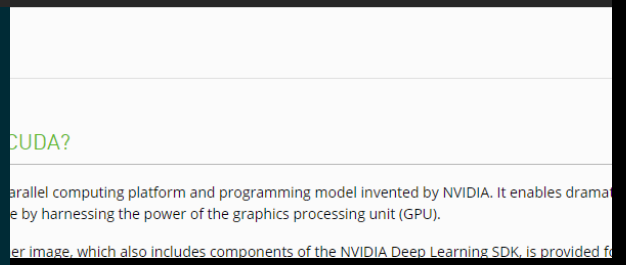
Contains *base* + CUDA libraries (e.g. cuBLAS, cuFFT)

Devel

Contains *runtime* + CUDA command line developer tools. Some *devel* tags also include cuDNN





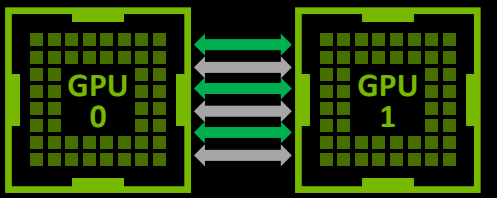


INCREASING CUDA CAPABILITIES ON WINDOWS

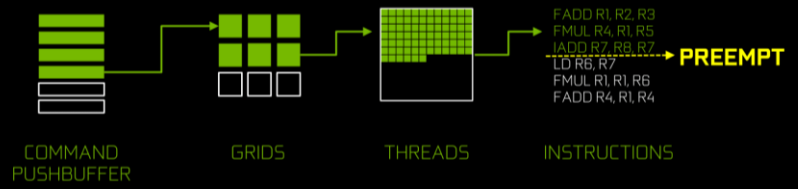
Additions Since CUDA 9

CUDA Interop with Vulkan and DX12

Windows Peer-to-Peer



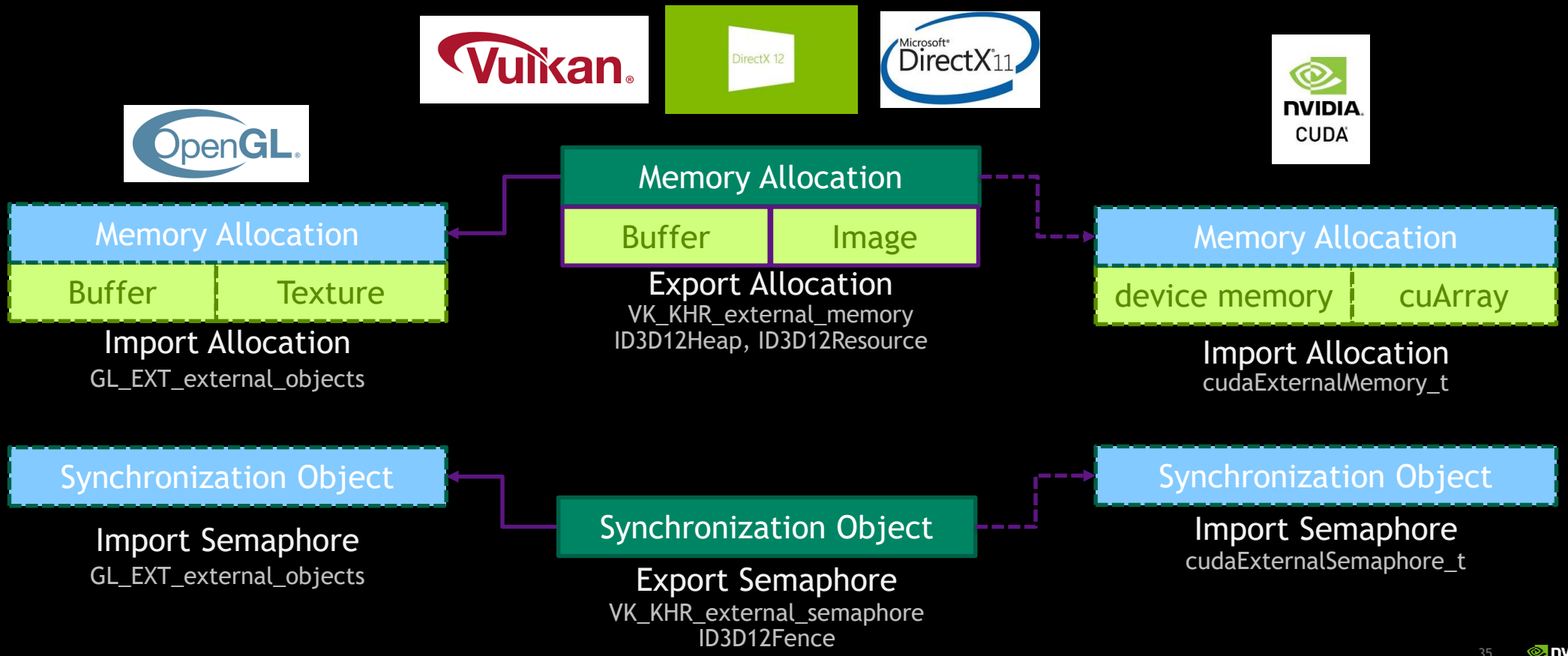
Compute Preemption (CILP) Support



S9957 – Using CUDA on Windows,
Wednesday 3-4pm

NEW GRAPHICS INTEROP

Direct Native Resource Mapping + CUDA-OpenGL interop via Vulkan



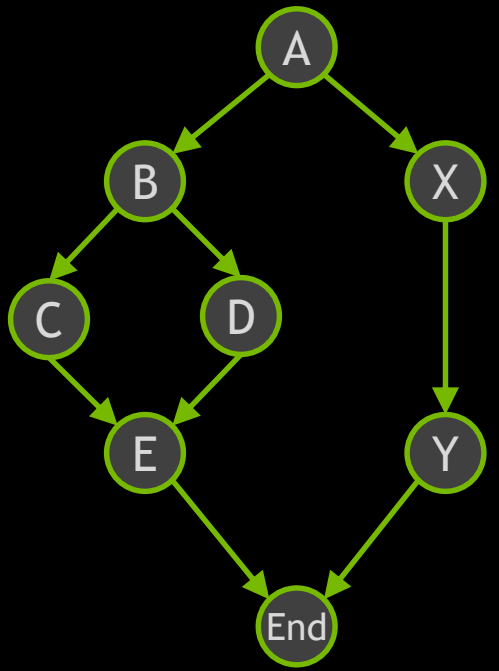
ASYNCHRONOUS TASK GRAPHS

A Graph Node Is A CUDA Operation

Sequence of operations, connected by dependencies

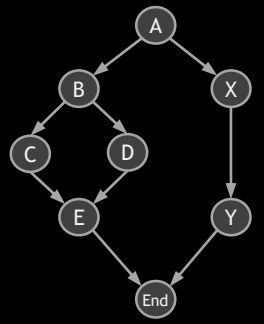
Operations are one of:

Kernel Launch	CUDA kernel running on GPU
CPU Function Call	Callback function on CPU
Memcopy/Memset	GPU data management
Sub-Graph	Graphs are hierarchical



THREE-STAGE EXECUTION MODEL

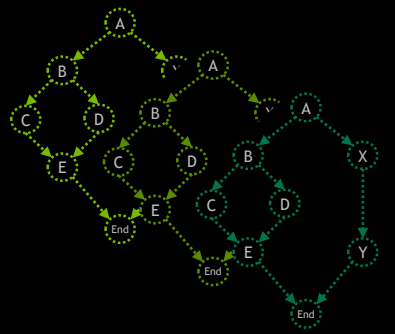
Define



Single Graph “Template”

Created in host code
or built up from libraries

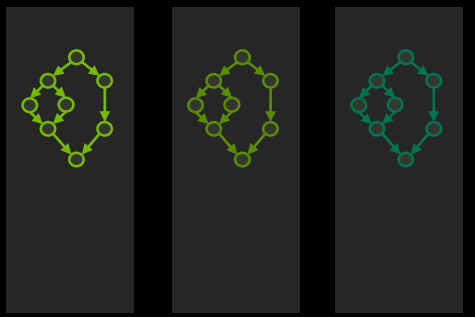
Instantiate



Multiple “Executable Graphs”

Snapshot of template
Sets up & initializes GPU
execution structures
(create once, run many times)

Execute



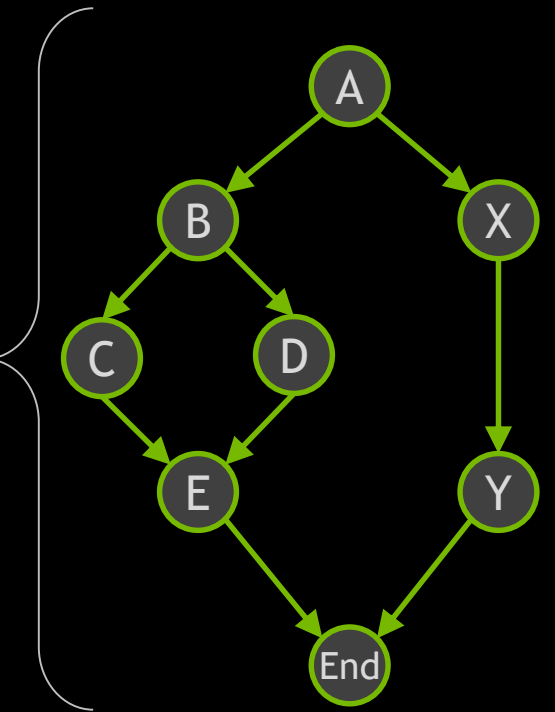
Executable Graphs
Running in CUDA Streams

Concurrency in graph
is not limited by stream

NEW EXECUTION MECHANISM

Graphs Can Be Generated Once Then Launched Repeatedly

```
for(int i=0; i<1000; i++) {
    launch_graph( G );
}
```



WORKFLOW EXECUTION OPTIMIZATIONS

Reducing System Overheads Around Short-Running Kernels

Breakdown of time spent during execution

53% Overhead

WORKFLOW EXECUTION OPTIMIZATIONS

Reducing System Overheads Around Short-Running Kernels

Breakdown of time spent during execution

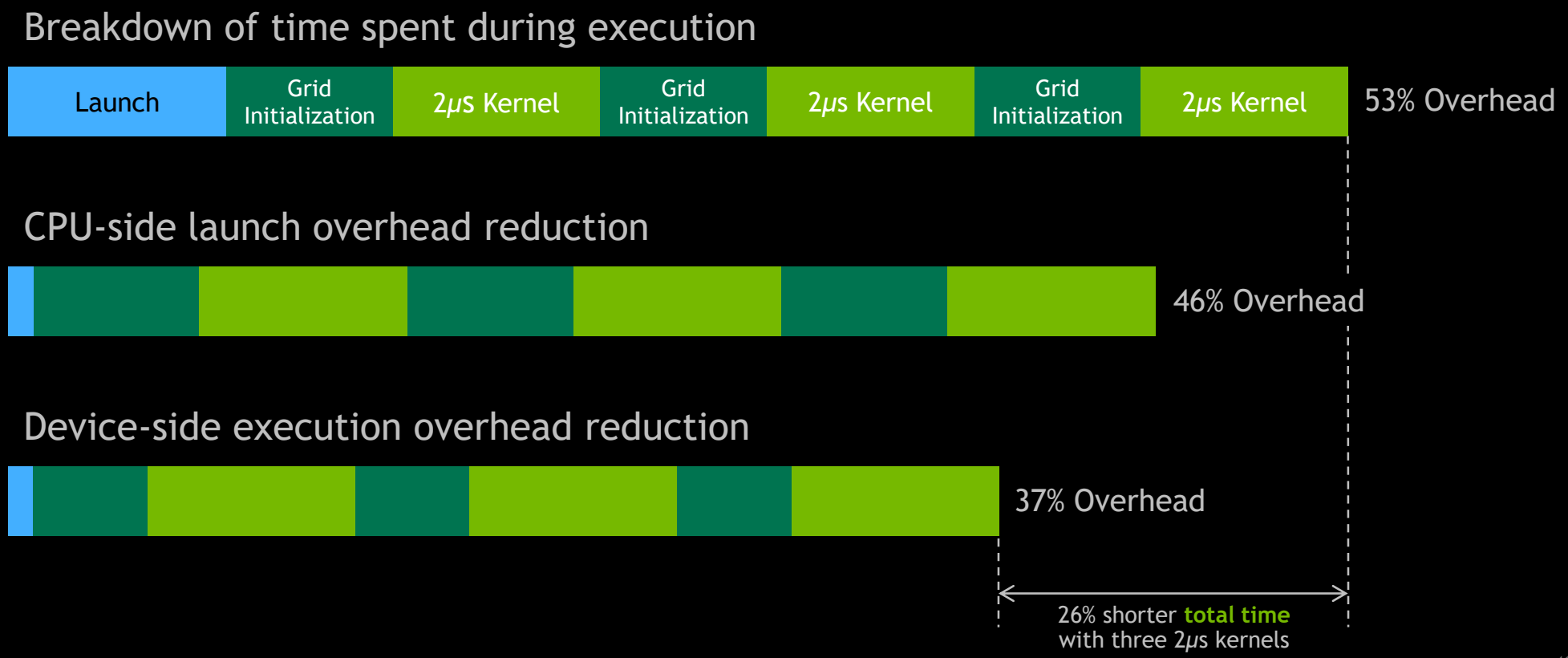
53% Overhead

CPU-side launch overhead reduction

46% Overhead

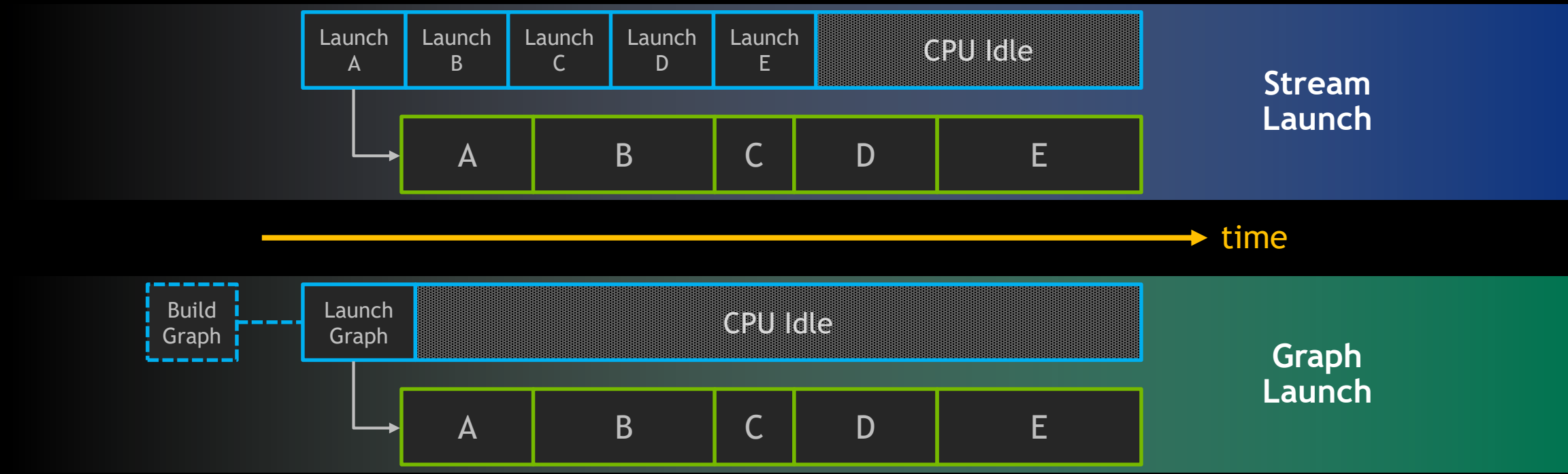
WORKFLOW EXECUTION OPTIMIZATIONS

Reducing System Overheads Around Short-Running Kernels



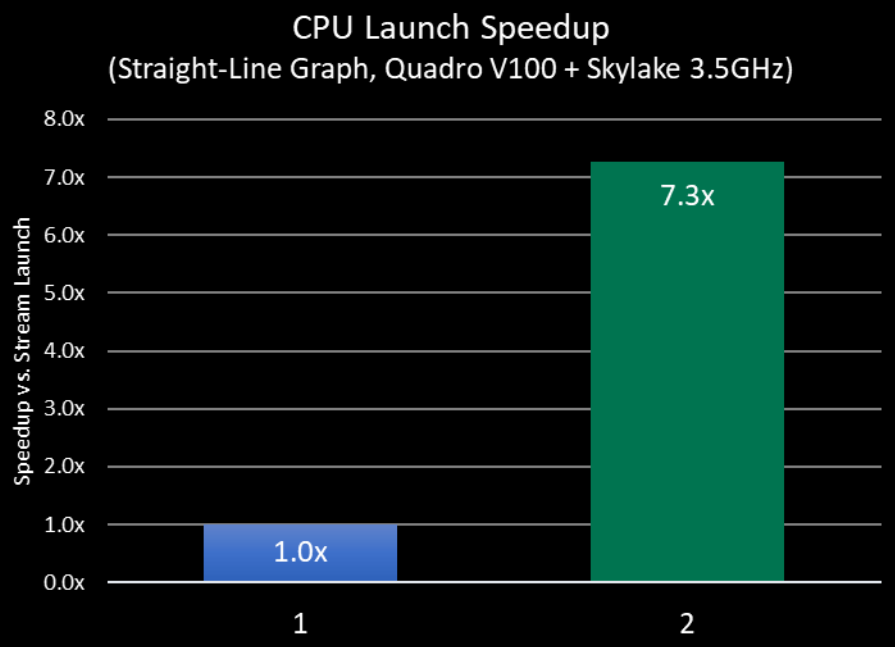
FREE UP CPU RESOURCES

Release CPU Time For Lower Power, or Running Other Work

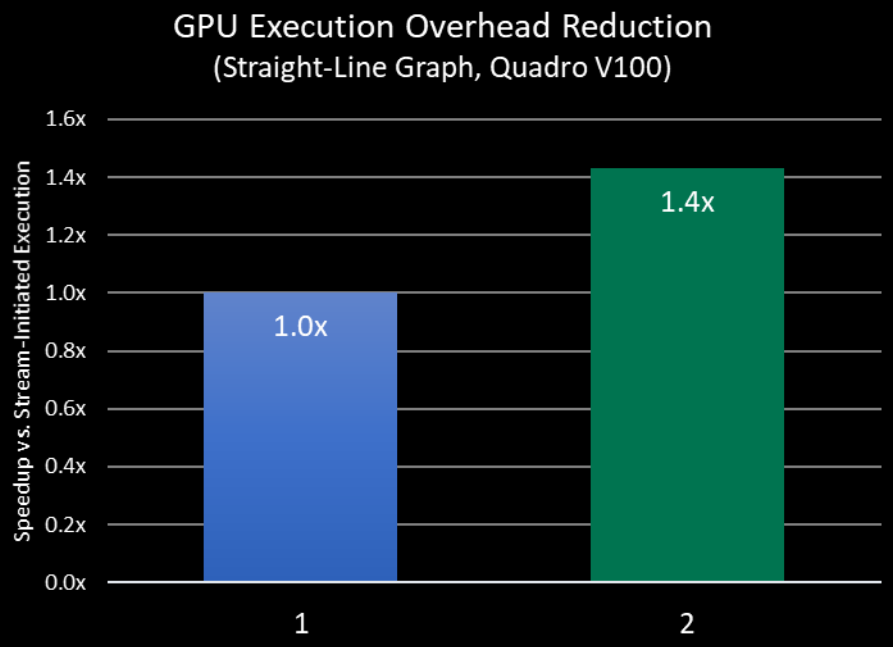


LAUNCH & EXECUTION SPEEDUP

Note: Reduction in **System** Overheads – Kernel Runtime is **Not** Affected



Launch of an already-created graph is **7-8x faster** than launching the same kernels into a stream



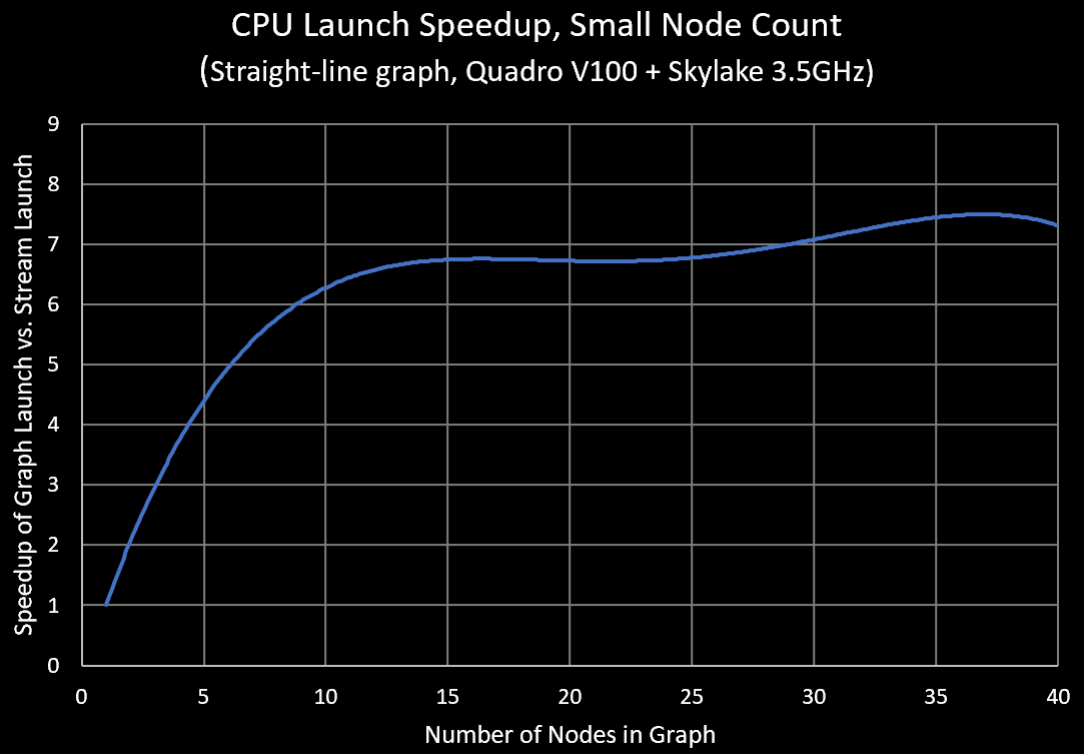
GPU overhead when running kernels is **1.4x lower** than equivalent work in a stream

SMALL-GRAPH PERFORMANCE

Speedup Decreases For Graphs Of <15 Nodes

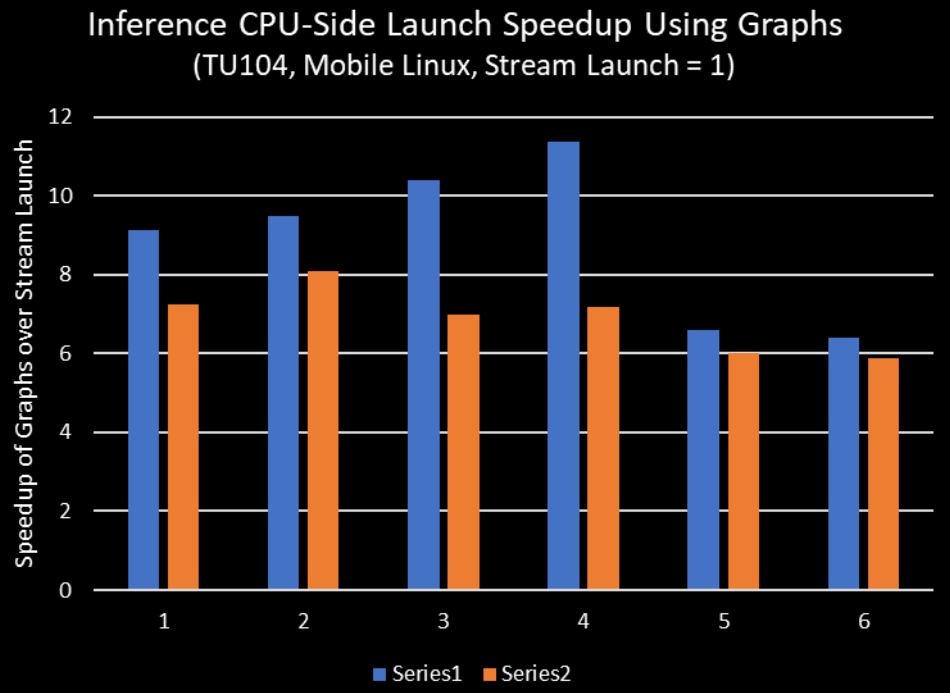
Fixed CPU/GPU transaction cost

- Is paid once for graph launch
- Is paid every kernel for streams
- Becomes insignificant when graph exceeds ~15 nodes

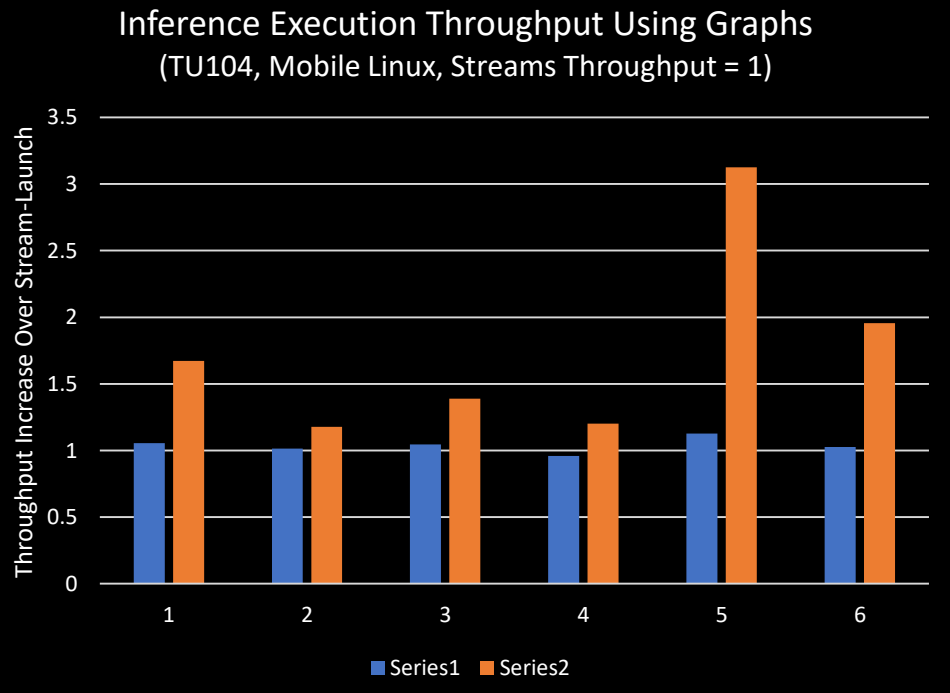


MOBILE INFERENCE

Embedded System Inference Benchmarks (Turing TU104 GPU)



Embedded system **launch** times improve up to 11x

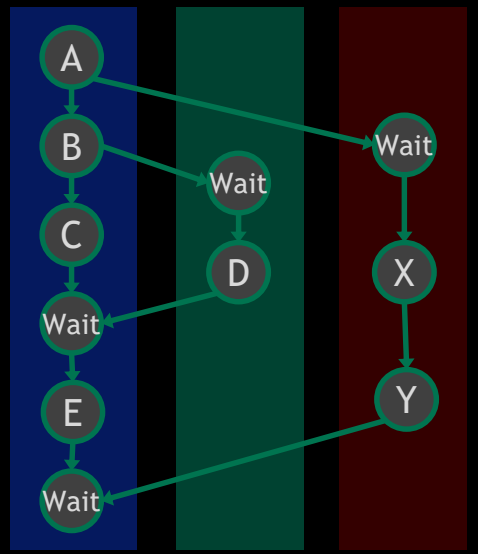


Embedded system **execution** times improve up to 3x

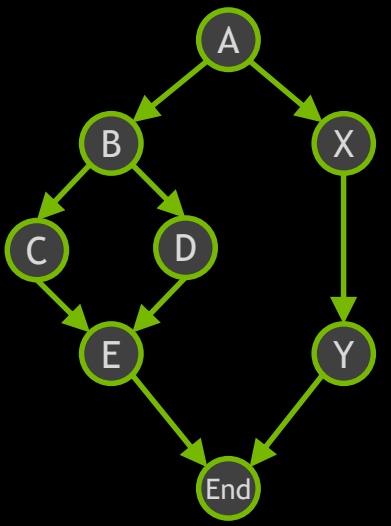
CREATING AND USING GRAPHS

All CUDA Stream Work Already Forms A Graph

CUDA Work in Streams



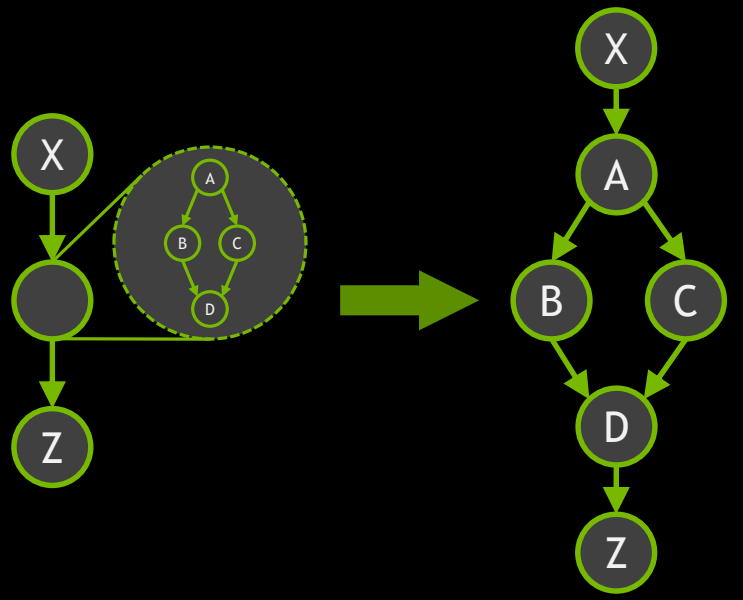
Graph of Dependencies



CAPTURE STREAM WORK INTO A GRAPH

Create A Graph With Two Lines Of Code

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

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

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

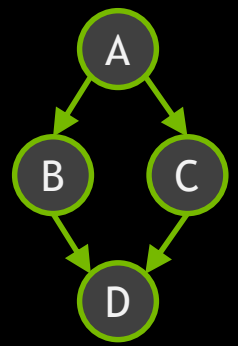


Launches by library also build graph

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

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

// Launch executable graph 1000 times
for(int i=0; i<1000; i++)
     cudaGraphLaunch(instance, stream);
```

FOR IN-DEPTH INFORMATION

See These Sessions This Week

S9956 – Best Practices When Benchmarkinig CUDA Applications, Wednesday 2-3pm

S9957 – Using CUDA on Windows, Wednesday 3-4pm

S9241 – All You Need To Know About Programming NVIDIA's DGX-2, Wednesday 1-2pm

S9329 – Synchronization Is Bad, But If You Must..., Thursday 9-10am

S9681 – Visualize Your Large Datasets, Wednesday 9-10am

S9768 – New Features in OptiX 6.0, Wednesday 1-2pm

NVCC ENHANCEMENTS

Improving Efficiency

Warp Matrix Functions (new C++ namespace)

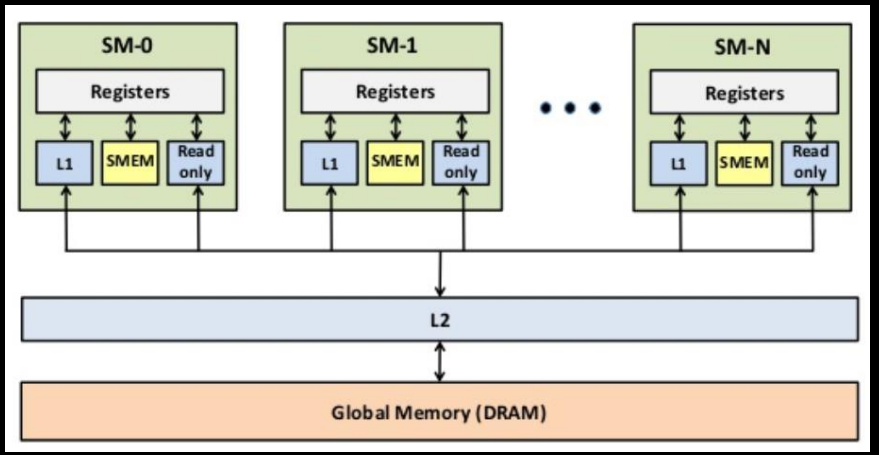
Extensible Whole Program (-ewp) mode compilation support

- Efficient compilation with use of CUDA run-time device library & with Cooperative Groups grid/multi-grid synchronization

New address predicate functions

`__isShared, __isConstant, __isLocal`

Ongoing C++17 language support



Efficient Code Generation for Chip Architecture

ENHANCED HALF-PRECISION FUNCTIONALITY

Includes Limited *half* Type Support For CPU Code

Half-precision atomic ADD (Volta+) (round-to-nearest mode)

```
half atomicAdd(half *address, half val);
half2 atomicAdd(half2 *address, half2 val);
```

Host-side conversion operators between *float* and *half* types

```
half pi = 3.1415f;              // Convert float to half
float fPI = (float)hPI;         // Convert half to float
```

Host-side construction and assignment operators for *half* and *half2* types

```
half pi = 3.1415f;
half also_pi = pi;              // Assign half to half
half2 vector_pi(pi, also_pi);   // Construct half2 from half
```

NOTE: Half-precision *arithmetic* operations remain only available in device code

DIRECTIVE-BASED HPC PROGRAMMING

Who's Using OpenACC?

3 OF TOP 5 HPC APPS

Intersect360 RESEARCH

5 OF 13 CAAR CODES

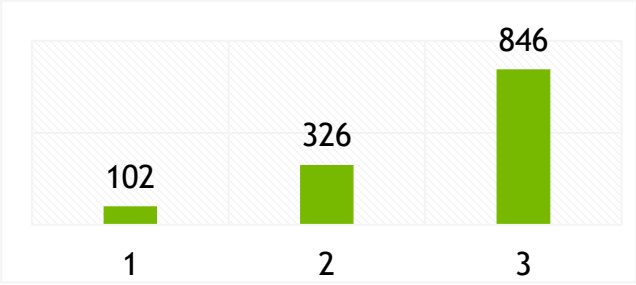
summit

ACCELERATED APPS

725 TRAINED EXPERTS

OpenACC HACKATHON
Advancing High Performance Computing

SLACK MEMBERS



160,000+ DOWNLOADS

PGI®
Community EDITION

PGI | 19.1

Fortran, C and C++ for the Tesla Platform

- CUDA Fortran Tensor Core Support
- OpenACC printf()
- OpenACC Deep Copy
- OpenACC Auto-compare
- OpenACC C++ Lambda
- CUDA 10.x support
- Full C++17 language
- OpenMP 4.5 for CPUs
- PGI in the Cloud

pgicompilers.com/whats-new

THE FUTURE OF GPU PROGRAMMING

Standard Languages | Directives | CUDA

```
std::for_each_n(POL, idx(0), n,
                [=](Index_t i){
    y[i] += a*x[i];
});
```

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

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

...

std::for_each_n(POL, idx(0), n,
                [=](Index_t i){
    y[i] += a*x[i];
});

...

}
```

```
__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, ...);
```

GPU Accelerated C++17 and Fortran 2018

Incremental Performance Optimization with OpenACC

Maximize GPU Performance with CUDA C++/Fortran

PGI SESSIONS AT GTC

S9279 - **OpenACC Programming Model — User Stories, Vendor Reaction, Relevance, and Roadmap**

with Duncan Poole and Michael Wolfe, Tuesday at 4:00 in room 210F

S9770 - **C++17 Parallel Algorithms for NVIDIA GPUs with PGI C++**

by David Olsen, Wednesday at 10:00 in room 210G

S9289 - **PGI Compilers, The NVIDIA HPC SDK: Updates for 2019**

by Michael Wolfe, Thursday at 10:00 in room 211A

SANITIZER: CODE ANALYSIS

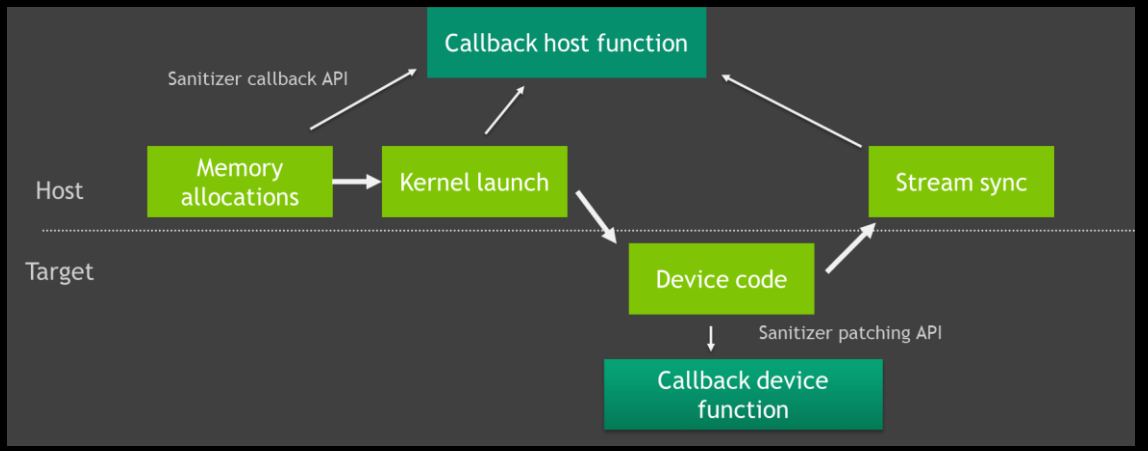
New APIs in CUDA 10.1

Tracks API calls and memory accesses during CUDA kernel execution

Support for Windows, Linux, Mac

Samples available on GitHub

https://github.com/NVIDIA/compute-sanitizer-samples



S9751 - Accelerate Your CUDA Development with Latest Debugging and Code Analysis Developer Tools

NSIGHT SYSTEMS

System-Wide Performance Analysis

Observe Application Behavior: CPU threads, GPU traces, Memory Bandwidth and more

Locate Optimization Opportunities: CUDA & OpenGL APIs, Unified Memory transfers, User Annotations using NVTX

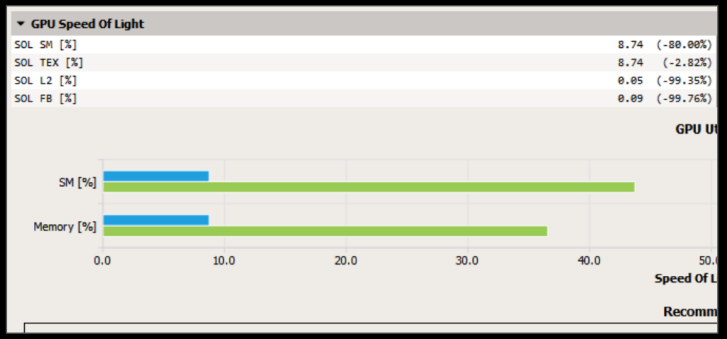
Ready for Big Data: Fast GUI capable of visualizing in excess of 10 million events on laptops, Container support, Minimum user privileges

https://developer.nvidia.com/nsight-systems

NVIDIA NSIGHT COMPUTE

Next Generation Kernel Profiler

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Improved Workflow and Fully Customizable (Baselining, Programmable UI/Rules)
- Command Line, Standalone, IDE Integration
- Platform Support
  - OS: Linux (x86, ARM), Windows
  - GPUs: Pascal, Volta, Turing



Kernel Profile Comparisons with Baseline

inst_executed [inst]	16,528.00; 16,528.00; …	13,476.00; 13,476.00; …
l1tex__sol_pct [%]	14.33	n/a
launch__block_size	128.00	128.00
launch__function_pcs	47,611,587,968.00	12,273,728.00
launch__grid_size	4,132.00	3,369.00
launch__occupancy_limit_blocks [block]	32.00	32.00
launch__occupancy_limit_registers [register]	21.00	21.00
launch__occupancy_limit_shared_mem [bytes]	384.00	384.00
launch__occupancy_limit_warps [warps]	16.00	16.00
launch__occupancy_per_block_size	3,638.00	3,638.00
launch__occupancy_per_register_count	5,792.00	5,792.00
launch__occupancy_per_shared_mem_size	2,260.00	2,260.00
launch__registers_per_thread [register/thread]	17.00	17.00
launch__shared_mem_config_size [bytes]	49,152.00	49,152.00
launch__shared_mem_per_block_dynamic [bytes/block]	0.00	0.00
launch__shared_mem_per_block_static [bytes/block]	20.00	20.00
launch__thread_count [thread]	528,896.00	431,232.00
launch__waves_per_multiprocessor	3.23	42.11
ltc__sol_pct [%]	6.93	7.18
memory_access_size_type [bytes]	2.00; 32.00; 32.00; 32…	2.00; 32.00; 32.00; 32…

Metric Data

Source	Live Registers	Sampling Data (All)	Sampling Data (No Issue)
@!PT SHFL.IDX PT, RZ, RZ, RZ, RZ;	0	223	0
MOV R1, c[0x0][0x28];	1	13	44
S2R R0, SR_CTAID.X;	2	143	75
S2R R2, SR_TID.X;	3	0	38
IMAD R0, R0, c[0x0][0x0], R2;	3	599	94
ISETP.GE.AND P0, PT, R0, c[0x0][0x170]	2	125	26
@P0 EXIT;	2	259	86
MOV R2, R0;	3	386	29
@!PT SHFL.IDX PT, RZ, RZ, RZ, RZ;	2	0	0
MOV R4, 0x4;	3	0	0
IMAD.WIDE R4, R2, R4, c[0x0][0x160];	4	0	0
LDG.E.SYS R3, [R4];	3	0	0
BSSY B0, 0xb00976780;	3	0	0
SHF.R.S32.HI R0, RZ, 0x1f, R2;	4	0	0

Source Correlation

TOOLS SESSIONS AT GTC

Talks

S9503 - Using Nsight Tools to Optimize the NAMD Molecular Dynamics Simulation Program

S9345 - CUDA Kernel Profiling using NVIDIA Nsight Compute

S9751 - Accelerate Your CUDA Development with Latest Debugging and Code Analysis Developer Tools

S9661 - Nsight Graphics - DXR/Vulkan Profiling/Vulkan Raytracing

Connect with the Experts

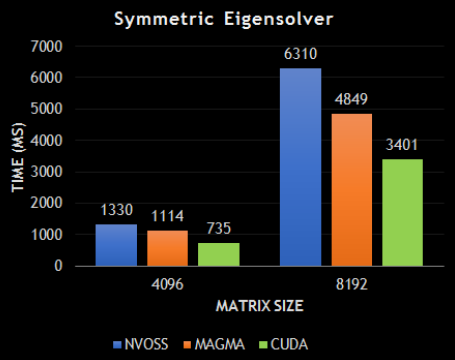
CE9123 - Connect with Experts: CUDA & Graphics Developer Tools

CE9137 - Connect with Jetson Embedded Platform Experts

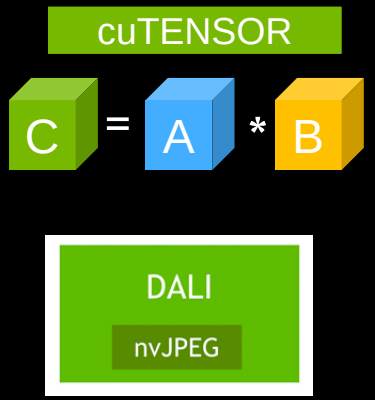
Devtools pod at NVIDIA booth on exhibition show-floor

CUDA MATH LIBRARIES

Major Initiatives

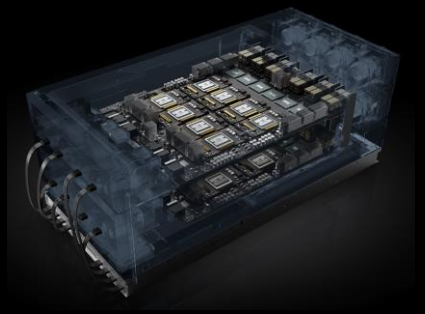


Performance
Tuning + new algorithms



Extended Features
New libraries & APIs

Functional Safety
Drive AV SW Stack



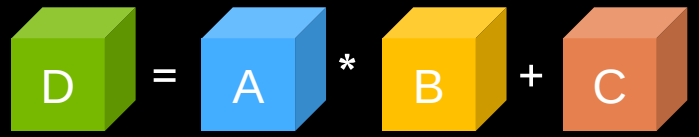
Multi-GPU
Strong/weak scaling

Single GPU
TC & low/mixed precision

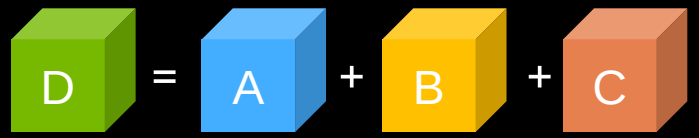
cuTENSOR

A New High-Performance CUDA Library for Tensor Primitives

Tensor Contractions



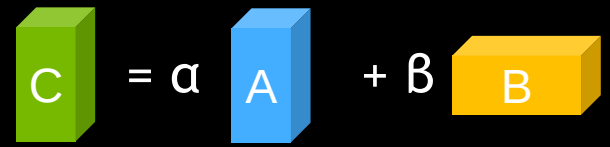
Elementwise operations

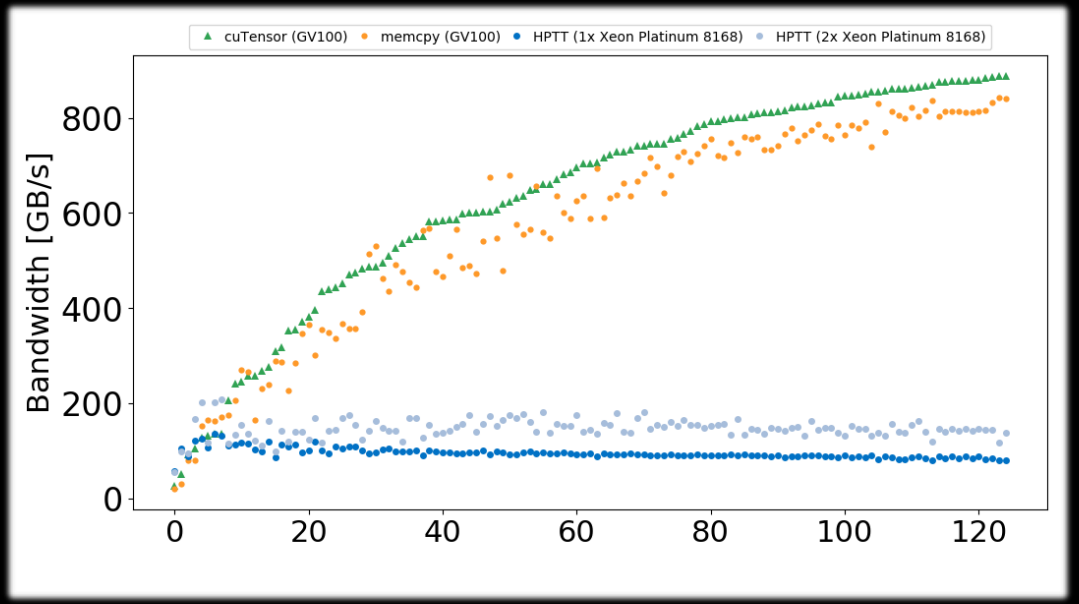


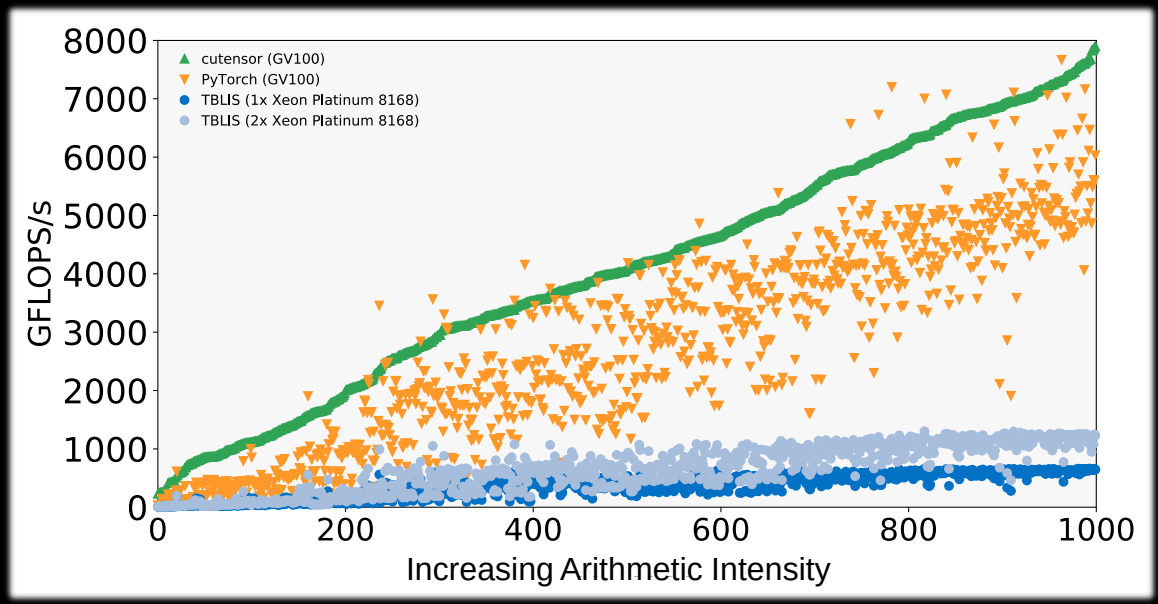
Pre-release version available

```
cutensorStatus_t cutensorCreateTensorDescriptor ( cutensorTensorDescriptor_t* desc,
                                                  unsigned int numModes,
                                                  const int64_t *extent,
                                                  const int64_t *stride,
                                                  cudaDataType_t dataType,
                                                  cutensorOperator_t unaryOp );


cutensorStatus_t cutensorContraction (cuTensorHandle_t handle,
      const void* alpha, const void *A, const cutensorTensorDescriptor *descA, const int modeA[],
                         const void *B, const cutensorTensorDescriptor *descB, const int modeB[],
      const void* beta,  const void *C, const cutensorTensorDescriptor *descC, const int modeC[],
                         void *D, const cutensorTensorDescriptor *descD, const int modeD[],
      cutensorOperator_t opOut, cudaDataType_t typeCompute, cutensorAlgo_t algo,
      void* workspace, size_t workspaceSize, cudaStream_t stream );


cutensorStatus_t cutensorElementwiseTrinary ( cuTensorHandle_t handle,
      const void* alpha, const void *A, const cutensorTensorDescripto r *descA, const int modeA[],
      const void* beta,  const void *B, const cutensorTensorDescriptor *descB, const int modeB[],
      const void* beta,  const void *C, const cutensorTensorDescriptor *descC, const int modeC[],
                         void *D, const cutensorTensorDescriptor *descD, const int modeD[],
      cutensorOperator_t opAB, cutensorOperator_t opABC, cudaDataType_t typeCompute, cudaStream_t stream );
```

CUTENSOR







Tensor transpositions: NCHW -> NHWC

Random tensor contractions: 3D to 6D tensors, increasing arithmetic Intensity

HPTT (https://github.com/springer13/hptt) TBLIS (https://github.com/devinamatthews/tblis)

cuBLASLt

New MATMUL Library with Full Algorithm Control

New header and binary with lightweight context

Targets power GEMM users

- Not a replacement for cuBLAS

Increased flexibility

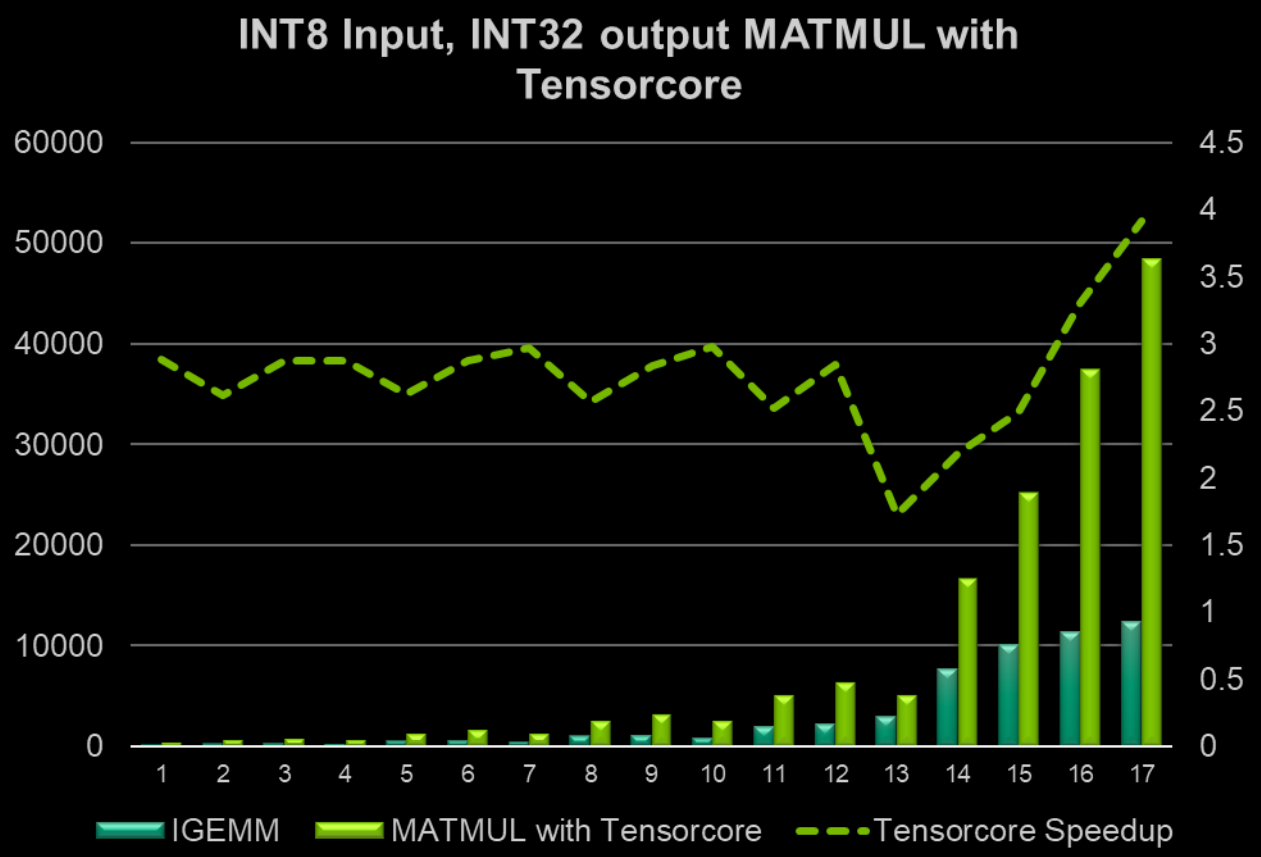
- Data layout
- Input and Compute types
- Algorithm choice and heuristics

Workspace enables new algorithms

Layout flexibility enables hardware optimization

```
#include <cublasLt.h>

cublasLtCreate()
cublasLtMatmul()
cublasLtMatmulAlgoGetHeuristic()
cublasLtMatmulAlgoConfigSetAttribute()
```

cuBLASLt



computeType	scaleType	Atype/Btype	Ctype
CUDA_R_16F	CUDA_R_16F	CUDA_R_16F	CUDA_R_16F
CUDA_R_32I	CUDA_R_32I	CUDA_R_8I	CUDA_R_32I
	CUDA_R_32F	CUDA_R_8I	CUDA_R_32I
CUDA_R_32F	CUDA_R_32F	CUDA_R_16F	CUDA_R_16F
		CUDA_R_8I	CUDA_R_32F
		CUDA_R_16F	CUDA_R_32F
		CUDA_R_32F	CUDA_R_32F
CUDA_R_64F	CUDA_R_64F	CUDA_R_64F	CUDA_R_64F
CUDA_C_32F	CUDA_C_32F	CUDA_C_8I	CUDA_C_32F
		CUDA_C_16F	CUDA_C_32F
		CUDA_C_32F	CUDA_C_32F
CUDA_C_64F	CUDA_C_64F	CUDA_C_64F	CUDA_C_64F

Average 2.8X, up to 3.9X Speedup with cuBLASLt Turing IMMA Support

cuFFTDx

New Library: cuFFT Device EXtention

Motivation

Performance

- FFTs are memory bound
- CPU issued commands → PCIe latency

Size

- Entire library required for single size use

Customization

- cuFFT launches own kernels
- No opportunity to inline

Key Features

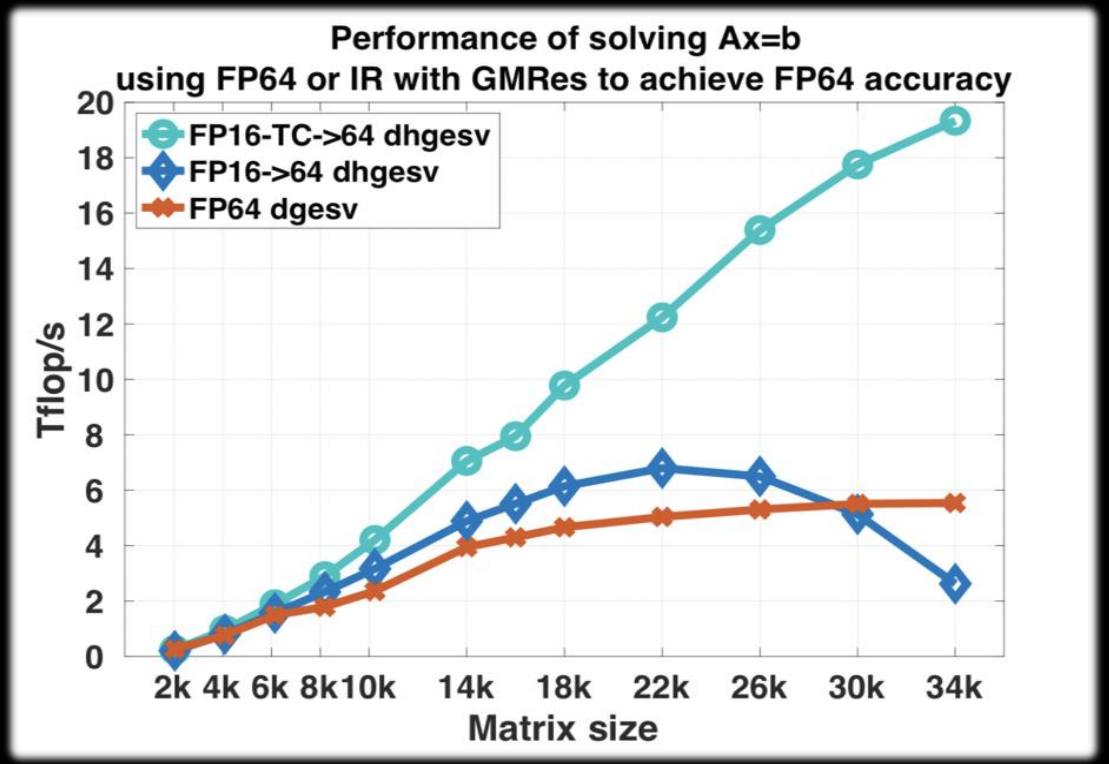
- Device callable library
- Retain and reuse on-chip data
- Inline FFTs in user kernel
- Combine FFT operations

When

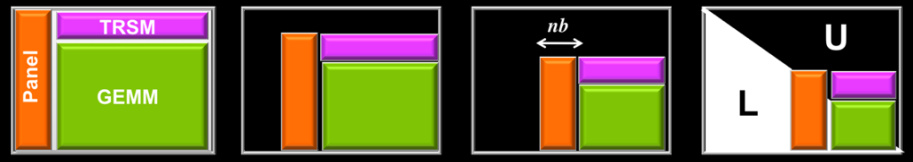
Initial release mid 2019

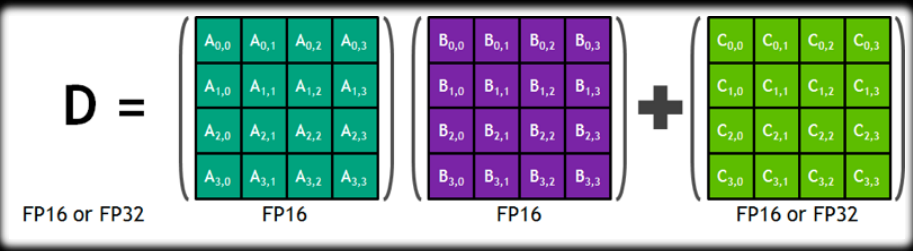
cuSOLVER

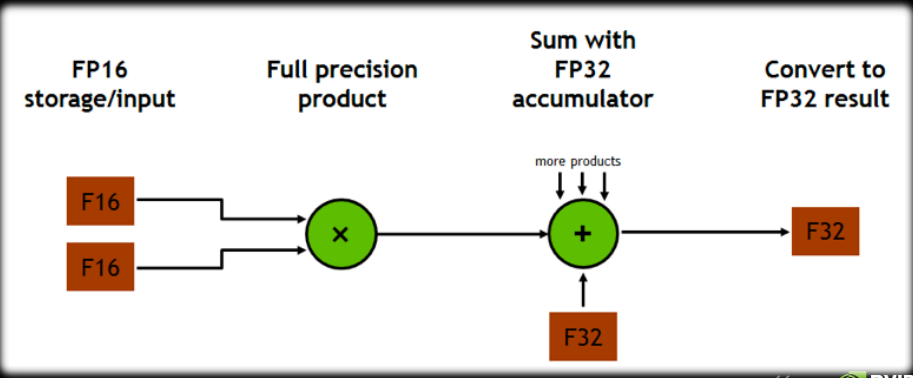
Tensor Core Accelerated Dense Linear Solver Coming Soon



Results obtained on GV100 using MAGMA



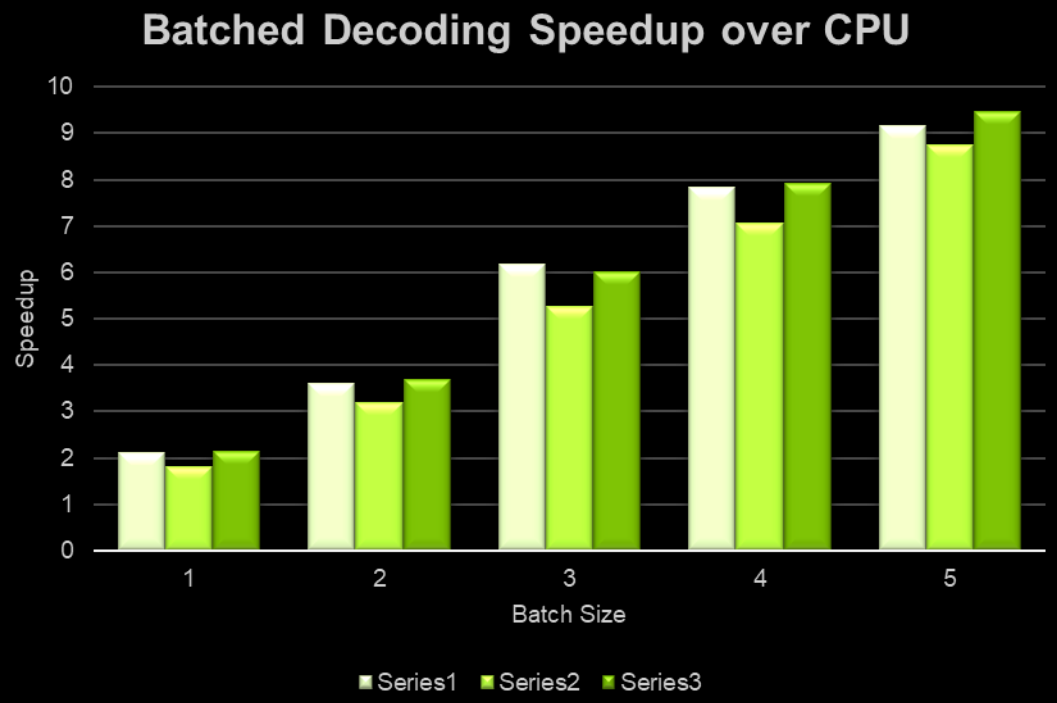




nvJPEG

New Features

- Batched Decoding
- Baseline Encoding
- Device and pinned memory control
- Linux-Power ppc64le platform support
- JPEG stream parsing
- Hybrid decode API
- ROI decoding



GPU Results obtained on GV100
CPU Results obtained with TJPEG on 2-socket Intel Xeon Gold 6140

CUDA LIBRARIES SESSIONS AT GTC

Come learn more about CUDA Libraries

S9593 - cuTENSOR: High-performance Tensor Operations in CUDA, Wednesday March 20, 1-2PM

S9226 - Fast Singular Value Decomposition on GPUs, Wednesday March 20, 2-3PM

CWE 9114 - Connect with the Experts: CUDA Libraries, Wednesday March 20, 5-6PM

S9257 - New FFT Library with Flexible C++ API, Thursday March 21, 3-4PM

TESLA UNIVERSAL ACCELERATION PLATFORM

Single Platform To Drive Utilization and Productivity

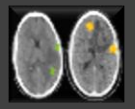
CUSTOMER USECASES


Speech


Translate


Recommender

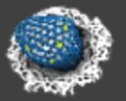
CONSUMER INTERNET

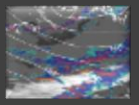

Healthcare

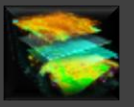

Manufacturing


Finance

INDUSTRIAL APPLICATIONS


Molecular Simulations


Weather Forecasting


Seismic Mapping

SUPERCOMPUTING

APPS & FRAMEWORKS

python

TensorFlow mxnet ONNX
PYTORCH RAPIDS KALDI Chainer

Amber ANSYS +550 Applications
NAMD SIMULIA

NVIDIA SDK & LIBRARIES

MACHINE LEARNING | RAPIDS
cuDF | cuML | cuGRAPH

DEEP LEARNING
cuDNN | cuBLAS | CUTLASS | NCCL | TensorRT

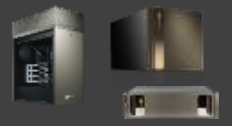
SUPERCOMPUTING
CuBLAS | CuFFT | OpenACC

CUDA

TESLA GPUs & SYSTEMS


TESLA GPU


VIRTUAL GPU


NVIDIA DGX FAMILY


NVIDIA HGX

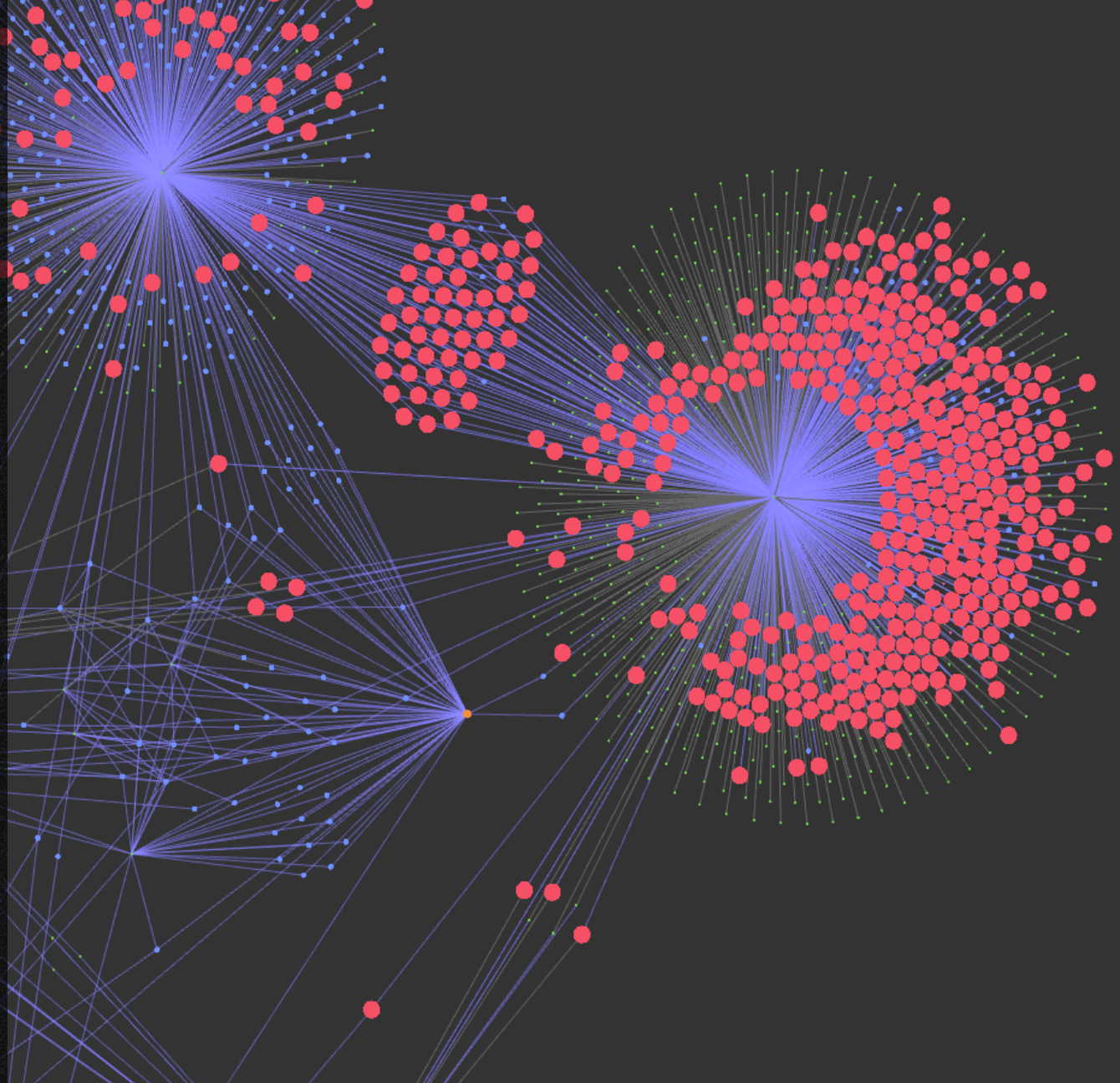
DELL Hewlett Packard Enterprise IBM
SYSTEM OEM

aws Google Cloud Platform Microsoft Azure
CLOUD

ACCELERATING DISCOVERIES WITH AI

New drugs typically take 12-14 years and $2.6 billion to bring to market. BenevolentAI is using GPU deep learning to bring new therapies to market quickly and more affordably. They've automated the process of identifying patterns within large amounts of research data, enabling scientists to form hypotheses and draw conclusions quicker than any human researcher could. And using the NVIDIA DGX-1 AI supercomputer, they identified two potential drug targets for Alzheimer's in less than one month.

benevolent.ai

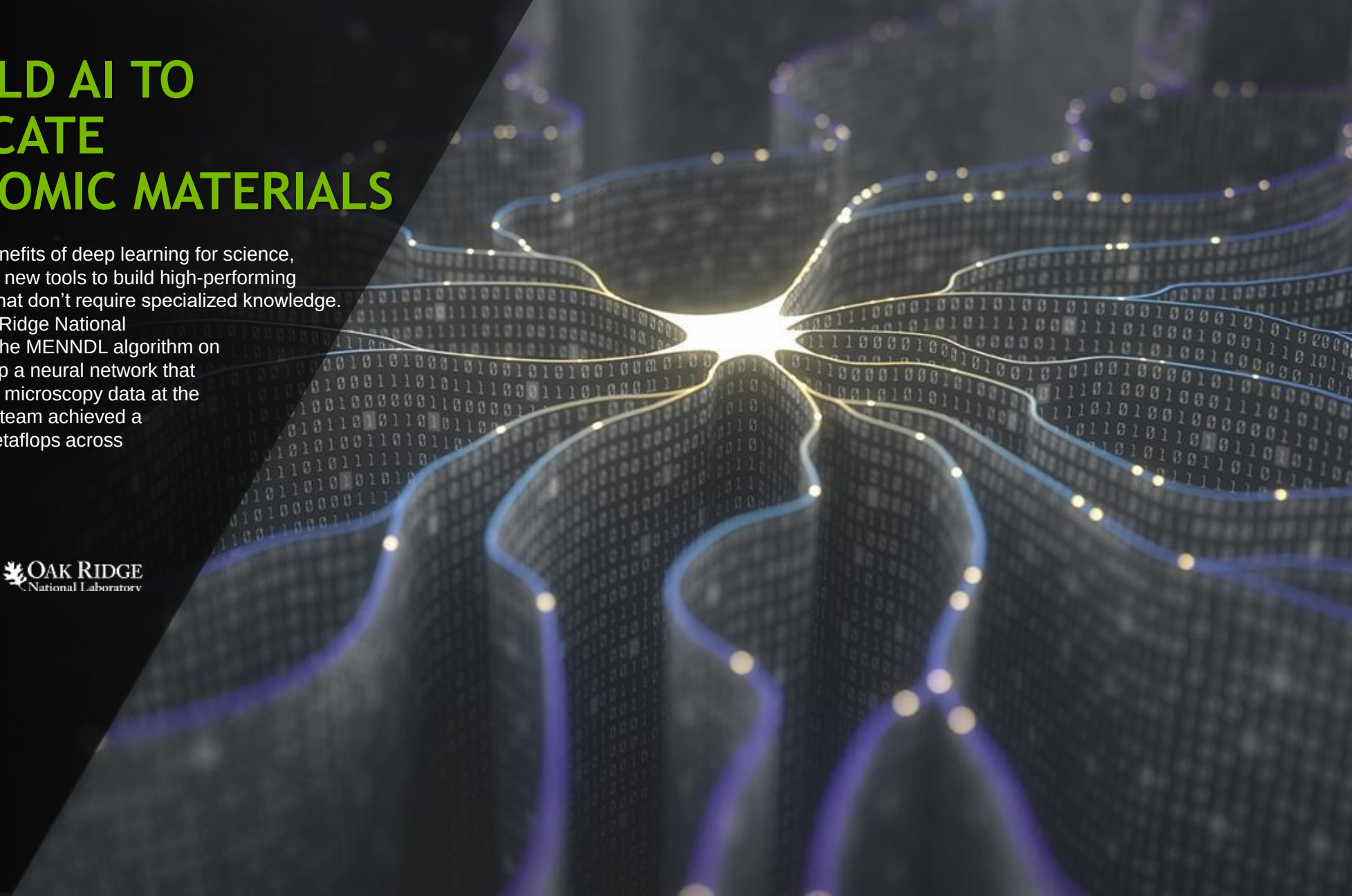


AI-BUILD AI TO FABRICATE SUBATOMIC MATERIALS

To expand the benefits of deep learning for science, researchers need new tools to build high-performing neural networks that don’t require specialized knowledge. Scientists at Oak Ridge National Laboratory used the MENNDL algorithm on Summit to develop a neural network that analyzes electron microscopy data at the atomic level. The team achieved a speed of 152.5 petaflops across 3,000 nodes.

NVIDIA

OAK RIDGE National Laboratory



A 21st CENTURY PLANNING TOOL BUILT ON AI

With the Earth's population at 7 billion and growing, understanding population distribution is essential to meeting societal needs for infrastructure, resources and vital services. Using GPUs and deep learning, Oak Ridge National Laboratory can quickly process high-resolution satellite imagery to map human settlements and changing urban dynamics. With the ability to process a major city in minutes, ORNL can provide emergency response teams critical information that used to take days to create.

OAK RIDGE National Laboratory

“SEEING” GRAVITY IN REAL-TIME

In 2015 gravitational waves (GW) were observed for the first time by astronomers at the Laser Interferometer Gravitational-wave Observatory (LIGO) originating from a pair of merging Black Holes 1.3B light years away. “Seeing” gravity opens the door to new discoveries and a daunting new challenge: observing GW in parallel with electromagnetic waves, and analyzing the combined data in real-time.

Scientists at NCSA are using GPU-powered deep learning to make this computationally intensive approach possible. Using a deep Convolutional Neural Network (CNN), NCSA trained its system to process gravitational wave data more than 5000 times faster than its previous machine learning methods — making real time analysis possible and putting us one step closer to understanding the universe’s oldest secrets.

Physics Letters B - Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced LIGO data
Daniel George, E.A. Huerta