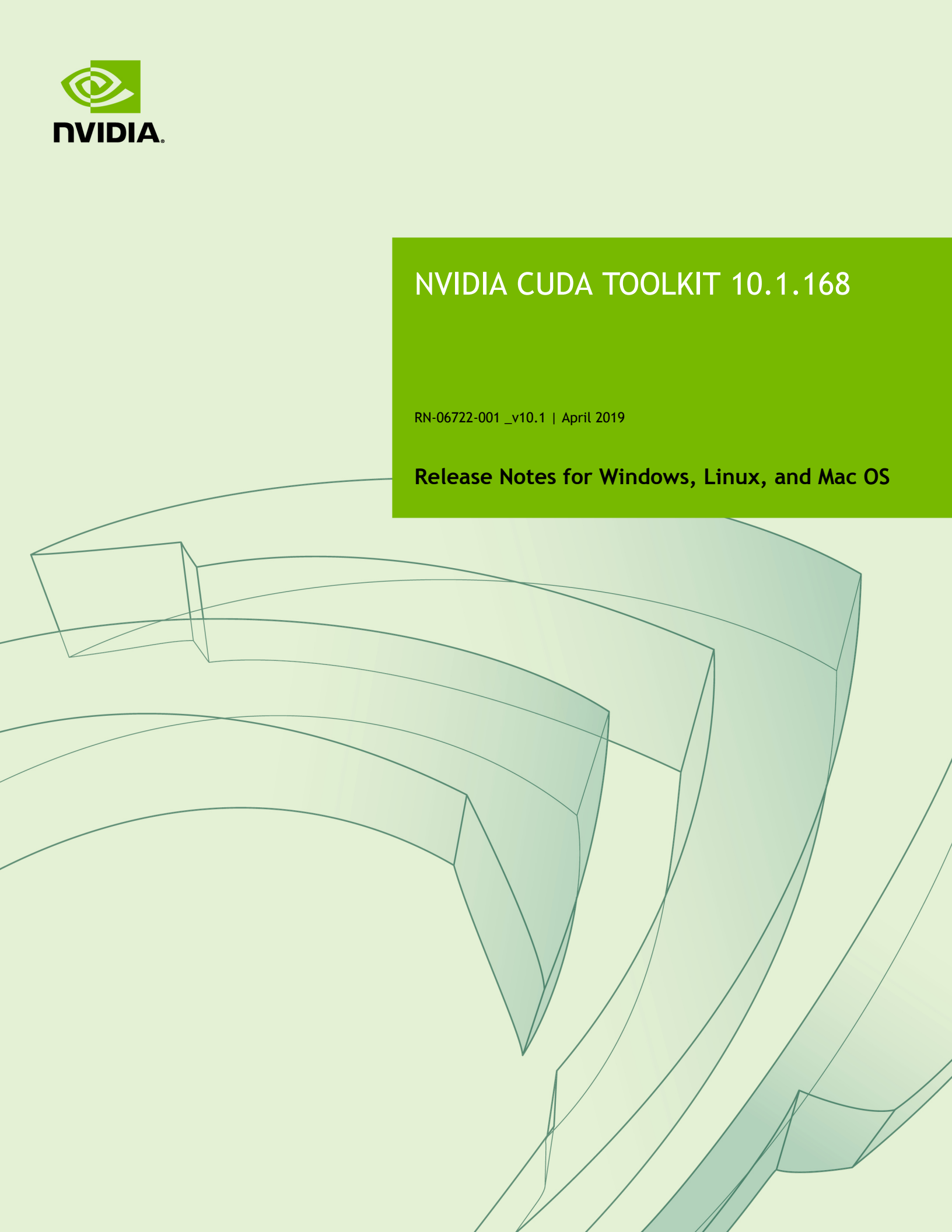

NVIDIA CUDA TOOLKIT 10.1.168

RN-06722-001 _v10.1 | April 2019

Release Notes for Windows, Linux, and Mac OS

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. CUDA TOOLKIT MAJOR COMPONENTS

This section provides an overview of the major components of the CUDA Toolkit and points to their locations after installation.

Compiler

The CUDA-C and CUDA-C++ compiler, **nvcc**, is found in the **bin/** directory. It is built on top of the NVVM optimizer, which is itself built on top of the LLVM compiler infrastructure. Developers who want to target NVVM directly can do so using the Compiler SDK, which is available in the **nvvm/** directory.
Please note that the following files are compiler-internal and subject to change without any prior notice.

- any file in **include/crt** and **bin/crt**
- **include/common_functions.h**, **include/device_double_functions.h**, **include/device_functions.h**, **include/host_config.h**, **include/host_defines.h**, and **include/math_functions.h**
- **nvvm/bin/cicc**
- **bin/cudafe++**, **bin/bin2c**, and **bin/fatbinary**

Tools

The following development tools are available in the **bin/** directory (except for Nsight Visual Studio Edition (VSE) which is installed as a plug-in to Microsoft Visual Studio, Nsight Compute and Nsight Systems are available in a separate directory).

- IDEs: **nsight** (Linux, Mac), Nsight VSE (Windows)
- Debuggers: **cuda-memcheck**, **cuda-gdb** (Linux), Nsight VSE (Windows)
- Profilers: Nsight Systems, Nsight Compute, **nvprof**, **nvvp**, Nsight VSE (Windows)
- Utilities: **cuobjdump**, **nvdisasm**, **gpu-library-advisor**

Libraries

The scientific and utility libraries listed below are available in the **lib/** directory (DLLs on Windows are in **bin/**), and their interfaces are available in the **include/** directory.

- **cublas** (BLAS)
- **cublas_device** (BLAS Kernel Interface)

- **cuda_occupancy** (Kernel Occupancy Calculation [header file implementation])
- **cudadevrt** (CUDA Device Runtime)
- **cudart** (CUDA Runtime)
- **cufft** (Fast Fourier Transform [FFT])
- **cupti** (CUDA Profiling Tools Interface)
- **curand** (Random Number Generation)
- **cusolver** (Dense and Sparse Direct Linear Solvers and Eigen Solvers)
- **cusparse** (Sparse Matrix)
- **nvJPEG** (JPEG encoding/decoding)
- **npp** (NVIDIA Performance Primitives [image and signal processing])
- **nvblas** ("Drop-in" BLAS)
- **nvcuvid** (CUDA Video Decoder [Windows, Linux])
- **nvgraph** (CUDA nvGRAPH [accelerated graph analytics])
- **nvml** (NVIDIA Management Library)
- **nvrtc** (CUDA Runtime Compilation)
- **nvtx** (NVIDIA Tools Extension)
- **thrust** (Parallel Algorithm Library [header file implementation])

CUDA Samples

Code samples that illustrate how to use various CUDA and library APIs are available in the **samples/** directory on Linux and Mac, and are installed to **C:\ProgramData\NVIDIA Corporation\CUDA Samples** on Windows. On Linux and Mac, the **samples/** directory is read-only and the samples must be copied to another location if they are to be modified. Further instructions can be found in the *Getting Started Guides* for Linux and Mac.

Documentation

The most current version of these release notes can be found online at http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html. Also, the **version.txt** file in the root directory of the toolkit will contain the version and build number of the installed toolkit.

Documentation can be found in PDF form in the **doc/pdf/** directory, or in HTML form at **doc/html/index.html** and online at http://docs.nvidia.com/cuda/index.html.

CUDA Driver

Running a CUDA application requires the system with at least one CUDA capable GPU and a driver that is compatible with the CUDA Toolkit. See Table 1. For more information various GPU products that are CUDA capable, visit https://developer.nvidia.com/cuda-gpus. Each release of the CUDA Toolkit requires a minimum version of the CUDA driver. The CUDA driver is backward compatible, meaning that applications compiled against a particular version of the CUDA will continue to work on subsequent (later) driver releases. More information on compatibility can be found at https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#cuda-runtime-and-driver-api-version.

Table 1 CUDA Toolkit and Compatible Driver Versions

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.1.105	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

For convenience, the NVIDIA driver is installed as part of the CUDA Toolkit installation. Note that this driver is for development purposes and is not recommended for use in production with Tesla GPUs. For running CUDA applications in production with Tesla GPUs, it is recommended to download the latest driver for Tesla GPUs from the NVIDIA driver downloads site at http://www.nvidia.com/drivers.

During the installation of the CUDA Toolkit, the installation of the NVIDIA driver may be skipped on Windows (when using the interactive or silent installation) or on Linux (by using meta packages). For more information on customizing the install process on Windows, see http://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html#install-cuda-software. For meta packages on Linux, see https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#package-manager-metas

CUDA-GDB Sources

CUDA-GDB sources are available as follows:

- For CUDA Toolkit 7.0 and newer, in the installation directory **`extras/`**. The directory is created by default during the toolkit installation unless the **`.rpm`** or **`.deb`** package installer is used. In this case, the **`cuda-gdb-src`** package must be manually installed.
- For CUDA Toolkit 6.5, 6.0, and 5.5, at https://github.com/NVIDIA/cuda-gdb.
- For CUDA Toolkit 5.0 and earlier, at ftp://download.nvidia.com/CUDAOpen64/.
- Upon request by sending an e-mail to mailto:oss-requests@nvidia.com.

Chapter 2. CUDA RELEASE NOTES

The release notes for the CUDA Toolkit can be found online at http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html.

2.1. General CUDA

- Introducing NVIDIA® Nsight™ Systems, a system-wide performance analysis tool designed to visualize an application's algorithms. This tool will help you identify the largest opportunities to optimize, and efficiently tune to scale across any quantity or size of CPUs and GPUs—from a large server to the smallest SoC. See more here.
- Added 6.4 version of the Parallel Thread Execution instruction set architecture (ISA). For more details on new (**`noreturn`**, **`mma`**) and deprecated instructions (**`satfinite`**, non-sync versions of **`shfl`** and **`vote`**), see this section in the PTX documentation.
- The following new operating systems are supported by CUDA. See the System Requirements section in the NVIDIA CUDA Installation Guide for Linux for a full list of supported operating systems.
  - Ubuntu 18.10
  - RHEL 7.6
  - Fedora 29
  - SUSE SLES 12.4
  - Windows Server 2019
  - Windows 10 (October 2018 Update)
- Improved the scalability of **`cudaFree`*** APIs on multi-GPU systems.
- Added support for cooperative group kernels (using the **`cudaLaunchCooperativeKernel`** API) with MPS.
- Relaxed IPC restrictions so that P2P can be enabled between devices that are not set by CUDA_VISIBLE_DEVICES.
- In CUDA 10.1 the CUDA Runtime API error codes are renumbered to match, wherever possible, their CUDA Driver API equivalents.
- Added GPU accounting, on Volta only, to keep track of open compute contexts and GPU utilization. This data is updated when the driver is loaded, but can be retrieved in driver-loaded and driver-unloaded modes via Out of Band (OOB).

- Added an out-of-band mechanism to fetch the instantaneous GPU and memory utilization See SMBPBI spec for the documentation.
- Added the ability to query GPU NVLink error rates / counts via out of band (OOB) with or without a driver present. See SMBPBI spec for the documentation.
- Added support for installing CUDA using runfiles on POWER (ppc64le) platforms.
- Added new CUDA samples for CUDA Graph APIs.

2.2. CUDA Tools

2.2.1. CUDA Compilers

- The following compilers are supported as host compilers in **nvcc**:
  - GCC 8.x
  - Clang 7.0
  - Microsoft Visual Studio 2017 (RTW, and all updates)
  - Microsoft Visual Studio 2019 (Preview releases)
  - PGI 19.x
  - ICC 19.0
  - Xcode 10.1 (10B61)
- New functions **__isShared()**, **__isConstant()** and **__isLocal()** have been added, to check if a generic pointer points to an object in **__shared__**, **__constant__** or local memory, respectively. These functions are documented in the CUDA C Programming Guide, along with the existing **__isGlobal()** function.
- The existing API functions **nvrtcGetLoweredName** and **nvrtcAddNameExpression** have been enhanced to allow looking up the mangled (lowered) name of **__constant__** and **__device__** variables. Details and example here: https://docs.nvidia.com/cuda/nvrtc/index.html#accessing-lowered-names.
- **nvcc** now supports the "-MF" and "-MM" flags related to dependency generation. See below the description of the new flags from "nvcc --help":
  - **--generate-nonsystem-dependencies (-MM)** : Same as **--generate-dependencies** but skip header files found in system directories (Linux only).
  - **--dependency-output (-MF)**: Specify the output file for the dependency file generated with -M or -MM. If this option is not specified, the output is the same as if -E has been specified.

2.2.2. CUDA Profiler

- For new features in Visual Profiler and **nvprof**, see the What's New section in the Profiler User's Guide.
- For new features available in CUPTI, see the What's New section in the CUPTI documentation.
- For system wide profiling, use Nsight Systems. Refer to the Nsight Systems Release Notes.

- For profiling specific CUDA kernels, use Nsight Compute. Refer to the Nsight Compute Release Notes.

2.2.3. CUDA-MEMCHECK

- For new features in CUDA-MEMCHECK, see the Release Notes in the CUDA-MEMCHECK documentation.

2.3. CUDA Libraries

This release of the CUDA toolkit is packaged with libraries that deliver new and extended functionality, bug fixes, and performance improvements for single and multi-GPU environments.

Also in this release the **soname** of the libraries has been modified to not include the minor toolkit version number. For example, the cuFFT library **soname** has changed from **libcufft.so.10.1** to **libcufft.so.10**. This is done to facilitate any future library updates that do not include API breaking changes without the need to relink.

2.3.1. cuBLAS Library

- With this release, on Linux systems, the cuBLAS libraries listed below are now installed in the **/usr/lib/<arch>-linux-gnu/** or **/usr/lib64/** directories as shared and static libraries. Their interfaces are available in the **/usr/include** directory:
  - cublas (BLAS)
  - cublasLt (new Matrix Multiply library)
- Note that the new installation locations of cuBLAS libraries are different from the past versions. In the past versions the libraries were installed in directories under the main toolkit installation directory.
- Package managers on Linux OSs will remove the previous version of cuBLAS and update to the new libraries in the new location. For linking and execution make sure the new location is specified within your paths such as LD_LIBRARY_PATH.
- With this update, the versioning scheme of the cuBLAS library has changed to a 4-digit version. Because of this change, version numbers might differ between the CUDA toolkit and cuBLAS libraries in future releases. For the new 4-digit version:
  - The first three digits follow semantic versioning, and
  - The last digit is the build number.
- A new library, the cuBLASLt, is added. The cuBLASLt is a new lightweight library dedicated to GEneral Matrix-to-matrix Multiply (GEMM) operations with a new flexible API. This new library adds flexibility in matrix data layouts, input types, compute types, and also in choosing the algorithmic implementations and heuristics through parameter programmability. Read more at: http://docs.nvidia.com/cuda/cublas/index.html#using-the-cublasLt-api.
- The new cuBLASLt library is packaged as a separate binary and a header file. Also, the cuBLASLt now adds support for:

- Utilization of IMMA tensor core operations on Turing GPUs for int8 input matrices.
- FP16 half-precision CGEMM split-complex matrix multiplies using tensor cores on Volta and Turing GPUs.

2.3.2. cuSOLVER Library

- For symmetric dense eigensolver:
  - Added a new selective eigensolver functionality for standard and generalized eigenvalue problems: SYEVDX and SYGVDX
  - Improved the performance for full eigenspectrum eigensolver.
- Added a new batched GESVDA API that computes the approximate singular value decomposition of a tall skinny **`m×n`** matrix A.
- Added a new POTRI API that computes the inverse of a symmetric positive definite matrix, using the Cholesky factorization computed by DPOTRF.

2.3.3. cuSPARSE Library

- Added a new generic Sparse x Dense Matrix Multiply (SpMM) APIs that encapsulates the functionality of many legacy APIs.
- Added a new COO matrix-matrix multiplication (cooMM) implementation with:
  - Deterministic and non-deterministic variants
  - Batched SpMM
  - Support for multiple data type combinations
  - Speed-ups w.r.t. csrMM for matrices with highly irregular nnzs/row
- Added two new algorithms for **`csr2csc`** format conversions with improved performance and reduced memory use.

2.3.4. cuFFT Library

- Improved the performance and scalability for the following use cases:
  - multi-GPU non-power of 2 transforms
  - R2C and Z2D odd sized transforms
  - 2D transforms with small sizes and large batch counts

2.3.5. cuRAND Library

- Improved the performance of the following random number generators:
  - MTGP32
  - MRG32k3a
  - Sobol32 and Scrambled Sobol32
  - Sobol64 and Scrambled Sobol64

2.3.6. NPP Library

- Some of the most commonly used image processing functions were extended to support the FP16 (**`__half`**) data type on GPU architectures Volta and beyond.
- Added support for application-managed stream contexts. Application-managed stream contexts make NPP truely stateless internally, allowing for rapid, stream context switching with no overhead.
- While it is recommended that all new NPP application code use application-managed stream contexts, existing application code can continue to use **`nppSetStream()`** and **`nppGetStream()`** to manage stream contexts (also with no overhead now). But over time NPP will likely deprecate the older non-application-managed stream context API.

2.3.7. nvJPEG Library

- Added baseline encoding functionality to the library that will be extended in future releases.
- Added new batched decoding that uses GPU acceleration for all phases of computation. This delivers significant performance gains for large batches of images where most images are baseline encoded JPEG images.
- Added new APIs for pinned memory allocator and for memory overallocations.
- The nvJPEG library is now added to the Linux ppc64le CUDA Toolkit distributions.

2.4. Deprecated Features

The following features are deprecated in the current release of the CUDA software. The features still work in the current release, but their documentation may have been removed, and they will become officially unsupported in a future release. We recommend that developers employ alternative solutions to these features in their software.

General CUDA

- Nsight Eclipse Edition standalone is deprecated in CUDA 10.1, and will be dropped in the release that immediately follows CUDA 10.1.
- Support for RHEL 6.x is deprecated with CUDA 10.1. It may be dropped in a future release of CUDA. Customers are encouraged to adopt RHEL 7.x to use new versions of CUDA.
- The following compilers are no longer supported as host compilers for **`nvcc`**
  - PGI 17.x
  - Microsoft Visual Studio 2010
  - Clang versions lower than 3.7
- Microsoft Visual Studio versions 2011, 2012 and 2013 are now deprecated as host compilers for nvcc. Support for these compilers may be removed in a future release of CUDA.
- 32-bit tools are no longer supported starting with CUDA 10.0.

- NVIDIA GPU Library Advisor (**gpu-library-advisor**) is now deprecated and will be removed in a future release of the toolkit.
- The non-sync definitions of warp shuffle functions (**__shfl**, **__shfl_up**, **__shfl_down**, and **__shfl_xor**) and warp vote functions (**__any**, **__all**, **__ballot**) have been removed when compilation is targeting devices with compute capability 7.x and higher.
- For WMMA operations with floating point accumulators, the **satf** (saturate-to-finite value) mode parameter is deprecated. Using it can lead to unexpected results. See http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#wmma-description for details.

CUDA Libraries

- The nvGRAPH library is deprecated. The library will no longer be shipped in future releases of the CUDA toolkit.
- The **nppGetGpuComputeCapability** function will be deprecated in the next NPP release. Users should instead call the **cudaGetDevice()** to get the GPU device ID, then call the function **cudaDeviceGetAttribute()** twice, once with the **cudaDevAttrComputeCapabilityMajor** parameter, and once with the **cudaDevAttrComputeCapabilityMinor** parameter.

2.5. Resolved Issues

2.5.1. CUDA Compilers

- In CUDA 9.2 **nvprune** crashes when running on a library with bss sections--for example, while pruning **libcusparse_static.a**. This is fixed in CUDA 10.1.
- Within a template function, the CUDA compiler previously incorrectly allowed the use of an undeclared function. In CUDA 10.1, this compiler bug has been fixed and may cause diagnostics to be emitted for code that was previously incorrectly accepted. For example:

```
//-- template <typename T>
__global__ void foo(T in) { bar(in); }
int main() { foo<<<1,1>>>(1); }
__device__ void bar(int) { }
//--
```

This example was accepted by the previous CUDA compiler, but will generate a compile error with CUDA 10.1 compiler, because the function **bar()** is used in **foo()** before it has been declared.

2.5.2. CUDA Libraries

- In earlier releases, cuBLAS GEMM calls might randomly crash when running multiple host threads sharing one cuBLAS handle despite adhering to recommended usage in Thread Safety. This bug affects cuBLAS in earlier CUDA releases, and is fixed in CUDA 10.1.
- This bug affects cuSOLVER dense linear algebra functionality in CUDA 10.0, and is fixed in CUDA 10.1. An internal routine (LARFT) in cuSOLVER dense linear algebra

has a race condition if the user stream is not null. Although this routine is not in the cuSOLVER public API, it affects all routines based on householder reflection, including ORGBR, ORMTR, ORGTR, ORMQR, ORMQL, ORGQR AND SYEVD, SYGVD.

The following routines are **not** affected:

- GEQRF and GESVD.
- Also, GESVDJ and SYEVDJ are not affected because they are based-on the Jacobi method.

- The cuSOLVER routine SYEVD in CUDA 10.0 has a bug that may potentially impact single and single complex eigenvalue problems. In affected cases, the eigensolver may deliver inaccurate eigenvalues and eigenvectors. This bug only affects cuSOLVER dense linear algebra functionality in CUDA 10.0 and is fixed in CUDA 10.1.
- In CUDA 10.0 cuBLAS library, a bug in the batched LU factorization, **`cublas[S|D|C|Z]getrfBatched`**, may lead to wrong result or inconsistent result from run to run. CUDA 10.1 fixes the issue.

2.6. Known Issues

2.6.1. General CUDA

- On systems with a new install of Ubuntu 18.04.2, note that the installation of CUDA 10.1 and NVIDIA 418 drivers may result in the following error:

```
The following packages have unmet dependencies:

xserver-xorg-video-nvidia-418 : Depends:

xserver-xorg-core (>= 2:1.19.6-1ubuntu2~)

E: Unable to correct problems, you have held broken packages.
```

  To recover from this error, install the *xserver-xorg-core* package and proceed with the installation of CUDA.

```
$ sudo apt-get install xserver-xorg-core
```

  

  This error is only observed on systems with a new install of Ubuntu 18.04.2. The error is not observed on systems that are upgraded from 18.04.1 to 18.04.2.

- Xwayland is not compatible with **`nvidia-settings`** and graphical CUDA samples. Recommend switching to Xorg session.

2.6.2. CUDA Tools

- When using separate compilation and object linking for the device code (**`nvcc --relocatable-device-code=true or nvcc --device-c`**) the resulting binary may crash at runtime when **all** conditions below are true:

1. Some of the objects linked into the binary are generated by a previously released compiler (i.e., compiler from CUDA 10.0 Toolkit or earlier), and
2. Objects generated by the previously released compiler contain CUDA kernel function definition (i.e., "**`__global__`**" functions).

A possible workaround in such case is to generate all the objects with the CUDA 10.1 compiler.

- For known issues in cuda-memcheck, see the Known Issues section in the cuda-memcheck documentation.
- For known issues in Nsight Compute, see the Known Issues section.
- When enabling the "Auto Profile" option in the Nsight Compute UI, profiling across different GPU architectures may fail.

  To workaround this issue, profile the relevant kernels using the Nsight Compute CLI, or disable "Auto Profile" in the UI and manually profile these kernels.
- The tools **`nv-nsight-cu`** and **`nv-nsight-cu-cli`** will not work on any Linux platform with GLIBC version lower than 2.15. Hence these tools will not work on RHEL 6.10 and CentOS 6.10, which use GLIBC 2.12.
- For known issues in CUDA Profiling tools **`nvprof`** and Visual Profiler, see the Profiler Known Issues section in the Nsight Eclipse Edition Getting Started Guide.
- For known issues in the CUPTI library, see the Limitations section in the CUPTI document.
- For known issues in Nsight Eclipse, see the Nsight Eclipse Known Issues section in the Profiler User's Guide.
- On Windows CUPTI samples and other applications using the CUPTI APIs will result in the error "cupti.dll was not found". This is due to a mismatch in the CUPTI dynamic library name referenced in the import library "cupti.lib".

  To workaround this issue rename the CUPTI dynamic library under the CUDA Toolkit directory (Default location is: "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\lib64") **from** "cupti64_101.dll" **to** "cupti.dll".
- A call to **`cuptiFinalize()`**/**`cuptiProfilerDeInitialize()`** API can result in a hang with 410.x driver. **Please use a 418 or later driver.**

2.6.3. CUDA Libraries

- cuSOLVER dense linear algebra routine GETRF might exit with error code 702 on GPUs that have only 2 SMs Jetson TX1 GPUs.

Chapter 3. THRUST V1.9.4 RELEASE NOTES

Thrust v1.9.4 adds asynchronous interfaces for parallel algorithms, a new allocator system including caching allocators and unified memory support, as well as a variety of other enhancements, mostly related to C++11/C++14/C++17/C++20 support.

The new asynchronous algorithms in the **`thrust::async`** namespace return **`thrust::event`** or **`thrust::future`** objects, which can be waited upon to synchronize with the completion of the parallel operation.

3.1. New Features

- **`thrust::event`** and **`thrust::future<T>`**, uniquely-owned asynchronous handles consisting of a state (ready or not ready), content (some value; for **`thrust::future`** only), and an optional set of objects that should be destroyed only when the future's value is ready and has been consumed.
  - The design is loosely based on C++11's **`std::future`**.
  - They can be **`.wait`**'d on, and the value of a future can be waited on and retrieved with .get or .extract.
  - Multiple **`thrust::event`**s and **`thrust::future`**s can be combined with **`thrust::when_all`**.
  - **`thrust::future`**s can be converted to **`thrust::event`**s.
  - Currently, these primitives are only implemented for the CUDA backend and are C++11 only.
- New asynchronous algorithms that return **`thrust::event`**/**`thrust::future`**s, implemented as C++20 range style customization points:
  - **`thrust::async::reduce`**.
  - **`thrust::async::reduce_into`**, which takes a target location to store the reduction result into.
  - **`thrust::async::copy`**, including a two-policy overload that allows explicit cross system copies which execution policy properties can be attached to.
  - **`thrust::async::transform`**.
  - **`thrust::async::for_each`**.

  - **thrust::async::stable_sort**.
  - **thrust::async::sort**.
  - By default the asynchronous algorithms use the new caching allocators. Deallocation of temporary storage is deferred until the destruction of the returned **thrust::future**. The content of **thrust::future**s is stored in either device or universal memory and transferred to the host only upon request to prevent unnecessary data migration.
  - Asynchronous algorithms are currently only implemented for the CUDA system and are C++11 only.
- **exec.after(f, g, ...)**, a new execution policy method that takes a set of **thrust::event**/**thrust::future**s and returns an execution policy that operations on that execution policy should depend upon.
- New logic and mindset for the type requirements for cross-system sequence copies (currently only used by **thrust::async::copy**), based on:
  - **thrust::is_contiguous_iterator** and THRUST_PROCLAIM_CONTIGUOUS_ITERATOR for detecting/indicating that an iterator points to contiguous storage.
  - **thrust::is_trivially_relocatable** and THRUST_PROCLAIM_TRIVIALLY_RELOCATABLE for detecting/indicating that a type is **memcpy**-able (based on principles from https://wg21.link/P1144).
  - The new approach reduces buffering, increases performance, and increases correctness.
  - The fast path is now enabled when copying fp16 and CUDA vector types with **thrust::async::copy**.
- All Thrust synchronous algorithms for the CUDA backend now actually synchronize. Previously, any algorithm that did not allocate temporary storage (counterexample: **thrust::sort**) and did not have a computation-dependent result (counterexample: **thrust::reduce**) would actually be launched asynchronously.

  Additionally, synchronous algorithms that allocated temporary storage would become asynchronous if a custom allocator was supplied that did not synchronize on allocation/deallocation, unlike **cudaMalloc** / **cudaFree**. So, now **thrust::for_each**, **thrust::transform**, **thrust::sort**, etc are truly synchronous.

  In some cases this may be a performance regression; if you need asynchrony, use the new asynchronous algorithms.
- Thrust's allocator framework has been rewritten. It now uses a memory resource system, similar to C++17's **std::pmr** but supporting static polymorphism. Memory resources are objects that allocate untyped storage and allocators are cheap handles to memory resources in this new model. The new facilities live in **<thrust/mr/*>** .
  - t**hrust::mr::memory_resource<Pointer>**, the memory resource base class, which takes a (possibly tagged) pointer to void type as a parameter.
  - **thrust::mr::allocator<T, MemoryResource>**, an allocator backed by a memory resource object.
  - **thrust::mr::polymorphic_adaptor_resource<Pointer>**, a type-erased memory resource adaptor.

- **thrust::mr::polymorphic_allocator<T>**, a C++17-style polymorphic allocator backed by a type-erased memory resource object.
- New tunable C++17-style caching memory resources, **thrust::mr::(disjoint_)?(un)?synchronized_pool_resource**, designed to cache both small object allocations and large repetitive temporary allocations. The disjoint variants use separate storage for management of the pool, which is necessary if the memory being allocated cannot be accessed on the host (e.g. device memory).
- System-specific allocators were rewritten to use the new memory resource framework.
- New **thrust::device_memory_resource** for allocating device memory.
- New **thrust::universal_memory_resource** for allocating memory that can be accessed from both the host and device (e.g. **cudaMallocManaged**).
- New **thrust::universal_host_pinned_memory_resource** for allocating memory that can be accessed from the host and the device but always resides in host memory (e.g. **cudaMallocHost**).
- **thrust::get_per_device_resource** and **thrust::per_device_allocator**, which lazily create and retrieve a per-device singleton memory resource.
- Rebinding mechanisms (**rebind_traits** and **rebind_alloc**) for **thrust::allocator_traits**.
- **thrust::device_make_unique**, a factory function for creating a **std::unique_ptr** to a newly allocated object in device memory.
- **<thrust/detail/memory_algorithms>** , a C++11 implementation of the C++17 uninitialized memory algorithms.
- **thrust::allocate_unique** and friends, based on the proposed C++23 **std::allocate_unique** (https://wg21.link/P0211).

- New type traits and metaprogramming facilities. Type traits are slowly being migrated out of **thrust::detail::** and **<thrust/detail/*>** ; their new home will be **thrust::** and **<thrust/type_traits/*>**.
  - **thrust::is_execution_policy**.
  - **thrust::is_operator_less_or_greater_function_object**, which detects **thrust::less**, **thrust::greater**, **std::less**, and **std::greater**.
  - **thrust::is_operator_plus_function_object**, which detects **thrust::plus** and **std::plus**.
  - **thrust::remove_cvref(_t)?**, a C++11 implementation of C++20's **thrust::remove_cvref(_t)?**.
  - **thrust::void_t**, and various other new type traits.
  - **thrust::integer_sequence** and friends, a C++11 implementation of C++20's **std::integer_sequence**.
  - **thrust::conjunction**, **thrust::disjunction**, and **thrust::disjunction**, a C++11 implementation of C++17's logical metafunctions.
  - Some Thrust type traits (such as **thrust::is_constructible**) have been redefined in terms of C++11's type traits when they are available.
- **<thrust/detail/tuple_algorithms.h>**, new **std::tuple** algorithms:

- **thrust::tuple_transform**.
- **thrust::tuple_for_each**.
- **thrust::tuple_subset**.

- Miscellaneous new **std::**-like facilities:
  - **thrust::optional**, a C++11 implementation of C++17's **std::optional**.
  - **thrust::addressof**, an implementation of C++11's **std::addressof**.
  - **thrust::next** and **thrust::prev**, an implementation of C++11's **std::next** and **std::prev**.
  - **thrust::square**, a **<functional>** style unary function object that multiplies its argument by itself.
  - **<thrust/limits.h>** and **thrust::numeric_limits**, a customized version of **<limits>** and **std::numeric_limits**.
- **<thrust/detail/preprocessor.h>**, new general purpose preprocessor facilities:
  - **THRUST_PP_CAT[2-5]**, concatenates two to five tokens.
  - **THRUST_PP_EXPAND(_ARGS)?**, performs double expansion.
  - **THRUST_PP_ARITY** and **THRUST_PP_DISPATCH**, tools for macro overloading.
  - **THRUST_PP_BOOL**, boolean conversion.
  - **THRUST_PP_INC** and **THRUST_PP_DEC**, increment/decrement.
  - **THRUST_PP_HEAD**, a variadic macro that expands to the first argument.
  - **THRUST_PP_TAIL**, a variadic macro that expands to all its arguments after the first.
  - **THRUST_PP_IIF**, bitwise conditional.
  - **THRUST_PP_COMMA_IF**, and **THRUST_PP_HAS_COMMA**, facilities for adding and detecting comma tokens.
  - **THRUST_PP_IS_VARIADIC_NULLARY**, returns true if called with a nullary **__VA_ARGS__** .
  - **THRUST_CURRENT_FUNCTION**, expands to the name of the current function.
- New C++11 compatibility macros:
  - **THRUST_NODISCARD**, expands to **[[nodiscard]]** when available and the best equivalent otherwise.
  - **THRUST_CONSTEXPR**, expands to **constexpr** when available and the best equivalent otherwise.
  - **THRUST_OVERRIDE**, expands to **override** when available and the best equivalent otherwise.
  - **THRUST_DEFAULT**, expands to **= default;** when available and the best equivalent otherwise.
  - **THRUST_NOEXCEPT**, expands to **noexcept** when available and the best equivalent otherwise.
  - **THRUST_FINAL**, expands to **final** when available and the best equivalent otherwise.
  - **THRUST_INLINE_CONSTANT**, expands to **inline constexpr** when available and the best equivalent otherwise.
- **<thrust/detail/type_deduction.h>**, new C++11-only type deduction helpers:

- **`THRUST_DECLTYPE_RETURNS*`**, expand to function definitions with suitable conditional noexcept qualifiers and trailing return types.
- **`THRUST_FWD(x)`**, expands to **`::std::forward<decltype(x)>(x)`**.
- **`THRUST_MVCAP`**, expands to a lambda move capture.
- **`THRUST_RETOF`**, expands to a **`decltype`** computing the return type of an invocable.

3.2. New Examples

- **`mr_basic`** demonstrates how to use the new memory resource allocator system.

3.3. Other Enhancements

3.3.1. Tagged Pointer Enhancements

- New **`thrust::pointer_traits`** specialization for **`void const*`**.
- **`nullptr`** support to Thrust tagged pointers.
- New explicit operator bool for Thrust tagged pointers when using C++11 for **`std::unique_ptr`** interoperability.
- Added **`thrust::reinterpret_pointer_cast`** and **`thrust::static_pointer_cast`** for casting Thrust tagged pointers.

3.3.2. Iterator Enhancements

- **`thrust::iterator_system`** is now SFINAE friendly.
- Removed **`cv`** qualifiers from iterator types when using **`thrust::iterator_system`**.
- Static assert enhancements:
- New **`THRUST_STATIC_ASSERT_MSG`**, takes an optional string constant to be used as the error message when possible.
- Update **`THRUST_STATIC_ASSERT(_MSG)`** to use C++11's **`static_assert`** when it's available.
- Introduce a way to test for static assertions.

3.3.3. Testing Enhancements

- Additional scalar and sequence types, including non-builtin types and vectors with unified memory allocators, have been added to the list of types used by generic unit tests.
- The generation of random input data has been improved to increase the range of values used and catch more corner cases.
- New **`truncate_to_max_representable`** utility for avoiding the generation of ranges that cannot be represented by the underlying element type in generic unit test code.

- The test driver now synchronizes with CUDA devices and check for errors after each test, when switching devices, and after each raw kernel launch.
- The **warningtester** uber header is now compiled with NVCC to avoid needing to disable CUDA-specific code with the preprocessor.
- Fixed the unit test framework's **ASSERT_*** to print **char**s as **int**s.
- New **DECLARE_INTEGRAL_VARIABLE_UNITTEST** test declaration macro.
- New **DECLARE_VARIABLE_UNITTEST_WITH_TYPES_AND_NAME** test declaration macro.
- **thrust::system_error** in the CUDA backend now print out its **cudaError_t** enumerator in addition to the diagnostic message.
- Stopped using conditionally signed types like **char**.

3.4. Resolved Issues

- Fixed compilation error when using **__device__ lambdas** with reduce on MSVC.
- Static asserted that **thrust::generate** / **thrust::fill** doesn't operate on **const** iterators.
- Fixed compilation failure with **thrust::zip_iterator** and **thrust::complex<float>**.
- Fixed dispatch for the CUDA backend's **thrust::reduce** to use two functions (one with the **pragma** for disabling **exec** checks, one with **THRUST_RUNTIME_FUNCTION**) instead of one. This fixes a regression with device compilation that started in CUDA 9.2.
- Added missing **__host__ __device__** annotations to a **thrust::complex::operator=** to satisfy GoUDA.
- Made **thrust::vector_base::clear** not depend on the element type being default constructible.
- Removed flaky **simple_cuda_streams** example.
- Added missing **thrust::device_vector** constructor that takes an allocator parameter.
- Updated the **range_view** example to not use device-side launch.
- Ensured that sized unit tests that use **counting_iterator** perform proper truncation.
- Refactored questionable **copy_if** unit tests.

Chapter 4.
CUDA TEGRA RELEASE NOTES

The release notes for CUDA Tegra contain only information that is specific to the following:

- CUDA Tegra Driver, and
- Mobile version of other CUDA components such as: compilers, tools, libraries, and samples.

The release notes for the desktop version of CUDA also apply to CUDA Tegra. On Tegra, the CUDA Toolkit version is 10.1.

4.1. New Features

CUDA Tegra Driver

- Support is added for GPUDirect RDMA on AGX Jetson platform. This now enables a direct path for data exchange between the GPU and third-party peer devices using standard features of PCI Express.
- Support for Android P added.
- Error Reporting enriched in CUDA on mobile RM.
- Support added for Ubuntu 18.04 for the AGX Drive platform.
- Resumed the support for Ubuntu 16.04 host on the AGX Jetson platform.
- Performance optimizations that were previously enabled for QNX are now also available on Linux through user-mode submits.

CUDA Compiler

- Support added for GCC 7.3 on AGX Jetson platforms.
- Support added for CLANG 6.0.2 for NVCC on Android platforms.

4.2. Known Issues and Limitations

CUDA Tegra Driver

- Only the below color formats are supported for Vulkan-CUDA interoperability on Jetson and Android:
  - VK_FORMAT_R8_UNORM
  - VK_FORMAT_R8_SNORM
  - VK_FORMAT_R8_UINT
  - VK_FORMAT_R8_SINT
  - VK_FORMAT_R8_SRGB
  - VK_FORMAT_R8G8_UNORM
  - VK_FORMAT_R8G8_SNORM
  - VK_FORMAT_R8G8_UINT
  - VK_FORMAT_R8G8_SINT
  - VK_FORMAT_R8G8_SRGB
  - VK_FORMAT_R16_UNORM
  - VK_FORMAT_R16_SNORM
  - VK_FORMAT_R16_UINT
  - VK_FORMAT_R16_SINT
  - VK_FORMAT_R16_SFLOAT

  Other formats are currently not supported.

CUDA Tools

- On NVIDIA DRIVE OS Linux systems, when using Nsight Compute CLI with "**`--mode attach`**" to attach to another process on the same machine, "**`--hostname 127.0.0.1`**" must be passed. This is because the default value of **`"localhost"`** for the **`"--hostname"`** parameter does not work.

4.3. Resolved Issues

General CUDA

- **CUDA-GDB on Linux:** The **`set cuda memcheck on`** command in cuda-gdb does not have any effect. This is fixed in CUDA 10.1.
- **CUDA-GDB on QNX:** **`ntoaarch64-gdb`** and **`cuda-qnx-gdb`** may hang when executing the run command. This is fixed in CUDA 10.1.

4.4. Deprecated Issues

General CUDA

- Deprecating support for Pascal product on Android.

Acknowledgments

NVIDIA extends thanks to Professor Mike Giles of Oxford University for providing the initial code for the optimized version of the device implementation of the double-precision `exp()` function found in this release of the CUDA toolkit.

NVIDIA acknowledges Scott Gray for his work on small-tile GEMM kernels for Pascal. These kernels were originally developed for OpenAI and included since cuBLAS 8.0.61.2.

Notice

Trademarks

Copyright

www.nvidia.com