ANNOUNCING CUDA 10.2
Download today at: https://developer.nvidia.com/cuda-downloads

Standard Language Support

Task Graph Enhancements

Low-Level Memory Management

Plus Compiler, Tools and Library Enhancements & Performance Improvements

ACCELERATED COMPUTING IS FULL-STACK OPTIMIZATION

2X More Performance With Software Optimizations Alone

HPC Applications Speedup

- CUDA 8
- CUBLAS 8
- CUFFT 8
- CUDA 10
- CUBLAS 10
- CUFFT 10

2X More Performance With Software Optimizations Alone

- 2x Broadwell vs 4xP100
- 2x Broadwell vs 4xV100

HPC Apps: AMBER, Chroma, GROMACS, GTC, LAMMPS, MILC, NAMD, QE, RTM, SPECFEM3D, VASP
# CUDA 10.2 PLATFORM SUPPORT

## New OS and Host Compilers

<table>
<thead>
<tr>
<th>PLATFORM</th>
<th>OS</th>
<th>VERSION</th>
<th>COMPILERS</th>
</tr>
</thead>
</table>
| Linux      | ![Ubuntu](https://weblate.org/icons/Ubuntu.svg) | 18.04.3 LTS  
16.04.6 LTS | GCC 8.x  
PGI 19.x  
Clang 7.0.x  
Clang 8  
ICC 19  
XLC 16.1.x (POWER) |
|            | ![CentOS](https://weblate.org/icons/CentOS.svg) | CentOS 7.7  
RHEL 8.1 | |
|            | ![SUSE](https://weblate.org/icons/SUSE.svg) | SLES 15.1  
SLES 12.4 | |
|            | ![Fedora](https://weblate.org/icons/Fedora.svg) | Fedora 29 | |
|            | ![SUSE](https://weblate.org/icons/SUSE.svg) | Leap 15.x | |
| Windows    | ![Windows](https://weblate.org/icons/Windows.svg) | Windows Server  
2019  
2016  
2012 R2 | Microsoft Visual Studio 2017 (15.x)  
Microsoft Visual Studio 2019 |
| Mac        | macOS   | 10.13.6                  | Xcode 10.2 |

NVIDIA
NVIDIA ACCELERATES HPC APPLICATIONS ON ARM

For more information: https://developer.nvidia.com/cuda-toolkit/arm

Benchmark Application [Dataset]: GROMACS [ADHDodeca-Dev Sandbox], LAMMPS [LJ 2.5], MILC [Apex Medium], NAMD [apoA1_npt_cuda], Quantum Espresso [AUSurf12-JR], Relion [Plasmodium Ribosome], SPEC.FEM3D [four_material_simple_model];

CPU node: 2x ThunderX2 9975; GPU node: 2x ThunderX2 + 2x V100 32GB PCIe. GROMACS and MILC data is for 1x V100 32GB PCIe.
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

Full compatibility matrix: https://docs.nvidia.com/deploy/cuda-compatibility/index.html

<table>
<thead>
<tr>
<th>Driver Branch</th>
<th>CUDA 10 Compatible</th>
<th>CUDA 10.2 Compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA 9.0</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CUDA 9.1</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CUDA 9.2</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CUDA 10.0</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>CUDA 10.1</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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
GRAPH UPDATE: MUTABLE PARAMETERS

New In CUDA 10.2

Stream Launch

A
B
C
A
B
C
repeat 10 times

Parameters: may change
Topology: may change

Graph Re-Lauch

A
B
C
Define Graph

launch graph

iterate 10 times

Parameters: may not change
Topology: may not change
GRAPH UPDATE: MUTABLE PARAMETERS

New In CUDA 10.2

Stream Launch

A → B → C (repeat 10 times)

Parameters: may change
Topology: may change

Graph Update

A → B → C

Update Graph

launch graph

? iterate 10 times

Parameters: may change
Topology: may not change

Graph Re-Launch

A → B → C

Define Graph

launch graph

? iterate 10 times

Parameters: may not change
Topology: may not change
GRAPH PERFORMANCE
Performance Gains For Both CPU & GPU

**CPU Launch Time Per Kernel**
(Large node count, Straight-line graph, Quadro GV100 + Skylake 3.5GHz)

- Stream Launch: 2.10 Microseconds
- Graph Relaunch: 0.29 Microseconds
- Update 100% Launch: 0.96 Microseconds
- Update 50% Launch: 0.76 Microseconds
- Update 0% Launch: 0.56 Microseconds

**GPU Per Kernel Execution Overhead**
(Microseconds, Large node count, Straight-line graph)

- Stream Exec Per Kernel: 1.57 Microseconds
- Relaunched Graph Per-Kernel Execution Overhead: 1.11 Microseconds
- Updated Graph Per-Kernel Execution Overhead: 1.20 Microseconds
MEMORY MAPPING & ALLOCATION CONTROL

Break Memory Allocation Into Its Constituent Parts

1. Reserve Virtual Address Range
   cuMemAddressReserve/Free

2. Allocate Physical Memory Pages
   cuMemCreate/Release

3. Map Pages To Virtual Addresses
   cuMemMap/Unmap

4. Manage Access Per-Device
   cuMemSetAccess

New in CUDA 10.2

Control & reserve address ranges
Map physical memory multiple times
Fine-grained access control
Manage inter-GPU peer-to-peer sharing on a per-address basis
Inter-process sharing
MEMORY MAPPING & ALLOCATION CONTROL

Basic Memory Allocation Example

1. Reserve Virtual Address Range
   `cuMemAddressReserve/Free`

2. Allocate Physical Memory Pages
   `cuMemCreate/Release`

3. Map Pages To Virtual Addresses
   `cuMemMap/Unmap`

4. Manage Access Per-Device
   `cuMemSetAccess`

// Allocate memory
`cuMemCreate(handle, size, &allocProps, 0);`

// Reserve address range
`cuMemAddressReserve(ptr, size, alignment, fixedVa, 0);`

// Map memory to address range
`cuMemMap(ptr, size, offset, handle, 0);`

// Make the memory accessible on all devices
`cuMemSetAccess(ptr, size, rwOnAllDevices, deviceCount);`
REMOVAL OF NON-SYNC WARP FUNCTIONS
Functions Deprecated In CUDA 9.0: Now Removed Since CUDA 10.1

<table>
<thead>
<tr>
<th>Removed Function</th>
<th>Replacement Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>__ballot()</td>
<td>__ballot_sync()</td>
</tr>
<tr>
<td>__any()</td>
<td>__any_sync()</td>
</tr>
<tr>
<td>__all()</td>
<td>__all_sync()</td>
</tr>
<tr>
<td>__shfl()</td>
<td>__shfl_sync()</td>
</tr>
<tr>
<td>__shfl_up()</td>
<td>__shfl_up_sync()</td>
</tr>
<tr>
<td>__shfl_down()</td>
<td>__shfl_down_sync()</td>
</tr>
<tr>
<td>__shfl_xor()</td>
<td>__shfl_xor_sync()</td>
</tr>
</tbody>
</table>

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
libcu++
The CUDA C++ Standard Library

ISO C++ == Language + Standard Library
CUDA C++ == Language + libcu++
libcu++
The CUDA C++ Standard Library

ISO C++ == Language + Standard Library
CUDA C++ == Language + libcu++

- Opt-in heterogeneous C++ standard library for CUDA
- Open source; port of LLVM’s libc++; contributing upstream
- **Version 1 (Out Now):** `<atomic>` (Pascal+), `<type_traits>

Future release priorities: C++20 sync library, `<chrono>`, `<complex>`, `<tuple>`, `<array>`, `<utility>`, `<cmath>`, ...
# libcu++

The CUDA C++ Standard Library

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Conforming to</th>
<th>Example</th>
</tr>
</thead>
</table>
| std::     | ISO C++, __host__ only  Provided by your host compiler | `#include <atomic>`  
`std::atomic<int> x;` |
| cuda::std:: | CUDA C++, __host__ __device__  Strictly conforming to ISO C++ | `#include <cuda/std/atomic>`  
`cuda::std::atomic<int> x;` |
| cuda::    | CUDA C++, __host__ __device__  Conforming extensions to ISO C++ | `#include <cuda/atomic>`  
`cuda::atomic<int, cuda::thread_scope_block> x;` |

For in depth information, see booth talk: Bryce Lelbach - The CUDA C++ Standard Library, Thursday @ 2pm
PGI FORTRAN, C AND C++

HPC Compilers for the Tesla Platform

Full C++17 | Fortran 2003/2008
Optimizing, SIMD vectorizing, OpenMP

Accelerated Computing Features
CUDA Fortran, OpenACC directives

Multi-platform solution
x86-64, OpenPOWER and Arm CPUs
NVIDIA Tesla GPUs
Linux, Windows, MacOS

New! Support for Arm and Rome CPUs
**OPENACC COMPILERS ARE FREELY AVAILABLE**

**PGI 19.10 Community Edition**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROGRAMMING MODELS</strong></td>
<td></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>
<td>✓</td>
</tr>
<tr>
<td><strong>PLATFORMS</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>X86, OpenPOWER, NVIDIA GPU</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>UPDATES</strong></td>
<td></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></td>
<td>User Forums</td>
<td>PGI Support</td>
<td>PGI Premier Services</td>
</tr>
<tr>
<td><strong>LICENSE</strong></td>
<td></td>
<td>Annual</td>
<td>Perpetual</td>
<td>Volume/Site</td>
</tr>
</tbody>
</table>

pgicompilers.com/community
THE FUTURE OF HPC GPU PROGRAMMING
Math Libraries | Standard Languages | Directives | CUDA

GPU Programming with C++17
David Olsen, 11:00am Thursday

```
#pragma acc data copy(x,y)
{
    ...
    std::transform(par, x, x+n, y, y,
                   [=](float x, float y) {
                      return y + a*x;
                   });
    ...
}

__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
CUDA FORTRAN TENSOR CORES

module mod1
use params ! Define matrix m, n, k
contains
  attributes(global) subroutine test1(a, b, c)
    use wmma
    real(2), device :: a(m, k), b(k, n)
    real(4), device :: c(m, n)
    WMMASubMatrix(WMMAMatrixA, 16, 16, 16, Real, WMMAColMajor):: sa
    WMMASubMatrix(WMMAMatrixB, 16, 16, 16, Real, WMMAColMajor):: sb
    WMMASubMatrix(WMMAMatrixC, 16, 16, 16, Real, WMMAKind4):: sc
    call wmmaLoadMatrix(sa, a(1,1), m)
    call wmmaLoadMatrix(sb, b(1,1), k)
    call wmmaMatmul(sc, sa, sb)
    call wmmaStoreMatrix(c(1,1), sc, m)
  end subroutine
end module
COMPUTE-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
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

ARM Support: Beta preview available now

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

MPI API Trace

Select the MPI implementation used by the target application to trace a default set of synchronous MPI calls. If the application uses a different MPI implementation, see the documentation for additional setup required to trace MPI. Note that NVTX tracing will also be enabled on selecting MPI tracing.

- OpenMPI
- MPICH and its derivatives

Collect MPI trace

Collect NVTX trace
NSIGHT COMPUTE
Next-Gen Kernel Profiling Tool

Key Features:
• Interactive CUDA API debugging and kernel profiling
• Fast Data Collection
• Improved Workflow (Diff’ing Results)
• Fully Customizable (Programmable UI/Rules)
• Command Line, Standalone, IDE Integration

OS: Linux (x86, Power, Tegra), Windows, MacOSX (host only)
GPUs: Pascal, Volta, Turing
ARM Support: Beta/Preview Available Now
HPC Libraries

Math and Communication

- cuBLAS
- cuSPARSE
- cuTENSOR
- cuSOLVER
- CUTLASS
- cuRAND
- cuFFT
- CUDA Math API
- NVSHMEM
- Legate
Template abstractions for high-performance matrix-multiplication

New in CUDA 10.2 & CUTLASS 2.0

• New PTX instructions enable maximum efficiency of Turing Tensor Cores

• Easier to use API, offering hierarchical decomposition

CUTLASS Performance Relative to cuBLAS
2080 Ti, TitanV - CUDA 10.2

CUTLASS 2.0
Open Source Framework for Tensor Core Programmability
cuTENSOR
A New High Performance CUDA Library for Tensor Primitives

v1.0.0 available 11/20 at developer.nvidia.com/cutensor
- Tensor Contractions and Reductions
- Elementwise Operations
- Mixed Precision Support
- Elementwise Fusion
- Tensor Core Acceleration

Impact
Up to 24X application end-to-end speedup over previously CPU-only Quantum Chemistry simulations with contraction API

See booth talk: Paul Springer - cuTENSOR: High-Performance CUDA Tensor Primitives - Wednesday @ 5:30pm
TENSOR CORE ACCELERATED LINEAR SOLVERS

Mixed Precision Dense Linear Solvers

- Common HPC Solvers dominated by matrix multiplication
  - LU, Cholesky, QR
- Can we accelerate with FP16 Tensor Core and retain FP64 accuracy? Yes!

V100 TFLOPS

<table>
<thead>
<tr>
<th></th>
<th>FP64</th>
<th>FP32</th>
<th>FP16 (TC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.8</td>
<td>15.7</td>
<td>125</td>
</tr>
</tbody>
</table>

- TRSM
- GEMM
- L
- U

n_b
TENSOR CORE ACCELERATED LINEAR SOLVERS

Mixed Precision Dense Linear Solvers

- CUDA Toolkit 10.2
- Real & Complex FP32 & FP64

LU Solver

- Solve dense linear system by one-sided factorizations
- Supports Real and Complex, FP32 and FP64 data
- Supports multiple right-hand sides
- Retain FP64 accuracy with up to 5X Speedup

Additional solvers coming soon
LEGATE
Accelerated and Distributed NumPy

Early Access launching this week
developer.nvidia.com/Legate

- Drop-in NumPy replacement
- Change one line of code to program a supercomputer
- Automatic GPU and Tensor Core Acceleration
- Automatic scalability

```python
# import numpy as np
import legate.numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
    return x
```
LEGATE
Accelerated and Distributed NumPy

Early Access launching this week
developer.nvidia.com/Legate

- Drop-in NumPy replacement
- Change one line of code to program a supercomputer
- Automatic GPU and Tensor Core Acceleration
- Automatic scalability

```python
import legate.numpy as np
```
### Math Library Early Access Program

Apply for access at [developer.nvidia.com/CUDAMathLibraryEA](developer.nvidia.com/CUDAMathLibraryEA)

Members gain access to pre-release versions of new math libraries and feature extensions

Currently includes:
- cuFFTDx
- cuBLASMg

### NVSHMEM Early Access Program

Apply for access at [developer.nvidia.com/NVSHMEM](developer.nvidia.com/NVSHMEM)

Members gain access to pre-release versions of NVSHMEM

### Legate Early Access Program

Apply for access at [developer.nvidia.com/Legate](developer.nvidia.com/Legate)

Members gain access to pre-release versions of Legate
CONNECT WITH LEADERS IN HPC

The HPC Summit at GTC brings together 300-500 leaders, scientists, and developers to advance the state of the art of HPC

Explore 10 hours of content, engage with experts, and learn about new innovations

www.nvidia.com/hpc-summit