NVIDIA CUDA Toolkit

Release Notes for CUDA 11.6

RN-06722-001_v11.6 | April 2022

Table of Contents

List of Tables

Chapter 1. CUDA 11.6 Release Notes

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

Note: The release notes have been reorganized into two major sections: the general CUDA release notes, and the CUDA libraries release notes including historical information for 11.x releases.

1.1. CUDA Toolkit Major Component Versions

CUDA Components

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

For CUDA 11.6, the table below indicates the versions:

Table 1. CUDA 11.6 Component Versions

Component Name	Version Information	Supported Architectures
CUDA C++ Core Compute Libraries	11.6.55	x86_64, POWER, Arm64
CUDA Runtime (cudart)	11.6.55	x86_64, POWER, Arm64
cuobjdump	11.6.55	x86_64, POWER, Arm64
CUPTI	11.6.55	x86_64, POWER, Arm64
CUDA cuxxfilt (demangler)	11.6.55	x86_64, POWER, Arm64
CUDA Demo Suite	11.6.55	x86_64
CUDA GDB	11.6.55	x86_64, POWER, Arm64
CUDA Memcheck	11.6.55	x86_64, POWER
CUDA Nsight	11.6.55	x86_64, POWER
CUDA NVCC	11.6.55	x86_64, POWER, Arm64
CUDA nvdisasm	11.6.55	x86_64, POWER, Arm64
CUDA NVML Headers	11.6.55	x86_64, POWER, Arm64

Component Name	Version Information	Supported Architectures
CUDA nvprof	11.6.55	x86_64, POWER, Arm64
CUDA nvprune	11.6.55	x86_64, POWER, Arm64
CUDA NVRTC	11.6.55	x86_64, POWER, Arm64
CUDA NVTX	11.6.55	x86_64, POWER, Arm64
CUDA NVVP	11.6.58	x86_64, POWER
CUDA Samples	11.6.101	x86_64, POWER, Arm64
CUDA Compute Sanitizer API	11.6.55	x86_64, POWER, Arm64
CUDA cuBLAS	11.8.1.74	x86_64, POWER, Arm64
CUDA cuFFT	10.7.0.55	x86_64, POWER, Arm64
CUDA cuFile	1.2.0.100	x86_64
CUDA cuRAND	10.2.9.55	x86_64, POWER, Arm64
CUDA cuSOLVER	11.3.2.55	x86_64, POWER, Arm64
CUDA cuSPARSE	11.7.1.55	x86_64, POWER, Arm64
CUDA NPP	11.6.0.55	x86_64, POWER, Arm64
CUDA nvJPEG	11.6.0.55	x86_64, POWER, Arm64
Nsight Compute	2022.1.0.12	x86_64, POWER, Arm64 (CLI only)
NVTX	1.21018621	x86_64, POWER, Arm64
Nsight Systems	2021.5.2.53	x86_64, POWER, Arm64 (CLI only)
Nsight Visual Studio Edition (VSE)	2022.1.0.21343	x86_64 (Windows)
nvidia_fs[1]	2.10.3	x86_64
Visual Studio Integration	11.6.55	x86_64 (Windows)
NVIDIA Linux Driver	510.39.01	x86_64, POWER, Arm64
NVIDIA Windows Driver	511.23	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 3. 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-compatibility-and-upgrades.

Note: Starting with CUDA 11.0, the toolkit components are individually versioned, and the toolkit itself is versioned as shown in the table below.

[1] Only available on select Linux distros

The minimum required driver version for CUDA minor version compatibility is shown below. CUDA minor version compatibility is described in detail in https://docs.nvidia.com/deploy/cuda-compatibility/index.html

Table 2. CUDA Toolkit and Minimum Required Driver Version for CUDA Minor Version Compatibility

CUDA Toolkit	Minimum Required Driver Version for CUDA Minor Version Compatibility*	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 11.6.x	>=450.80.02	>=452.39
CUDA 11.5.x	>=450.80.02	>=452.39
CUDA 11.4.x	>=450.80.02	>=452.39
CUDA 11.3.x	>=450.80.02	>=452.39
CUDA 11.2.x	>=450.80.02	>=452.39
CUDA 11.1 (11.1.0)	>=450.80.02	>=452.39
CUDA 11.0 (11.0.3)	>=450.36.06**	>=451.22**

* Using a Minimum Required Version that is **different** from Toolkit Driver Version could be allowed in compatibility mode -- please read the CUDA Compatibility Guide for details.

** CUDA 11.0 was released with an earlier driver version, but by upgrading to Tesla Recommended Drivers 450.80.02 (Linux) / 452.39 (Windows), minor version compatibility is possible across the CUDA 11.x family of toolkits.

The version of the development NVIDIA GPU Driver packaged in each CUDA Toolkit release is shown below.

Table 3. CUDA Toolkit and Corresponding Driver Versions

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 11.6 GA	>=510.39.01	>=511.23
CUDA 11.5 Update 1	>=495.29.05	>=496.13
CUDA 11.5 GA	>=495.29.05	>=496.04
CUDA 11.4 Update 3	>=470.82.01	>=472.50
CUDA 11.4 Update 2	>=470.57.02	>=471.41
CUDA 11.4 Update 1	>=470.57.02	>=471.41
CUDA 11.4.0 GA	>=470.42.01	>=471.11
CUDA 11.3.1 Update 1	>=465.19.01	>=465.89
CUDA 11.3.0 GA	>=465.19.01	>=465.89

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 11.2.2 Update 2	>=460.32.03	>=461.33
CUDA 11.2.1 Update 1	>=460.32.03	>=461.09
CUDA 11.2.0 GA	>=460.27.03	>=460.82
CUDA 11.1.1 Update 1	>=455.32	>=456.81
CUDA 11.1 GA	>=455.23	>=456.38
CUDA 11.0.3 Update 1	>= 450.51.06	>= 451.82
CUDA 11.0.2 GA	>= 450.51.05	>= 451.48
CUDA 11.0.1 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

1.2. General CUDA

11.6

- Added a new API, `cudaGraphNodeSetEnabled()`, to allow disabling nodes in an instantiated graph. Support is limited to kernel nodes in this release. A corresponding API, `cudaGraphNodeGetEnabled()`, allows querying the enabled state of a node.
- Full release of 128-bit integer (`__int128`) data type including compiler and developer tools support. The host-side compiler must support the `__int128` type to use this feature.
- Cooperative groups namespace is updated with new functions to improve consistency in naming, function scope, and unit dimension/size:

Implicit Group/Member	Threads	Blocks

`thread_block::`	`dim_threads`
	`num_threads`
	`thread_rank`
	`thread_index`
		(not needed)

`grid_group::`	`num_threads`
	`thread_rank`
		`num_blocks`
		`block_rank`
		`block_index`

- Added ability to disable NULL kernel graph node launches.
- Added new NVML public APIs for querying functionality under Wayland.
- Added L2 cache control descriptors for atomics.
- Large CPU page support for UVM managed memory.

1.3. CUDA Compilers

11.6

- **VS2022 Support:** CUDA 11.6 officially supports the latest VS2022 as host compiler. A separate Nsight Visual Studio installer 2022.1.1 must be downloaded from **here.** A future CUDA release will have the Nsight Visual Studio installer with VS2022 support integrated into it.
- **New instructions in public PTX:** New instructions for bit mask creation - BMSK and sign extension - SZEXT are added to the public PTX ISA. You can find documentation for these instructions in the PTX ISA guide: **BMSK** and **SZEXT**.
- **Unused Kernel Optimization:** In CUDA 11.5, unused kernel pruning was introduced with the potential benefits of reducing binary size and improving performance through more

efficient optimizations. This was an opt-in feature but in 11.6, this feature is enabled by default. As mentioned in the 11.5 blog **here**, there is an opt-out flag that can be used in case it becomes necessary for debug purposes or for other special situations.

```
$ nvcc -rdc=true user.cu testlib.a -o user -Xnvlink -ignore-host-info
```

- In addition to the `-arch=all` and `-arch=all-major` options added in CUDA 11.5, NVCC introduced `-arch= native` in CUDA 11.5 update1. This `-arch=native` option is a convenient way for users to let NVCC determine the right target architecture to compile the CUDA device code to based on the GPU installed on the system. This can be particularly helpful for testing when applications are run on the same system they are compiled in.
- **Generate PTX from nvlink:** Using the following command line, device linker, nvlink will produce PTX as an output in addition to CUBIN:

  ```
  nvcc -dlto -dlink -ptx
  ```

  Device linking by nvlink is the final stage in the CUDA compilation process. Applications that have multiple source translation units have to be compiled in separate compilation mode. LTO (introduced in CUDA 11.4) allowed nvlink to perform optimizations at device link time instead of at compile time so that separately compiled applications with several translation units can be optimized to the same level as whole program compilations with a single translation unit. However, without the option to output PTX, applications that cared about forward compatibility of device code could not benefit from Link Time Optimization or had to constrain the device code to a single source file.

  With the option for nvlink that performs LTO to generate the output in PTX, customer applications that require forward compatibility across GPU architectures can span across multiple files and can also take advantage of Link Time Optimization.
- **Bullseye support:** NVCC compiled source code will work with code coverage tool Bullseye. The code coverage is only for the CPU or the host functions. Code coverage for device function is not supported through bullseye.
- **INT128 developer tool support:** In 11.5, CUDA C++ support for 128-bit was added. In this release, developer tools supports the datatype as well. With the latest version of libcu++, int 128 data type is supported by math functions.

1.4. Resolved Issues

1.4.1. CUDA Compilers

11.5. Update 1

- When using the `--fmad=false` compiler option, even the explicitly requested fused multiply-add instructions were decomposed into separate multiply and add, leading to loss of algorithm semantics intended by the programmer. One of the consequences was that CUDA Math APIs could not be trusted to deliver correct results; worst case errors became unbounded. This issue was introduced in 11.5, and is now resolved.

- Fixed a compiler optimization bug that may move memory access instructions across memory barriers that may lead to incorrect runtime results with certain synchronization dependencies.
- An issue in the PTX optimizer sometimes produced incorrect results. This issue is resolved.

11.5

- Linking with cubins larger than 2 GB is now supported.
- Certain C++17 features that were backported to C++14 in MSVC are now supported.
- An issue with the use of lambda function when an object is passed-by-value is resolved. https://github.com/Ahdhn/nvcc_bug_maybe

1.5. 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

- The `cudaDeviceSynchronize()` function used for on-device fork/join parallelism is deprecated in preparation for a replacement programming model with higher performance. These functions continue to work in this release, but the tools will emit a warning about the upcoming change.
- CentOS Linux 8 has reached End-of-Life on Dec 31, 2021 and support for this OS is now deprecated in the CUDA Toolkit. CentOS Linux 8 support will be completely removed in a future release.

1.6. Known Issues

1.6.1. General CUDA

- Intermittent crashes were seen when CUDA binaries were running on a system with a GLIBC version older than 2.17-106.el7_2.1. This is due to a known bug in older versions of GLIBC (Bug reference: https://bugzilla.redhat.com/show_bug.cgi?id=1293976) and has been fixed in later versions (>= glibc-2.17-107.el7).

Chapter 2. CUDA Libraries

This section covers CUDA Libraries release notes for 11.x releases.

- CUDA Math Libraries toolchain uses C++11 features, and a C++11-compatible standard library (libstdc++ >= 20150422) is required on the host.
- CUDA Math libraries are no longer shipped for SM30 and SM32.
- Support for the following compute capabilities are deprecated for all libraries:
  - sm_35 (Kepler)
  - sm_37 (Kepler)

2.1. cuBLAS Library

2.1.1. cuBLAS: Release 11.6

- **New Features**
  - New epilogue options have been added to support fusion in DLtraining: `CUBLASLT_EPILOGUE_{DRELU,DGELU}` which are similar to `CUBLASLT_EPILOGUE_{DRELU,DGELU}_BGRAD` but don't compute bias gradient.
- **Resolved Issues**
  - Some syrk-related functions (`cublas{D,Z}syrk, cublas{D,Z}syr2k, cublas{D,Z}syrkx`) may fail for matrices which size is greater than 2^31.

2.1.2. cuBLAS: Release 11.4 Update 3

- **Resolved Issues**
  - Some cublas and cublasLt functions sometimes returned `CUBLAS_STATUS_EXECUTION_FAILED` if the dynamic library was loaded and unloaded several times during application lifetime within the same CUDA context. This issue has been resolved.

2.1.3. cuBLAS: Release 11.4 Update 2

- **New Features**
  - Vector (and batched) alpha support for per-row scaling in TN int32 math Matmul with int8 output. See `CUBLASLT_POINTER_MODE_ALPHA_DEVICE_VECTOR_BETA_HOST` and `CUBLASLT_MATMUL_DESC_ALPHA_VECTOR_BATCH_STRIDE`.
  - New epilogue options have been added to support fusion in DLtraining: `CUBLASLT_EPILOGUE_BGRADA` and `CUBLASLT_EPILOGUE_BGRADB` which compute bias gradients based on matrices A and B respectively.
  - New auxiliary functions `cublasGetStatusName()`, `cublasGetStatusString()` have been added to cuBLAS that return the string representation and the description of the cuBLAS status (`cublasStatus_t`) respectively. Similarly, `cublasLtGetStatusName()`, `cublasLtGetStatusString()` have been added to cuBlasLt.
- **Known Issues**
  - cublasGemmBatchedEx() and cublas<t>gemmBatched() check the alignment of the input/output arrays of the pointers like they were the pointers to the actual matrices. These checks are irrelevant and will be disabled in future releases. This mostly affects half-precision inputGEMMs which might require 16-byte alignment, and array of pointers could only be aligned by 8-byte boundary.
- **Resolved Issues**
  - cublasLtMatrixTransform can now operate on matrices with dimensions greater than 65535.
  - Fixed out-of-bound access in GEMM and Matmul functions, when split K or non-default epilogue is used and leading dimension of the output matrix exceeds `int32_t` limit.
  - NVBLAS now uses lazy loading of the CPU BLAS library on Linux to avoid issues caused by preloading `libnvblas.so` in complex applications that use `fork` and similar APIs.
  - Resolved symbols name conflict when using cuBlasLt static library with static TensorRT or cuDNN libraries.

2.1.4. cuBLAS: Release 11.4

- **Resolved Issues**
  - Some gemv cases were producing incorrect results if the matrix dimension (n or m) was large, for example 2^20.

2.1.5. cuBLAS: Release 11.3 Update 1

- **New Features**

- Some new kernels have been added for improved performance but have the limitation that only host pointers are supported for scalars (for example, alpha and beta parameters). This limitation is expected to be resolved in a future release.
- New epilogues have been added to support fusion in ML training. These include:
  - ReLuBias and GeluBias epilogues that produce an auxiliary output which is used on backward propagation to compute the corresponding gradients.
  - DReLuBGrad and DGeluBGrad epilogues that compute the backpropagation of the corresponding activation function on matrix C, and produce bias gradient as a separate output. These epilogues require auxiliary input mentioned in the bullet above.

- **Resolved Issues**
  - Some tensor core accelerated strided batched GEMM routines would result in misaligned memory access exceptions when batch stride wasn't a multiple of 8.
  - Tensor core accelerated cublasGemmBatchedEx (pointer-array) routines would use slower variants of kernels assuming bad alignment of the pointers in the pointer array. Now it assumes that pointers are well aligned, as noted in the documentation.
- **Known Issues**
  - To be able to access the fastest possible kernels through `cublasLtMatmulAlgoGetHeuristic()` you need to set CUBLASLT_MATMUL_PREF_POINTER_MODE_MASK in search preferences to CUBLASLT_POINTER_MODE_MASK_HOST or CUBLASLT_POINTER_MODE_MASK_NO_FILTERING. By default, heuristics query assumes the pointer mode may change later and only returns algo configurations that support both _HOST and _DEVICE modes. Without this, newly added kernels will be excluded and it will likely lead to a performance penalty on some problem sizes.
- **Deprecated Features**
  - Linking with static cublas and cublasLt libraries on Linux now requires using gcc-5.2 and compatible or higher due to C++11 requirements in these libraries.

2.1.6. cuBLAS: Release 11.3

- **Known Issues**
  - The planar complex matrix descriptor for batched matmul has inconsistent interpretation of batch offset.
  - Mixed precision operations with reduction scheme CUBLASLT_REDUCTION_SCHEME_OUTPUT_TYPE (might be automatically selected based on problem size by `cublasSgemmEx()` or `cublasGemmEx()` too, unless `CUBLAS_MATH_DISALLOW_REDUCED_PRECISION_REDUCTION` math mode bit is set) not only stores intermediate results in output type but also accumulates them internally in the same precision, which may result in lower than expected accuracy. Please use `CUBLASLT_MATMUL_PREF_REDUCTION_SCHEME_MASK` or

`CUBLAS_MATH_DISALLOW_REDUCED_PRECISION_REDUCTION` if this results in numerical precision issues in your application.

2.1.7. cuBLAS: Release 11.2

- **Known Issues**
  - `cublas<s/d/c/z>Gemm()` with very large n and m=k=1 may fail on Pascal devices.

2.1.8. cuBLAS: Release 11.1 Update 1

- **New Features**
  - cuBLASLt Logging is officially stable and no longer experimental. cuBLASLt Logging APIs are still experimental and may change in future releases.
- **Resolved Issues**
  - cublasLt Matmul fails on Volta architecture GPUs with `CUBLAS_STATUS_EXECUTION_FAILED` when n dimension > 262,137 and epilogue bias feature is being used. This issue exists in 11.0 and 11.1 releases but has been corrected in 11.1 Update 1

2.1.9. cuBLAS: Release 11.1

- **Resolved Issues**
  - A performance regression in the `cublasCgetrfBatched` and `cublasCgetriBatched` routines has been fixed.
  - The IMMA kernels do not support padding in matrix C and may corrupt the data when matrix C with padding is supplied to `cublasLtMatmul`. A suggested work around is to supply matrix C with leading dimension equal to 32 times the number of rows when targeting the IMMA kernels: computeType = CUDA_R_32I and CUBLASLT_ORDER_COL32 for matrices A,C,D, and CUBLASLT_ORDER_COL4_4R2_8C (on NVIDIA Ampere GPU architecture or Turing architecture) or CUBLASLT_ORDER_COL32_2R_4R4 (on NVIDIA Ampere GPU architecture) for matrix B. Matmul descriptor must specify CUBLAS_OP_T on matrix B and CUBLAS_OP_N (default) on matrix A and C. The data corruption behavior was fixed so that CUBLAS_STATUS_NOT_SUPPORTED is returned instead.
  - Fixed an issue that caused an Address out of bounds error when calling `cublasSgemm()`.
  - A performance regression in the `cublasCgetrfBatched` and `cublasCgetriBatched` routines has been fixed.

2.1.10. cuBLAS: Release 11.0 Update 1

- **New Features**

- The cuBLAS API was extended with a new function, `cublasSetWorkspace()`, which allows the user to set the cuBLAS library workspace to a user-owned device buffer, which will be used by cuBLAS to execute all subsequent calls to the library on the currently set stream.
- cuBLASLt experimental logging mechanism can be enabled in two ways:
  - By setting the following environment variables before launching the target application:
    - `CUBLASLT_LOG_LEVEL=<level>` -- where level is one of the following levels:
      - "0" - Off - logging is disabled (default)
      - "1" - Error - only errors will be logged
      - "2" - Trace - API calls that launch CUDA kernels will log their parameters and important information
      - "3" - Hints - hints that can potentially improve the application's performance
      - "4" - Heuristics - heuristics log that may help users to tune their parameters
      - "5" - API Trace - API calls will log their parameter and important information
    - `CUBLASLT_LOG_MASK=<mask>` -- where mask is a combination of the following masks:
      - "0" - Off
      - "1" - Error
      - "2" - Trace
      - "4" - Hints
      - "8" - Heuristics
      - "16" - API Trace
    - `CUBLASLT_LOG_FILE=<value>` -- where value is a file name in the format of "`<file_name>.%i`", %i will be replaced with process id.If CUBLASLT_LOG_FILE is not defined, the log messages are printed to stdout.
  - By using the runtime API functions defined in the cublasLt header:
    - `typedef void(*cublasLtLoggerCallback_t)(int logLevel, const char* functionName, const char* message)` -- A type of callback function pointer.
    - `cublasStatus_t cublasLtLoggerSetCallback(cublasLtLoggerCallback_t callback)` --

Allows to set a call back functions that will be called for every message that is logged by the library.

- `cublasStatus_t cublasLtLoggerSetFile(FILE* file)` -- Allows to set the output file for the logger. The file must be open and have write permissions.
- `cublasStatus_t cublasLtLoggerOpenFile(const char* logFile)` -- Allows to give a path in which the logger should create the log file.
- `cublasStatus_t cublasLtLoggerSetLevel(int level)` -- Allows to set the log level to one of the above mentioned levels.
- `cublasStatus_t cublasLtLoggerSetMask(int mask)` -- Allows to set the log mask to a combination of the above mentioned masks.
- `cublasStatus_t cublasLtLoggerForceDisable()` -- Allows to disable to logger for the entire session. Once this API is being called, the logger cannot be reactivated in the current session.

2.1.11. cuBLAS: Release 11.0

- **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.

2.1.12. cuBLAS: Release 11.0 RC

- **New Features**
  - Many performance improvements have been implemented for NVIDIA Ampere, Volta, and Turing Architecture based GPUs.
  - 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
    - `CUBLAS`**`LT`**`_LOG_FILE=<value>` - while value is a file name in the format of "`<file_name>.%i`", %i will be replaced with process id. If CUBLAS**LT**_LOG_FILE is not defined, the log messages are printed to stdout.
  - For matrix multiplication APIs:

- `cublasGemmEx`, `cublasGemmBatchedEx`, `cublasGemmStridedBatchedEx` and `cublasLtMatmul` added new data type support for `__nv_bfloat16` (CUDA_R_16BF).
- 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.
- 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.

- **Deprecated Features**
  - 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.

2.2. cuFFT Library

2.2.1. cuFFT: Release 11.5

- **Known Issues**
  - FFTs of certain sizes in single and double precision (multiples of size 6) could fail on future devices. This issue will be fixed in an upcoming release.

2.2.2. cuFFT: Release 11.4 Update 2

- **Resolved Issues**

- Since cuFFT 10.3.0 (CUDA Toolkit 11.1), cuFFT may require user to make sure that all operations on input and output buffers are complete before calling `cufft[Xt]Exec*` if:
  - sm70 or later, 3D FFT, batch > 1, total size of transform is greater than 4.5MB
  - FFT size for all dimensions is in the set of the following sizes: {2, 4, 8, 16, 32, 64, 128, 3, 9, 81, 243, 729, 2187, 6561, 5, 25, 125, 625, 3125, 6, 36, 216, 1296, 7776, 7, 49, 343, 2401, 11, 121}
- Some V100 FFTs were slower than expected. This issue is resolved.

- **Known Issues**
  - Some T4 FFTs are slower than expected.
  - Plans for FFTs of certain sizes in single precision (including some multiples of 1024 sizes, and some large prime numbers) could fail on future devices with less than 64 kB of shared memory. This issue will be fixed in an upcoming release.

2.2.3. cuFFT: Release 11.4 Update 1

- **Resolved Issues**
  - Some cuFFT multi-GPU plans exhibited very long creation times.
  - cuFFT sometimes produced incorrect results for real-to-complex and complex-to-real transforms when the total number of elements across all batches in a single execution exceeded 2147483647.
- **Known Issues**
  - Some V100 FFTs are slower than expected.
  - Some T4 FFTs are slower than expected.

2.2.4. cuFFT: Release 11.4

- **New Features**
  - Performance improvements.
- **Known Issues**
  - Some T4 FFTs are slower than expected.
  - cuFFT may produce incorrect results for real-to-complex and complex-to-real transforms when the total number of elements across all batches in a single execution exceeds 2147483647.
  - Some cuFFT multi-GPU plans may exhibit very long creation time. Issue will be fixed in the next update.
  - cuFFT may produce incorrect results for transforms with strides when the index of the last element across all batches exceeds 2147483647 (see Advanced Data Layout).

- **Deprecated Features**
  - Support for callback functionality using separately compiled device code is deprecated on all GPU architectures. Callback functionality will continue to be supported for all GPU architectures.

2.2.5. cuFFT: Release 11.3

- **New Features**
  - cuFFT shared libraries are now linked statically against libstdc++ on Linux platforms.
  - Improved performance of certain sizes (multiples of large powers of 3, powers of 11) in SM86.
- **Known Issues**
  - cuFFT planning and plan estimation functions may not restore correct context affecting CUDA driver API applications.
  - Plans with strides, primes larger than 127 in FFT size decomposition and total size of transform including strides bigger than 32GB produce incorrect results.

2.2.6. cuFFT: Release 11.2 Update 2

- **Known Issues**
  - cuFFT planning and plan estimation functions may not restore correct context affecting CUDA driver API applications.
  - Plans with strides, primes larger than 127 in FFT size decomposition and total size of transform including strides bigger than 32GB produce incorrect results.

2.2.7. cuFFT: Release 11.2 Update 1

- **Resolved Issues**
  - Previously, reduced performance of power-of-2 single precision FFTs was observed on GPUs with sm_86 architecture. This issue has been resolved.
  - Large prime factors in size decomposition and real to complex or complex to real FFT type no longer cause cuFFT plan functions to fail.
- **Known Issues**
  - cuFFT planning and plan estimation functions may not restore correct context affecting CUDA driver API applications.
  - Plans with strides, primes larger than 127 in FFT size decomposition and total size of transform including strides bigger than 32GB produce incorrect results.

2.2.8. cuFFT: Release 11.2

- **New Features**

- Multi-GPU plans can be associated with a stream using the `cufftSetStream` API function call.
- Performance improvements for R2C/C2C/C2R transforms.
- Performance improvements for multi-GPU systems.

- **Resolved Issues**
  - cuFFT is no longer stuck in a bad state if previous plan creation fails with `CUFFT_ALLOC_FAILED`.
  - Previously, single dimensional multi-GPU FFT plans ignored user input on `cufftXtSetGPUs`*whichGPUs* argument and assumed that GPUs IDs are always numbered from 0 to N-1. This issue has been resolved.
  - Plans with primes larger than 127 in FFT size decomposition or FFT size being a prime number bigger than 4093 do not perform calculations on second and subsequent `cufftExecute*` calls. The regression was introduced in cuFFT 11.1.
- **Known Issues**
  - cuFFT planning and plan estimation functions may not restore correct context affecting CUDA driver API applications.

2.2.9. cuFFT: Release 11.1

- **New Features**
  - cuFFT is now L2-cache aware and uses L2 cache for GPUs with more than 4.5MB of L2 cache. Performance may improve in certain single-GPU 3D C2C FFT cases.
  - After successfully creating a plan, cuFFT now enforces a lock on the cufftHandle. Subsequent calls to any planning function with the same cufftHandle will fail.
  - Added support for very large sizes (3k cube) to multi-GPU cuFFT on DGX-2.
  - Improved performance on multi-gpu cuFFT for certain sizes (1k cube).
- **Resolved Issues**
  - Resolved an issue that caused cuFFT to crash when reusing a handle after clearing a callback.
  - Fixed an error which produced incorrect results / NaN values when running a real-to-complex FFT in half precision.
- **Known Issues**
  - cuFFT will always overwrite the input for out-of-place C2R transform.
  - Single dimensional multi-GPU FFT plans ignore user input on the `whichGPUs` parameter of `cufftXtSetGPUs()` and assume that GPUs IDs are always numbered from 0 to N-1.

2.2.10. cuFFT: Release 11.0 RC

- **New Features**
  - 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.
- **Resolved Issues**
  - 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.
- **Known Issues**
  - 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.

2.3. cuRAND Library

2.3.1. cuRAND: Release 11.5 Update 1

- **New Features**
  - Improved performance of `CURAND_RNG_PSEUDO_MRG32K3A` pseudorandom number generator when using ordering `CURAND_ORDERING_PSEUDO_BEST` or `CURAND_ORDERING_PSEUDO_DEFAULT`.
  - Added a new type of order parameter: `CURAND_ORDERING_PSEUDO_DYNAMIC`.
    - Supported PRNGs:
      - `CURAND_RNG_PSEUDO_XORWOW`
      - `CURAND_RNG_PSEUDO_MRG32K3A`
      - `CURAND_RNG_PSEUDO_MTGP32`
      - `CURAND_RNG_PSEUDO_PHILOX4_32_10`
    - Improved performance compared to `CURAND_ORDERING_PSEUDO_DEFAULT`, especially on NVIDIA Ampere architecture GPUs.

- The output ordering of generated random numbers for `CURAND_ORDERING_PSEUDO_DYNAMIC` depends on the number of SMs on a GPU, and thus can be different on different GPUs.
- The `CURAND_ORDERING_PSEUDO_DYNAMIC` ordering can't be used with a host generator created using `curandCreateGeneratorHost()`.

- **Resolved Issues**
  - Added information about cuRAND thread safety.
- **Known Issues**
  - `CURAND_RNG_PSEUDO_XORWOW` with ordering `CURAND_ORDERING_PSEUDO_DYNAMIC` can produce incorrect results on architectures newer than SM86.

2.3.2. cuRAND: Release 11.3

- **Resolved Issues**
  - Fixed inconsistency between random numbers generated by GPU and host generators when `CURAND_ORDERING_PSEUDO_LEGACY` ordering is selected for certain generator types.

2.3.3. cuRAND: Release 11.0 Update 1

- **Resolved Issues**
  - Fixed an issue that caused linker errors about the multiple definitions of `mtgp32dc_params_fast_11213` and `mtgpdc_params_11213_num` when including `curand_mtgp32dc_p_11213.h` in different compilation units.

2.3.4. cuRAND: Release 11.0

- **Resolved Issues**
  - Fixed an issue that caused linker errors about the multiple definitions of `mtgp32dc_params_fast_11213` and `mtgpdc_params_11213_num` when including `curand_mtgp32dc_p_11213.h` in different compilation units.

2.3.5. cuRAND: Release 11.0 RC

- **Resolved Issues**
  - 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.

2.4. cuSOLVER Library

2.4.1. cuSOLVER: Release 11.4

- **New Features**
  - Introducing `cusolverDnXtrtri`, a new generic API for triangular matrix inversion (trtri).
  - Introducing `cusolverDnXsytrs`, a new generic API for solving systems of linear equations using a given factorized symmetric matrix from SYTRF.

2.4.2. cuSOLVER: Release 11.3

- **Known Issues**
  - For values N<=16, `cusolverDn[S|D|C|Z]syevjBatched` hits out-of-bound access and may deliver the wrong result. The workaround is to pad the matrix A with a diagonal matrix D such that the dimension of [A 0 ; 0 D] is bigger than 16. The diagonal entry D(j,j) must be bigger than maximum eigenvalue of A, for example, norm(A, 'fro'). After the syevj, W(0:n-1) contains the eigenvalues and A(0:n-1,0:n-1) contains the eigenvectors.

2.4.3. cuSOLVER: Release 11.2 Update 2

- **New Features**
  - New singular value decomposition (GESVDR) is added. GESVDR computes partial spectrum with random sampling, an order of magnitude faster than GESVD.
  - `libcusolver.so` no longer links `libcublas_static.a`; instead, it depends on `libcublas.so`. This reduces the binary size of `libcusolver.so`. However, it breaks backward compatibility. The user has to link `libcusolver.so` with the correct version of `libcublas.so`.

2.4.4. cuSOLVER: Release 11.2

- **Resolved Issues**
  - `cusolverDnIRSXgels` sometimes returned `CUSOLVER_STATUS_INTERNAL_ERROR` when the precision is 'z'. This issue has been fixed in CUDA 11.2; now `cusolverDnIRSXgels` works for all precisions.

- ZSYTRF sometimes returned CUSOLVER_STATUS_INTERNAL_ERROR due to insufficient resources to launch the kernel. This issue has been fixed in CUDA 11.2.
- GETRF returned early without finishing the whole factorization when the matrix was singular. This issue has been fixed in CUDA 11.2.

2.4.5. cuSOLVER: Release 11.1 Update 1

- **Resolved Issues**
  - cusolverDnDDgels reports IRS_NOT_SUPPORTED when m > n. The issue has been fixed in release 11.1 U1, so cusolverDnDDgels will support m > n.
  - cusolverMgDeviceSelect can consume over 1GB device memory. The issue has been fixed in release 11.1 U1. The hidden memory allocation inside cusolverMG handle is about 30 MB per device.
- **Known Issues**
  - cusolverDnIRSXgels may return CUSOLVER_STATUS_INTERNAL_ERROR. when the precision is 'z' due to insufficient workspace which causes illegal memory access.

    The cusolverDnIRSXgels_bufferSize() does not report the correct size of workspace. To workaround the issue, the user has to add more workspace than what is reported by cusolverDnIRSXgels_bufferSize(). For example, if x is the size of workspace returned by cusolverDnIRSXgels_bufferSize(), then the user has to allocate (x + min(m,n)*sizeof(cuDoubleComplex)) bytes.

2.4.6. cuSOLVER: Release 11.1

- **New Features**
  - Added new 64-bit APIs:
    - cusolverDnXpotrf_bufferSize
    - cusolverDnXpotrf
    - cusolverDnXpotrs
    - cusolverDnXgeqrf_bufferSize
    - cusolverDnXgeqrf
    - cusolverDnXgetrf_bufferSize
    - cusolverDnXgetrf
    - cusolverDnXgetrs
    - cusolverDnXsyevd_bufferSize
    - cusolverDnXsyevd

- `cusolverDnXsyevdx_bufferSize`
- `cusolverDnXsyevdx`
- `cusolverDnXgesvd_bufferSize`
- `cusolverDnXgesvd`

- Added a new SVD algorithm based on polar decomposition, called GESVDP which uses the new 64-bit API, including `cusolverDnXgesvdp_bufferSize` and `cusolverDnXgesvdp`.

- **Deprecated Features**

  The following 64-bit APIs are deprecated:

  - `cusolverDnPotrf_bufferSize`
  - `cusolverDnPotrf`
  - `cusolverDnPotrs`
  - `cusolverDnGeqrf_bufferSize`
  - `cusolverDnGeqrf`
  - `cusolverDnGetrf_bufferSize`
  - `cusolverDnGetrf`
  - `cusolverDnGetrs`
  - `cusolverDnSyevd_bufferSize`
  - `cusolverDnSyevd`
  - `cusolverDnSyevdx_bufferSize`
  - `cusolverDnSyevdx`
  - `cusolverDnGesvd_bufferSize`
  - `cusolverDnGesvd`

2.4.7. cuSOLVER: Release 11.0

- **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.

- **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.

2.5. cuSPARSE Library

2.5.1. cuSPARSE: Release 11.6

- **New Features**
  - Better performance for **cusparseSpGEMM**, **cusparseSpGEMMreuse**, **cusparseCsr2cscEx2**, and **cusparseDenseToSparse** routines.
- **Resolved Issues**
  - Fixed forward compatibility issues with `axpby`, `rot`, `spvv`, `scatter`, `gather`.
  - Fixed incorrect results in COO SpMM Alg1 which occurred in some rare cases.

2.5.2. cuSPARSE: Release 11.5 Update 1

- **New Features**
  - New routine **cusparseSpMMOp** that exploits Just-In-Time Link-Time-Optimization (JIT LTO) for providing sparse matrix-dense matrix multiplication with custom (user-defined) operators. See https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-generic-function-spmm-op.
  - cuSPARSE now supports logging functionalities. See https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-logging.
- **Resolved Issues**
  - Added memory requirements, graph capture, and asynchronous notes for `cusparseXcsrsm2_analysis`.
  - CSR, CSC, and COO format descriptions wrongly reported sorted column indices requirement. All routines support unsorted column indices, except where strictly indicated
  - Clarified `cusparseSpSV` and `cusparseSpSM` memory management.
  - `cusparseSpSM` produced wrong results in some cases when the matB operation is `CUSPARSE_ OPERATION_NON_TRANSPOSE` or `CUSPARSE_OPERATION_CONJUGATE_TRANSPOSE`.
  - `cusparseSpSM` produced wrong results in some cases when the matrix layout is row-major.

2.5.3. cuSPARSE: Release 11.4 Update 1

- **Resolved Issues**
  - `cusparseSpSV` and `cusparseSpSM` could produce wrong results
  - `cusparseSpSV` and `cusparseSpSM` did not work correctly when vecX == vecY or matB == matC.

2.5.4. cuSPARSE: Release 11.4

- **Known Issues**
  - `cusparseSpSV` and `cusparseSpSM` could produce wrong results
  - `cusparseSpSV` and `cusparseSpSM` do not work correctly when vecX == vecY or matB == matC.

2.5.5. cuSPARSE: Release 11.3 Update 1

- **New Features**
  - Introduced a new routine for sparse matrix - sparse matrix multiplication (`cusparseSpGEMMreuse`) where the output matrix structure is reused for multiple computation. The new routine supports CSR storage format and mixed-precision computation.
  - Sparse triangular solver adds support for COO format.
  - Introduced a new routine for sparse triangular solver with multiple right-hand sides `cusparseSpSM()`.
  - `cusparseDenseToSparse()` routine adds the conversion from dense matrix (row-major/column-major) to Blocked-ELL format.
  - Blocke-ELL format now support empty blocks
  - Better performance for Blocked-ELL SpMM with block size > 64, double data type, and alignments smaller than 128-byte on NVIDIA Ampere sm80.
  - All cuSPARSE APIs are now asynchronous on platforms that support stream ordered memory allocators https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#stream-ordered-querying-memory-support.
  - Improved NTVX trace with distinction between light calls and kernel routines
- **Resolved Issues**
  - `cusparseCnnz_compress` produced wrong results when the number of rows are greater than 128 * resident CTAs.
  - `cusparseSnnz` produced wrong results for some particular sparsity pattern.

- **Deprecated Features**
  - `cusparseXcsrsm2_zeroPivot`, `cusparseXcsrsm2_solve`, `cusparseXcsrsm2_analysis`, and `cusparseScsrsm2_bufferSizeExt` have been deprecated in favor of `cusparseSpSM` Generic APIs

2.5.6. cuSPARSE: Release 11.3

- **New Features**

  Added new routine `cusparesSpSV` for sparse triangular solver with better performance. The new Generic API supports:
  - CSR storage format
  - Non-transpose, transpose, and transpose-conjugate operations
  - Upper, lower fill mode
  - Unit, non-unit diagonal type
  - 32-bit and 64-bit indices
  - Uniform data type computation
- **Deprecated Features**
  - `cusparseScsrsv2_analysis`, `cusparseScsrsv2_solve`, `cusparseXcsrsv2_zeroPivot`, and `cusparseScsrsv2_bufferSize` have been deprecated in favor of `cusparseSpSV`.

2.5.7. cuSPARSE: Release 11.2 Update 2

- **Resolved Issues**
  - `cusparseDestroy(NULL)` no longer crashes on Windows.
- **Known Issues**
  - `cusparseDestroySpVec`, `cusparseDestroyDnVec`, `cusparseDestroySpMat`, `cusparseDestroyDnMat`, `cusparseDestroy` with `NULL` argument could cause segmentation fault on Windows.

2.5.8. cuSPARSE: Release 11.2 Update 1

- **New Features**
  - New Tensor Core-accelerated Block Sparse Matrix - Matrix Multiplication (`cusparseSpMM`) and introduction of the Blocked-Ellpack storage format.
  - New algorithms for CSR/COO Sparse Matrix - Vector Multiplication (`cusparseSpMV`) with better performance.
  - Extended functionalities for `cusparseSpMV`:

- Support for the CSC format.
- Support for regular/complex bfloat16 data types for both uniform and mixed-precision computation.
- Support for mixed regular-complex data type computation.
- Support for deterministic and non-deterministic computation.

- New algorithm (`CUSPARSE_SPMM_CSR_ALG3`) for Sparse Matrix - Matrix Multiplication (`cusparseSpMM`) with better performance especially for small matrices.
- New routine for Sampled Dense Matrix - Dense Matrix Multiplication (`cusparseSDDMM`) which deprecated `cusparseConstrainedGeMM` and provides better performance.
- Better accuracy of `cusparseAxpby`, `cusparseRot`, `cusparseSpVV` for bfloat16 and half regular/complex data types.
- All routines support NVTX annotation for enhancing the profiler time line on complex applications.

- **Resolved Issues**
  - `cusparseAxpby`, `cusparseGather`, `cusparseScatter`, `cusparseRot`, `cusparseSpVV`, `cusparseSpMV` now support zero-size matrices.
  - `cusparseCsr2cscEx2` now correctly handles empty matrices (nnz = 0).
  - `cusparseXcsr2csr_compress` now uses 2-norm for the comparison of complex values instead of only the real part.
- **Known Issues**

  `cusparseDestroySpVec`, `cusparseDestroyDnVec`, `cusparseDestroySpMat`, `cusparseDestroyDnMat`, `cusparseDestroy` with `NULL` argument could cause segmentation fault on Windows.
- **Deprecated Features**
  - `cusparseConstrainedGeMM` has been deprecated in favor of `cusparseSDDMM`.
  - `cusparseCsrmvEx` has been deprecated in favor of `cusparseSpMV`.
  - COO Array of Structure (CooAoS) format has been deprecated including `cusparseCreateCooAoS`, `cusparseCooAoSGet`, and its support for `cusparseSpMV`.

2.5.9. cuSPARSE: Release 11.2

- **Known Issues**
  - `cusparseXdense2csr` provides incorrect results for some matrix sizes.

2.5.10. cuSPARSE: Release 11.1 Update 1

- **New Features**

- `cusparseSparseToDense`
  - CSR, CSC, or COO conversion to dense representation
  - Support row-major and column-major layouts
  - Support all data types
  - Support 32-bit and 64-bit indices
  - Provide performance 3x higher than `cusparseXcsc2dense`, `cusparseXcsr2dense`
- `cusparseDenseToSparse`
  - Dense representation to CSR, CSC, or COO
  - Support row-major and column-major layouts
  - Support all data types
  - Support 32-bit and 64-bit indices
  - Provide performance 3x higher than `cusparseXcsc2dense`, `cusparseXcsr2dense`

- **Known Issues**
  - `cusparseXdense2csr` provides incorrect results for some matrix sizes.
- **Deprecated Features**
  - Legacy conversion routines: `cusparseXcsc2dense`, `cusparseXcsr2dense`, `cusparseXdense2csc`, `cusparseXdense2csr`

2.5.11. cuSPARSE: Release 11.0

- **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

2.5.12. cuSPARSE: Release 11.0 RC

- **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
  - 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`.
- **Deprecated Features**

  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`

2.6. Math Library

2.6.1. CUDA Math: Release 11.6

- **New Features**
  - New half and bfloat16 APIs for addition/multiplication in round-to-nearest-even mode that do not get contracted into an fma instruction. Please see `__hadd_rn`, `__hsub_rn`, `__hmul_rn`, `__hadd2_rn`, `__hsub2_rn`, and `__hmul2_rn` in https://docs.nvidia.com/cuda/cuda-math-api/index.html.

2.6.2. CUDA Math: Release 11.5

- **Deprecations**
  - The following undocumented CUDA Math APIs are deprecated and will be removed in a future release. Please consider switching to similar intrinsic APIs documented here: https://docs.nvidia.com/cuda/cuda-math-api/index.html
    - `__device__ int mulhi(const int a, const int b)`
    - `__device__ unsigned int mulhi(const unsigned int a, const unsigned int b)`
    - `__device__ unsigned int mulhi(const int a, const unsigned int b)`
    - `__device__ unsigned int mulhi(const unsigned int a, const int b)`
    - `__device__ long long int mul64hi(const long long int a, const long long int b)`

- `__device__ unsigned long long int mul64hi(const unsigned long long int a, const unsigned long long int b)`
- `__device__ unsigned long long int mul64hi(const long long int a, const unsigned long long int b)`
- `__device__ unsigned long long int mul64hi(const unsigned long long int a, const long long int b)`
- `__device__ int float_as_int(const float a)`
- `__device__ float int_as_float(const int a)`
- `__device__ unsigned int float_as_uint(const float a)`
- `__device__ float uint_as_float(const unsigned int a)`
- `__device__ float saturate(const float a)`
- `__device__ int mul24(const int a, const int b)`
- `__device__ unsigned int umul24(const unsigned int a, const unsigned int b)`
- `__device__ int float2int(const float a, const enum cudaRoundMode mode = cudaRoundZero)`
- `__device__ unsigned int float2uint(const float a, const enum cudaRoundMode mode = cudaRoundZero)`
- `__device__ float int2float(const int a, const enum cudaRoundMode mode = cudaRoundNearest)`
- `__device__ float uint2float(const unsigned int a, const enum cudaRoundMode mode = cudaRoundNearest)`

2.6.3. CUDA Math: Release 11.3

- **Resolved Issues**
  - Previous releases of CUDA were potentially delivering incorrect results in some Linux distributions for the following host Math APIs: `sinpi`, `cospi`, `sincospi`, `sinpif`, `cospif`, `sincospif`. If passed huge inputs like 7.3748776e+15 or 8258177.5 the results were not equal to 0 or 1. These have been corrected with this release.

2.6.4. CUDA Math: Release 11.1

- **New Features**

- Added host support for half and `nv_bfloat16` converts to/from integer types.
- Added `__hcmadd()` device only API for fast half2 and nv_bfloat162 based complex multiply-accumulate.

2.6.5. CUDA Math: Release 11.0 Update 1

- **Resolved Issues**
  - `nv_bfloat16` comparison functions could trigger a fault with misaligned addresses.
  - Performance improvements in half and `nv_bfloat16` basic arithmetic implementations.

2.6.6. CUDA Math: Release 11.0 RC

- **New Features**
  - 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`.
- **Resolved Issues**
  - 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. NVIDIA Performance Primitives (NPP)

2.7.1. NPP: Release 11.5

- **New Features**
  - New APIs added to compute Signed Anti-aliased Distance Transform using PBA, the anti-aliased Euclidean distance between pixel sites in images. This will improve the accuracy of distance transform.
    - `nppiSignedDistanceTransformAbsPBA_xxxxx_C1R_Ctx()` – Input and output combination supports (xxxxxx) - 32f, 32f64f, 64f
  - New API for Absolute Manhattan distance transform; another method to improve the accuracy of distance transform using Manhattan distance transform between pixels.

- `nppiDistanceTransformAbsPBA_xxxxx_C1R_Ctx()` – Input and output combination supports (xxxxxx) - 8u16u, 8s16u, 16u16u, 16s16u, 8u32f, 8s32f, 16u32f, 16s32f, 8u64f, 8s64f, 16u64f, 16s64f, 32f64f, 64f

- **Resolved Issues**
  - Fixed an issue in `FilterMedian()` API with add interpolation when mask even size.
  - Improved Contour function performance by parallelizing more of it and also improving quality.
  - Resolved an issue with Alpha composition used to accumulate output buffers multiple times.
  - Resolved an issue with `nppiLabelMarkersUF_8u32u_C1R` column processing incorrect results.

2.7.2. NPP: Release 11.4

- **New Features**
  - New API `FindContours` .`FindContours` can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

2.7.3. NPP: Release 11.3

- **New Features**
  - Added `nppiDistanceTransformPBA` functions.

2.7.4. NPP: Release 11.2 Update 2

- **New Features**
  - Added `nppiDistanceTransformPBA` functions.

2.7.5. NPP: Release 11.2 Update 1

- **New Features**

  New APIs added to compute Distance Transform using Parallel Banding Algorithm (PBA):

  - `nppiDistanceTransformPBA_`*xxxxx*`_C1R_Ctx()` – where *xxxxx* specifies the input and output combination: 8u16u, 8s16u, 16u16u, 16s16u, 8u32f, 8s32f, 16u32f, 16s32f
  - `nppiSignedDistanceTransformPBA_32f_C1R_Ctx()`

- **Resolved Issues**
  - Fixed the issue in which Label Markers adds zero pixel as object region.

2.7.6. NPP: Release 11.0

- **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.

2.7.7. NPP: Release 11.0 RC

- **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.
  - 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.
- **Resolved Issues**
  - 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.
- **Known Issues**
  - 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. nvJPEG Library

2.8.1. nvJPEG: Release 11.5 Update 1

- **Resolved Issues**
  - Fixed the issue in which `nvcuvid()` released uncompressed frames causing a memory leak.

2.8.2. nvJPEG: Release 11.4

- **Resolved Issues**
  - Additional subsampling added to solve the `NVJPEG_CSS_2x4`.

2.8.3. nvJPEG: Release 11.2 Update 1

- **New Features**

  nvJPEG decoder added new APIs to support region of interest (ROI) based decoding for batched hardware decoder:

  - `nvjpegDecodeBatchedEx()`
  - `nvjpegDecodeBatchedSupportedEx()`

2.8.4. nvJPEG: Release 11.1 Update 1

- **New Features**
  - Added error handling capabilities for nonstandard JPEG images.

2.8.5. nvJPEG: Release 11.0 Update 1

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

2.8.6. nvJPEG: Release 11.0

- **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.

2.8.7. nvJPEG: Release 11.0 RC

- **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.
  - 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()`.
- **Known Issues**
  - `NVJPEG_BACKEND_GPU_HYBRID` has an issue when handling bit-streams which have corruption in the scan.
- **Deprecated Features**

  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`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com