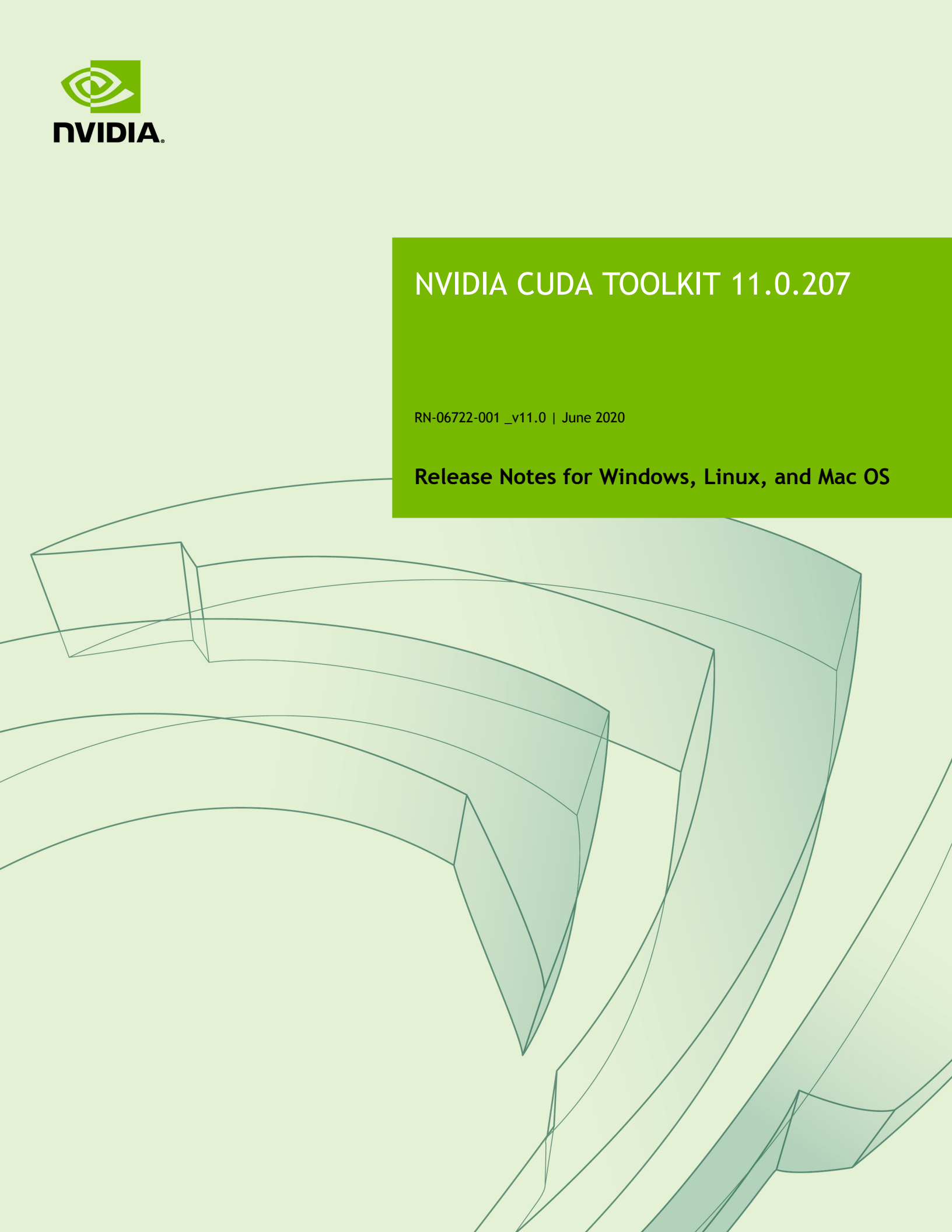

NVIDIA CUDA TOOLKIT 11.0.207

RN-06722-001 _v11.0 | June 2020

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 NVIDIA® 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**, **ncu**, Nsight VSE (Windows)
- Utilities: **cuobjdump**, **nvdisasm**

Libraries

The scientific and utility libraries listed below are available in the **lib64/** directory (DLLs on Windows are in **bin/**), and their interfaces are available in the **include/** directory.

- **cub** (High performance primitives for CUDA)
- **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)
- **libcu++** (CUDA Standard C++ Library)
- **nvJPEG** (JPEG encoding/decoding)
- **npp** (NVIDIA Performance Primitives [image and signal processing])
- **nvblas** ("Drop-in" BLAS)
- **nvcuvid** (CUDA Video Decoder [Windows, Linux])
- **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-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 11.0 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. What's New

This section summarizes the changes in CUDA 11.0 GA since the 11.0 RC release.

General CUDA

- Added support for Ubuntu 20.04 LTS on **`x86_64`** platforms.
- Arm server platforms (arm64 sbsa) are supported with NVIDIA T4 GPUs.

NPP New Features

- Batched Image Label Markers Compression that removes sparseness between marker label IDs output from LabelMarkers call.
- Image Flood Fill functionality fills a connected region of an image with a specified new value.
- Stability and performance fixes to Image Label Markers and Image Label Markers Compression.

nvJPEG New Features

- nvJPEG allows the user to allocate separate memory pools for each chroma subsampling format. This helps avoid memory re-allocation overhead. This can be controlled by passing the newly added flag **`NVJPEG_FLAGS_ENABLE_MEMORY_POOLS`** to the **`nvjpegCreateEx`** API.
- nvJPEG encoder now allow compressed bitstream on the GPU Memory.

cuBLAS New Features

- cuBLASLt Matrix Multiplication adds support for fused ReLU and bias operations for all floating point types except double precision (FP64).
- Improved batched TRSM performance for matrices larger than 256.

cuSOLVER New Features

- Add 64-bit API of GESVD. The new routine cusolverDnGesvd_bufferSize() fills the missing parameters in 32-bit API **`cusolverDn[S|D|C|Z]gesvd_bufferSize()`** such that it can estimate the size of the workspace accurately.
- Added the single process multi-GPU Cholesky factorization capabilities POTRF, POTRS and POTRI in cusolverMG library.

cuSOLVER Resolved Issues

- Fixed an issue where SYEVD/SYGVD would fail and return error code 7 if the matrix is zero and the dimension is bigger than 25.

cuSPARSE New Features

- Added new Generic APIs for Axpby (cusparseAxpby), Scatter (cusparseScatter), Gather (cusparseGather), Givens rotation (cusparseRot). __nv_bfloat16/ __nv_bfloat162 data types and 64-bit indices are also supported.
- This release adds the following features for cusparseSpMM:
  - Support for row-major layout for cusparseSpMM for both CSR and COO format
  - Support for 64-bit indices
  - Support for __nv_bfloat16 and __nv_bfloat162 data types
  - Support for the following strided *batch* mode:
    - Ci=A#Bi
    - Ci=Ai#B
    - Ci=Ai#Bi

cuFFT New Features

- cuFFT now accepts __nv_bfloat16 input and output data type for power-of-two sizes with single precision computations within the kernels.

Known Issues

- cuFFT now accepts __nv_bfloat16 input and output data type for power-of-two sizes with single precision computations within the kernels.
- Note that starting with CUDA 11.0, the minimum recommended GCC compiler is at least GCC 5 due to C++11 requirements in CUDA libraries e.g. cuFFT and CUB. On distributions such as RHEL 7 or CentOS 7 that may use an older GCC toolchain by default, it is recommended to use a newer GCC toolchain with CUDA 11.0. Newer GCC toolchains are available with the Red Hat Developer Toolset.

Deprecations

The following functions have been removed:

- **`cusparse<t>gemmi()`**
- **`cusparseXaxpyi`**, **`cusparseXgthr`**, **`cusparseXgthrz`**, **`cusparseXroti`**, **`cusparseXsctr`**

2.2. CUDA Toolkit Major Component Versions

CUDA Components

Starting with CUDA 11, the various components in the toolkit are versioned independently.

For CUDA 11, the table below indicates the versions:

Table 1 CUDA 11 Component Versions

Component Name	Version Information	Supported Architectures
CUDA Runtime (cudart)	11.0.194	x86_64, POWER, Arm64
cuobjdump	11.0.194	x86_64, POWER, Arm64
CUPTI	11.0.194	x86_64, POWER, Arm64
CUDA Demo Suite	11.0.167	x86_64
CUDA GDB	11.0.194	x86_64, POWER, Arm64
CUDA Memcheck	11.0.194	x86_64, POWER
CUDA NVCC	11.0.194	x86_64, POWER, Arm64
CUDA nvdisasm	11.0.194	x86_64, POWER, Arm64
CUDA NVML Headers	11.0.167	x86_64, POWER, Arm64
CUDA nvprof	11.0.194	x86_64, POWER, Arm64
CUDA nvprune	11.0.167	x86_64, POWER, Arm64
CUDA NVRTC	11.0.194	x86_64, POWER, Arm64
CUDA NVTX	11.0.167	x86_64, POWER, Arm64
CUDA NVVP	11.0.194	x86_64, POWER
CUDA Samples	11.0.194	x86_64, POWER, Arm64
CUDA Compute Sanitizer API	11.0.194	x86_64, POWER, Arm64
CUDA cuBLAS	11.1.0.229	x86_64, POWER, Arm64
CUDA cuFFT	10.2.0.218	x86_64, POWER, Arm64
CUDA cuRAND	10.2.1.218	x86_64, POWER, Arm64
CUDA cuSOLVER	10.5.0.218	x86_64, POWER, Arm64
CUDA cuSPARSE	11.1.0.218	x86_64, POWER, Arm64
CUDA NPP	11.1.0.218	x86_64, POWER, Arm64
CUDA nvJPEG	11.1.0.218	x86_64, POWER, Arm64
Nsight Eclipse Plugins	11.0.194	x86_64, POWER
Nsight Compute	2020.1.1.8	x86_64, POWER, Arm64
Nsight Windows NVTX	1.21018621	x86_64, POWER, Arm64

Component Name	Version Information	Supported Architectures
Nsight Systems	2020.3.2.6	x86_64, POWER, Arm64
Nsight Visual Studio Edition (VSE)	2020.1.1.20163	x86_64 (Windows)
NVIDIA Linux Driver	450.51.05	x86_64, POWER, Arm64
NVIDIA Windows Driver	451.48	x86_64 (Windows)

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 2. 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 2 CUDA Toolkit and Compatible Driver Versions

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 11.0.207	>= 450.51.05	>= 451.48
CUDA 11.0.171 RC	>= 450.36.06	>= 451.22
CUDA 10.2.89	>= 440.33	>= 441.22
CUDA 10.1 (10.1.105 general release, and updates)	>= 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

2.3. General CUDA

- CUDA 11.0 adds support for the NVIDIA Ampere GPU microarchitecture (**`compute_80`** and **`sm_80`**).
- CUDA 11.0 adds support for NVIDIA A100 GPUs and systems that are based on A100. The A100 GPU adds the following capabilities for compute via CUDA:
  - Alternate floating point data format Bfloat16 (**`__nv_bfloat16`**) and compute type TF32 (**`tf32`**)
  - Double precision matrix multiply accumulate through the DMMA instruction (see note on WMMA in CUDA C++ and mma in PTX)
  - Support for asynchronous copy instructions that allow copying of data asynchronously (LDGSTS instruction and the corresponding **`cp.async.*`** PTX instructions)
  - Cooperative groups improvements, which allow reduction operation across threads in a warp (using the **`redux.sync`** instruction)
  - Support for hardware partitioning via Multi-Instance GPU (MIG). See the driver release notes on more information on the corresponding NVML APIs and **`nvidia-smi`** CLI tools for configuring MIG instances
- Added the 7.0 version of the Parallel Thread Execution instruction set architecture (ISA). For more details on new (**`sm_80`** target, new instructions, new floating point data types in **`.bf16`**, **`.tf32`**, and new mma shapes) and deprecated instructions, see this section in the PTX documentation.
- CUDA 11.0 adds support for the Arm server platform (arm64 SBSA). Note that with this release, only the following platforms are supported with Tesla V100 GPU:
  - HPE Apollo 70 (using Marvell ThunderX2™ CN99XX)
  - Gigabyte R2851 (using Marvell ThunderX2™ CN99XX)
  - Huawei TaiShan 2280 V2 (using Huawei Kunpeng 920)
- CUDA supports a wide range of Linux and Windows distributions. For a full list of supported operating systems, see system requirements for more information. The following new Linux distributions are supported in CUDA 11.0.

  For x86 (x86_64):

- Red Hat Enterprise Linux (RHEL) 8.1
- Ubuntu 18.04.4 LTS

For Arm (arm64):

- SUSE SLES 15.1

For POWER (ppc64le):

- Red Hat Enterprise Linux (RHEL) 8.1

- CUDA C++ includes support for new data types to support new 16-bit floating point data (with 1-sign bit, 8-bit exponent and 7-bit mantissa): **`__nv_bfloat16`** and **`__nv_bfloat162`**. See **`include/cuda_bf16.hpp`** and the CUDA Math API for more information on the datatype definition and supported arithmetic operations.
- CUDA 11.0 adds the following support for WMMA:
  - Added support for double (FP64) to the list of available input/output types for 8x8x4 shapes (DMMA.884)
  - Added support for **`__nv_bfloat16`** and **`tf32`** precision formats for the HMMA 16x16x8 shape
- Added support for cooperative kernels in CUDA graphs, including stream capture for **`cuLaunchCooperativeKernel`**.
- The CUDA_VISIBLE_DEVICES variable has been extended to add support for enumerating Multiple Instance GPUs (MIG) in NVIDIA A100/GA100 GPUs.
- Added support for PCIe Relaxed Ordering for GPU initiated writes. This is not enabled by default but can be enabled by setting the following module parameter on Linux x86_64: **`NVreg_EnablePCIERelaxedOrderingMode`**.
- CUDA 11.0 adds a specification for inter-task memory ordering in the "API Synchronization" subsection of the PTX memory model and allows CUDA's implementation to be optimized consistent with this addition. In rare cases, code may have assumed a stronger ordering than required by the added specification and may notice a functional regression. The environment variable **`CUDA_FORCE_INTERTASK_SYSTEM_FENCE`** may be set to a value of "0" to disable post-10.2 inter-task fence optimizations, or "1" to enable them for 445 and newer drivers. If the variable is not set, code compiled entirely against CUDA 10.2 or older will disable the optimizations and code compiled against 11.0 or newer will enable them. Code with mixed versions may see a combination.

2.4. CUDA Tools

2.4.1. CUDA Compilers

- The following new compilers are supported as host compilers for the CUDA compiler (nvcc)
  - Clang 9
  - GCC 9
  - PGI 20.1

- ICC 19.1
- Arm C/C++ 19.2

- The default compilation target for nvcc is now **sm_52**. Other older targets are either deprecated or no longer supported. See the Deprecated Features section for more details.
- Added support for Link-Time Optimization (LTO). LTO enables cross-file inlining and optimization when doing separate compilation. To use LTO, add **-dlto** to both the compile and link commands, for example "**nvcc -arch=sm_70 -dlto a.cu b.cu**". LTO is currently in technical preview. See the section titled "Optimization of Separate Compilation" in the nvcc manual for more information.
- nvcc added two new flags ('**-Wdefault-stream-launch**') and ('**-Werror=default-stream-launch**') to generate a warning and an error, respectively, when a stream argument is not explicitly specified in the **<<<...>>>** kernel launch syntax. For example:

```
$ cat j1.cu

__global__ void foo() { }

int main() { foo<<<1,1>>>();

}

$nvcc -Wdefault-stream-launch j1.cu -ptx

j1.cu(2): warning: explicit stream argument not provided in
kernel launch

$nvcc -Werror=default-stream-launch j1.cu -c

j1.cu(2): error: explicit stream argument not provided in kernel
launch
```

- The compiler optimizer now implements more aggressive dead code elimination for **__shared__** variables whose value is not used. For example:

```
//--__device__ void foo() {

__shared__ int xxx;

xxx = 1;

}
```

  In previous CUDA toolkits, the variable "xxx" is still present in the generated PTX. With CUDA 11 or later, the variable may be removed in the generated PTX, because its value is not used. Marking the variable as "**volatile**" will inhibit this compiler optimization.
- In previous CUDA toolkits, NVRTC on Linux incorrectly added "**/usr/include**" to the default header file search path. This issue has been fixed; NVRTC in CUDA 11.0 and later will not implicitly add '**/usr/include**' to the header file search path.

  If some included files are present inside **/usr/include**, the NVRTC **nvrtcCompileProgram()** API call must now be explicitly passed the "**/usr/include**" path with the "**-I**" flag.

- nvcc now allows options that take a single argument to be redefined. If the redefinition is incompatible with the earlier instance, a warning is issued. For example:

  ```
  // the following command line is now accepted, previously nvcc gave an error

  $nvcc -rdc=true -rdc=true -c j1.cu

  // the following command line is now accepted with a warning (due to incompatible redefinition of '-rdc' argument), previously nvcc gave an error

  $nvcc -rdc=true -rdc=false -c j1.cu

  nvcc warning : incompatible redefinition for option 'relocatable-device-code'
  ```
- nvcc implements a new flag '**-extra-device-vectorization**' , which enables more aggressive vectorization of device code.
- Added support for C++17.
- Added support for **__attribute__((visibility("default")))**.

2.4.2. CUDA Developer Tools

- The following developer tools are supported for remote (target) debugging/profiling of applications on macOS hosts:
  - Nsight Compute
  - Nsight Systems
  - cuda-gdb
  - NVVP
- For new features, improvements, and bug fixes in CUPTI, see the changelog.
- For new features, improvements, and bug fixes in Nsight Compute, see the changelog.
- Cuda-gdb is now upgraded to support GDB 8.2.
- A new tool called Compute Sanitizer, for memory and race condition checking, is now included as part of CUDA 11.0.

2.5. CUDA Libraries

This release of the toolkit includes the following updates:

- CUDA Math libraries toolchain uses C++11 features, and a C++11-compatible standard library is required on the host.
- cuBLAS 11.0.0
- cuFFT 10.1.3
- cuRAND 10.2.0
- cuSPARSE 11.0.0
- cuSOLVER 10.4.0

- NPP 11.0.0
- nvJPEG 11.0.0

2.5.1. cuBLAS Library

- cuBLASLt Matrix Multiplication adds support for fused ReLU and bias operations for all floating point types except double precision (FP64).
- Improved batched TRSM performance for matrices larger than 256.
- Many performance improvements have been implemented for the NVIDIA Ampere, Volta, and Turing Architecture based GPUs.
- With this release, on Linux systems, the cuBLAS libraries listed below are now installed in the **`/usr/local/cuda-11.0`** (**`./lib64/`** for lib and **`./include/`** for headers) directories as shared and static libraries.
- The **`cuBLASLt`** logging mechanism can be enabled by setting the following environment variables before launching the target application:
  - CUBLASLT_LOG_LEVEL=<level> - while level is one of the following levels:
    - "0" - Off - logging is disabled (default)
    - "1" - Error - only errors will be logged
    - "2" - Trace - API calls will be logged with their parameters and important information
  - CUBLASLT_LOG_FILE=<value> - while value is a file name in the format of "<file_name>.%i", %i will be replaced with the process id. If CUBLASLT_LOG_FILE is not defined, the log messages are printed to stdout.
- For matrix multiplication APIs:
  - **`cublasGemmEx`**, **`cublasGemmBatchedEx`**, **`cublasGemmStridedBatchedEx`** and **`cublasLtMatmul`** has new data type support for BFLOAT16 (CUDA_R_16BF).
  - The newly introduced **`computeType_t`** changes function prototypes on the API: **`cublasGemmEx`**, **`cublasGemmBatchedEx`**, and **`cublasGemmStridedBatchedEx`** have a new signature that uses **`cublasComputeType_t`** for the **`computeType`** parameter. Backward compatibility is ensured with internal mapping for C users and with added overload for C++ users.
  - **`cublasLtMatmulDescCreate`**, **`cublasLtMatmulAlgoGetIds`**, and **`cublasLtMatmulAlgoInit`** have new signatures that use **`cublasComputeType_t`**.
  - A new compute type TensorFloat32 (TF32) has been added to provide tensor core acceleration for FP32 matrix multiplication routines with full dynamic range and increased precision compared to BFLOAT16.
  - New compute modes Default, Pedantic, and Fast have been introduced to offer more control over compute precision used.
  - **`*Init`** versions of **`*Create`** functions are introduced in cublasLt to allow for simple wrappers that hold all descriptors on stack.
  - Experimental feature of cuBLASLt API logging is introduced.
  - Tensor cores are now enabled by default for half-, and mixed-precision- matrix multiplications.
  - Double precision tensor cores (DMMA) are used automatically.

- Tensor cores can now be used for all sizes and data alignments and for all GPU architectures:
  - Selection of these kernels through cuBLAS heuristics is automatic and will depend on factors such as math mode setting as well as whether it will run faster than the non-tensor core kernels.
  - Users should note that while these new kernels that use tensor cores for all unaligned cases are expected to perform faster than non-tensor core based kernels but slower than kernels that can be run when all buffers are well aligned.

2.5.2. cuFFT Library

- cuFFT now accepts __nv_bfloat16 input and output data type for power-of-two sizes with single precision computations within the kernels.
- Reoptimized power of 2 FFT kernels on Volta and Turing architectures.

2.5.3. cuSPARSE Library

- Added new Generic APIs for Axpby (cusparseAxpby), Scatter (cusparseScatter), Gather (cusparseGather), Givens rotation (cusparseRot). __nv_bfloat16/ __nv_bfloat162 data types and 64-bit indices are also supported.
- This release adds the following features for cusparseSpMM:
  - Support for row-major layout for cusparseSpMM for both CSR and COO format
  - Support for 64-bit indices
  - Support for __nv_bfloat16 and __nv_bfloat162 data types
  - Support for the following strided *batch* mode:
    - Ci=A#Bi
    - Ci=Ai#B
    - Ci=Ai#Bi
- Added new generic APIs and improved performance for sparse matrix-sparse matrix multiplication (SpGEMM): **`cusparseSpGEMM_workEstimation`**, **`cusparseSpGEMM_compute`**, and **`cusparseSpGEMM_copy`**.
- SpVV: added support for **`__nv_bfloat16`**.

2.5.4. cuSOLVER Library

- Add 64-bit API of GESVD. The new routine cusolverDnGesvd_bufferSize() fills the missing parameters in 32-bit API cusolverDn[S|D|C|Z]gesvd_bufferSize() such that it can estimate the size of the workspace accurately.
- Added the single process multi-GPU Cholesky factorization capabilities POTRF, POTRS and POTRI in cusolverMG library.
- Added 64-bit APIs for **`getrf`**, **`getrs`**, **`potrf`**, **`potrs`**, **`geqrf`**, **`syevd`** and **`syevdx`**.
- This release adds more control and helpful functionalities for the Tensor Cores Accelerated Iterative Refinement Solver TCAIRS.

- In addition to the previously released TCAIRS-LU based solver a new TCAIRS-QR based solver for real and complex systems with one or multiple right hand sides is introduced.
- In addition to the FP64, FP32 and FP16 computational precisions two new computational precisions types are supported: the BFLOAT16 and the TensorFloat32 (TF32). Both TCAIRS-LU and TCAIRS-QR come with the five computational precisions options. Tensor Float (TF32), introduced with NVIDIA Ampere Architecture GPUs, is the most robust tensor core accelerated compute mode for the iterative refinement solver. It is able to solve the widest range of problems in HPC arising from different applications and provides up to 4X and 5X speedup for real and complex systems, respectively. On Volta and Turing architecture GPUs, half precision tensor core acceleration is recommended. In cases where the iterative refinement solver fails to converge to the desired accuracy (double precision in most cases), it is recommended to use full double precision factorization and solve (such as [D,Z]GETRF and [D,Z]GETRS or cusolverDn[DD,ZZ]gesv).
- TCAIRS (LU and QR) are released with easy LAPACK-style APIs (drop-in replacement) as well as expert generic APIs that give users a lot of control of the internal of the solver. These support all five computational precisions.
- Simple and Expert APIs now support all five computational precisions.
- Expert TCAIRS solvers APIs allow users to choose between 4 methods of refinement.
- Expert TCAIRS solvers APIs now support a no-refinement option which means they behave as standard Xgesv/Xgels solvers without refinement.

- Performance improvements of the TCAIRS solver for NVIDIA Ampere, Volta, and Turing Architecture based GPUs.

2.5.5. NVIDIA Performance Primitives (NPP)

- Batched Image Label Markers Compression that removes sparseness between marker label IDs output from LabelMarkers call.
- Image Flood Fill functionality fills a connected region of an image with a specified new value.
- Added batching support for nppiLabelMarkersUF functions.
- Added the **`nppiCompressMarkerLabelsUF_32u_C1IR`** function.
- Added **`nppiSegmentWatershed`** functions.
- Added sample apps on GitHub demonstrating the use of NPP application managed stream contexts along with watershed segmentation and batched and compressed UF image label markers functions.
- Added support for non-blocking streams.

2.5.6. nvJPEG

- nvJPEG allows the user to allocate separate memory pools for each chroma subsampling format. This helps avoid memory re-allocation overhead. This can be controlled by passing the newly added flag **`NVJPEG_FLAGS_ENABLE_MEMORY_POOLS`** to the **`nvjpegCreateEx`** API.

- nvJPEG encoder now allow compressed bitstream on the GPU Memory.
- Hardware accelerated decode is now supported on NVIDIA A100.
- The nvJPEG decode API (**`nvjpegDecodeJpeg()`**) now has the flexibility to select the backend when creating **`nvjpegJpegDecoder_t`** object. The user has the option to call this API instead of making three separate calls to **`nvjpegDecodeJpegHost()`**, **`nvjpegDecodeJpegTransferToDevice()`**, and **`nvjpegDecodeJpegDevice()`**.

2.5.7. CUDA Math API

- Add arithmetic support for **`__nv_bfloat16`** floating-point data type with 8 bits of exponent, 7 explicit bits of mantissa.
- Performance and accuracy improvements in single precision math functions: **`fmodf`**, **`expf`**, **`exp10f`**, **`sinhf`**, and **`coshf`**.

2.6. Deprecated and Dropped Features

The following features are deprecated or dropped in the current release of the CUDA software. Deprecated 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

- Support for Red Hat Enterprise Linux (RHEL) and CentOS 6.x is dropped.
- Support for Kepler **`sm_30`** and **`sm_32`** architecture based products is dropped.
- Support for the following compute capabilities are deprecated in the CUDA Toolkit:
  - **`sm_35`** (Kepler)
  - **`sm_37`** (Kepler)
  - **`sm_50`** (Maxwell)

  For more information on GPU products and compute capability, see https://developer.nvidia.com/cuda-gpus.
- Support for Linux cluster packages is dropped.
- CUDA 11.0 does not support macOS for developing and running CUDA applications. Note that some of the CUDA developer tools are still supported on macOS hosts for remote (target) debugging and profiling. See the CUDA Tools section for more information.
- CUDA 11.0 no longer supports development of CUDA applications on the following Windows distributions:
  - Windows 7
  - Windows 8
  - Windows Server 2012 R2

- nvGraph is no longer included as part of the CUDA Toolkit installers. See the cuGraph project as part of RAPIDS; the project includes algorithms from nvGraph and more.
- The context creation flag CU_CTX_MAP_HOST (to support mapped pinned allocations) is deprecated and will be removed in a future release of CUDA.

CUDA Developer Tools

- Nsight Eclipse Edition standalone is dropped in CUDA 11.0.
- Nsight Compute does not support profiling on Pascal architectures.
- Nsight VSE, Nsight EE Plugin, cuda-gdb, nvprof, Visual Profiler, and memcheck are reducing support for the following architectures:
  - Support for Kepler **`sm_30`** and **`sm_32`** architecture based products (deprecated since CUDA 10.2) has beeen dropped.
  - Support for the following compute capabilities (deprecated since CUDA 10.2) will be dropped in an upcoming CUDA release:
    - **`sm_35`** (Kepler)
    - **`sm_37`** (Kepler)
    - **`sm_50`** (Maxwell)

CUDA Libraries - cuBLAS

- Algorithm selection in **`cublasGemmEx`** APIs (including batched variants) is non-functional for NVIDIA Ampere Architecture GPUs. Regardless of selection it will default to a heuristics selection. Users are encouraged to use the **`cublasLt`** APIs for algorithm selection functionality.
- The matrix multiply math mode **`CUBLAS_TENSOR_OP_MATH`** is being deprecated and will be removed in a future release. Users are encouraged to use the new **`cublasComputeType_t`** enumeration to define compute precision.

CUDA Libraries -- cuSOLVER

- TCAIRS-LU expert **`cusolverDnIRSXgesv()`** and some of its configuration functions undergo a minor API change.

CUDA Libraries -- cuSPARSE

The following functions have been removed:

- **`cusparse<t>gemmi()`**
- **`cusparseXaxpyi`**, **`cusparseXgthr`**, **`cusparseXgthrz`**, **`cusparseXroti`**, **`cusparseXsctr`**
- Hybrid format enums and helper functions: **`cusparseHybPartition_t`**, **`cusparseHybPartition_t`**, **`cusparseCreateHybMat`**, **`cusparseDestroyHybMat`**
- Triangular solver enums and helper functions: **`cusparseSolveAnalysisInfo_t`**, **`cusparseCreateSolveAnalysisInfo`**, **`cusparseDestroySolveAnalysisInfo`**
- Sparse dot product: **`cusparseXdoti`**, **`cusparseXdotci`**
- Sparse matrix-vector multiplication: **`cusparseXcsrmv`**, **`cusparseXcsrmv_mp`**
- Sparse matrix-matrix multiplication: **`cusparseXcsrmm`**, **`cusparseXcsrmm2`**

- Sparse triangular-single vector solver: **cusparseXcsrsv_analysis**, **cusparseCsrsv_analysisEx**, **cusparseXcsrsv_solve**, **cusparseCsrsv_solveEx**
- Sparse triangular-multiple vectors solver: **cusparseXcsrsm_analysis**, **cusparseXcsrsm_solve**
- Sparse hybrid format solver: **cusparseXhybsv_analysis**, **cusparseShybsv_solve**
- Extra functions: **cusparseXcsrgeamNnz**, **cusparseScsrgeam**, **cusparseXcsrgemmNnz**, **cusparseXcsrgemm**
- Incomplete Cholesky Factorization, level 0: **cusparseXcsric0**
- Incomplete LU Factorization, level 0: **cusparseXcsrilu0**, **cusparseCsrilu0Ex**
- Tridiagonal Solver: **cusparseXgtsv**, **cusparseXgtsv_nopivot**
- Batched Tridiagonal Solver: **cusparseXgtsvStridedBatch**
- Reordering: **cusparseXcsc2hyb**, **cusparseXcsr2hyb**, **cusparseXdense2hyb**, **cusparseXhyb2csc**, **cusparseXhyb2csr**, **cusparseXhyb2dense**

The following functions have been deprecated:

- SpGEMM: **cusparseXcsrgemm2_bufferSizeExt**, **cusparseXcsrgemm2Nnz**, **cusparseXcsrgemm2**

CUDA Libraries -- nvJPEG

- The following multiphase APIs have been removed:
  - **nvjpegStatus_t NVJPEGAPI nvjpegDecodePhaseOne**
  - **nvjpegStatus_t NVJPEGAPI nvjpegDecodePhaseTwo**
  - **nvjpegStatus_t NVJPEGAPI nvjpegDecodePhaseThree**
  - **nvjpegStatus_t NVJPEGAPI nvjpegDecodeBatchedPhaseOne**
  - **nvjpegStatus_t NVJPEGAPI nvjpegDecodeBatchedPhaseTwo**

2.7. Resolved Issues

2.7.1. General CUDA

- Fixed an issue where GPU passthrough on arm64 systems was not functional. GPU passthrough is now supported on arm64, but there may be a small performance impact to workloads (compared to bare-metal) on some system configurations.
- Fixed an issue where starting X on systems with arm64 CPUs and NVIDIA GPUs would result in a crash.

2.7.2. CUDA Tools

- Fixed an issue where NVCC throws a compilation error when a value > 32768 was used in an **__attribute__((aligned(value)))**.
- Fixed an issue in PTXAS where a 64-bit integer modulo operation resulted in illegal memory access.

- Fixed an issue with NVCC where code using the **__is_implicitly_default_constructible** type trait would result in an access violation.
- Fixed an issue where NVRTC (**nvrtcCompileProgram()**) would enter into infinite loops triggered by some code patterns.
- Fixed implementation of **nvrtcGetTypeName()** on Windows to call **UnDecorateSymbolName()** with the correct flags. The string returned by **UnDecorateSymbolName()** may contain Microsoft specific keywords '**__cdecl**' and '**__ptr64**'. NVRTC has been updated to define these symbols to empty during compilation. This allows use of names returned by **nvrtcGetTypeName()** to be directly used in **nvrtcAddNameExpression**/**nvrtcGetLoweredName()**.
- Fixed a compilation time issue in NVCC to improve handling of large numbers of explicit specialization of function templates.

2.7.3. cuFFT Library

- Reduced R2C/C2R plan memory usage to previous levels.
- Resolved bug introduced in 10.1 update 1 that caused incorrect results when using custom strides, batched 2D plans and certain sizes on Volta and later.

2.7.4. cuRAND Library

- Introduced CURAND_ORDERING_PSEUDO_LEGACY ordering. Starting with CUDA 10.0, the ordering of random numbers returned by MTGP32 and MRG32k3a generators are no longer the same as previous releases despite being guaranteed by the documentation for the CURAND_ORDERING_PSEUDO_DEFAULT setting. The CURAND_ORDERING_PSEUDO_LEGACY provides pre-CUDA 10.0 ordering for MTGP32 and MRG32k3a generators.
- Starting with CUDA 11.0 CURAND_ORDERING_PSEUDO_DEFAULT is the same as CURAND_ORDERING_PSEUDO_BEST for all generators except MT19937. Only CURAND_ORDERING_PSEUDO_LEGACY is guaranteed to provide the same for all future cuRAND releases.

2.7.5. cuSOLVER Library

- Fixed an issue where SYEVD/SYGVD would fail and return error code 7 if the matrix is zero and the dimension is bigger than 25.
- Fixed a race condition of GETRF when running with other kernels concurrently.
- Fixed the pivoting strategy of [c|z]getrf to be compliant with LAPACK.
- Fixed NAN and INF values that might result in the TCAIRS-LU solver when FP16 was used and matrix entries are outside FP16 range.
- Fixed the pivoting strategy of [c|z]getrf to be compliant with LAPACK.
- Previously, **cusolverSpDcsrlsvchol** could overflow 32-bit signed integer when zero fill-in is huge. Such overflow causes memory corruption. **cusolverSpDcsrlsvchol** now returns **CUSOLVER_STATUS_ALLOC_FAILED** when integer overflow happens.

2.7.6. CUDA Math API

- Corrected documented maximum ulp error thresholds in **`erfcinvf`** and **`powf`**.
- Improved **`cuda_fp16.h`** interoperability with Visual Studio C++ compiler.
- Updated libdevice user guide and CUDA math API definitions for **`j1`**, **`j1f`**, **`fmod`**, **`fmodf`**, **`ilogb`**, and **`ilogbf`** math functions.

2.7.7. NVIDIA Performance Primitives (NPP)

- Stability and performance fixes to Image Label Markers and Image Label Markers Compression.
- Improved quality of **`nppiLabelMarkersUF`** functions.
- **`nppiCompressMarkerLabelsUF_32u_C1IR`** can now handle a huge number of labels generated by the **`nppiLabelMarkersUF`** function.

2.7.8. CUDA Profiling Tools Interface (CUPTI)

- The **`cuptiFinalize()`** API now allows on-demand detachability of the profiling tool.

2.8. Known Issues

2.8.1. General CUDA

- The nanosleep PTX instruction for Volta and Turing is not supported in this release of CUDA. It may be fully supported in a future releaseof CUDA. There may be references to nanosleep in the compiler headers (such as **`include/crt/sm_70_rt*`**). Developers are encouraged to not use this instruction in their CUDA applications on Volta and Turing until it is fully supported.
- Read-only memory mappings (via **`CU_MEM_ACCESS_FLAGS_PROT_READ`** in **`CUmemAccess_flags`**) with **`cuMemSetAccess()`** API will result in an error. Read-only memory mappings are currently not supported and may be added in a future release of CUDA.
- Note that the R450 driver bundled with this release of CUDA 11 does not officially support Windows 10 May 2020 Update and may have issues
- GPU workloads are executed on GPU hardware engines. On Windows, these engines are represented by “nodes”. With Hardware Scheduling disabled for Windows 10 May 2020 Update, some NVIDIA GPU engines are represented by virtual nodes, and multiple virtual nodes may represent more than one GPU hardware engine. This is done to achieve better parallel execution of workloads. Examples of these virtual nodes are “Cuda”, “Compute_0”, “Compute_1”, and “Graphics_1” as shown in Windows Task Manager. These correspond to the same underlying hardware engines as the “3D” node in Windows Task Manager. With Hardware Scheduling enabled, the virtual nodes are no longer needed, and Task

Manager shows only the “3D”node for the previous “3D” node and multiple virtual nodes shown before, combined. CUDA is still supported in this scenario.

2.8.2. CUDA Tools

- The legacy profiling tools nvprof and NVVP do not support the NVIDIA Ampere architecture.
- Arithmetic is not supported on **`__nv_bfloat16`** floating point variables in the Nsight debugger watch window.
- In some cases, cuda-gdb has a dependency on Python that can be resolved by installing the libpython-dev packages on Linux. For example, on Ubuntu use: **`sudo apt install libpython-dev`**.
- For remote debugging on macOS with cuda-gdb, disassembly of code is not supported and may return an error. This issue will be addressed in the production release of CUDA 11.0.

2.8.3. CUDA Compiler

- **`Sample 0_Simple/simpleSeparateCompilation`** fails to build with the error "cc: unknown target 'gcc_ntox86". The workaround to allow the build to pass is by passing additionally **EXTRA_NVCCFLAGS="-arbin $QNX_HOST/usr/bin/aarch64-unknown-nto-qnx7.0.0-ar**".

2.8.4. cuFFT Library

- cuFFT modifies C2R input buffer for some non-strided FFT plans.
- There is a known issue with certain cuFFT plans that causes an assertion in the execution phase of certain plans. This applies to plans with all of the following characteristics: real input to complex output (R2C), in-place, native compatibility mode, certain even transform sizes, and more than one batch.

2.8.5. NVIDIA Performance Primitives (NPP)

- The **`nppiCopy`** API is limited by CUDA thread for large image size. Maximum image limits is a minimum of 16 * 65,535 = 1,048,560 horizontal pixels of any data type and number of channels and 8 * 65,535 = 524,280 vertical pixels for a maximum total of 549,739,036,800 pixels.

2.8.6. nvJPEG

- **`NVJPEG_BACKEND_GPU_HYBRID`** has an issue when handling bit-streams which have corruption in the scan.

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