CUDA: NEW AND UPCOMING FEATURES
Stephen Jones, GTC 2018
CUDA ECOSYSTEM 2018

CUDA DOWNLOADS IN 2017
3,500,000

CUDA REGISTERED DEVELOPERS
800,000

GTC ATTENDEES
8,000+
CUDA DEVELOPMENT ECOSYSTEM
From Ease of Use to Specialized Performance

Applications | Frameworks | Libraries | Directives and Standard Languages | Specialized Languages

CUDA-C++ | CUDA Fortran

cuDNN

OpenACC

cuBLAS

cuEMMIE

cuFFT

cuSPLITE

cuPDEV

cuHDL

cuDNN

cuASAP

cuBLAS

cuEMMIE

cuFFT

cuSPLITE

cuPDEV

cuHDL

GROMACS

NAMD

FLUENT

ANSYS
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)

**NEW ALGORITHMS**

Mixed-precision Batched GEMM for attention models (cuBLAS)

Image Augmentation and batched image processing routines (NPP)

Batched pentadiagonal solver (cuSPARSE)

**PERFORMANCE**

GEMM optimizations for RNNs (cuBLAS)

Faster image processing (NPP)

Prime factor FFT performance (cuFFT)

SpMV performance (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

The following chart shows a summary view of the memory hierarchy of the CUDA programming model. The green nodes in the diagram depict logical memory space whereas blue nodes depict actual hardware unit on the chip. For the various caches the reported percentage numbers state the cache hit rate; that is the ratio of requests that could be served with data locally available to the cache over all requests made.

The links between the nodes in the diagram depict the data paths between the SMs to the memory spaces in the memory system. Different metrics are shown per data path. The data paths from the SMs to the memory spaces report the total number of memory instructions executed, it includes both read and write operations. The data path between memory spaces and "Unified Cache" or "Shared Memory" reports the total amount of memory requests made (read or write). All other data paths report the total amount of transferred memory in bytes.
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

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

```c
__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)

$$D = \begin{pmatrix} \text{Dmat} & \text{Amat} & \text{Bmat} & \text{Cmat} \end{pmatrix}$$

FP16 or FP32
FP16
FP16
FP16 or FP32

$$D = AB + C$$
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 = D 32x8 + A 32x16 + B 16x8 + C 32x8

WMMA 8x32x16 = D 8x32 + A 8x16 + B 16x32 + C 8x32
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

```c
#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
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];
      ...
    }
  }
  ...
}
```

CUDA Unified Memory

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

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];
    ...
  }
}
...

C malloc, C++ new, Fortran allocate all mapped to CUDA Unified Memory
GYROKINETIC TOROIDAL CODE
Being ported for runs on the ORNL Summit supercomputer

The Gyrokinetic Toroidal Code (GTC)

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming Models</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Platforms</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>X86, OpenPOWER, NVIDIA GPU</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>1-2 times a year</td>
<td>6-9 times a year</td>
<td>6-9 times a year</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>User Forums</td>
<td>PGI Support</td>
<td>PGI Premier Services</td>
</tr>
<tr>
<td><strong>License</strong></td>
<td>Annual</td>
<td>Perpetual</td>
<td>Volume/Site</td>
</tr>
</tbody>
</table>

### Additional Information
- **FREE**
- **Update频率**:
  - 1-2 times a year
  - 6-9 times a year
- **Support Options**:
  - User Forums
  - PGI Support
  - PGI Premier Services
- **License Options**:
  - 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 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

GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
GPU 5
GPU 6
GPU 7
GPU 8
GPU 9
GPU 10
GPU 11
GPU 12
GPU 13
GPU 14
GPU 15

NVSwitch Fabric
FULL 6-WAY POINT-TO-POINT

NVSwitch Fabric
LOAD & STORE TO ANY GPU

NVSwitch Fabric
LOAD & STORE TO ANY GPU

NVSwitch Fabric
LOAD & STORE TO ANY GPU
16-WAY ALL-REDUCE PERFORMANCE
16-WAY ALL-REDUCE PERFORMANCE

8x smaller packet size with the same performance
16-WAY ALL-REDUCE PERFORMANCE

8x smaller packet size with the same performance

4x higher performance for a given packet size
2X HIGHER PERFORMANCE WITH NVSWITCH

2X FASTER

Physics (MILC benchmark)  
4D Grid

2.4X FASTER

Weather (ECMWF benchmark)  
All-to-all

2X FASTER

Recommender (Sparse Embedding)  
Reduce & Broadcast

2.7X FASTER

Language Model (Transformer with MoE)  
All-to-all

2x DGX-1 (Volta)  
DGX-2 with NVSwitch
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

Graph of Dependencies

All CUDA streams can be mapped to a graph
GRAPHS CARRY RICHER INFORMATION

CUDA Work in Streams

Graph of Dependencies

Inline dependencies
Based on order that work is submitted

Explicit dependencies
Defined when graph is created
// Start by initating stream capture
cudaStreamBeginCapture(&stream);

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

// Now convert the stream to a graph
cudaStreamEndCapture(stream, &graph);
CREATE GRAPHS DIRECTLY
Map Graph-Based Workflows Directly Into CUDA

```
// 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

(input) → 3x3 convolution → ReLU → 5x5 convolution → ReLU → 1x1 convolution → ReLU → max pool → ReLU → concat

(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

CUDA 9.2

Beyond