CUDA 9.2 UPDATE
OLCF User Group Call, 4/25/2018

NVIDIA
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
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
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
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)
CUDA TENSOR CORE PROGRAMMING
16x16x16 Warp Matrix Multiply and Accumulate (WMMA)

\[ D = \text{wmma::mma_sync}(D\text{mat}, A\text{mat}, B\text{mat}, C\text{mat}); \]

\[ D = AB + C \]
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

\[
\begin{align*}
D & : 32x8 \\
A & : 32x16 \\
B & : 16x8 \\
C & : 32x8
\end{align*}
\]

=  

WMMA 8x32x16

\[
\begin{align*}
D & : 8x32 \\
A & : 8x16 \\
B & : 16x32 \\
C & : 8x32
\end{align*}
\]

=  

\[
\begin{align*}
\text{WMMA 32x16} \\
\text{WMMA 8x16}
\end{align*}
\]
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 number states 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 into 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
UNIFIED MEMORY WITH ATS ON IBM POWER9
IBM Power9 CPUs With NVLink Interconnect

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

**ACCESS**
All data accessible concurrently from any processor, anytime
Atomic operations resolved directly over NVLink
UNIFIED MEMORY WITH ATS ON IBM POWER9
IBM Power9 CPUs With NVLink Interconnect

ATS & POWER9 FEATURES

ATS allows GPUDirect RDMA to unified memory

Managed memory is cache-coherent between CPU & GPU

CPU has direct access to GPU memory without need for migration
WHAT YOU CAN DO WITH UNIFIED MEMORY

Works everywhere today

```c
int *data;
cudaMallocManaged(&data, sizeof(int) * n);
kernel<<< grid, block >>>(data);
```

Works on POWER9 + CUDA 9.2

```c
int *data = (int*)malloc(sizeof(int) * n);
kernelpart<<< grid, block >>>(data);

int data[1024];
kernelpart<<< grid, block >>>(data);

int *data = (int*)alloca(sizeof(int) * n);
kernelpart<<< grid, block >>>(data);

extern int *data;
kernelpart<<< grid, block >>>(data);
```
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