Release Notes

Release 12.8

NVIDIA Corporation

May 05, 2025

Contents

NVIDIA CUDA Toolkit Release Notes

The Release Notes for the CUDA Toolkit.

The release notes for the NVIDIA® CUDA® Toolkit can be found online at https://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 12.x releases.

Chapter 1. CUDA Toolkit Major Component Versions

CUDA Components

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

For CUDA 12.8, the table below indicates the versions:

Table 1: CUDA 12.8 Update 1 Component Versions

Component Name		Version Information	Supported Architectures	Supported Platforms
CUDA C++ Core Compute Libraries	Thrust	2.7.0	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows
	CUB	2.7.0		
	libcu++	2.7.0		
	Cooperative Groups	12.8.90		
CUDA Compatibility		12.8.39468522	aarch64-jetson	Linux
CUDA Runtime (cudart)		12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
cuobjdump		12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows
CUPTI		12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA cuxxfilt (demangler)		12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows
CUDA Demo Suite		12.8.90	x86_64	Linux, Windows
CUDA GDB		12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, WSL

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA Nsight Eclipse Plugin	12.8.90	x86_64	Linux
CUDA NVCC	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA nvdisasm	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows
CUDA NVML Headers	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA nvprof	12.8.90	x86_64	Linux, Windows
CUDA nvprune	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA NVRTC	12.8.93	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
NVTX	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA NVVP	12.8.93	x86_64	Linux, Windows
CUDA OpenCL	12.8.90	x86_64	Linux, Windows
CUDA Profiler API	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA Compute Sanitizer API	12.8.93	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA cuBLAS	12.8.4.1	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
cuDLA	12.8.90	aarch64-jetson	Linux
CUDA cuFFT	11.3.3.76	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA cuFile	1.13.1.3	x86_64, arm64-sbsa, aarch64-jetson	Linux

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA cuRAND	10.3.9.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA cuSOLVER	11.7.3.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA cuSPARSE	12.5.8.88	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA NPP	12.3.3.100	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA nvFatbin	12.8.90	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA nvJitLink	12.8.93	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
CUDA nvJPEG	12.3.5.92	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL
Nsight Compute	2025.1.1.2	x86_64, arm64-sbsa, aarch64-jetson	Linux, Windows, WSL (Windows 11)
Nsight Systems	2024.6.2.225	x86_64, arm64-sbsa	Linux, Windows, WSL
Nsight Visual Studio Edition (VSE)	2025.1.0.25055	x86_64 (Windows)	Windows
nvidia_fs[1]	2.24.3	x86_64, arm64-sbsa, aarch64-jetson	Linux
Visual Studio Integration	12.8.90	x86_64 (Windows)	Windows
NVIDIA Linux Driver	570.124.06	x86_64, arm64-sbsa	Linux
NVIDIA Windows Driver	572.61	x86_64 (Windows)	Windows, WSL

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

[1] Only available on select Linux distros

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.

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 12.x	>=525.60.13	>=528.33
CUDA 11.8.x CUDA 11.7.x CUDA 11.6.x CUDA 11.5.x CUDA 11.4.x CUDA 11.3.x CUDA 11.2.x CUDA 11.1.x	>=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 12.8 Update 1	>=570.124.06	>=572.61
CUDA 12.8 GA	>=570.117	>=572.30
CUDA 12.6 Update 3	>=560.35.05	>=561.17
CUDA 12.6 Update 2	>=560.35.03	>=560.94
CUDA 12.6 Update 1	>=560.35.03	>=560.94
CUDA 12.6 GA	>=560.28.03	>=560.76

continues on next page

Table 3 – continued from previous page

CUDA Toolkit	Toolkit Driver Version	
CUDA 12.5 Update 1	>=555.42.06	>=555.85
CUDA 12.5 GA	>=555.42.02	>=555.85
CUDA 12.4 Update 1	>=550.54.15	>=551.78
CUDA 12.4 GA	>=550.54.14	>=551.61
CUDA 12.3 Update 1	>=545.23.08	>=546.12
CUDA 12.3 GA	>=545.23.06	>=545.84
CUDA 12.2 Update 2	>=535.104.05	>=537.13
CUDA 12.2 Update 1	>=535.86.09	>=536.67
CUDA 12.2 GA	>=535.54.03	>=536.25
CUDA 12.1 Update 1	>=530.30.02	>=531.14
CUDA 12.1 GA	>=530.30.02	>=531.14
CUDA 12.0 Update 1	>=525.85.12	>=528.33
CUDA 12.0 GA	>=525.60.13	>=527.41
CUDA 11.8 GA	>=520.61.05	>=520.06
CUDA 11.7 Update 1	>=515.48.07	>=516.31
CUDA 11.7 GA	>=515.43.04	>=516.01
CUDA 11.6 Update 2	>=510.47.03	>=511.65
CUDA 11.6 Update 1	>=510.47.03	>=511.65
CUDA 11.6 GA	>=510.39.01	>=511.23
CUDA 11.5 Update 2	>=495.29.05	>=496.13
CUDA 11.5 Update 1	>=495.29.05	>=496.13
CUDA 11.5 GA	>=495.29.05	>=496.04
CUDA 11.4 Update 4	>=470.82.01	>=472.50
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 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

continues on next page

Table 3 – continued from previous page

CUDA Toolkit	Toolkit Driver Version	
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 https://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 https://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.

Chapter 2. New Features

- This release adds compiler support for the following Nvidia Blackwell GPU architectures:
  - SM_100
  - SM_101
  - SM_120
- Tegra-Specific:
  - Added MPS support for DRIVE OS QNX
  - Added support for GCC 13.2.0
- Added support for Unified Virtual Memory (UVM) with Extended GPU Memory (EGM) arrays
- Hopper Confidential Computing:
  - Added multi-GPU support for protected PCIe mode
  - Added key rotation capability for single GPU passthrough mode
- NVML Updates:
  - Fixed per-process memory usage reporting for Docker containers using Open GPU Kernel Module drivers
  - Added support for DRAM encryption query and control (Blackwell)
  - Added checkpoint/restore functionality for userspace applications
  - Added support for Blackwell reduced bandwidth mode (RBM)
- CUDA Graphs:
  - Added conditional execution features for CUDA Graphs:
  - ELSE graph support for IF nodes
  - SWITCH node support
  - Introduced additional performance optimizations
- CUDA Usermode Driver (UMD):
  - Added PCIe device ID to CUDA device properties
  - Added cudaStreamGetDevice and cuStreamGetDevice APIs to retrieve the device associated with a CUDA stream
  - Added CUDA support for INT101010 texture/surface format

  - Added batch CUDA asynchronous memory copy APIs (cuMemcpyBatchAsync and cuMemcpyBatch3DAsync) for variable-sized transfers between multiple source and destination buffers
- Userspace Checkpoint and Restore:
  - Added new driver API for checkpoint/restore operations

2.1. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-7.
- Added two new nvcc flags:
  - `static-global-template-stub {true|false}`: Controls host side linkage for global/device/constant/managed templates in whole program mode
  - `device-entity-has-hidden-visibility {true|false}`: Controls ELF visibility of global/device/constant/managed symbols

  The current default value for both flags is false. These defaults will change to true in our future release. For detailed information about these flags and their impact on existing programs, refer to the `nvcc --help` command or the online CUDA documentation.
- **libNVVM**

  `libNVVM` now supports compilation for the Blackwell family of architectures. Compilation of compute capabilities `compute_100` and greater (Blackwell and future architectures) uses an updated NVVM IR dialect, based on LLVM 18.1.8 IR (the "modern" dialect) that differs from the older dialect used for pre-Blackwell architectures (a compute capability less than compute_100). NVVM IR bitcode using the older dialect generated for pre-Blackwell architectures can be used to target Blackwell and later architectures, with the exception of debug metadata.
- **nvdisasm**

  `Nvdisasm` now supports emitting JSON formatted SASS disassembly.

2.2. CUDA Developer Tools

- For changes to nvprof and Visual Profiler, see the changelog.
- For new features, improvements, and bug fixes in Nsight Systems, see the changelog.
- For new features, improvements, and bug fixes in Nsight Visual Studio Edition, see the changelog.
- 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.
- For new features, improvements, and bug fixes in Compute Sanitizer, see the changelog.
- For new features, improvements, and bug fixes in CUDA-GDB, see the changelog.

Chapter 3. Resolved Issues

3.1. CUDA Compiler

- Resolved compilation issues where code that successfully built with GCC would fail to compile with NVCC on Ubuntu 24.04. This improves cross-compiler compatibility and ensures consistent behavior between GCC and NVIDIA's CUDA compiler toolchain. [*4893699*]
- Fixed incorrect handling of C++20 requires expressions, restoring proper functionality and standard compliance. This ensures that compile-time requirements on template parameters now evaluate correctly. [*4843353*]
- Fixed an issue where NVCC (NVIDIA Compiler Driver) was ignoring the global namespace prefix of a type and thus incorrectly resolving it to a local type that shares the same name. [*4804685*]
- Fixed a compilation error in NVCC that occurred when code contained three or more nested lambda expressions with variadic arguments. The compiler now properly handles deeply nested variadic lambdas. [*4782817*]
- Fixed a limitation in NVRTC that caused compilation failures when kernel functions had long identifiers. The runtime compiler now properly handles kernel functions with extended name lengths. [*4781023*]
- Resolved an issue where template alias resolution could produce incorrect template instances. Previously, when an alias template and its underlying type-id template had different default arguments, the compiler would sometimes incorrectly omit the differing default argument when substituting the alias with its underlying type. This resulted in references to incorrect template instances. The template argument resolution now properly preserves all necessary default arguments during alias substitution. [*4721362*]
- Fixed invalid error reporting when using variables as template arguments from outside their visible scope. This resolves incorrect diagnostic messages particularly affecting cases involving braced initializers. The compiler now properly validates scope accessibility for template arguments. [*4717351*]
- Added the ability to cancel ongoing NVRTC compilations through callback mechanisms. This new feature allows developers to safely interrupt and terminate compilation processes programmatically. [*4082060*]
- The semantics of the `-expt-relaxed-constexpr` nvcc flag are now documented in the "C++ Language Support" section of the CUDA Programming Guide. [*3288543*]

Chapter 4. Known Issues and Limitations

4.1. CUDA

- Certain Linux kernels with KASLR enabled have a known issue in HMM initialization, causing CUDA initialization to fail. This issue is indicated by the following debug message:

  ```
  [64689.125237] nvidia-uvm: uvm_pmm_gpu.c:3176 devmem_alloc_
  ↪pagemap[pid:92821] request_free_mem_region() err -34
  ```

  Fixes to this issue are being handled in upstream kernels. In the meantime, you can use one of the following workarounds:

  - Option 1: Disable KASLR (Preferred option)

    If using GRUB, edit /etc/default/grub and add `nokaslr` to `GRUB_CMDLINE_LINUX_DEFAULT`:

    ```
    GRUB_CMDLINE_LINUX_DEFAULT="quiet splash nokaslr"
    ```

    Then, update GRUB and reboot:

    ```
    sudo update-grub
    sudo reboot
    ```

  - Option 2: Disable HMM for UVM

    1. Create or edit /etc/modprobe.d/uvm.conf.
    2. Add or update the following line:

       ```
       options nvidia_uvm uvm_disable_hmm=1
       ```

    3. Unload and reload the `nvidia_uvm` kernel module or reboot the system:

       ```
       sudo modprobe -r nvidia_uvm
       sudo modprobe nvidia_uvm
       ```

4.2. CUDA Compiler

- Some GPUs may experience higher-than-normal context creation times with driver version 570.xx.yyy. For many applications this will likely be unnoticeable, as context creation is usually done at initialization and amortized over the application lifetime. However, applications that create and destroy CUDA contexts frequently may see higher impact. NVIDIA will address this issue in an upcoming driver 570 release. [*4886848*]

Chapter 5. Deprecated or Dropped Features

Features deprecated in the current release of the CUDA software 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.

5.1. Deprecated Architectures

- Architecture support for Maxwell, Pascal, and Volta is considered feature-complete and will be frozen in an upcoming release.

5.2. Deprecated or Dropped Operating Systems

- Support for Microsoft Windows 10 21H2 has been dropped.
- Support for Debian 11 has been dropped.
- Support for versions prior to SLES 15 Service Pack 4 / OpenSUSE 15.4 has been dropped.
- NVTX v2 is deprecated. To migrate to NVTX v3. Change your code from:

  `#include <nvtoolsext.h>` to `#include "nvtx3/nvtoolsext.h"`. This header is included in the toolkit.

  For the latest NVTX version and extensions, visit NVIDIA NVTX.

5.3. Deprecated CUDA Tools

- Profiling tools supporting pre-turing architectures, Visual Profiler and nvprof, are now deprecated will be dropped in an upcoming release.
- The CUPTI Event API (from `header cupti_events.h`) and CUPTI Metric API (from `cupti_metrics.h`) are now deprecated and will be dropped in an upcoming release.
- Nsight Eclipse plugins will no longer be included in Tegra (SOC) packages, such as DriveOS or Jetson. Users of these packages are encouraged to use Nsight Visual Studio Code, available in the VSCode Extension Gallery or from the Microsoft VSCode Marketplace.
- Support for the macOS host client of CUDA-GDB has been dropped.

Chapter 6. CUDA Libraries

This section covers CUDA Libraries release notes for 12.x releases.

- CUDA Math Libraries toolchain uses C++11 features, and a C++11-compatible standard library (libstdc++ >= 20150422) is required on the host.

6.1. cuBLAS Library

6.1.1. cuBLAS: Release 12.8 Update 1

- **New Features**
  - Performance Improvements on Nvidia Blackwell GPU Architecure:
    - Matrix Multiplication (Matmuls): Enhanced performance for FP8 (both block-scaled and tensor-wide scaled), FP4, and FP16/BF16.
    - BLAS Level 3: Optimized SSYRK, CSYRK, and CHERK operations, especially for unaligned problems.
    - Batched Operations: Improved efficiency for batched GEMMs and batched GEMVs.
  - Added support for block-scaled FP8 and FP4 datatypes on Blackwell GeForce-class GPUs.
  - Improved performance on Blackwell GeForce-class GPUs.
- **Resolved Issues**
  - Using `cublasLtMatmul` with m or n equal to 1 and leading dimensions that cause the input or output matrices to exceed 2^31 elements may result in illegal memory access. [*5113092, 4959900*]
  - Using `cublasLtMatmul` with m or n equal to 1 and the `CUBLASLT_EPILOGUE_BIAS` epilogue may produce incorrect results. [*5104822*]
  - Under rare circumstances, `cublasLtMatmul` running FP8, FP16, or BF16 on a Blackwell GPU may result in a "CUDA Exception: Cluster target block not present" or a "CUDA Error 719: Unspecified launch failure". [*5124406*]

6.1.2. cuBLAS: Release 12.8

- **New Features**
  - Added support for NVIDIA Blackwell GPU architecture.
  - Extended the cuBLASLt API to support micro-scaled 4-bit and 8-bit floating-point mixed-precision tensor core-accelerated matrix multiplication for compute capability 10.0 (Blackwell) and higher. Extensions include:
    - CUDA_R_4F_E2M1: Integration with `CUDA_R_UE4M3` scales and 16-element scaling blocks.
    - CUDA_R_8F variants: Compatibility with `CUDA_R_UE8` scales and 32-element scaling blocks.
    - FP8 Matmul Attribute extensions
      - Support for block-scaled use cases with scaling factor tensors instead of scalars.
      - Ability to compute scaling factors dynamically for output tensors when the output is a 4-bit or 8-bit floating-point data type.
  - Introduced initial support for CUDA in Graphics (CIG) on Windows x64 for NVIDIA Ampere GPU architecture and Blackwell GeForce-class GPUs. CIG contexts are now auto-detected, and cuBLAS selects kernels that comply with CIG shared memory usage limits.
  - Performance improvement on all Hopper GPUs for non-aligned INT8 matmuls.
- **Resolved Issues**
  - The use of `cublasLtMatmul` with `CUBLASLT_EPILOGUE_BGRAD{A,B}` epilogue allowed the output matrix to be in `CUBLASLT_ORDER_ROW` layout, which led to incorrectly computed bias gradients. This layout is now disallowed when using `CUBLASLT_EPILOGUE_BGRAD{A,B}` epilogue. [*4910924*]
- **Deprecations**
  - The experimental feature for Atomics Synchronization along rows (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_ROWS`) or columns (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_COLS`) of the output matrix is now deprecated. The functional implementation is still available but not performant and will be removed in a future release.

6.1.3. cuBLAS: Release 12.6 Update 2

- **New Features**
  - Broad performance improvement on all Hopper GPUs for FP8, FP16 and BF16 matmuls. This improvement also includes the following fused epilogues `CUBLASLT_EPILOGUE_BIAS`, `CUBLASLT_EPILOGUE_RELU`, `CUBLASLT_EPILOGUE_RELU_BIAS`, `CUBLASLT_EPILOGUE_RELU_AUX`, `CUBLASLT_EPILOGUE_RELU_AUX_BIAS`, `CUBLASLT_EPILOGUE_GELU`, and `CUBLASLT_EPILOGUE_GELU_BIAS`.
- **Known Issues**
  - cuBLAS in multi context scenarios may hang with R535 Driver for version below <535.91. [*CUB-7024*]

- Users may observe suboptimal performance on Hopper GPUs for FP64 GEMMs. A potential workaround is to conditionally turn on swizzling. To do this, users can take the algo returned via `cublasLtMatmulAlgoGetHeuristic` and query if swizzling can be enabled by calling `cublasLtMatmulAlgoCapGetAttribute` with `CUBLASLT_ALGO_CAP_CTA_SWIZZLING_SUPPORT`. If swizzling is supported, you can enable swizzling by calling `cublasLtMatmulAlgoConfigSetAttribute` with `CUBLASLT_ALGO_CONFIG_CTA_SWIZZLING`. [*4872420*]

- **Resolved Issues**
  - `cublasLtMatmul` could ignore the user specified Bias or Aux data types (`CUBLASLT_MATMUL_DESC_BIAS_DATA_TYPE` and `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_DATA_TYPE`) for FP8 matmul operations if these data types do not match the documented limitations in *cublasLtMatmulDescAttributes_t <https://docs.nvidia.com/cuda/cublas/#cublasltmatmuldescattributes-t>__*. [*44750343, 4801528*]
  - Setting `CUDA_MODULE_LOADING` to `EAGER` could lead to longer library load times on Hopper GPUs due to JIT compilation of PTX kernels. This can be mitigated by setting this environment variable to LAZY. [*4720601*]
  - `cublasLtMatmul` with INT8 inputs, INT32 accumulation, INT8 outputs, and FP32 scaling factors could have produced numerical inaccuracies when a `splitk` reduction was used. [*4751576*]

6.1.4. cuBLAS: Release 12.6 Update 1

- **Known Issues**
  - `cublasLtMatmul` could ignore the user specified Bias or Aux data types (`CUBLASLT_MATMUL_DESC_BIAS_DATA_TYPE` and `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_DATA_TYPE`) for FP8 matmul operations if these data types do not match the documented limitations in cublasLtMatmulDescAttributes_t. [*4750343*]
  - Setting `CUDA_MODULE_LOADING` to `EAGER` could lead to longer library load times on Hopper GPUs due to JIT compilation of PTX kernels. This can be mitigated by setting this environment variable to LAZY. [*4720601*]
  - `cublasLtMatmul` with INT8 inputs, INT32 accumulation, INT8 outputs, and FP32 scaling factors may produce accuracy issues when a `splitk` reduction is used. To workaround this issue, you can use `cublasLtMatmulAlgoConfigSetAttribute` to set the reduction scheme to none and set the `splitk` value to 1. [*4751576*]

6.1.5. cuBLAS: Release 12.6

- **Known Issues**
  - Computing matrix multiplication and an epilogue with INT8 inputs, INT8 outputs, and FP32 scaling factors can have numerical errors in cases when a second kernel is used to compute the epilogue. This happens because the first GEMM kernel converts the intermediate result from FP32 into INT8 and stores it for the subsequent epilogue kernel to use. If a value is outside of the range of INT8 before the epilogue and the epilogue would bring it into the

range of INT8, there will be numerical errors. This issue has existed since before CUDA 12 and there is no known workaround. [*CUB-6831*]

- `cublasLtMatmul` could ignore the user specified Bias or Aux data types (`CUBLASLT_MATMUL_DESC_BIAS_DATA_TYPE` and `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_DATA_TYPE`) for FP8 matmul operations if these data types do not match the documented limitations in cublasLtMatmulDescAttributes_t. [*4750343*]

- **Resolved Issues**
  - `cublasLtMatmul` produced incorrect results when data types of matrices `A` and `B` were different FP8 (for example, `A` is `CUDA_R_8F_E4M3` and `B` is `CUDA_R_8F_E5M2`) and matrix `D` layout was `CUBLASLT_ORDER_ROW`. [*4640468*]
  - `cublasLt` may return not supported on Hopper GPUs in some cases when A, B, and C are of type `CUDA_R_8I` and the compute type is `CUBLAS_COMPUTE_32I`. [*4381102*]
  - cuBLAS could produce floating point exceptions when running GEMM with `K` equal to 0. [*4614629*]

6.1.6. cuBLAS: Release 12.5 Update 1

- **New Features**
  - Performance improvement to matrix multiplication targeting large language models, specifically for small batch sizes on Hopper GPUs.
- **Known Issues**
  - The bias epilogue (without ReLU or GeLU) may be not supported on Hopper GPUs for strided batch cases. A workaround is to implement batching manually. This will be fixed in a future release.
  - `cublasGemmGroupedBatchedEx` and `cublas<t>gemmGroupedBatched` have large CPU overheads. This will be addressed in an upcoming release.
- **Resolved Issues**
  - Under rare circumstances, executing SYMM/HEMM concurrently with GEMM on Hopper GPUs might have caused race conditions in the host code, which could lead to an Illegal Memory Access CUDA error. [*4403010*]
  - `cublasLtMatmul` could produce an Illegal Instruction CUDA error on Pascal GPUs under the following conditions: batch is greater than 1, and beta is not equal to 0, and the computations are out-of-place (C != D). [*4566993*]

6.1.7. cuBLAS: Release 12.5

- **New Features**
  - cuBLAS adds an experimental API to support mixed precision grouped batched GEMMs. This enables grouped batched GEMMs with FP16 or BF16 inputs/outputs with the FP32 compute type. Refer to cublasGemmGroupedBatchedEx for more details.
- **Known Issues**

  - cublasLtMatmul ignores inputs to CUBLASLT_MATMUL_DESC_D_SCALE_POINTER and CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_SCALE_POINTER if the elements of the respective matrix are not of FP8 types.
- **Resolved Issues**
  - cublasLtMatmul ignored the mismatch between the provided scale type and the implied by the documentation, assuming the latter. For instance, an unsupported configuration of cublasLtMatmul with the scale type being FP32 and all other types being FP16 would run with the implicit assumption that the scale type is FP16 and produce incorrect results.
  - cuBLAS SYMV failed for large n dimension: 131072 and above for ssymv, 92673 and above for csymv and dsymv, and 65536 and above for zsymv.

6.1.8. cuBLAS: Release 12.4 Update 1

- **Known Issues**
  - Setting a cuBLAS handle stream to cudaStreamPerThread and setting the workspace via cublasSetWorkspace will cause any subsequent cublasSetWorkspace calls to fail. This will be fixed in an upcoming release.
  - cublasLtMatmul ignores mismatches between the provided scale type and the scale type implied by the documentation and assumes the latter. For example, an unsupported configuration of cublasLtMatmul with the scale type being FP32 and all other types being FP16 would run with the implicit assumption that the scale type is FP16 which can produce incorrect results. This will be fixed in an upcoming release.
- **Resolved Issues**
  - cublasLtMatmul ignored the CUBLASLT_MATMUL_DESC_AMAX_D_POINTER for unsupported configurations instead of returning an error. In particular, computing absolute maximum of D is currently supported only for FP8 Matmul when the output data type is also FP8 (CUDA_R_8F_E4M3 or CUDA_R_8F_E5M2).
  - Reduced host-side overheads for some of the cuBLASLt APIs: cublasLtMatmul(), cublasLtMatmulAlgoCheck(), and cublasLtMatmulAlgoGetHeuristic(). The issue was introduced in CUDA Toolkit 12.4.
  - cublasLtMatmul() and cublasLtMatmulAlgoGetHeuristic() could have resulted in floating point exceptions (FPE) on some Hopper-based GPUs, including Multi-Instance GPU (MIG). The issue was introduced in cuBLAS 11.8.

6.1.9. cuBLAS: Release 12.4

- **New Features**
  - cuBLAS adds experimental APIs to support grouped batched GEMM for single precision and double precision. Single precision also supports the math mode, CUBLAS_TF32_TENSOR_OP_MATH. Grouped batch mode allows you to concurrently solve GEMMs of different dimensions (m, n, k), leading dimensions (lda, ldb, ldc), transpositions (transa, transb), and scaling factors (alpha, beta). Please see gemmGroupedBatched for more details.
- **Known Issues**

- When the current context has been created using `cuGreenCtxCreate()`, cuBLAS does not properly detect the number of SMs available. The user may provide the corrected SM count to cuBLAS using an API such as `cublasSetSmCountTarget()`.
- BLAS level 2 and 3 functions might not treat alpha in a BLAS compliant manner when alpha is zero and the pointer mode is set to `CUBLAS_POINTER_MODE_DEVICE`. This is the same known issue documented in cuBLAS 12.3 Update 1.
- `cublasLtMatmul` with K equals 1 and epilogue `CUBLASLT_EPILOGUE_D{RELU,GELU}_BGRAD` could out-of-bound access the workspace. The issue exists since cuBLAS 11.3 Update 1.
- `cublasLtMatmul` with K equals 1 and epilogue `CUBLASLT_EPILOGUE_D{RELU,GELU}` could produce illegal memory access if no workspace is provided. The issue exists since cuBLAS 11.6.
- When captured in CUDA Graph stream capture, cuBLAS routines can create memory nodes through the use of stream-ordered allocation APIs, `cudaMallocAsync` and `cudaFreeAsync`. However, as there is currently no support for memory nodes in child graphs or graphs launched from the device, attempts to capture cuBLAS routines in such scenarios may fail. To avoid this issue, use the cublasSetWorkspace() function to provide user-owned workspace memory.

6.1.10. cuBLAS: Release 12.3 Update 1

- **New Features**
  - Improved performance of heuristics cache for workloads that have a high eviction rate.
- **Known Issues**
  - BLAS level 2 and 3 functions might not treat alpha in a BLAS compliant manner when alpha is zero and the pointer mode is set to `CUBLAS_POINTER_MODE_DEVICE`. The expected behavior is that the corresponding computations would be skipped. You may encounter the following issues: (1) HER{,2,X,K,2K} may zero the imaginary part on the diagonal elements of the output matrix; and (2) HER{,2,X,K,2K}, SYR{,2,X,K,2K} and others may produce NaN resulting from performing computation on matrices A and B which would otherwise be skipped. If strict compliance with BLAS is required, the user may manually check for alpha value before invoking the functions or switch to `CUBLAS_POINTER_MODE_HOST`.
- **Resolved Issues**
  - cuBLASLt matmul operations might have computed the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.
  - When an application compiled with cuBLASLt from CUDA Toolkit 12.2 update 1 or earlier runs with cuBLASLt from CUDA Toolkit 12.2 update 2 or CUDA Toolkit 12.3, matrix multiply descriptors initialized using `cublasLtMatmulDescInit()` sometimes did not respect attribute changes using `cublasLtMatmulDescSetAttribute()`.
  - Fixed creation of cuBLAS or cuBLASLt handles on Hopper GPUs under the Multi-Process Service (MPS).
  - `cublasLtMatmul` with K equals 1 and epilogue `CUBLASLT_EPILOGUE_BGRAD{A,B}` might have returned incorrect results for the bias gradient.

6.1.11. cuBLAS: Release 12.3

- **New Features**
  - Improved performance on NVIDIA L40S Ada GPUs.
- **Known Issues**
  - cuBLASLt matmul operations may compute the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.
  - When an application compiled with cuBLASLt from CUDA Toolkit 12.2 update 1 or earlier runs with cuBLASLt from CUDA Toolkit 12.2 update 2 or later, matrix multiply descriptors initialized using `cublasLtMatmulDescInit()` may not respect attribute changes using `cublasLtMatmulDescSetAttribute()`. To workaround this issue, create the matrix multiply descriptor using `cublasLtMatmulDescCreate()` instead of `cublasLtMatmulDescInit()`. This will be fixed in an upcoming release.

6.1.12. cuBLAS: Release 12.2 Update 2

- **New Features**
  - cuBLASLt will now attempt to decompose problems that cannot be run by a single gemm kernel. It does this by partitioning the problem into smaller chunks and executing the gemm kernel multiple times. This improves functional coverage for very large m, n, or batch size cases and makes the transition from the cuBLAS API to the cuBLASLt API more reliable.
- **Known Issues**
  - cuBLASLt matmul operations may compute the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.

6.1.13. cuBLAS: Release 12.2

- **Known Issues**
  - cuBLAS initialization fails on Hopper architecture GPUs when MPS is in use with `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` set to a value less than 100%. There is currently no workaround for this issue.
  - Some Hopper kernels produce incorrect results for batched matmuls with `CUBLASLT_EPILOGUE_RELU_BIAS` or `CUBLASLT_EPILOGUE_GELU_BIAS` and a non-zero `CUBLASLT_MATMUL_DESC_BIAS_BATCH_STRIDE`. The kernels apply the first batch's bias vector to all batches. This will be fixed in a future release.

6.1.14. cuBLAS: Release 12.1 Update 1

- **New Features**
  - Support for FP8 on NVIDIA Ada GPUs.
  - Improved performance on NVIDIA L4 Ada GPUs.
  - Introduced an API that instructs the cuBLASLt library to not use some CPU instructions. This is useful in some rare cases where certain CPU instructions used by cuBLASLt heuristics negatively impact CPU performance. Refer to https://docs.nvidia.com/cuda/cublas/index.html#disabling-cpu-instructions.
- **Known Issues**
  - When creating a matrix layout using the `cublasLtMatrixLayoutCreate()` function, the object pointed at by `cublasLtMatrixLayout_t` is smaller than `cublasLtMatrixLayoutOpaque_t` (but enough to hold the internal structure). As a result, the object should not be dereferenced or copied explicitly, as this might lead to out of bound accesses. If one needs to serialize the layout or copy it, it is recommended to manually allocate an object of size `sizeof(cublasLtMatrixLayoutOpaque_t)` bytes, and initialize it using `cublasLtMatrixLayoutInit()` function. The same applies to `cublasLtMatmulDesc_t` and `cublasLtMatrixTransformDesc_t`. The issue will be fixed in future releases by ensuring that `cublasLtMatrixLayoutCreate()` allocates at least `sizeof(cublasLtMatrixLayoutOpaque_t)` bytes.

6.1.15. cuBLAS: Release 12.0 Update 1

- **New Features**
  - Improved performance on NVIDIA H100 SXM and NVIDIA H100 PCIe GPUs.
- **Known Issues**
  - For optimal performance on NVIDIA Hopper architecture, cuBLAS needs to allocate a bigger internal workspace (64 MiB) than on the previous architectures (8 MiB). In the current and previous releases, cuBLAS allocates 256 MiB. This will be addressed in a future release. A possible workaround is to set the `CUBLAS_WORKSPACE_CONFIG` environment variable to :32768:2 when running cuBLAS on NVIDIA Hopper architecture.
- **Resolved Issues**
  - Reduced cuBLAS host-side overheads caused by not using the cublasLt heuristics cache. This began in the CUDA Toolkit 12.0 release.
  - Added forward compatible single precision complex GEMM that does not require workspace.

6.1.16. cuBLAS: Release 12.0

- **New Features**
  - cublasLtMatmul now supports FP8 with a non-zero beta.
  - Added int64 APIs to enable larger problem sizes; refer to 64-bit integer interface.
  - Added more Hopper-specific kernels for cublasLtMatmul with epilogues:
    - CUBLASLT_EPILOGUE_BGRAD{A,B}
    - CUBLASLT_EPILOGUE_{RELU,GELU}_AUX
    - CUBLASLT_EPILOGUE_D{RELU,GELU}
  - Improved Hopper performance on arm64-sbsa by adding Hopper kernels that were previously supported only on the x86_64 architecture for Windows and Linux.
- **Known Issues**
  - There are no forward compatible kernels for single precision complex gemms that do not require workspace. Support will be added in a later release.
- **Resolved Issues**
  - Fixed an issue on NVIDIA Ampere architecture and newer GPUs where cublasLtMatmul with epilogue CUBLASLT_EPILOGUE_BGRAD{A,B} and a nontrivial reduction scheme (that is, not CUBLASLT_REDUCTION_SCHEME_NONE) could return incorrect results for the bias gradient.
  - cublasLtMatmul for gemv-like cases (that is, m or n equals 1) might ignore bias with the CUBLASLT_EPILOGUE_RELU_BIAS and CUBLASLT_EPILOGUE_BIAS epilogues.

Deprecations

  - Disallow including cublas.h and cublas_v2.h in the same translation unit.
  - Removed:
    - CUBLAS_MATMUL_STAGES_16x80 and CUBLAS_MATMUL_STAGES_64x80 from cublasLtMatmulStages_t. No kernels utilize these stages anymore.
    - cublasLt3mMode_t, CUBLASLT_MATMUL_PREF_MATH_MODE_MASK, and CUBLASLT_MATMUL_PREF_GAUSSIAN_MODE_MASK from cublasLtMatmulPreferenceAttributes_t. Instead, use the corresponding flags from cublasLtNumericalImplFlags_t.
    - CUBLASLT_MATMUL_PREF_POINTER_MODE_MASK, CUBLASLT_MATMUL_PREF_EPILOGUE_MASK, and CUBLASLT_MATMUL_PREF_SM_COUNT_TARGET from cublasLtMatmulPreferenceAttributes_t. The corresponding parameters are taken directly from cublasLtMatmulDesc_t.
    - CUBLASLT_POINTER_MODE_MASK_NO_FILTERING from cublasLtPointerModeMask_t. This mask was only applicable to CUBLASLT_MATMUL_PREF_MATH_MODE_MASK which was removed.

6.2. cuFFT Library

6.2.1. cuFFT: Release 12.8 Update 1

- **Resolved Issues**
  - Fixed an issue where SM120 was only supported via PTX JIT for legacy callback kernels. SASS support is now available.
  - Fixed an issue where large applications (over 2 GB in total binary size) linking against the static cuFFT libraries (`libcufft_static.a`, `libcufft_static_nocallback.a`) on x86_64 systems without using the `-mcmodel=medium` flag would run into linking errors.

6.2.2. cuFFT: Release 12.8

- **New Features**
  - Added support for the NVIDIA Blackwell GPU architecture.
- **Deprecations**
  - The static library `libcufft_static_nocallback.a` is deprecated and scheduled for removal in a future release. Users should migrate to `libcufft_static.a`, as both libraries provide equivalent functionality following the introduction of LTO callbacks in cuFFT with CUDA Toolkit 12.6 Update 2.
- **Known Issues**
  - SM120 is only supported via PTX JIT for legacy callback kernels. As a result, non-LTO device callback code intended to be linked with `libcufft_static.a` must be compiled to PTX, not SASS.
  - Large applications (over 2 GB in total binary size) linking against the static cuFFT libraries (`libcufft_static.a`, `libcufft_static_nocallback.a`) in x86_64 systems without using the `-mcmodel=medium` flag will run into linking errors (For example: `.gcc_except_table relocation R_X86_64_PC32 out of range; references DW.ref._ZTI13cufftResult_t`) This issue will be fixed in an upcoming release.

    Existing workarounds include:

    - Building or linking the application with `-mcmodel=medium flag`
    - Using `readelf` to analyze the `libcufft_static.a` symbols, it is possible to move the reference `ref._ZTI13cufftResult_t` from the large data section `.ldata.DW.ref._ZTI13cufftResult_t` to the non-large data section `.data.DW.ref._ZTI13cufftResult_t`

6.2.3. cuFFT: Release 12.6 Update 2

- **New Features**
  - Introduced LTO callbacks as a replacement for the deprecated legacy callbacks. LTO callbacks offer:
    - Additional performance vs. legacy callbacks
    - Support for callbacks on Windows and on dynamic (shared) libraries

    See the cuFFT documentation page for more information.
- **Resolved Issues**
  - Several issues present in our cuFFT LTO EA preview binary have been addressed.
- **Deprecations**
  - cuFFT LTO EA, our preview binary for LTO callback support, is deprecated and will be removed in the future.

6.2.4. cuFFT: Release 12.6

- **Known Issues**
  - FFT of size 1 with `istride/ostride > 1` is currently not supported for FP16. There is a known memory issue for this use case in CTK 12.1 or before. A `CUFFT_INVALID_SIZE` error is thrown in CTK 12.2 or after. [*4662222*]

6.2.5. cuFFT: Release 12.5

- **New Features**
  - Added Just-In-Time Link-Time Optimized (JIT LTO) kernels for improved performance in R2C and C2R FFTs for many sizes.
    - We recommend testing your R2C / C2R use cases with and without JIT LTO kernels and comparing the resulting performance. You can enable JIT LTO kernels using the per-plan properties cuFFT API.

6.2.6. cuFFT: Release 12.4 Update 1

- **Resolved Issues**
  - A routine from the cuFFT LTO EA library was added by mistake to the cuFFT Advanced API header (`cufftXt.h`) in CUDA 12.4. This routine has now been removed from the header.

6.2.7. cuFFT: Release 12.4

- **New Features**
  - Added Just-In-Time Link-Time Optimized (JIT LTO) kernels for improved performance in FFTs with 64-bit indexing.
  - Added per-plan properties to the cuFFT API. These new routines can be leveraged to give users more control over the behavior of cuFFT. Currently they can be used to enable JIT LTO kernels for 64-bit FFTs.
  - Improved accuracy for certain single-precision (fp32) FFT cases, especially involving FFTs for larger sizes.
- **Known Issues**
  - A routine from the cuFFT LTO EA library was added by mistake to the cuFFT Advanced API header (`cufftXt.h`). This routine is not supported by cuFFT, and will be removed from the header in a future release.
- **Resolved Issues**
  - Fixed an issue that could cause overwriting of user data when performing out-of-place real-to-complex (R2C) transforms with user-specified output strides (i.e. using the `ostride` component of the Advanced Data Layout API).
  - Fixed inconsistent behavior between `libcufftw` and FFTW when both `inembed` and `onembed` are `nullptr / NULL`. From now on, as in FFTW, passing `nullptr / NULL` as `inembed/onembed` parameter is equivalent to passing n, that is, the logical size for that dimension.

6.2.8. cuFFT: Release 12.3 Update 1

- **Known Issues**
  - Executing a real-to-complex (R2C) or complex-to-real (C2R) plan in a context different to the one used to create the plan could cause undefined behavior. This issue will be fixed in an upcoming release of cuFFT.
- **Resolved Issues**
  - Complex-to-complex (C2C) execution functions (`cufftExec` and similar) now properly error-out in case of error during kernel launch, for example due to a missing CUDA context.

6.2.9. cuFFT: Release 12.3

- **New Features**
  - Callback kernels are more relaxed in terms of resource usage, and will use fewer registers.
  - Improved accuracy for double precision prime and composite FFT sizes with factors larger than 127.
  - Slightly improved planning times for some FFT sizes.

6.2.10. cuFFT: Release 12.2

- **New Features**
  - `cufftSetStream` can be used in multi-GPU plans with a stream from any GPU context, instead of from the primary context of the first GPU listed in `cufftXtSetGPUs`.
  - Improved performance of 1000+ of FFTs of sizes ranging from 62 to 16380. The improved performance spans hundreds of single precision and double precision cases for FFTs with contiguous data layout, across multiple GPU architectures (from Maxwell to Hopper GPUs) via PTX JIT.
  - Reduced the size of the static libraries when compared to cuFFT in the 12.1 release.
- **Resolved Issues**
  - cuFFT no longer exhibits a race condition when threads simultaneously create and access plans with more than 1023 plans alive.
  - cuFFT no longer exhibits a race condition when multiple threads call `cufftXtSetGPUs` concurrently.

6.2.11. cuFFT: Release 12.1 Update 1

- **Known Issues**
  - cuFFT exhibits a race condition when one thread calls `cufftCreate` (or `cufftDestroy`) and another thread calls any API (except `cufftCreate` or `cufftDestroy`), and when the total number of plans alive exceeds 1023.
  - cuFFT exhibits a race condition when multiple threads call `cufftXtSetGPUs` concurrently on different plans.

6.2.12. cuFFT: Release 12.1

- **New Features**
  - Improved performance on Hopper GPUs for hundreds of FFTs of sizes ranging from 14 to 28800. The improved performance spans over 542 cases across single and double precision for FFTs with contiguous data layout.
- **Known Issues**
  - Starting from CUDA 11.8, CUDA Graphs are no longer supported for callback routines that load data in out-of-place mode transforms. An upcoming release will update the cuFFT callback implementation, removing this limitation. cuFFT deprecated callback functionality based on separate compiled device code in cuFFT 11.4.
- **Resolved Issues**
  - cuFFT no longer produces errors with compute-sanitizer at program exit if the CUDA context used at plan creation was destroyed prior to program exit.

6.2.13. cuFFT: Release 12.0 Update 1

- **Resolved Issues**
  - Scratch space requirements for multi-GPU, single-batch, 1D FFTs were reduced.

6.2.14. cuFFT: Release 12.0

- **New Features**
  - PTX JIT kernel compilation allowed the addition of many new accelerated cases for Maxwell, Pascal, Volta and Turing architectures.
- **Known Issues**
  - cuFFT plan generation time increases due to PTX JIT compiling. Refer to Plan Initialization TIme.
- **Resolved Issues**
  - cuFFT plans had an unintentional small memory overhead (of a few kB) per plan. This is resolved.

6.3. cuSOLVER Library

6.3.1. cuSOLVER: Release 12.8

- **New Features**
  - `cusolverDn{SDCZ}sytrf` and `cusolverDnXsytrs` now support symmetric factorization without pivoting when the input pivot array `devIpiv=NULL`, providing improved performance.
  - `cusolver{DZ}gesvdaStridedBatched` now offers improved accuracy and performance for a wide range of problems.
  - `cusolver{SDCZ}gesvdaStridedBatched` now returns the number of leading valid singular values and vectors in case of a convergence failure.
- **Resolved Issues**
  - Fixed an issue with `cusolverDnXsyevBatched` when using `cuComplex` or `cuDoubleComplex` with a batch size of at least two, where an incorrect result could be returned if the workspace was not initialized to zero upon entry.
- **Deprecations**
  - The following APIs in `cuSOLVERSp` and `cuSOLVERRf` include deprecation warning in 12.8 [*4674686*]:
    - `cusolverSp{SDCZ}csrlsvluHost`
    - `cusolverSp{SDCZ}csrlsvcholHost`
    - `cusolverSp{SDCZ}csrlsvchol`

- `cusolverRfSetupHost`
- `cusolverRfSetupDevice`
- `cusolverRfResetValues`
- `cusolverRfAnalyze`
- `cusolverRfRefactor`
- `cusolverRfAccessBundledFactorsDevice`
- `cusolverRfExtractBundledFactorsHost`
- `cusolverRfExtractSplitFactorsHost`
- `cusolverRfSolve`

The deprecation warning can be removed by adding a compiler flag `-DDISABLE_CUSOLVER_DEPRECATED`.

Users are encouraged to use the cuDSS library for better performance and ongoing support. Refer to the cuDSS samples for the transition.

6.3.2. cuSOLVER: Release 12.6 Update 2

- **New Features**
  - New API `cusolverDnXgeev` to solve non-Hermitian eigenvalue problems.
  - New API `cusolverDnXsyevBatched` to solve uniform batched Hermitian eigenvalue problems.

6.3.3. cuSOLVER: Release 12.6

- **New Features**
  - Performance improvements of `cusolverDnXgesvdp()`.

6.3.4. cuSOLVER: Release 12.5 Update 1

- **Resolved Issues**
  - The potential out-of-bound accesses on `bufferOnDevice` by calls of `cusolverDnXlarft` have been resolved.

6.3.5. cuSOLVER: Release 12.5

- **New Features**
  - Performance improvements of cusolverDnXgesvd and cusolverDn<t>gesvd if jobu != 'N' or jobvt != 'N'.
  - Performance improvements of cusolverDnXgesvdp if jobz = CUSOLVER_EIG_MODE_NOVECTOR.
  - Lower workspace requirement of cusolverDnXgesvdp for tall-and-skinny-matrices.
- **Known Issues**
  - With CUDA Toolkit 12.4 Update 1, values ldt > k in calls of cusolverDnXlarft can result in out-of-bound memory accesses on bufferOnDevice. As a workaround it is possible to allocate a larger device workspace buffer of size workspaceInBytesOnDevice=ALIGN_32((ldt*k + n*k)*sizeofCudaDataType(dataTypeT)), with

```
auto ALIGN_32=[](int64_t val) {
   return ((val + 31)/32)*32;
};
```

  and

```
auto sizeofCudaDataType=[](cudaDataType dt) {
   if (dt == CUDA_R_32F) return sizeof(float);
   if (dt == CUDA_R_64F) return sizeof(double);
   if (dt == CUDA_C_32F) return sizeof(cuComplex);
   if (dt == CUDA_C_64F) return sizeof(cuDoubleComplex);
};
```

6.3.6. cuSOLVER: Release 12.4 Update 1

- **New Features**
  - The performance of cusolverDnXlarft has been improved. For large matrices, the speedup might exceed 100x. The performance on H100 is now consistently better than on A100. The change in cusolverDnXlarft also results in a modest speedup in cusolverDn<t>ormqr, cusolverDn<t>ormtr, and cusolverDnXsyevd.
  - The performance of cusolverDnXgesvd when singular vectors are sought has been improved. The job configuration that computes both left and right singular vectors is up to 1.5x faster.
- **Resolved Issues**
  - cusolverDnXtrtri_bufferSize now returns the correct workspace size in bytes.
- **Deprecations**
  - Using long-deprecated cusolverDnPotrf, cusolverDnPotrs, cusolverDnGeqrf, cusolverDnGetrf, cusolverDnGetrs, cusolverDnSyevd, cusolverDnSyevdx, cusolverDnGesvd, and their accompanying bufferSize functions will result in a deprecation warning. The warning can be turned off by using the -DDISABLE_CUSOLVER_DEPRECATED flag while compiling; however, users should use cusolverDnXpotrf, cusolverDnXpotrs, cusolverDnXgeqrf, cusolverDnXgetrf, cusolverDnXgetrs, cusolverDnXsyevd,

cusolverDnXsyevdx, cusolverDnXgesvd, and the corresponding bufferSize functions instead.

6.3.7. cuSOLVER: Release 12.4

- **New Features**
  - cusolverDnXlarft and cusolverDnXlarft_bufferSize APIs were introduced. cusolverDnXlarft forms the triangular factor of a real block reflector, while cusolverDnXlarft_bufferSize returns its required workspace sizes in bytes.
- **Known Issues**
  - cusolverDnXtrtri_bufferSize returns an incorrect required device workspace size. As a workaround the returned size can be multiplied by the size of the data type (for example, 8 bytes if matrix A is of type double) to obtain the correct workspace size.

6.3.8. cuSOLVER: Release 12.2 Update 2

- **Resolved Issues**
  - Fixed an issue with cusolverDn<t>gesvd(), cusolverDnGesvd(), and cusolverDnXgesvd(), which could cause wrong results for matrices larger than 18918 if jobu or jobvt was unequal to 'N'.

6.3.9. cuSOLVER: Release 12.2

- **New Features**
  - A new API to ensure deterministic results or allow non-deterministic results for improved performance. See cusolverDnSetDeterministicMode() and cusolverDnGetDeterministicMode(). Affected functions are: cusolverDn<t>geqrf(), cusolverDn<t>syevd(), cusolverDn<t>syevdx(), cusolverDn<t>gesvdj(), cusolverDnXgeqrf(), cusolverDnXsyevd(), cusolverDnXsyevdx(), cusolverDnXgesvdr(), and cusolverDnXgesvdp().
- **Known Issues**
  - Concurrent executions of cusolverDn<t>getrf() or cusolverDnXgetrf() in different non-blocking CUDA streams on the same device might result in a deadlock.

6.4. cuSPARSE Library

6.4.1. cuSPARSE: Release 12.8 Update 1

- **Resolved Issues** - `cusparseSpMM` and `cusparseSDDMM` previously produced incorrect results if the output matrix had multiple batches with `batchStride = 0`. This case now returns an error code instead. [*CUSPARSE-2141*]
- **Known Issues**
  - Many cuSPARSE routines may not function correctly with large matrices when nnz approaches the signed 32-bit integer limit (e.g., `nnz = 2^31 - 1`), even when using 64-bit indices. [[*4966852*]
  - Some cuSPARSE routines may not function correctly with small matrices, particularly those with very few elements or a zero dimension. This issue affects at least `cusparseDenseToSparse` and `cusparseSpMV` with CSR matrices. [*CUSPARSE-2263*]
  - Many cuSPARSE routines require 16-byte alignment for data arrays to function correctly. This applies to matrix values, indices, offsets, and the temporary buffer. [*5053391*]
  - `CUSPARSE_SPMM_CSR_ALG1` may return incorrect results when the dense matrix has more than 2^20 - 16 columns.

6.4.2. cuSPARSE: Release 12.8

- **New Features**
  - Added support for NVIDIA Blackwell GPUs with significant performance improvements in sparse matrix operations:
    - SpMV (Sparse Matrix-Vector multiplication): Up to 2.3x faster than Hopper
    - SpMM (Sparse Matrix-Matrix multiplication): Up to 2.4x faster than Hopper
- **Resolved Issues**
  - Fixed an issue in cusparseSpMM that caused "misaligned address" errors when using the CUSPARSE_SPMM_CSR_ALG3 algorithm with CUDA_R_64F data type and mismatched memory layouts between two dense matrices - op(B) and C. [*CUSPARSE-2081*]
  - Fixed an issue where subsequent calls to SpMV preprocess on the same matrix would fail after the first call. [*CUSPARSE-1897*]
  - Fixed an issue where SpMV preprocess would not execute when alpha=0. [*CUSPARSE-1897*]
  - Fixed issues to enable preprocessing operations (SpMV, SpMM, SDDMM) with different memory buffers. [*CUSPARSE-1962*]
  - Addressed an issue in SpSV where incorrect results occurred when the matrix was in SlicedELL format with lower triangular structure and diagonal elements. [*CUSPARSE-1996*]
- **Known Issues**
  - SpMM and certain other routines are currently limited when processing matrices approaching 2^31 non-zero elements. [*CUSPARSE-2133*]
- **Deprecations**

- The following cuSPARSE functions are deprecated and planned for removal in a future major release [*4687069*]:
  - `cusparseSpVV()`
  - `cusparseAxpby()`
  - `cusparseXgemvi()`
  - `cusparseSbsr2csr()`
  - `cusparseSgebsr2csr()`
  - `cusparseSgebsr2gebsr()`
  - `cusparseXbsrmm()` (use `cusparseSpMM` instead)

  Contact Math-Libs-Feedback@nvidia.com or visit https://forums.developer.nvidia.com/ with any concerns.
- Support for 16-bit complex floating-point (CUDA_C_16F) and 16-bit complex bfloat floating-point (CUDA_C_16BF) data types will be removed from cuSPARSE in a future release. These data types have been marked as deprecated since CUDA 12.2. [*CUSPARSE-2225*]

6.4.3. cuSPARSE: Release 12.6 Update 2

- **Resolved Issues**
  - Re-wrote the documentation for `cusparseSpMV_preprocess()`, `cusparseSpMM_preprocess()`, and `cusparseSDDMM_preprocess()`. The documentation now explains the additional constraints that code must satisfy when using these functions. [*CUSPARSE-1962*]
  - `cusparseSpMV()` would expect the values in the external buffer to be maintained from one call to the next. If this was not true, it could compute the incorrect result or crash. [*CUSPARSE-1897*]
  - `cusparseSpMV_preprocess()` wouldn't run correctly if `cusparseSpMM_preprocess()` was executed on the same matrix, and vice versa. [*CUSPARSE-1897*]
  - `cusparseSpMV_preprocess()` runs SpMV computation if it's called two or more times on the same matrix. [*CUSPARSE-1897*]
  - `cusparseSpMV()` could cause subsequent calls to `cusparseSpMM()` with the same matrix to produce incorrect results or crash. [*CUSPARSE-1897*]
  - With a single sparse matrix `A` and a dense matrix `X` that has only a single column, calling both `cusparseSpMM_preprocess(A,X,...)` could cause subsequent calls to `cusparseSpMV()` to crash or produce incorrect results. The same is true with the roles of SpMV and SpMM swapped. [*CUSPARSE-1921*]

6.4.4. cuSPARSE: Release 12.6

- **Known Issues**
  - `cusparseSpMV_preprocess()` runs SpMV computation if it is called two or more times on the same matrix. [*CUSPARSE-1897*]
  - `cusparseSpMV_preprocess()` will not run if `cusparseSpMM_preprocess()` was executed on the same matrix, and vice versa. [*CUSPARSE-1897*]
  - The same external_buffer must be used for all `cusparseSpMV` calls. [*CUSPARSE-1897*]

6.4.5. cuSPARSE: Release 12.5 Update 1

- **New Features**
  - Added support for BSR format in `cusparseSpMM`.
- **Resolved Issues**
  - `cusparseSpMM()` would sometimes get incorrect results when `alpha=0`, `num_batches>1`, `batch_stride` indicates that there is padding between batches.
  - `cusparseSpMM_bufferSize()` would return the wrong size when the sparse matrix is Blocked Ellpack and the dense matrices have only a single column (n=1).
  - `cusparseSpMM` returned the wrong result when `k=0` (for example when A has zero columns). The correct behavior is doing `C \*= beta`. The bug behavior was not modifying `C` at all.
  - `cusparseCreateSlicedEll` would return an error when the slice size is greater than the matrix number of rows.
  - Sliced-ELLPACK `cusparseSpSV` produced wrong results for diagonal matrices.
  - Sliced-ELLPACK `cusparseSpSV_analysis()` failed due to insufficient resources for some matrices and some slice sizes.

6.4.6. cuSPARSE: Release 12.5

- **New Features**
  - Added support for mixed input types in SpMV: single precision input matrix, double precision input vector, double precision output vector.
- **Resolved Issues**
  - `cusparseSpMV()` introduces invalid memory accesses when the output vector is not aligned to 16 bytes.

6.4.7. cuSPARSE: Release 12.4

- **New Features**
  - Added the preprocessing step for sparse matrix-vector multiplication cusparseSpMV_preprocess().
  - Added support for mixed real and complex types for cusparseSpMM().
  - Added a new API cusparseSpSM_updateMatrix() to update the sparse matrix between the analysis and solving phase of cusparseSpSM().
- **Known Issues**
  - cusparseSpMV() introduces invalid memory accesses when the output vector is not aligned to 16 bytes.
- **Resolved Issues**
  - cusparseSpVV() provided incorrect results when the sparse vector has many non-zeros.

6.4.8. cuSPARSE: Release 12.3 Update 1

- **New Features**
  - Added support for block sizes of 64 and 128 in cusparseSDDMM().
  - Added a preprocessing step cusparseSDDMM_preprocess() for BSR cusparseSDDMM() that helps improve performance of the main computing stage.

6.4.9. cuSPARSE: Release 12.3

- **New Features**
  - The cusparseSpSV_bufferSize() and cusparseSpSV_analysis() routines now accept NULL pointers for the dense vector.
  - The cusparseSpSM_bufferSize() and cusparseSpSM_analysis() routines now accept dense matrix descriptors with NULL pointer for values.
- **Known Issues**
  - The cusparseSpSV_analysis() and cusparseSpSM_analysis() routines are blocking calls/not asynchronous.
  - Wrong results can occur for cusparseSpSV() using sliced ELLPACK format and transpose/transpose conjugate operation on matrix A.
- **Resolved Issues**
  - cusparseSpSV() provided indeterministic results in some cases.
  - Fixed an issue that caused cusparseSpSV_analysis() to hang sometimes in a multi-thread environment.
  - Fixed an issue with cusparseSpSV() and cusparseSpSV() that sometimes yielded wrong output when the output vector/matrix or input matrix contained NaN.

6.4.10. cuSPARSE: Release 12.2 Update 1

- **New Features**
  - The library now provides the opportunity to dump sparse matrices to files during the creation of the descriptor for debugging purposes. See logging API https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-logging-api.
- **Resolved Issues**
  - Removed `CUSPARSE_SPMM_CSR_ALG3` fallback to avoid confusion in the algorithm selection process.
  - Clarified the supported operations for `cusparseSDDMM()`.
  - `cusparseCreateConstSlicedEll()` now uses `const` pointers.
  - Fixed wrong results in rare edge cases of `cusparseCsr2CscEx2()` with base 1 indexing.
  - `cusparseSpSM_bufferSize()` could ask slightly less memory than needed.
  - `cusparseSpMV()` now checks the validity of the buffer pointer only when it is strictly needed.
- **Deprecations**
  - Several legacy APIs have been officially deprecated. A compile-time warning has been added to all of them.

6.4.11. cuSPARSE: Release 12.1 Update 1

- **New Features**
  - Introduced Block Sparse Row (BSR) sparse matrix storage for the Generic APIs with support for SDDMM routine (`cusparseSDDMM`).
  - Introduced Sliced Ellpack (SELL) sparse matrix storage format for the Generic APIs with support for sparse matrix-vector multiplication (`cusparseSpMV`) and triangular solver with a single right-hand side (`cusparseSpSV`).
  - Added a new API call (`cusparseSpSV_updateMatrix`) to update matrix values and/or the matrix diagonal in the sparse triangular solver with a single right-hand side after the analysis step.

6.4.12. cuSPARSE: Release 12.0 Update 1

- **New Features**
  - `cusparseSDDMM()` now supports mixed precision computation.
  - Improved `cusparseSpMM()` alg2 mixed-precision performance on some matrices on NVIDIA Ampere architecture GPUs.
  - Improved `cusparseSpMV()` performance with a new load balancing algorithm.
  - `cusparseSpSV()` and `cusparseSpSM()` now support in-place computation, namely the output and input vectors/matrices have the same memory address.

- **Resolved Issues**
  - cusparseSpSM() could produce wrong results if the leading dimension (ld) of the RHS matrix is greater than the number of columns/rows.

6.4.13. cuSPARSE: Release 12.0

- **New Features**
  - JIT LTO functionalities (cusparseSpMMOp()) switched from driver to nvJitLto library. Starting from CUDA 12.0 the user needs to link to libnvJitLto.so, see cuSPARSE documentation. JIT LTO performance has also been improved for cusparseSpMMOpPlan().
  - Introduced const descriptors for the Generic APIs, for example, cusparseConstSpVecGet(). Now the Generic APIs interface clearly declares when a descriptor and its data are modified by the cuSPARSE functions.
  - Added two new algorithms to cusparseSpGEMM() with lower memory utilization. The first algorithm computes a strict bound on the number of intermediate product, while the second one allows partitioning the computation in chunks.
  - Added int8_t support to cusparseGather(), cusparseScatter(), and cusparseCsr2cscEx2().
  - Improved cusparseSpSV() performance for both the analysis and the solving phases.
  - Improved cusparseSpSM() performance for both the analysis and the solving phases.
  - Improved cusparseSDDMM() performance and added support for batch computation.
  - Improved cusparseCsr2cscEx2() performance.
- **Resolved Issues**
  - cusparseSpSV() and cusparseSpSM() could produce wrong results.
  - cusparseDnMatGetStridedBatch() did not accept batchStride == 0.
- **Deprecations**
  - Removed deprecated CUDA 11.x APIs, enumerators, and descriptors.

6.5. Math Library

6.5.1. CUDA Math: Release 12.8 Update 1

- Users of the E8M0 (__nv_fp8_e0m8) types defined in cuda_fp8.h should be aware of a change in the rounding behavior for the C++ converting constructors when converting from other floating-point and integer types. The constructors now take the absolute value of the input and apply round-toward-positive-infinity rounding with saturation to convert to the E8M0 representation. Previously, the constructors used absolute value with round-toward-zero rounding and saturation. This previous behavior can now be accessed through specific conversion functions, such as __nv_cvt_bfloat16raw_to_e8m0. [*5066830*]

6.5.2. CUDA Math: Release 12.8

- **New Features**
  - Added support for several new floating point datatypes:
    - E2M1 (2-bit exponent, 1-bit mantissa)
    - E2M3 (2-bit exponent, 3-bit mantissa)
    - E3M2 (3-bit exponent, 2-bit mantissa)
    - E8M0 (8-bit exponent, 0-bit mantissa)

    For detailed information about FP4, FP6, and FP8 types, including conversion operators and intrinsics, refer to the CUDA Math API documentation. [*CUMATH-1385*]
  - Conversion operations for these types are natively supported by specific devices (e.g. devices of compute capability 10.0a), other devices use emulation path.
  - Optimized standard single precision hyperbolic tangent (`tanhf()`) function, achieving 30-40% faster performance. [*4557267*]
  - Added several new tanh implementations:
    - `__tanhf(float x)`: New fast reduced-accuracy math intrinsic
    - `htanh()` and `h2tanh()`: tanh functions for half and bfloat16 types in scalar and packed formats
    - `htanh_approx()` and `h2tanh_approx()`: Fast reduced-accuracy versions

    Refer to CUDA Math API documentation for detailed usage information. [*CUMATH-6821*]
  - Added support for quad-precision `__float128` data type and select math library operations in device computations on GPUs with compute capability 10.0 and above. Refer to CUDA Math API documentation for details. [*CUMATH-5463*]
- **Known Issues**
  - When converting to MXFP4/MXFP6/MXFP8 formats developers should not use the C++ converting constructors, which currently implement only round-toward-zero behavior. Conversions to MXFP formats should use round-toward-positive-infinity, which is implemented as an option in conversion functions like `__nv_cvt_bfloat16raw_to_e8m0`. C++ converting constructors behavior will change in a future update.

6.5.3. CUDA Math: Release 12.6 Update 1

- **Resolved Issues**
  - Issue 4731352 from release 12.6 is resolved.

6.5.4. CUDA Math: Release 12.6

- **Known Issues**
  - As a result of ongoing compatibility testing NVIDIA identified that a number of CUDA Math Integer SIMD APIs silently produced wrong results if used on the CPU in programs compiled with MSVC 17.10. The root cause is found to be the coding error in the header-based implementation of the APIs exposed to the undefined behavior during narrowing integer conversion when doing a host-based emulation of the GPU functionality. The issue will be fixed in a future release of CUDA. Applications affected are those calling `__vimax3_s16x2`, `__vimin3_s16x2`, `__vibmax_s16x2`, and `__vibmin_s16x2` on the CPU and not in CUDA kernels. [*4731352*]

6.5.5. CUDA Math: Release 12.5

- **Known Issues**
  - As a result of ongoing testing we updated the interval bounds in which double precision `lgamma()` function may experience greater than the documented 4 ulp accuracy loss. New interval shall read (-23.0001; -2.2637). This finding is applicable to CUDA 12.5 and all previous versions. [*4662420*]

6.5.6. CUDA Math: Release 12.4

- **Resolved Issues**
  - Host-specific code in `cuda_fp16/bf16` headers is now free from type-punning and shall work correctly in the presence of optimizations based on strict-aliasing rules. [*4311216*]

6.5.7. CUDA Math: Release 12.3

- **New Features**
  - Performance of SIMD Integer CUDA Math APIs was improved.
- **Resolved Issues**
  - The `__hisinf()` Math APIs from `cuda_fp16.h` and `cuda_bf16.h` headers were silently producing wrong results if compiled with the `-std=c++20` compiler option because of an underlying nvcc compiler issue, resolved in version 12.3.
- **Known Issues**
  - Users of `cuda_fp16.h` and `cuda_bf16.h` headers are advised to disable host compilers strict aliasing rules based optimizations (e.g. pass `-fno-strict-aliasing` to host GCC compiler) as these may interfere with the type-punning idioms used in the `__half`, `__half2`, `__nv_bfloat16`, `__nv_bfloat162` types implementations and expose the user program to undefined behavior. Note, the headers suppress GCC diagnostics through: #pragma GCC diagnostic ignored `-Wstrict-aliasing`. This behavior may improve in future versions of the headers.

6.5.8. CUDA Math: Release 12.2

- **New Features**
  - CUDA Math APIs for `__half` and `__nv_bfloat16` types received usability improvements, including host side <emulated> support for many of the arithmetic operations and conversions.
  - `__half` and `__nv_bfloat16` types have implicit conversions to/from integral types, which are now available with host compilers by default. These may cause build issues due to ambiguous overloads resolution. Users are advised to update their code to select proper overloads. To opt-out user may want to define the following macros (these macros will be removed in the future CUDA release):
    - `__CUDA_FP16_DISABLE_IMPLICIT_INTEGER_CONVERTS_FOR_HOST_COMPILERS__`
    - `__CUDA_BF16_DISABLE_IMPLICIT_INTEGER_CONVERTS_FOR_HOST_COMPILERS__`
- **Resolved Issues**
  - During ongoing testing, NVIDIA identified that due to an algorithm error the results of 64-bit floating-point division in default round-to-nearest-even mode could produce spurious overflow to infinity. NVIDIA recommends that all developers requiring strict IEEE754 compliance update to CUDA Toolkit 12.2 or newer. The affected algorithm was present in both offline compilation as well as just-in-time (JIT) compilation. As JIT compilation is handled by the driver, NVIDIA recommends updating to driver version greater than or equal to R535 (R536 on Windows) when IEEE754 compliance is required and when using JIT. This is a software algorithm fix and is not tied to specific hardware.
  - Updated the observed worst case error bounds for single precision intrinsic functions `__expf()`, `__exp10f()` and double precision functions `asinh()`, `acosh()`.

6.5.9. CUDA Math: Release 12.1

- **New Features**
  - Performance and accuracy improvements in `atanf`, `acosf`, `asinf`, `sinpif`, `cospif`, `powf`, `erff`, and `tgammaf`.

6.5.10. CUDA Math: Release 12.0

- **New Features**
  - Introduced new integer/fp16/bf16 CUDA Math APIs to help expose performance benefits of new DPX instructions. Refer to https://docs.nvidia.com/cuda/cuda-math-api/index.html.
- **Known Issues**
  - Double precision inputs that cause the double precision division algorithm in the default 'round to nearest even mode' produce spurious overflow: an infinite result is delivered where `DBL_MAX 0x7FEF_FFFF_FFFF_FFFF` is expected. Affected CUDA Math APIs: `__ddiv_rn()`. Affected CUDA language operation: double precision / operation in the device code.
- **Deprecations**

- All previously deprecated undocumented APIs are removed from CUDA 12.0.

6.6. NVIDIA Performance Primitives (NPP)

6.6.1. NPP: Release 12.4

- **New Features**
  - Enhanced large file support with `size_t`.

6.6.2. NPP: Release 12.0

- **Deprecations**
  - Deprecating non-CTX API support from next release.
- **Resolved Issues**
  - A performance issue with the NPP `ResizeSqrPixel` API is now fixed and shows improved performance.

6.7. nvJPEG Library

6.7.1. nvJPEG: Release 12.8

- **New Features**
  - Added hardware-accelerated JPEG decoding support in nvJPEG for NVIDIA Blackwell architecture GPUs.
  - The nvJPEG library now uses significantly less GPU memory during encoding, achieving memory savings of 30% to 50%, depending on image size and chroma subsampling mode. For images larger than 5 MB (approximately 2K x 1K pixels) and popular subsampling modes such as 4:2:2 and 4:2:0, memory savings are around 50%. Additionally, nvJPEG no longer artificially runs out of memory when processing large or complex images, enhancing its reliability and performance.
- **Resolved Issues**
  - Resolved an issue in nvJPEG that prevented the correct encoding of very small images with dimensions less than 25 pixels. [*4655922*]
  - Fixed an issue that caused out-of-bound reads when decoding a truncated JPEG file using `nvjpegDecodeJpegHost` with the `NVJPEG_BACKEND_GPU_HYBRID backend`. [*4663831*]

6.7.2. nvJPEG: Release 12.4

- **New Features**
  - IDCT performance optimizations for single image CUDA decode.
  - Zero Copy behavior has been changed: Setting NVJPEG_FLAGS_REDUCED_MEMORY_DECODE_ZERO_COPY flag will no longer enable NVJPEG_FLAGS_REDUCED_MEMORY_DECODE.

6.7.3. nvJPEG: Release 12.3 Update 1

- **New Features**
  - New APIs: nvjpegBufferPinnedResize and nvjpegBufferDeviceResize which can be used to resize pinned and device buffers before using them.

6.7.4. nvJPEG: Release 12.2

- **New Features**
  - Added support for JPEG Lossless decode (process 14, FO prediction).
  - nvJPEG is now supported on L4T.

6.7.5. nvJPEG: Release 12.0

- **New Features**
  - Immproved the GPU Memory optimisation for the nvJPEG codec.
- **Resolved Issues**
  - An issue that causes runtime failures when nvJPEGDecMultipleInstances was tested with a large number of threads is resolved.
  - An issue with CMYK four component color conversion is now resolved.
- **Known Issues**
  - Backend NVJPEG_BACKEND_GPU_HYBRID - Unable to handle bistreams with extra scans lengths.
- **Deprecations**
  - The reuse of Huffman table in Encoder (nvjpegEncoderParamsCopyHuffmanTables).

Chapter 7. Notices

7.1. 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.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

7.2. OpenCL

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

7.3. 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