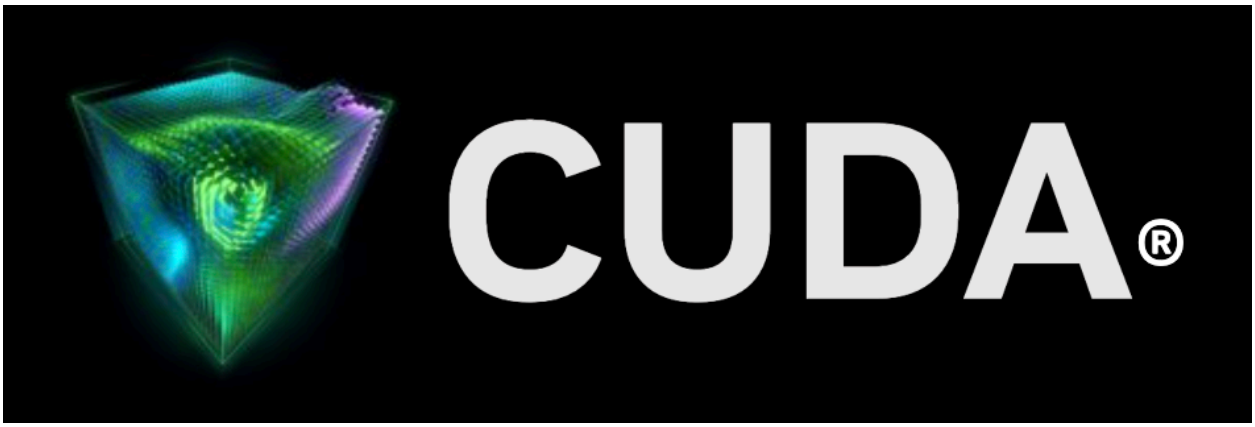


Release Notes

Release 13.5

NVIDIA Corporation

Nov 04, 2026

Contents

1 Overview **3**

2 CUDA Platform **5**
2.1 CUDA Toolkit Major Components . . . 5
2.2 CUDA Driver . . . 6
2.3 New Features . . . 9
2.3.1 CUDA Platform . . . 9
2.3.2 CUDA Developer Tools . . . 10
2.3.3 CUDA Compiler . . . 10
2.3.4 CUDA C++ Core Libraries (CCCL) . . . 11
2.3.5 CUDA Python . . . 12
2.3.6 CUDA Tile . . . 12
2.3.7 CUDA Tile IR . . . 13
2.4 Resolved Issues . . . 14
2.4.1 General CUDA . . . 14
2.4.2 CUDA Compiler . . . 15
2.4.3 CUDA Tools . . . 15
2.5 Known Issues . . . 15
2.5.1 CUDA Platform . . . 15
2.5.2 CUDA Compiler . . . 16
2.5.3 CUDA Tile IR . . . 16
2.6 Deprecated or Dropped Features . . . 16
2.6.1 CUDA Platform . . . 16
2.6.2 Architectures . . . 17
2.6.3 Operating Systems . . . 17
2.6.4 CUDA Toolchains . . . 17
2.6.5 Host Compiler . . . 17

3 CUDA Libraries **19**
3.1 cuBLAS Library . . . 19
3.1.1 cuBLAS: Release 13.5 . . . 19
3.1.2 cuBLAS: Release 13.4 Update 1 . . . 20
3.1.3 cuBLAS: Release 13.4 . . . 21
3.1.4 cuBLAS: Release 13.3 Update 1 . . . 23
3.1.5 cuBLAS: Release 13.3 . . . 24
3.1.6 cuBLAS: Release 13.2 Update 2 . . . 25
3.1.7 cuBLAS: Release 13.2 Update 1 . . . 26
3.1.8 cuBLAS: Release 13.2 . . . 27
3.1.9 cuBLAS: Release 13.1 Update 1 . . . 28
3.1.10 cuBLAS: Release 13.1 . . . 29
3.1.11 cuBLAS: Release 13.0 Update 2 . . . 31
3.1.12 cuBLAS: Release 13.0 Update 1 . . . 32
3.1.13 cuBLAS: Release 13.0 . . . 32

3.2 cuFFT Library . . . 33
3.2.1 cuFFT: Release 13.5 . . . 33
3.2.2 cuFFT: Release 13.4 . . . 34
3.2.3 cuFFT: Release 13.3 Update 1 . . . 34
3.2.4 cuFFT: Release 13.3 . . . 34
3.2.5 cuFFT: Release 13.2 . . . 34
3.2.6 cuFFT: Release 13.1 . . . 35
3.2.7 cuFFT: Release 13.0 Update 1 . . . 35
3.2.8 cuFFT: Release 13.0 . . . 35
3.3 cuSOLVER Library . . . 36
3.3.1 cuSOLVER: Release 13.5 . . . 36
3.3.2 cuSOLVER: Release 13.4 Update 1 . . . 37
3.3.3 cuSOLVER: Release 13.4 . . . 37
3.3.4 cuSOLVER: Release 13.3 Update 1 . . . 38
3.3.5 cuSOLVER: Release 13.3 . . . 38
3.3.6 cuSOLVER: Release 13.2 Update 1 . . . 38
3.3.7 cuSOLVER: Release 13.2 . . . 39
3.3.8 cuSOLVER: Release 13.1 . . . 39
3.3.9 cuSOLVER: Release 13.0 Update 1 . . . 39
3.3.10 cuSOLVER: Release 13.0 . . . 40
3.4 cuSPARSE Library . . . 41
3.4.1 cuSPARSE: Release 13.5 . . . 41
3.4.2 cuSPARSE: Release 13.4 . . . 41
3.4.3 cuSPARSE: Release 13.3 Update 1 . . . 42
3.4.4 cuSPARSE: Release 13.3 . . . 42
3.4.5 cuSPARSE: Release 13.2 Update 1 . . . 42
3.4.6 cuSPARSE: Release 13.2 . . . 43
3.4.7 cuSPARSE: Release 13.1 Update 1 . . . 43
3.4.8 cuSPARSE: Release 13.1 . . . 43
3.4.9 cuSPARSE: Release 13.0 Update 1 . . . 44
3.4.10 cuSPARSE: Release 13.0 . . . 44
3.5 Math Library . . . 45
3.5.1 CUDA Math: Release 13.5 . . . 45
3.5.2 CUDA Math: Release 13.4 . . . 45
3.5.3 CUDA Math: Release 13.3 . . . 45
3.5.4 CUDA Math: Release 13.2 Update 1 . . . 46
3.5.5 CUDA Math: Release 13.2 . . . 46
3.5.6 CUDA Math: Release 13.0 . . . 46
3.6 nvJPEG Library . . . 47
3.6.1 nvJPEG: Release 13.5 . . . 47
3.6.2 nvJPEG: Release 13.4 Update 1 . . . 47
3.6.3 nvJPEG: Release 13.3 Update 1 . . . 47
3.6.4 nvJPEG: Release 13.3 . . . 47
3.6.5 nvJPEG: Release 13.2 Update 2 . . . 48
3.6.6 nvJPEG: Release 13.2 Update 1 . . . 48
3.6.7 nvJPEG: Release 13.1 Update 1 . . . 48
3.6.8 nvJPEG: Release 13.1 . . . 48
3.6.9 nvJPEG: Release 13.0 Update 1 . . . 48
3.6.10 nvJPEG: Release 13.0 . . . 48
3.7 NPP Library . . . 49
3.7.1 NPP: Release 13.4 . . . 49
3.7.2 NPP: Release 13.1 . . . 49
3.7.3 NPP: Release 13.0 . . . 49

4 Notices **51**
4.1 Notice 51
4.2 OpenCL 52
4.3 Trademarks 52

CUDA Toolkit 13.5 - Release Notes

Chapter 1. Overview

Welcome to the release notes for NVIDIA® CUDA® Toolkit 13.5. This release includes enhancements and fixes across the CUDA Toolkit and its libraries.

This documentation is organized into two main sections:

- **CUDA Platform**

  Focuses on the core CUDA infrastructure including component versions, driver compatibility, compiler/runtime features, issues, and deprecations.

- **CUDA Libraries**

  Covers the specialized computational libraries with their feature updates, performance improvements, API changes, and version history across CUDA 13.x releases.

Chapter 2. CUDA Platform

2.1. CUDA Toolkit Major Components

For CUDA 13.5, the table below indicates the versions:

Table 1: CUDA 13.5 Component Versions

Component Name		Version Information	Supported Architec
CUDA C++ Core Compute Libraries	Thrust	3.5.0	x86_64, arm64-sbsa, arm64 (Windows)
	CUB	3.5.0	
	libcu++	3.5.0	
	Cooperative Groups	13.3.5.0.6	
CUDA Compatibility Package (Orin)		13.5.47659993	arm64-sbsa
CUDA Application Compiler (crt)		13.5.37	x86_64, arm64-sbsa
CUDA Compilation Optimizer (ctadvisor)		13.5.37	x86_64, arm64-sbsa
CUDA Runtime (cudart)		13.5.37	x86_64, arm64-sbsa
CUDA culibos		13.5.25	x86_64, arm64-sbsa
CUDA cuobjdump		13.5.37	x86_64, arm64-sbsa
CUPTI		13.5.25	x86_64, arm64-sbsa
CUDA cuxxfilt (demangler)		13.5.37	x86_64, arm64-sbsa
CUDA Documentation		13.5.25	x86_64, arm64-sbsa
CUDA GDB		13.5.25	x86_64, arm64-sbsa
CUDA NVCC		13.5.37	x86_64, arm64-sbsa
CUDA nvdisasm		13.5.37	x86_64, arm64-sbsa
CUDA NVML Headers		13.5.37	x86_64, arm64-sbsa
CUDA nvprune		13.5.37	x86_64, arm64-sbsa
CUDA NVRTC		13.5.37	x86_64, arm64-sbsa
CUDA NVTX		13.5.25	x86_64, arm64-sbsa

Table 1 – continued from previous page

Component Name	Version Information	Supported Architec
CUDA OpenCL	13.5.25	x86_64, arm64 (Win
CUDA Profiler API	13.5.25	x86_64, arm64-sbsa
CUDA Sandbox dev	13.5.25	x86_64, arm64-sbsa
CUDA Compute Sanitizer API	13.5.25	x86_64, arm64-sbsa
CUDA TILE-IR AS	13.5.37	x86_64, arm64-sbsa
CUDA cuBLAS	13.10.0.20	x86_64, arm64-sbsa
CUDA cuDLA	13.5.25	arm64-sbsa, arm64
CUDA cuFFT	12.5.0.14	x86_64, arm64-sbsa
CUDA cuFile	1.20.0.107	x86_64, arm64-sbsa
CUDA cuobjclient	1.4.0.107	x86_64, arm64-sbsa
CUDA cuRAND	10.4.5.25	x86_64, arm64-sbsa
CUDA cuSOLVER	12.4.1.2	x86_64, arm64-sbsa
CUDA cuSPARSE	12.9.4.2	x86_64, arm64-sbsa
CUDA NPP	13.2.0.93	x86_64, arm64-sbsa
CUDA nvFatbin	13.5.37	x86_64, arm64-sbsa
CUDA nvJitLink	13.5.37	x86_64, arm64-sbsa
CUDA nvJPEG	13.3.2.43	x86_64, arm64-sbsa
CUDA nvptxcompiler	13.5.37	x86_64, arm64-sbsa
CUDA nvvm	13.5.37	x86_64, arm64-sbsa
Nsight Compute	2026.4.0.10	x86_64, arm64-sbsa
Nsight Systems	2026.5.2.323	x86_64, arm64-sbsa
Nsight Visual Studio Edition (VSE)	2026.4.0.26265	x86_64, arm64 (Win
Visual Studio Integration	13.5.25	x86_64, arm64 (Win

2.2. CUDA Driver

Note: The NVIDIA driver is **no longer bundled** with the CUDA Toolkit – on Windows starting with CUDA 13.1, and on Linux starting with CUDA 13.4. Download and install the appropriate driver from the official NVIDIA Driver Downloads page.

Running a CUDA application requires a system with at least one CUDA-capable GPU and a driver that is compatible with the CUDA Toolkit. For more information about various GPU products that are CUDA-capable, visit https://developer.nvidia.com/cuda/gpus.

The NVIDIA driver branch corresponding to each CUDA Toolkit release is shown below. Update releases

within a CUDA minor version use the same driver branch.

Table 2: CUDA Toolkit and Corresponding Driver Branch

CUDA Toolkit	Corresponding Driver Branch
CUDA 13.5	R620
CUDA 13.4	R615
CUDA 13.3	R610
CUDA 13.2	R595
CUDA 13.1	R590
CUDA 13.0	R580

Note: Existing CUDA 13.x applications run on drivers >=580 under CUDA minor version compatibility. CUDA 13.5 new features and newly enabled platforms require an R620 or later driver that supports them. The Windows driver for the RTX Spark device is 616.41 or later.

The CUDA driver is backward compatible: applications compiled against a particular CUDA Toolkit version continue to work on subsequent (later) driver releases. In addition, CUDA minor version compatibility allows applications to run on a driver older than the corresponding driver branch, within the ranges shown below. The installed driver must meet or exceed the minimum required version for the CUDA Toolkit. For details, see the CUDA Compatibility Guide and CUDA Compatibility and Upgrades.

Table 3: CUDA Toolkit and Minimum Required Driver Version for CUDA Minor Version Compatibility

CTK Version	Driver Range for Minor Version Compatibility	
	Min	Max
13.x	>= 580	N/A
12.x	>= 525	< 580
11.x	>= 450	< 525

CUDA 11.0 shipped with earlier driver versions. Minor-version compatibility across the CUDA 11.x family requires driver version 450.80.02 or later on Linux, or 452.39 or later on Windows.

Older CUDA versions (12.9 and earlier)

Table 4: Corresponding Driver Versions (CUDA 12.9 and earlier)

CUDA Toolkit	Corresponding Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.9 Update 1	>=575.57.08	>=576.57

continues on next page

Table 4 – continued from previous page

CUDA Toolkit	Corresponding Driver Version	
CUDA 12.9 GA	>=575.51.03	>=576.02
CUDA 12.8 Update 1	>=570.124.06	>=572.61
CUDA 12.8 GA	>=570.26	>=570.65
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
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

continues on next page

Table 4 – continued from previous page

CUDA Toolkit	Corresponding Driver Version	
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
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

2.3. New Features

2.3.1. CUDA Platform

- CUDA APIs that set up or validate peer access, such as `cuDeviceCanAccessPeer()` and `cuCtxEnablePeerAccess()`, now also check whether the IOMMU and PCI ACS system topology is supported for P2P usage. This applies only to multi-GPU systems that use PCIe-based P2P. To override the result of these checks, re-run the application with `CUDA_DISABLE_IOMMU_ACS_P2P_CAPABILITY_CHECK=1` set. *[6977011]*

- Added Confidential Computing (CC) support for NVENC on NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs. Supported Linux confidential virtual machines can perform hardware-accelerated H.264, HEVC, and AV1 encoding through the existing CUDA-based NVENCODE APIs in the NVIDIA Video Codec SDK. This feature requires a qualified R620 driver and firmware configuration. The driver manages protected encoder buffers and encrypted, authenticated CPU-GPU transfers. *[6324055]*
- Added low-overhead CUDA, NVTX, and HWPM tracing support through Nsight Perf Always-On AI profiling. For more information, see the Nsight Perf documentation. *[6957884]*

2.3.2. CUDA Developer Tools

For details on new features, improvements, and bug fixes, see the changelogs for:

- Nsight Systems.
- Nsight Visual Studio Edition.
- CUPTI.
- Nsight Compute.
- Compute Sanitizer.
- NVML.
- cuda-gdb.

2.3.3. CUDA Compiler

- PTX ISA 9.5 is supported. For new PTX features, see the PTX ISA 9.5 documentation.
- On devices with compute capability 10.0 and higher, added compiler support for 1-byte, 2-byte, and 16-byte atomic operations with explicit memory ordering and synchronization scope, and for single-precision and double-precision floating-point atomic min and max operations.
- The NVCC and PTXAS `--position-independent-code` (`-pic`) option is deprecated. Whole-program device compilation continues to generate position-independent code by default in eligible compilation modes. Previously generated position-independent CUDA binaries remain supported. This change does not affect position-independent host code generation.
- Added the `__sin2pif`, `__cos2pif`, and `__sincos2pif` device intrinsics for fast approximate single-precision trigonometric calculations with inputs expressed in cycles. Corresponding PTX `.2pi` variants of the `sin` and `cos` instructions avoid an additional angle-conversion operation for workloads that already use cycle-based phase.
- Extended support for Advanced Controls Files (ACFs) to the PTX Compiler and NVVM libraries. Applications and frameworks can apply CompileIQ-generated compiler controls within their existing compilation workflows, extending access to workload-specific compiler tuning.
- Added device-side control of TMA out-of-bounds addressing. CUDA 13.5 extends PTX tensor-map operations with allocation-class controls for Tensor Memory Accelerator (TMA) out-of-bounds addressing. Kernels can select a small or large allocation class, preserve the existing class when replacing a tensor map's global address, and query whether a tensor map uses the large

class. This enables kernels to update tensor-map addressing behavior when reusing descriptors across different tensor allocations. Replacing the global address without an allocation-class modifier selects the small class by default.

- Added PTX queries for GPU locality domains and supplemental SMs on Blackwell and newer architectures, enabling kernels to adapt memory accesses and work distribution to GPU locality.
- Added a `--precompile` option to `nvprune` that compiles intermediate device code (PTX, Tile IR) to native SASS/ELF during the prune step. This eliminates runtime JIT overhead for downstream consumers who deploy fat binaries to known target architectures, such as cloud providers and package distributors.
- Improved ELF symbol-table readability in `cuobjdump`.
- Added opt-in support for the newer LLVM-based compiler backend in libNVVM, nvlink, and nvJitLink for pre-Blackwell GPUs. Enable it with `-nvvm-latest`: pass it to `nvvmCompileProgram()` for libNVVM and to `nvJitLinkCreate()` for nvJitLink. For nvlink, it applies to link-time optimization, so use it with `-dlto`, or through NVCC as `-dlto --nvlink-options=-nvvm-latest`. Blackwell and newer GPU architectures already use the latest LLVM-based compiler backend by default.

Note: There can be performance impacts between different versions of LLVM-based compiler backends.

2.3.4. CUDA C++ Core Libraries (CCCL)

- Added public tuning APIs for CUB device-wide algorithms through `cuda::execution::tune`, supporting customization of block size, items per thread, vectorization, and algorithm selection by GPU compute capability. See the tuning documentation.
- Added run-to-run reproducibility for floating-point summation in `cub::DeviceScan` through `cuda::execution::determinism::run_to_run`. Results are reproducible for the same input, GPU, and build configuration. See the DeviceScan documentation.
- Added `cub::DeviceBatchedTopK` for selecting the smallest or largest K keys or key-value pairs across multiple segments. Supports variable segment sizes and per-segment K values. Initial support requires bounded segment sizes and explicitly requested nondeterministic, unsorted output. See the API reference.
- Added device-resident problem-size support to `cub::DeviceReduce::{Reduce, Sum, Min, Max, TransformReduce}` using `cuda::args::deferred`. Captured reductions can consume updated element counts on each CUDA Graph replay without graph updates or recapture. See the DeviceReduce documentation.
- Added `cub::DeviceFind::LowerBoundSortedValues` and `UpperBoundSortedValues` for batched binary searches when both the searched range and query values are sorted under the same comparator. See PR #8780.
- Added Programmatic Dependent Launch support to `cub::DeviceRadixSort` and vectorized `cub::BlockLoad`/`cub::BlockStore` operations for contiguous iterators.
- Added `cuda::std::ranges::zip_view` and `cuda::std::views::zip`, with expanded tuple-like construction and assignment support in `cuda::std::tuple` and `cuda::std::pair`.
- Extended `<cuda/std/bit>` with `bit_compress`, `bit_expand`, `bit_repeat`, `bit_reverse`, `shl`, and `shr`.

For complete details, see the CCCL 3.5 release notes.

2.3.5. CUDA Python

- Added batched memory copies through `cuda.core.utils.copy_batch()`, with `CopyOptions` controlling source access ordering, memory-location hints, and overlap behavior. Requires a CUDA 13 build and CUDA bindings/driver 13.0 or newer.
- Added Programmatic Dependent Launch through `LaunchConfig.programmatic_stream_serialization` on GPUs with compute capability 9.0 or newer. Cooperating kernels can overlap execution within the same stream.
- Added `ProgramOptions.use_bundled_headers` to compile with CUDA and CCCL headers bundled with NVRTC, without requiring a full CUDA Toolkit installation. Requires NVRTC 13.3 or newer.
- Added graph-definition node updates and executable-node updates through `graph[node]`, allowing parameter changes without reinstantiating the executable graph. Kernel, memcpy, and memset nodes can also be enabled or disabled. Requires CUDA bindings/driver 12.2 or newer.
- Improved CUDA-GDB source-level debugging for JIT-compiled kernels by preserving NVRTC input source when debug or lineinfo is enabled.

For complete details, see the cuda.core 1.2 release notes.

2.3.6. CUDA Tile

Tile C++

- Added support for `return` statements inside `for`, `while`, `do`, and `switch` statements in tile code.
- Added `ct::scaled_mma()` for block-scaled matrix multiplication.
- Added a `ct::tile` constructor that accepts a list of compile-time constant scalars.
- Added `ct::insert()` for inserting a subtile into a larger tile.
- Added `ct::repack()` for reinterpreting data across multiple tile elements.
- Added masked atomic read-modify-write methods to `ct::partition_view` and `ct::strided_view`, with defined behavior for partially out-of-bounds chunks.
- Added `ct::powi()` and integer-exponent support in `ct::pow()`.
- Added support for `__nv_fp8_e8m0`, `__nv_fp8_ue5m3`, and `__nv_fp4_e2m1` as tile elements.
- Added rounding and saturation modes to `ct::element_cast()`.
- Added padding support when loading restricted floating-point types through partition, strided, and gather-scatter views.
- Added support for optimization hints on variable declarations and return statements.
- Expanded `num_worker_warps_per_cta` to accept powers of two from 1 through 32 (previously limited to 4 and 8).

For a complete list of new features, see the CUDA Tile C++ API Reference release notes.

Tile Python

- Added Windows ARM64 support, including `win_arm64` wheels.
- Added `CUDA_TILE_LOGS=TILEIR` and `CUDA_TILE_DUMP_TILEIR` for printing and saving Tile IR MLIR in text format. Requires CUDA Toolkit 13.4 or later.
- Added support for `str.join()` and string alternatives to enum arguments in built-in functions.
- Added compile-time exceptions through `raise ct.static_exception(...)`.
- Extended dataclass support with default comparisons, user-defined arithmetic and comparison operators, and `len()`.
- Added a `flush_l2` option to `ct.tune.exhaustive_search()` to control L2 cache flushing before each benchmarked kernel execution. Defaults to `True`.
- Removed the requirement for a `ct.Constant` parameter annotation when exporting kernels with `ConstantConstraint`.

For details, see the cuTile Python 1.7.0 release notes.

2.3.7. CUDA Tile IR

Supported Architectures

- Added support for `f8E4M3FN` and `f8E5M2` types on Ada (`sm_89`).

New Operations

- Added op `cuda_tile.multimem_load_reduce_view_tko` to load and reduce corresponding elements across a multicast-backed tile view, returning the reduced tile. Supported for multicast-backed `cuda_tile.partition_view` on `sm_100`, `sm_103`, and `sm_107`, when GPUs are connected through an NVSwitch fabric with NVLink SHARP support.
- Added op `cuda_tile.multimem_reduce_view_tko` to atomically reduce an input tile elementwise into each destination associated with a multicast-backed view. Supported for multicast-backed `cuda_tile.partition_view` on `sm_100`, `sm_103`, and `sm_107`, when GPUs are connected through an NVSwitch fabric with NVLink SHARP support.
- Added op `cuda_tile.multimem_store_view_tko` to store the same tile to each destination of a multicast-backed view. Supported for multicast-backed `cuda_tile.partition_view` on `sm_100`, `sm_103`, and `sm_107`, when GPUs are connected through an NVSwitch fabric with NVLink SHARP support.
- (`--preview=simt` only) Added op `cuda_tile.preview$call` to invoke SIMT library functions from within CUDA Tile IR kernels. This feature is currently in preview and is not portable to future versions. For details, see the SIMT interoperability documentation.

Modified Operations

- Modified op `cuda_tile.atomic_red_view_tko` to add the `inbounds` boolean array attribute, allowing programs to guarantee that the indicated dimensions are always in-bounds and permitting the compiler to omit bounds checks for those dimensions.
- Modified op `cuda_tile.atomic_red_view_tko` to accept views carrying `padding_value`.

New and Modified Attributes

- Added the `multimem_reduction_kind` attribute with `add`, `min`, `max`, `umin`, `umax`, `and`, `or`, and `xor` variants.
- Modified the `ptr_attr` attribute to add `unicast` and `multicast` variants to represent pointers to memory across multiple GPUs. The classification carries over to a tile view created from the pointer. `unicast` denotes peer-to-peer access over NVLink; unicast-backed views are supported in normal view-based memory operations. `multicast` denotes one-to-many and additionally requires an NVSwitch fabric with NVLink SHARP support. Memory accesses through multicast-backed views must use explicit multimem operations. These pointers must be created externally and passed to CUDA Tile IR kernels. For details, see the multi-GPU communication documentation.

Compiler Flags

- Added support for `--apply-controls=<file>` in `tileiras` to apply encrypted compiler controls to the NVVM and PTXAS backends during Tile IR compilation. This enables backend tuning without modifying the Tile IR program.

Tile Compiler Improvements

- Introduced a new preview classification for compiler features. Preview features are unstable, not compatible across versions, and always explicitly opt-in. The debut preview category, `simt`, allows calling SIMT-implemented library functions and requires explicit enablement with `--preview=simt`. For details, see the preview features documentation.
- The `num_worker_warps_per_cta` optimization hint now accepts any power-of-two value from 1 to 32 (formerly only 4 and 8).
- Added a new opt-in optimization remark indicating when the compiler cannot automatically narrow a tensor view index to `i32`.
- Added a warning for unusually high estimated per-thread register usage to help identify excessive compilation times, with suggestions to reduce tile size or lower the occupancy hint.

Documentation Improvements

- Clarified the `f4E2M1FN` memory access contract and explicitly detailed the view conventions required for access to be well-formed.
- Clarified floating-point behavior in several places, including strict widening conversions for `cuda_tile.ftof`, sign-preserving flush-to-zero behavior, and the lack of bit-identical MMA results across programs, toolchains, or architectures.
- Expanded token ordering documentation with examples of races caused by missing dependencies and legal reordering of independent weak memory accesses.

2.4. Resolved Issues

2.4.1. General CUDA

- The CUDA Installation Guide for Linux (Host Compiler Support Policy) incorrectly stated that only libstdc++ is supported. It now documents libc++ support, available since CUDA 13.4 on Linux x86_64 and Arm64 sbsa with Clang 22 or later, libc++ 22 or later, and the `-stdlib=libc++` flag. libc++ is not supported on Windows. *[6930543]*

2.4.2. CUDA Compiler

- Fixed intermittent illegal memory accesses on Blackwell and newer GPUs caused by incompatible TMA out-of-bounds addressing modes. Recompile kernels that update tensor-map global addresses with `tensormap.replace` to obtain the corrected behavior. *[5795304]*
- Fixed a segmentation fault in PTXAS for PTX programs containing `stmatrix` instructions compiled with the `-g` option. This occurred for a specific usage pattern of `stmatrix` instructions whose source registers are used across basic blocks. *[6713855]*

2.4.3. CUDA Tools

- None

2.5. Known Issues

2.5.1. CUDA Platform

- When using CFT with MPS MLOPart enabled, importing unicast logical endpoints via `cuLogicalEndpointImport()` may fail with `CUDA_ERROR_NOT_SUPPORTED`, or `cuLogicalEndpointQuery()` may fail with `CUDA_ERROR_INVALID_VALUE`. This will be fixed in a future driver version.
- NVENC Confidential Computing support has the following limitations:
  - Internal and external lookahead, weighted prediction, external motion-estimation hints, temporal SVC, motion-estimation-only mode, and sub-frame write are unsupported.
  - Ultra-high-quality tuning for H.264 and HEVC, output bitstream in video memory, and H.264/HEVC buffering-period SEI insertion are unsupported.
  - Encoding configurations with both width >= 7680 and height >= 4320 are unsupported.
  - OpenGL device interoperability is unsupported.
  - Registering CUDA-array input resources with ARGB, ABGR, ARGB10, ABGR10, AYUV, YV12, or IYUV formats is unsupported.

  Gated requests return `NV_ENC_ERR_UNSUPPORTED_DEVICE` during encoder initialization or resource registration. Applications can use `nvEncGetLastErrorString` to obtain the failure reason. *[6324055]*

2.5.2. CUDA Compiler

- Fabric `try_atom` instructions may write to an incorrect destination when the logical endpoint offset is not aligned to a 16-byte boundary, potentially producing incorrect results. This will be fixed in the upcoming toolkit version. *[6875657]*

2.5.3. CUDA Tile IR

- A `cuda_tile.loop` or `cuda_tile.for` operation may fail compilation or produce incorrect results when a tile produced by a load is carried between iterations. The issue may occur when the next iteration's tile is loaded before the carried tile's final use. As a workaround, place the next load after the carried tile's final use, or restructure the loop so the loaded tile is not carried between iterations.
- Converting a `tf32` tile loaded from global memory to `f32` may produce incorrect values if the lower 13 mantissa bits of any loaded value are nonzero. These bits do not affect MMA operations on the loaded tile; the issue becomes observable only after conversion to `f32`. As a workaround, ensure that the lower 13 mantissa bits of every `tf32` value loaded from global memory are zero.
- An indexing signedness issue can cause miscompilation to TMA for view-based memory operations when the underlying tensor has a large number of elements. Loading or storing to a view whose underlying `cuda_tile.tensor_view` has 2^31 elements or more along a single dimension can trigger the miscompilation. As a workaround, limit the size of tensors to 2^31-1 per dimension, or set the `allow_tma` optimization hint to `false` on affected loads and stores.
- The specifications of `cuda_tile.scan` and `cuda_tile.reduce` implicitly allow several behaviors that are not yet fully supported. The body of a scan or reduce op capturing SSA values from an enclosing scope, using non-scalar intermediate tiles, or nesting region-bearing operations may cause miscompilation.
- Floating-point atomic addition (`addf`) in `cuda_tile.atomic_rmw_tko` and `cuda_tile.atomic_red_view_tko` currently flushes subnormal values to zero. This behavior is not intended and is expected to change in a future compiler version; programs should not rely on it.

2.6. Deprecated or Dropped Features

2.6.1. CUDA Platform

- Support for the `nvidia-peermem.ko` kernel module is deprecated and will be removed in the upcoming CUDA Toolkit release. Since GPU driver R515, GPUDirect RDMA is supported on NVIDIA Mellanox HCAs/NICs via the Linux kernel DMA-BUF mechanism provided by `nvidia.ko`, when using the open GPU kernel module and a Linux kernel 5.12 or newer. Customers using the proprietary GPU kernel modules should use a long-term support branch of the GPU driver. *[6958218]*

2.6.2. Architectures

- CUDA 14.0 will move to Armv8.2-A as the minimum supported architecture for ARM64-SBSA.

2.6.3. Operating Systems

- Debian 12 is deprecated as of CUDA 13.5. Support ends with this release; it will be dropped in the next CUDA release. Migrate to Debian 13.
- openSUSE Leap 15.6 is deprecated as of CUDA 13.5. Support ends with this release; it will be dropped in the next CUDA release. Migrate to openSUSE Leap 16.

2.6.4. CUDA Toolchains

- None

2.6.5. Host Compiler

- Support for the Arm C/C++ compiler included in Arm Compiler for Linux (ACfL) 24.10 is deprecated in CUDA 13.5 and will be removed in a future CUDA release. Migrate to a supported host compiler listed in the CUDA Installation Guide for Linux.

Chapter 3. CUDA Libraries

This section covers CUDA Libraries release notes for 13.x releases.

Note: Documentation will be updated to accurately reflect supported C++ standard libraries for CUDA Math Libraries.

3.1. cuBLAS Library

3.1.1. cuBLAS: Release 13.5

- **New Features**
  - Improved performance for low-alignment BF16 matmuls on Hopper and newer GPUs. *[6599136]*
  - Enabled Ozaki-II support for `{D,Z}SYR{K,2K,KX}` and `ZHER{K,2K,KX}` on NVIDIA Ampere and newer GPUs. Peak performance is improved by up to 1.7x on Vera Rubin, 2.9x on RTX PRO 6000 Blackwell Server Edition, and 1.2x on GB200 GPUs. *[CUB-10008]*
  - Improved performance of `cublas<t>getrsBatched` on all NVIDIA GPUs, with end-to-end speedups of up to 1.14x for complex, large batch-count, small problems on Hopper and Blackwell data center GPUs. *[5758350]*
- **Known Issues**
  - Ozaki-II double-complex BLAS Level 3 routines can produce incorrect results in rare cases. This can happen for elements produced by dot products of nearly collinear vectors whose components are roughly the same magnitude. This impacts Ampere and newer GPUs and has been present since CUDA Toolkit 13.4. *[CUB-10730]*
  - Fixed-point emulation can produce incorrect results when the leading dimensions differ between the C and D matrices or when split-K is used. This impacts Ampere and newer GPUs and has been present since CUDA Toolkit 13.0 Update 2. *[CUB-10744]*
- **Resolved Issues**
  - Fixed an issue where GEMM kernels could produce incorrect results on Hopper GPUs for FP16, BF16, and FP8 matrices when the process ran as an MPS client with `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` below 100. This affected only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 with cluster configurations using 4 or more CTAs. This issue was introduced in CUDA Toolkit 12.6 Update 2 (cuBLAS 12.6.3). *[6862950]*

- Fixed an issue where fixed-point emulation could produce incorrect results when run with the experimental exponent span capacity (ESC) algorithm, which is enabled only with `CUBLAS_ENABLE_EXPERIMENTAL_ESC=1`. This could occur on Ampere and newer GPUs and was introduced in CUDA Toolkit 13.4. *[6714114]*
- Fixed an issue where Ozaki-I fixed-point emulation kernels (FP64 and complex FP64) could produce incorrect results or data corruption on Blackwell GPUs when run concurrently with third-party kernels that also allocate tensor memory. *[CUB-9633]*
- Fixed an issue where, under concurrent load on the GPU, GEMM kernels with beta not equal to zero could produce incorrect results on Blackwell, Blackwell Ultra, and Rubin GPUs with algorithm configurations in which `CUBLASLT_ALGO_CONFIG_ID` is equal to 66 and the cluster has 8 or more CTAs. This issue affected all cuBLAS versions starting with CUDA Toolkit 12.8 (cuBLAS 12.8.1). In testing, the issue was reproducible only for configurations returned by heuristics for NVFP4, but problematic algorithm configurations can be constructed manually for all precisions. *[6854729]*
- `cublas<t>getrsBatched` now returns `CUBLAS_STATUS_INVALID_VALUE` for a negative `batchSize`.
- Fixed an issue where cuBLASLt Grouped GEMM with per-batch tensor-wide scales accessed input scales for groups with `m > 0`, `n > 0`, and `k = 0`, causing an invalid memory access. This issue was introduced in CUDA Toolkit 13.1 (cuBLAS 13.2.0). *[CUB-10481]*
- Fixed an issue where cuBLASLt Grouped GEMM operations could produce incorrect results on Blackwell and Rubin GPUs. This issue was introduced in cuBLAS 13.7.0. *[6798616]*

3.1.2. cuBLAS: Release 13.4 Update 1

- **New Features**
  - Emulated FP64 matrix multiplications:
    - Improved ZGEMM peak performance by up to 25% on Rubin GPUs.
    - When the per-handle workspace is insufficient, fixed-point FP64 emulation now allocates its temporary workspace from a cuBLAS-managed, per-device CUDA memory pool that retains memory across stream synchronizations. This avoids repeated allocation overhead without requiring users to tune the default memory pool.
- **Known Issues**
  - When running NVIDIA Compute Sanitizer (versions up to CUDA Toolkit 13.4) with cuBLAS-linked workloads, the tool may report false-positive errors regarding invalid global reads or out-of-bounds (OOB) memory accesses inside cuBLAS accelerated kernels. This is a known issue caused by a limitation in Compute Sanitizer's ability to accurately identify the PTX semantics of the asynchronous global-to-shared memory copy instruction `cp.async.cg.shared.global [dst], [src], cp-size, src-size` when `src-size < cp-size`. In this specific scenario, where the number of bytes to copy exceeds the source buffer size, the hardware guarantees that the remaining bytes (`cp-size - src-size`) in the destination shared memory are automatically padded with zeros, ensuring that the memory access is neither invalid nor OOB. Hence, Compute Sanitizer reports for these specific cases can be safely ignored. *[6196059]*
  - cuBLASLt Grouped GEMM with per-batch tensor-wide scales causes an invalid memory access for groups where `m > 0`, `n > 0`, and `k = 0`. As a workaround, always pass valid scale

pointers for each group. This issue was introduced in CUDA Toolkit 13.1 (cuBLAS 13.2.0). *[CUB-10481]*

- cuBLASLt Grouped GEMM operations can lead to incorrect results on Blackwell and Rubin GPUs. This issue was introduced in CUDA Toolkit 13.4 (cuBLAS 13.7.0). *[6798616]*

- **Resolved Issues**
  - Fixed an issue where cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` could lead to incorrect results on Hopper GPUs when the number of waves was larger than 2. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - Fixed an issue where cuBLASLt Grouped GEMM operations could lead to incorrect results on B300 and Rubin GPUs when the input matrices used the NVFP4 data type, the number of waves was larger than 3, and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` was `CUBLASLT_MATMUL_STAGES_768xAUTO`. This issue was introduced in CUDA Toolkit 13.4 (cuBLAS 13.7.0). *[6681084]*
  - Fixed an issue where `cublasLtMatmul()` could return incorrect output when using split-K if the number of K splits (`SPLITK_NUM`) did not evenly divide the K dimension and the floor division of K / `SPLITK_NUM` was an exact multiple of the stage size. This affected only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 9.0, 10.x, and 11.x. This issue was introduced in CUDA Toolkit 12.6 Update 2 (cuBLAS 12.6.3). *[6580689]*
  - Fixed an issue where `cublasLtMatmul()` could produce incorrect results on B300 and Rubin GPUs when the input matrices used the NVFP4 data type. This affected only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` set to `CUBLASLT_MATMUL_STAGES_768xAUTO`. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[CUB-10573]*

3.1.3. cuBLAS: Release 13.4

- **New Features**
  - Emulated FP64 matrix multiplications:
    - Emulated FP64 matrix multiplications now leverage the Ozaki-II scheme when it provides a performance benefit over the Ozaki-I scheme; the Ozaki-II scheme is supported on NVIDIA Ampere and newer GPUs.
    - On B200 and RTX PRO 6000 Blackwell Server Edition GPUs, this enables up to 175 and 45 TFLOPS of emulated DGEMM performance, respectively, and up to 295 and 70 TFLOPS of emulated ZGEMM performance, respectively, by leveraging the 2M algorithm.
    - Emulated FP64 matrix multiplications now support Rubin GPUs (compute capability 10.7), leveraging the Ozaki-I and Ozaki-II schemes and the new TI16 type via the `tcgen05.mma` instruction to achieve up to 212 TFLOPS of emulated DGEMM performance. Emulated ZGEMM can reach up to 301 TFLOPS, with further improvements to come.
  - cuBLASLt adds experimental support for an alternative scaling-factor layout for MXFP8 matmuls through the `CUBLASLT_MATMUL_MATRIX_SCALE_VEC32_MN_K4_UE8M0` and `CUBLASLT_MATMUL_MATRIX_SCALE_VEC128_MN_K4_UE8M0` scaling modes. For more information, see the cuBLAS documentation.
  - Added support for the NVIDIA Rubin (compute capability 10.7) GPU architecture.

- cuBLASLt Grouped GEMM performance is improved on Blackwell data center GPUs through dynamic scheduling of matrix computations. Improvements of up to 20% can be seen for Grouped GEMM calls with a large number of groups (for example, 32). *[5913842] [5992105] [CUB-9925]*

- **Known Issues**
  - When running NVIDIA Compute Sanitizer (versions up to CUDA Toolkit 13.4) with cuBLAS-linked workloads, the tool may report false-positive errors regarding invalid global reads or out-of-bounds (OOB) memory accesses inside cuBLAS accelerated kernels. This is a known issue caused by a limitation in Compute Sanitizer's ability to accurately identify the PTX semantics of the asynchronous global-to-shared memory copy instruction `cp.async.cg.shared.global [dst], [src], cp-size, src-size` when `src-size < cp-size`. In this specific scenario, where the number of bytes to copy exceeds the source buffer size, the hardware guarantees that the remaining bytes (`cp-size - src-size`) in the destination shared memory are automatically padded with zeros, ensuring that the memory access is neither invalid nor OOB. Hence, Compute Sanitizer reports for these specific cases can be safely ignored. *[6196059]*
  - cuBLASLt Grouped GEMM with per-batch tensor-wide scales causes an invalid memory access for groups where `m > 0`, `n > 0`, and `k = 0`. As a workaround, always pass valid scale pointers for each group. This issue was introduced in CUDA Toolkit 13.1 (cuBLAS 13.2.0). *[CUB-10481]*
  - `cublasLtMatmul()` can return incorrect output when using split-K if the number of K splits (`SPLITK_NUM`) does not evenly divide the K dimension and the floor division of K / `SPLITK_NUM` is an exact multiple of the stage size. This affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 9.0, 10.x, and 11.x. As a workaround, use `cublasLtMatmulAlgoConfigGetAttribute()` to query the number of K splits (`CUBLASLT_ALGO_CONFIG_SPLITK_NUM`) and the stage size (derived from `CUBLASLT_ALGO_CONFIG_STAGES_ID`), then use `cublasLtMatmulAlgoConfigSetAttribute()` to set a value that evenly divides the K dimension, or for which K / `SPLITK_NUM` is not an exact multiple of the stage size. This issue was first introduced in CUDA Toolkit 12.6 Update 2 (cuBLAS 12.6.3). *[6580689]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - cuBLASLt Grouped GEMM operations can lead to incorrect results on B300 and Rubin GPUs when the input matrices use the NVFP4 data type, the number of waves is larger than 3, and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` is `CUBLASLT_MATMUL_STAGES_768xAUTO`. As a workaround, skip such candidates when using `cublasLtMatmulAlgoGetHeuristic()`. *[6681084]*
  - `cublasLtMatmul()` can produce incorrect results on B300 and Rubin GPUs when the input matrices use the NVFP4 data type. This affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` is `CUBLASLT_MATMUL_STAGES_768xAUTO`. *[CUB-10573]*

- **Resolved Issues**
  - Between CUDA Toolkit 13.3 Update 1 and 13.4, cuBLAS released an independent patch release (cuBLAS 13.6.1) that resolves several issues. Refer to the associated cuBLAS patch release notes for details.

- Fixed an issue where `cublasLtMatmulAlgoGetHeuristic()` could return no algorithms, and `cublasLtMatmul()` could return `CUBLAS_STATUS_NOT_SUPPORTED`, for some NVFP4 matmuls with NVFP4 output and the `CUBLASLT_EPILOGUE_BIAS` epilogue on GPUs with compute capability 12.x. *[CUB-10222]*
- Fixed an issue where `cublasLtMatmulAlgoGetHeuristic()` returned `CUBLAS_STATUS_INTERNAL_ERROR` and invalidated an in-flight CUDA graph capture when called while a non-default blocking stream was being captured. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
- Fixed an issue where `cublas<t>gemv()` with `trans` equal to `CUBLAS_OP_T` or `CUBLAS_OP_C` could perform an illegal memory access when the number of elements addressed by the output vector (`n * incy`) exceeded the 32-bit index range. `cublasZgemv()` could additionally return `CUBLAS_STATUS_NOT_SUPPORTED` when `n * lda` exceeded that range. *[6207926]*
- Fixed an issue where `cublasZgemv()` with `trans` equal to `CUBLAS_OP_C` and `incy` greater than 1 produced incorrect results when FP64 fixed-point emulation was enabled (`CUBLAS_FP64_EMULATED_FIXEDPOINT_MATH` with `CUBLAS_EMULATION_STRATEGY_EAGER`). *[6207926]*

3.1.4. cuBLAS: Release 13.3 Update 1

- **New Features**
  - The TMA-based kernel (Hopper and newer) now accelerates DSYMV in addition to the already-enabled SSYMV. The 16-byte alignment requirement for the A pointer was dropped for this kernel, and support for atomics was added through `cublasSetAtomicsMode()`. The geomean speedup across architectures and datatypes is 1.3x, and up to 5.9x.
- **Known Issues**
  - Non-default epilogues are unintentionally allowed for `cublasLtMatmul()` with int8 inputs using regular data ordering and scale type `CUDA_R_32F`. This is an undocumented and lightly tested feature that users are discouraged from using, and it is planned for removal in the next major release. *[CUB-10067]*
  - In cuBLASLt, the heuristics for the Grouped GEMM API return sub-optimal algorithms when the C and D matrices use `CUBLASLT_ORDER_ROW` ordering. As a workaround, swap `CUBLASLT_MATMUL_PREF_GROUPED_DESC_D_AVERAGE_ROWS` and `CUBLASLT_MATMUL_PREF_GROUPED_DESC_D_AVERAGE_COLS` in the preferences before calling `cublasLtMatmulAlgoGetHeuristic()`. *[6335555]*
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - Calling `cublasLtMatmulAlgoGetHeuristic()` while a non-default blocking stream is being captured returns `CUBLAS_STATUS_INTERNAL_ERROR`. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
  - GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may return `CUBLAS_STATUS_NOT_SUPPORTED` for certain shapes and workspace configurations. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1).
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared

across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*

- GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*
- GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may produce incorrect results on GPUs with compute capability 12.0 or 12.1 when the matrix is transposed, its non-accumulation dimension is greater than 3145680, and beta is not equal to 0. This issue was introduced in CUDA Toolkit 13.3 Update 1 (cuBLAS 13.6.0). *[CUB-10445]*
- cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*

- **Resolved Issues**
  - Fixed an issue where `cublasXt<t>spmm()` could produce incorrect results with `m` greater than 46340. *[6155165]*
  - Fixed an issue where `cublasLtMatmul()` could run an unsupported combination of data types: an FP32-like compute type with FP32 C and D and non-FP32 A and B, in which case A and B are incorrectly interpreted as FP32 matrices. *[CUB-9942]*
  - Fixed an issue where `cublasLtMatmul()` returned `CUBLAS_STATUS_NOT_SUPPORTED` for FP8 Grouped GEMM problems with scale modes `CUBLASLT_MATMUL_MATRIX_SCALE_VEC128_32F` and `CUBLASLT_MATMUL_MATRIX_SCALE_BLK128x128_32F` on Hopper GPUs. *[CUB-10031]*
  - Fixed an issue where `cublasLtMatmul()` with int8 inputs and scale type `CUDA_R_32I` would allow non-default epilogues on Blackwell GPUs with compute capability 10.x and return incorrect results. The correct behavior is to disallow all but the default epilogue, as documented. *[CUB-10066]*

3.1.5. cuBLAS: Release 13.3

- **New Features**
  - Enabled memory-parsimonious tiling for FP64 emulated matrix multiplications. This improvement ensures that the workspace memory budget no longer exceeds 8 GB.
  - Added support for CUDA Green contexts.
  - Improved FP4 matrix multiplication performance on Blackwell Ultra GPUs by a geometric mean of 5% across a wide range of problems, with up to 7% speedup for some small problems.
  - Improved TF32 matrix multiplication performance on Blackwell and Blackwell Ultra GPUs by a geometric mean of 27% across a wide range of problems and layouts, with up to 3.5x speedup for some small problems.

- Improved TF32 TN matrix multiplication performance on Hopper GPUs by a geometric mean of 11% across a wide range of problems, with up to 40% speedup for some small problems.
- Improved SYMV performance with TMA-based acceleration for Hopper, Blackwell, and Blackwell Ultra kernels with up to 27% geomean speedup.

- **Known Issues**
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - Calling `cublasLtMatmulAlgoGetHeuristic()` while a non-default blocking stream is being captured returns `CUBLAS_STATUS_INTERNAL_ERROR`. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
  - GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may return `CUBLAS_STATUS_NOT_SUPPORTED` for certain shapes and workspace configurations. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1).
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.6. cuBLAS: Release 13.2 Update 2

- **Known Issues**
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after

calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue where `cublasLtMatmul()` ignored the tensor-wide scaling value specified by `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER` for NVFP4 matrix multiplications with NVFP4 output, resulting in incorrect results. This affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6059292]*

3.1.7. cuBLAS: Release 13.2 Update 1

Note: CUDA Toolkit 13.2 Update 1 contains a critical cuBLAS bug for an issue where `cublasLtMatmul()` could ignore tensor-wide scaling for NVFP4 matrix multiplications, resulting in incorrect results. Please see the cuBLAS patch release notes for an available cuBLAS patch (13.4.1) to resolve this issue.

- **New Features**
  - Extended the experimental Grouped GEMM API in cuBLASLt to support NVFP4 inputs and bias epilogues on Blackwell GPUs with Compute Capability 10.x and 11.0. Grouped GEMM NVFP4 support currently uses only MMA tile sizes with K equal to 64.
  - Extended the experimental Grouped GEMM API in cuBLASLt to support BF16, FP16, and FP8 input data types with BF16, FP16, and FP32 output data types on Hopper GPUs. For FP8 inputs, tensorwide scaling and block scaling (`VEC128` and `BLK128x128`) are supported.
  - Improved Grouped GEMM performance on Blackwell GPUs, providing up to 20% higher performance for large problem sizes where the matrices exceed the L2 cache size.
- **Known Issues**
  - `cublasLtMatmul()` ignores the tensor-wide scaling value provided by `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER` for NVFP4 matrix multiplications with NVFP4 output, leading to incorrect results. This affects algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x. This issue was introduced in CUDA Toolkit 13.2 Update 1. *[6059292]*
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a

WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue in `cublasLtMatmulAlgoGetHeuristic()` that could result in no algorithm candidates being returned for Grouped GEMM on Blackwell GPUs. *[CUB-9657]*

3.1.8. cuBLAS: Release 13.2

- **New Features**
  - Extended the experimental Grouped GEMM API in cuBLASLt to support MXFP8 inputs on GPUs with Compute Capability 10.x and 11.0.
  - Added control over special-case handling in FP32 emulation via the environment variable `CUBLAS_EMULATION_SPECIAL_VALUES_SUPPORT_MASK`. Setting `CUBLAS_EMULATION_SPECIAL_VALUES_SUPPORT_MASK=0` can improve performance for applications that do not require preservation of infinity and NaN values, without requiring code changes. For more information, see the `cudaEmulationSpecialValuesSupport_t` documentation.
  - Added FP64 fixed-point emulation support to the `cublas[D|Z]syrk`, `cublas[D|Z]syr2k`, `cublasZherk`, and `cublasZher2k` routines. When the math mode is set to `CUBLAS_FP64_EMULATED_FIXEDPOINT_MATH`, cuBLAS will automatically use FP64 emulation for sufficiently large SYRK and HERK problems. Current support is limited to GPUs with Compute Capability 10.0.
  - Improved performance on RTX PRO 6000 GPUs, delivering up to 20% speedup for FP8, FP16/BF16, TF32, and INT8 precisions.
  - Improved GEMM performance on DGX Spark systems for MXFP8 and NVFP4 data types in large M and N problem sizes, with up to 3× performance improvement for selected matrix shapes.
- **Known Issues**
  - On Blackwell GPUs, FP64 fixed-point emulation kernels may produce incorrect results or experience data corruption when executed concurrently with third-party kernels that allocate tensor memory. *[CUB-9633]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*
- **Resolved Issues**

- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results when it ran concurrently with another kernel that uses Tensor Memory. This issue only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with Compute Capability 10.x and 11.x, and existed since cuBLAS 12.8. *[5807900]*
- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results or invalid memory access errors for large leading dimensions, specifically when the product of the data type size and the leading dimension of a matrix exceeded the bounds of a signed 32 bit integer. This issue affected GPUs with Compute Capability 9.0, 10.x, or 11.0, existed since cuBLAS 12.6 Update 2, and only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66. *[CUB-9572]*
- Fixed an issue in the cuBLASLt Matmul API that could cause FP8 kernels to hang on GPUs with Compute Capability 9.0 when `beta != 0` and `scale_C = 0`. This issue only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66. *[CUB-9627]*
- Fixed an issue in the cuBLASLt Grouped GEMM API that ignored groups with `k = 0`, leading to incorrect results. This issue existed since CUDA 13.1. *[CUB-9529]*
- Fixed an issue in the cuBLASLt Matmul API that could cause incorrect results when C broadcasting was used (`LDC = 0`). *[5845724]*
- Added missing checks for matrix pointer alignment in the `cublasLtMatmul` API. *[CUB-9577] [CUB-9599] [CUB-9585]*
- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results for NVFP4 precision on B300 and GB300 GPUs when the `m` dimension was not a multiple of 64. *[CUB-9577]*
- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results for NVFP4 precision on future GPUs, impacting future hardware compatibility. *[CUB-9570]*
- Fixed an issue in GEMM and Matmul APIs with BF16 and FP16 inputs on DGX Spark and FP8 inputs on GeForce that could potentially cause illegal memory accesses. *[5846563]*
- Fixed an issue in cuBLASLt to enable `CUBLASLT_EPILOGUE_BGRADA` and `CUBLASLT_EPILOGUE_BGRADB` epilogues when the C matrix `CUBLASLT_MATRIX_LAYOUT_ORDER` was set to `CUBLASLT_ORDER_ROW`. *[4617436]*
- Fixed an integer overflow bug in complex, emulated FP64 matrix multiplication. The affected routines include `cublasZgemm`, `cublasZtrsm`, `cublasGemmEx`, and `cublasLtMatmul`. The overflow occurred when `2*m*n + m` exceeded `UINT_MAX`, where `m` is the number of rows of `op(A)` and C, and `n` is the number of columns of `op(B)` and C. *[5720478]*
- Improved GB200 and B200 performance for MXFP8 and NVFP4 precisions when `M` and `N` were less than or equal to 32. *[CUB-9646]*

3.1.9. cuBLAS: Release 13.1 Update 1

- **Known Issues**
  - The cuBLASLt Grouped GEMM API ignores groups with `k = 0`, which can lead to incorrect results. As a workaround, initialize output matrices `D` with `beta*C` for all groups, and then compute Grouped GEMM as `D += A*B` so the result for groups with `k = 0` is computed properly. This issue applies to the experimental cuBLASLt Grouped GEMM API introduced in CUDA 13.1. *[CUB-9529]*

- Complex FP64 GEMM routines using fixed-point emulation can produce incorrect results when matrix dimensions are large enough that `m*n > 2^31` due to integer overflow in an address calculation. *[5720478]*
- `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
- Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
- cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue where fixed point emulation with 7 mantissa bits or less could trigger unspecified launch failures. *[5692684]*
  - Fixed an issue where `cublasLtMatmul` with FP8 arguments and `CUBLASLT_MATMUL_MATRIX_SCALE_SCALAR_32F` scaling mode (default) incorrectly required scaling factor addresses to be 16-byte aligned. This issue existed since cuBLAS 12.9. *[5728938]*

3.1.10. cuBLAS: Release 13.1

- **New Features**
  - Introduced experimental support for grouped GEMM in cuBLASLt. Users can create a matrix with grouped layout using `cublasLtGroupedMatrixLayoutCreate` or `cublasLtGroupedMatrixLayoutInit`, where matrix shapes are passed as device arrays. `cublasLtMatmul` now accepts matrices with grouped layout, in which case matrices are passed as a device array of pointers, where each pointer is a separate matrix that represents a group with its own shapes. Initial support covers A/B types FP8 (E4M3/E5M2), FP16, and BF16, with C/D types FP16, BF16, and FP32; column-major only, default epilogue, 16-byte alignment; requires GPUs with compute capability 10.x or 11.0.

    In addition, the following experimental features were added as part of grouped GEMM:

    - Per-batch tensor-wide scaling for FP8 inputs, enabled by the new `cublasLtMatmulDescAttributes_t` entry `CUBLASLT_MATMUL_MATRIX_SCALE_PER_BATCH_SCALAR_32F`.
    - Per-batch device-side alpha and beta, enabled by the new `cublasLtMatmulDescAttributes_t` entries `CUBLASLT_MATMUL_DESC_ALPHA_BATCH_STRIDE` and `CUBLASLT_MATMUL_DESC_BETA_BATCH_STRIDE`.
  - Improved performance on NVIDIA DGX Spark for CFP32 GEMMs. *[5514146]*
  - Added `sm_121` DriveOS support.

- Improved performance on Blackwell (sm_100 and sm_103) via heuristics tuning for FP32 GEMMs whose shapes satisfy M, N >> K. *[CUB-8572]*
- Improved performance of FP16, FP32, and CFP32 GEMMs on Blackwell Thor.

- **Resolved Issues**
  - Fixed missing memory initialization in cublasCreate() that could result in emulation environment variables being ignored. *[CUB-9302]*
  - Removed unnecessary overhead related to loading kernels on GPUs with compute capability 10.3. *[5547886]*
  - Fixed FP8 matmuls potentially failing to launch on multi-device Blackwell GeForce systems. *[CUB-9487]*
  - Added stricter checks for in-place matmul to prevent invalid use cases (C == D is allowed if and only if Cdesc == Ddesc). As a side effect, users are no longer able to use D as a dummy pointer for C when using CUBLASLT_POINTER_MODE_DEVICE with beta = 0. However, a distinct dummy pointer may still be passed. The stricter checking was added in CUDA Toolkit 13.0 Update 2. *[5471880]*
  - Fixed cublasLtMatmul with INT8 inputs, INT32 accumulation, and INT32 outputs potentially returning CUBLAS_STATUS_NOT_SUPPORTED when dimension N is larger than 65,536 or when batch count is larger than 1. *[5541380]*
  - Added validation for batched matmul to reject invalid configurations where the batch counts differ (Adesc batch count != Bdesc batch count). *[5645772]*
- **Known Issues**
  - The Grouped GEMM cuBLASLt API ignores groups with k = 0, which can lead to incorrect results. As a workaround, initialize each output matrix D with beta * C for all groups before the call, then compute Grouped GEMM as D += A * B so that the result for groups with k = 0 is preserved. This issue applies to the experimental Grouped GEMM cuBLASLt API released in CUDA 13.1. *[CUB-9529]*
  - cublasLtMatmul() may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with CUBLASLT_ALGO_CONFIG_ID equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling cudaGridDependencySynchronize(). This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling cudaTriggerProgrammaticLaunchCompletion(). This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.11. cuBLAS: Release 13.0 Update 2

- **New Features**
  - Enabled opt-in fixed-point emulation for FP64 matmuls (D/ZGEMM) which improves performance and power-efficiency. The implementation follows the Ozaki-1 Scheme and leverages an automatic dynamic precision framework to ensure FP64-level accuracy. See here for more details on fixed-point emulation along with the table of supported compute-capabilities and the CUDA library samples for example usages.
  - Improved performance on NVIDIA DGX Spark for FP16/BF16 and FP8 GEMMs.
  - Added support for BF16x9 FP32 emulation to `cublas[SC]syr[2]k` and `cublasCher[2]k` routines. With the math mode set to `CUBLAS_FP32_EMULATED_BF16X9_MATH`, for large enough problems, cuBLAS will automatically dispatch SYRK and HERK to BF16x9-accelerated algorithms.
- **Resolved Issues**
  - Fixed undefined behavior caused by dereferencing a `nullptr` when passing an uninitialized matrix layout descriptor for `Cdesc` in `cublasLtMatmul`. *[CUB-8911]*
  - Improved performance of `cublas[SCDZ]syr[2]k` and `cublas[CZ]her[2]k` on Hopper GPUs when dimension `N` is large. *[CUB-8293] [5384826]*
- **Known Issues**
  - `cublasLtMatmul` with INT8 inputs, INT32 accumulation, and INT32 outputs might return `CUBLAS_STATUS_NOT_SUPPORTED` when dimension `N` is larger than 65,536 or when the batch count is larger than 1. The issue has existed since CUDA Toolkit 13.0 Update 1 and will be fixed in a later release. *[5541380]*
  - FP8 matmuls may fail to launch on multi-device Blackwell GeForce systems. As a workaround, run a separate process per device. *[CUB-9487]*
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.12. cuBLAS: Release 13.0 Update 1

- **New Features**
  - Improved performance:
    - Block-scaled FP4 GEMMs on NVIDIA Blackwell and Blackwell Ultra GPUs
    - SYMV on NVIDIA Blackwell GPUs *[5171345]*
    - `cublasLtMatmul` for small cases when run concurrently with other CUDA kernels *[5238629]*
    - TF32 GEMMs on Thor GPUs *[5313616]*
    - Programmatic Dependent Launch (PDL) is now supported in some cuBLAS kernels for architectures `sm_90` and above, decreasing kernel launch latencies when executed alongside other PDL kernels.
- **Resolved Issues**
  - Fixed an issue where some `cublasSsyrkx` kernels produced incorrect results when `beta = 0` on NVIDIA Blackwell GPUs. *[CUB-8846]*
  - Resolved issues in `cublasLtMatmul` with INT8 inputs, INT32 accumulation, and INT32 outputs where:
    - `cublasLtMatmul` could have produced incorrect results when A and B matrices used regular ordering (`CUBLASLT_ORDER_COL` or `CUBLASLT_ORDER_ROW`). *[CUB-8874]*
    - `cublasLtMatmul` could have been run with unsupported configurations of `alpha`/`beta`, which must be 0 or 1. *[CUB-8873]*
- **Known Issues**
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*

3.1.13. cuBLAS: Release 13.0

- **New Features**
  - The `cublasGemmEx`, `cublasGemmBatchedEx`, and `cublasGemmStridedBatchedEx` functions now accept `CUBLAS_GEMM_AUTOTUNE` as a valid value for the `algo` parameter. When this option is used, the library benchmarks a selection of available algorithms internally and chooses the optimal one based on the given problem configuration. The selected algorithm is cached within the current `cublasHandle_t`, so subsequent calls with the same problem descriptor will reuse the cached configuration for improved performance.

    This is an experimental feature. Users are encouraged to transition to the cuBLASLt API, which provides fine-grained control over algorithm selection through the heuristics API and includes support for additional data types such as FP8 and block-scaled formats, as well as kernel fusion. (See autotuning example in cuBLASLt).
  - Improved performance of BLAS Level 3 non-GEMM kernels (SYRK, HERK, TRMM, SYMM, HEMM) for FP32 and CF32 precisions on NVIDIA Blackwell GPUs.
  - This release adds support for `sm_110` GPUs for arm64-sbsa on Linux.

- **Resolved Issues**
  - Fixed an issue where some `cublasZtrmm()` kernels produced incorrect results when M is equal to 1 and `side` is `CUBLAS_SIDE_RIGHT` on NVIDIA Ada and Blackwell GeForce-class GPUs. *[5452663]*
- **Known Issues**
  - `cublasLtMatmul` previously ignored user-specified auxiliary (Aux) data types for ReLU epilogues and defaulted to using a bitmask. The correct behavior is now enforced: an error is returned if an invalid Aux data type is specified for ReLU epilogues. *[CUB-7984]*
  - Some `cublasSsyrkx()` kernels produce incorrect results when beta is equal to 0 on Blackwell GPUs. *[CUB-8846]*
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
- **Deprecations**
  - The experimental feature for atomic synchronization along the rows (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_ROWS`) and columns (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_COLS`) of the output matrix which was deprecated in 12.8 has now been **removed**.
  - Starting with this release, cuBLAS will return `CUBLAS_STATUS_NOT_SUPPORTED` if any of the following descriptor attributes are set but the corresponding scale is not supported:
    - `CUBLASLT_MATMUL_DESC_A_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_B_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_D_OUT_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_SCALE_POINTER`
  - Previously, this restriction applied only to non-narrow precision matmuls. It now also applies to narrow precision matmuls when a scale is set for a non-narrow precision tensor.

3.2. cuFFT Library

3.2.1. cuFFT: Release 13.5

- **New Features**
  - Improved performance on Rubin GPUs for power-of-2 FFTs in single and double precision.
- **Known Issues**
  - The cuFFT documentation states that work areas must not overlap to avoid indeterminate results, but does not mention input and output areas. Currently, passing an input or output pointer as the work area, when a work area is needed, can produce silently incorrect results. In a future release, cuFFT will return a new error when a work area is needed and the work-area pointer is equal to the input or output pointer.

3.2.2. cuFFT: Release 13.4

- **New Features**
  - Added new plan property `NVFFT_PLAN_PROPERTY_INT64_BLUESTEIN_ONLY` to enforce Bluestein-only kernels.
  - Added new plan property `NVFFT_PLAN_PROPERTY_INT64_DISABLE_FMA` to disable fused multiply-add contraction when using LTO kernels.
- **Resolved Issues**
  - Added alignment checks that prevent unaligned memory access when using user-provided work areas.
  - Fixed a correctness issue in real-side LTO callback kernels for R2C and C2R transforms. This issue was first identified in CUDA Toolkit 13.3 Update 1.

3.2.3. cuFFT: Release 13.3 Update 1

- **Known Issues**
  - An issue identified in CUDA 13.3 affects the correctness of real-side LTO callback kernels for R2C and C2R transforms. It affects only even sizes above certain large thresholds (8192 or greater in single precision, or 4096 or greater in double precision) whose length has a largest prime factor of at least 127.

3.2.4. cuFFT: Release 13.3

- **New Features**
  - Expanded LTO support to include transform sizes divisible by primes larger than 127, along with increased callback support.
- **Resolved Issues**
  - Fixed an issue where `cufftXtQueryPlan` could result in floating-point exceptions when querying multi-GPU plans that are not single-batch one-dimensional FFTs. *[5923044]*

3.2.5. cuFFT: Release 13.2

- **New Features**
  - Using cuFFT link-time optimized (LTO) kernels now requires NVRTC.
- **Deprecations**
  - `cufftDebug` is deprecated and will be removed in a future release.

3.2.6. cuFFT: Release 13.1

- **New Features**
  - Improved performance for transforms whose sizes are powers of 2, 3, 5, and 7 on Blackwell GPUs, in both single and double precision.
  - Improved performance for selected power-of-two sizes in 2D and 3D transforms, in both single and double precision.
  - Introduced an experimental cuFFT device API that provides host functions to query or generate device function code and exposes database metadata through a C++ header for use with the cuFFTDx library.
- **Resolved Issues**
  - Fixed a correctness issue, identified in CUDA 13.0, that affected a very specific subset of kernels: half- and bfloat16-precision strided R2C and C2R FFTs of size 1.

3.2.7. cuFFT: Release 13.0 Update 1

- **Known Issues**
  - In CUDA 13.0, a correctness issue affects a specific subset of kernels, namely half and bfloat precision size 1 strided R2C and C2R kernels. A fix will be included in a future CUDA release.

3.2.8. cuFFT: Release 13.0

- **New Features**
  - Added new error codes:
    - `CUFFT_MISSING_DEPENDENCY`
    - `CUFFT_NVRTC_FAILURE`
    - `CUFFT_NVJITLINK_FAILURE`
    - `CUFFT_NVSHMEM_FAILURE`
  - Introduced `CUFFT_PLAN_NULL`, a value that can be assigned to a `cufftHandle` to indicate a null handle. It is safe to call `cufftDestroy` on a null handle.
  - Improved performance for single-precision C2C multi-dimensional FFTs and large power-of-2 FFTs.
- **Known Issues**
  - An issue identified in CUDA 13.0 affects the correctness of a specific subset of cuFFT kernels, specifically half-precision and bfloat16 size-1 strided R2C and C2R transforms. A fix will be included in a future CUDA release.
- **Deprecations**
  - Removed support for Maxwell, Pascal, and Volta GPUs, corresponding to compute capabilities earlier than Turing.
  - Removed legacy cuFFT error codes:

- `CUFFT_INCOMPLETE_PARAMETER_LIST`
- `CUFFT_PARSE_ERROR`
- `CUFFT_LICENSE_ERROR`

- Removed the `libcufft_static_nocallback.a` static library. Users should link against `libcufft_static.a` instead, as both are functionally equivalent.

3.3. cuSOLVER Library

3.3.1. cuSOLVER: Release 13.5

- **New Features**
  - Added `cusolverDnXsygvStridedBatched` and `cusolverDnXsygvStridedBatched_bufferSize`, which solve a batch of generalized symmetric-definite eigenproblems A x = lambda B x located by a base pointer and a stride rather than an array of pointers. This version supports only `itype = CUSOLVER_EIG_TYPE_1`. The device workspace requirement is not monotonic in n, so query the buffer size for each problem size you intend to run.
  - `cusolverDnXgeev` now offers an optional mixed-precision eigensolver, selected with `cusolverDnSetAdvOptions` using `CUSOLVERDN_GEEV` and `CUSOLVER_ALG_1`. It computes the Schur decomposition in reduced precision and recovers double precision through iterative refinement. It is available for `CUDA_C_64F` with `jobvl = CUSOLVER_EIG_MODE_NOVECTOR` and `jobvr = CUSOLVER_EIG_MODE_VECTOR`. It is intended for large, well-separated spectra on GPUs with limited double-precision throughput; it is slower for clustered spectra or small matrix sizes and requires substantially more device workspace.
  - The singular-vector update in `cusolverDnXgesvd` is now user-selectable through `cusolverDnSetAdvOptions` with `CUSOLVERDN_GESVD`. `CUSOLVER_ALG_0` is the hybrid blocked update introduced in CUDA Toolkit 13.4 Update 1 and remains the default; `CUSOLVER_ALG_1` is the classical update used in earlier releases.
- **Known Issues**
  - `cusolverDnXsyevBatched` fails without computing when `batchSize` exceeds 65,535 and n is at least 128 (256 for `CUDA_R_32F`), even though the documented `n * lda * batchSize <= INT32_MAX` limit holds. As a workaround, split the batch into chunks of at most 65,535. *[6657796]*
  - The cuSOLVERDn logging control routines, including `cusolverDnLoggerSetCallback` and `cusolverDnLoggerSetLevel`, are declared in the header and documented but are not exported by the shared library, so they cannot be linked or resolved. *[6738440]*
- **Resolved Issues**
  - Fixed a performance regression in `cusolverDnSgesvdjBatched` on Blackwell GPUs. *[6557683]*

3.3.2. cuSOLVER: Release 13.4 Update 1

- **New Features**
  - `cusolverDnXgesvd` now computes singular vectors with a hybrid blocked update by default, in which the Givens rotations of the bidiagonal SVD are computed on the host and applied on the device. The previous classical LASR update remains available as `CUSOLVER_ALG_1` via `cusolverDnSetAdvOptions`. Best performance of the default path is attained with pinned host memory.
  - A two-stage symmetric eigenvalue decomposition is now available for `cusolverDnXsyevd`, reducing the dense symmetric matrix to tridiagonal form through an intermediate banded form (`sytrd_sy2sb` + `sytrd_sb2st`). The default algorithm (`CUSOLVER_ALG_0`) now selects automatically between the one-stage and two-stage paths based on the problem size, architecture, and math mode; `CUSOLVER_ALG_2` forces the two-stage path and `CUSOLVER_ALG_1` forces the one-stage path.
- **Resolved Issues**
  - Fixed a hang when cuSOLVERDn routines were used inside a CUDA Green Context. The handle cached the physical device SM count at `cusolverDnCreate` time, so persistent kernels launched more CTAs than the green context could schedule and the spin-wait barriers never completed. *[4256378]*
  - Fixed an integer overflow in the internal pivot handling of `cusolverDnXgetrf` and `cusolverDnXgetrf64` when the column offset times the leading dimension exceeded the `INT32_MAX` range, which could cause incorrect results or invalid memory accesses for matrices with large leading dimensions. *[5411488]*

3.3.3. cuSOLVER: Release 13.4

- **New Features**
  - Further reduced device workspace for `cusolverDnXgesvd` when singular vectors are requested and the input matrix is tall-and-skinny (`m > n`).
  - Reduced device workspace for `cusolverDnXgeev` by optimizing the Householder reflector accumulation (ORGHR) stage, enabling computation of larger eigenvalue problems within available GPU memory.
  - Added `cusolverDnXsytrf` and `cusolverDnXsytrf_bufferSize`, which provide a 64-bit generic API for computing LDL^T and UDU^T factorizations in symmetric indefinite linear systems.
  - Added `cusolverDnXhetrf` and `cusolverDnXhetrf_bufferSize`, which provide a 64-bit generic API for computing LDL^H and UDU^H factorizations in Hermitian indefinite linear systems, with support for the `CUDA_C_32F` and `CUDA_C_64F` data types.
  - Added `cusolverDnXhetrs` and `cusolverDnXhetrs_bufferSize`, which provide a 64-bit generic API for solving linear systems of the form AX = B using the factorization obtained via `cusolverDnXhetrf`, supporting the `CUDA_C_32F` and `CUDA_C_64F` data types.
- **Resolved Issues**
  - Fixed an issue where `cusolverDnXgetrf` silently discarded a NaN value in the input matrix instead of propagating it to the factored output. *[5849138]*

3.3.4. cuSOLVER: Release 13.3 Update 1

- **New Features**
  - Improved `cusolverDnXgetrf` performance with pivoting for `sm_90`, `sm_100`, `sm_103`, and `sm_120`.
- **Resolved Issues**
  - Fixed an issue where `cusolverDn{C/Z}sytrf()` and `cusolverDn{C/Z}sytrs()` with `devIpiv == nullptr` could treat the complex symmetric input as Hermitian instead of symmetric, which could lead to incorrect results for complex symmetric problems. The documented symmetric factorization and solve behaviors have been restored.
  - Fixed accuracy issues on ill-conditioned and rank-deficient matrices for `cusolverDnXgesvdp` and `cusolverDnXpolar`.

3.3.5. cuSOLVER: Release 13.3

- **New Features**
  - Improved `cusolverDnXgeev` performance when computing eigenvectors by moving eigenvector post-processing from the host to the device.
- **Known Issues**
  - The `cusolverDn{C,Z}sytrf` and `cusolverDnXsytrs` APIs assume that the complex input matrix `A` is Hermitian instead of symmetric when `devIpiv` is set to `NULL`. This issue exists starting with CUDA Toolkit 13.1. *[5797471]*

3.3.6. cuSOLVER: Release 13.2 Update 1

- **New Features**
  - Improved performance of `cusolverDnXgeqrf()` and `cusolverDn<S,D,C,Z>geqrf()` on `sm_90`, `sm_100`, `sm_103`, and `sm_120` for matrices with `m <= 65536`.
  - Added the new public 64-bit interface `cusolverDnXpolar()`, which exposes the QDWH algorithm implementation for polar decomposition in cuSOLVERDn.
  - Added the new public 64-bit interface `cusolverDnXstedc()`, which computes the eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer method.

3.3.7. cuSOLVER: Release 13.2

- **New Features**
  - Added FP64 fixed-point emulation support to cuSOLVERDn. The following new APIs are available:
  - `cusolverDnSetFixedPointEmulationMantissaControl()`
  - `cusolverDnGetFixedPointEmulationMantissaControl()`
  - `cusolverDnSetFixedPointEmulationMaxMantissaBitCount()`
  - `cusolverDnGetFixedPointEmulationMaxMantissaBitCount()`
  - `cusolverDnSetFixedPointEmulationMantissaBitOffset()`
  - `cusolverDnGetFixedPointEmulationMantissaBitOffset()`
  - `cusolverDnSetEmulationSpecialValuesSupport()`
  - `cusolverDnGetEmulationSpecialValuesSupport()`
  - Added the `cusolverDnXsygvd` API to support larger problem sizes.
- **Known Issues**
  - Starting with CUDA Toolkit 13.1, `cusolverDn{C,Z}sytrf` and `cusolverDnXsytrs` assume the complex input matrix `A` is Hermitian (instead of symmetric) when `devIpiv == NULL`. *[5797471]*

3.3.8. cuSOLVER: Release 13.1

- **Resolved Issues**
  - Fixed a bug that prevented users from changing the algorithm for `cusolverDnXsyevBatched` by using `cusolverDnSetAdvOptions`. *[5539844]*

3.3.9. cuSOLVER: Release 13.0 Update 1

- **Resolved Issues**
  - Fixed a race condition in `cusolverDnXgeev` that could occur when using multiple host threads with either separate handles per thread or a shared handle, which caused execution to abort and returned `CUSOLVER_STATUS_INTERNAL_ERROR`.

3.3.10. cuSOLVER: Release 13.0

- **New Features**
  - cuSOLVER offers a new math mode to leverage improved performance of emulated FP32 arithmetic on NVIDIA Blackwell GPUs.

    To enable and control this feature, the following new APIs have been added:
    - `cusolverDnSetMathMode()`
    - `cusolverDnGetMathMode()`
    - `cusolverDnSetEmulationStrategy()`
    - `cusolverDnGetEmulationStrategy()`
  - Performance improvements for `cusolverDnXsyevBatched()` have been made by introducing an internal algorithm switch on Blackwell GPUs for matrices of size `n <= 32`.

    To revert to the previous algorithm for all problem sizes, use cusolverDnSetAdvOptions().

    For more details, refer to the cusolverDnXsyevBatched() documentation.
- **Deprecations**
  - `cuSOLVERMg` is deprecated and may be removed in an upcoming major release. Users are encouraged to use cuSOLVERMp for multi-GPU functionality across both single and multi-node environments. To disable the deprecation warning, add the compiler flag `-DDISABLE_CUSOLVERMG_DEPRECATED`.
  - `cuSOLVERSp` and `cuSOLVERRf` are fully deprecated and may be removed in an upcoming major release. Users are encouraged to use the cuDSS library for better performance and ongoing support.

    For help with the transition, refer to the cuDSS samples or CUDA samples for migrating from `cuSOLVERSp` to `cuDSS`.

    To disable the deprecation warning, add the compiler flag: `-DDISABLE_CUSOLVER_DEPRECATED`.
- **Resolved Issues**
  - The supported input matrix size for `cusolverDnXsyevd`, `cusolverDnXsyevdx`, `cusolverDnXsyevBatched`, `cusolverDn<t>syevd`, and `cusolverDn<t>syevdx` is no longer limited to `n <= 32768`.

    This update also applies to routines that share the same internal implementation: `cusolverDnXgesvdr`, `cusolverDnXgesvdp`, `cusolverDn<t>sygvd`, `cusolverDn<t>sygvdx`, and `cusolverDn<t>gesvdaStridedBatched`.

3.4. cuSPARSE Library

3.4.1. cuSPARSE: Release 13.5

- **New Features**
  - Added support for mixed-index CSR and SELL matrices (64-bit offsets, 32-bit column indices) in `SpMV`. *[CUSPARSE-3437] [CUSPARSE-3501]*
  - Added support for mixed-index CSR matrices (64-bit offsets, 32-bit column indices) in `SpMM`. *[CUSPARSE-3436]*
  - Added support for mixed-index SELL matrices (64-bit offsets, 32-bit column indices) in `SpSV`. *[CUSPARSE-3502]*
  - Added support for mixed-index CSR and CSC matrices (64-bit offsets, 32-bit column indices) in the sparse-to-dense and dense-to-sparse APIs. *[CUSPARSE-3574]*
  - Added mixed-precision support (`FP16` or `BF16` matrix, `FP32` input and output vectors, `FP32` compute type) for CSR `cusparseSpMV`. *[CUSPARSE-3686]*
  - Added the experimental `cusparseSpMatConvert` API for matrix format conversion. It currently supports conversion between CSR and CSC. *[CUSPARSE-3595] [CUSPARSE-3573]*
  - Added the experimental `cusparseSpMVGeneric` API, which supports sparse matrix-vector multiplication (SpMV) on symmetric matrices using half storage (storing only the upper or lower half of the entries). *[CUSPARSE-3543]*
  - Improved `SpSV` analysis performance by 2.79x (geometric mean). *[CUSPARSE-3528]*
  - Reduced `SpSV` memory usage. *[CUSPARSE-3590]*
  - Added a new `SpSV` algorithm (`ALG2`) with 2.7x faster solve performance than `ALG1` (geometric mean). *[CUSPARSE-3576] [CUSPARSE-3583] [CUSPARSE-3746]*
  - Improved `csrcolor` performance by approximately 3x (geometric mean). *[CUSPARSE-3615]*
  - Improved `FP32` and `FP64` CSR `SpMM` (`ALG3`) performance by 1.65x to 2.27x (geometric mean) across data center GPUs. *[CUSPARSE-3593]*
- **Resolved Issues**
  - Fixed an out-of-bounds issue in `SpGEMM ALG3`.
  - Fixed an index overflow issue in `SpGEMMreuse`.
  - Fixed a 32-bit overflow in BSR `SDDMM` for value buffers larger than 4 GB.

3.4.2. cuSPARSE: Release 13.4

- **Known Issues**
  - Mixed-precision CSR/COO `SpMM` is not supported in some cases. Refer to the cuSPARSE documentation for details.
- **Resolved Issues**
  - Fixed a bug with cached parameters in `SpMVOp`.

3.4.3. cuSPARSE: Release 13.3 Update 1

- **New Features**
  - Improved SpGEAM performance by an average of 40%. *[CUSPARSE-3361]*
  - Reduced preprocessing time for SpMM ALG3.
- **Known Issues**
  - Incorrect result when running BSR SDDMM on a very large matrix.
- **Resolved Issues**
  - Fixed a rarely occurring issue in CSC and transposed CSR SpMV. *[5975307]*
  - Fixed an accuracy issue in mixed-precision SELL SpMV.
  - Fixed an issue with the algorithm configuration cache in SpMVOp.

3.4.4. cuSPARSE: Release 13.3

- **New Features**
  - Added support for the CSC format in SpSV and SpSM.
  - Improved CSR SpMV ALG2 performance by an average of 11%.
  - Added the Generic API SpGEAM for sparse matrix-matrix addition.
  - Added SpMVOp ALG1 with reduced preprocessing overhead.
  - Added support for mixed index types in SpMVOp computation for CSR matrices with 64-bit offsets and 32-bit indices.
  - Added support for the FP32 data type in SpMVOp.
  - Avoided recompilation for the same epilogue in SpMVOp.
  - Added mixed-precision support in SpMV for 32-bit input matrices and 64-bit input vectors.
  - Added support for updating matrix values after preprocessing in SpMVOp ALG1.
- **Resolved Issues**
  - Fixed a memory leak in SpMVOp when `destroy_lrb()` was called. *[5974043]*

3.4.5. cuSPARSE: Release 13.2 Update 1

- **New Features**
  - Improved `cusparseSpMVOp_createDescr()` performance by up to 2.5x.
  - Reduced `cusparseSpMVOp_createPlan()` planning latency for default epilogues through ahead-of-time compilation, avoiding JIT compilation in this case.
- **Resolved Issues**
  - Fixed an issue that caused performance regressions in BSR SpMM for certain block sizes. *[5860241]*

- **Deprecation**
  - Deprecated the `SpMMOp` and `SpGEMMreuse` APIs.

3.4.6. cuSPARSE: Release 13.2

- **New Features**
  - Improved the runtime of the `SpMVOp::buffer_size_estimate` API.

3.4.7. cuSPARSE: Release 13.1 Update 1

- **New Features**
  - Added a new `cusparseSpMVOp_bufferSize` API that returns the size of the workspace buffer required for SpMVOp computations. Users provide this buffer when creating `cusparseSpMVOpDescr_t`, removing internal memory allocations.
  - Improved SpMVOp performance on B200. *[CUSPARSE-2931] [CUSPARSE-2932] [CUSPARSE-2933]*
- **Resolved Issues**
  - Fixed an accuracy issue in mixed-precision CSR/COO SpMM computations. *[CUSPARSE-2349]*
  - Fixed an issue in CSR SpMM computations when the input dense matrix has a high number of columns. *[CUSPARSE-2301]*

3.4.8. cuSPARSE: Release 13.1

- **New Features**
  - Introduced an experimental Sparse Matrix-Vector Multiplication (SpMVOp) API that provides improved performance compared with the existing generic CsrMV API. This API supports CSR format with 32-bit indices, double precision, and user-defined epilogues.
  - The nvJitLink shared library is now loaded dynamically at runtime.
  - Improved `cusparseXcsrsort` with reduced memory usage and higher performance. *[CUSPARSE-2630]*
- **Known Issues**
  - When using 32-bit indexing, `cusparseSpSV` and `cusparseSpSM` may crash if the number of nonzero elements (nnz) approaches `2^31 - 1`. *[CUSPARSE-2211]*
- **Resolved Issues**
  - Fixed potential issues when input and output pointers are not 16-byte aligned in `cusparseCsr2cscEx2`, `cusparseSparseToDense`, and CSR/COO `cusparseSpMM`. *[CUSPARSE-2380]*
  - Fixed a determinism issue in CSR `cusparseSpMM` ALG3. *[CUSPARSE-2612]*

- All routines now support matrices with up to 2^31 - 1 nonzero elements (nnz) when using 32-bit indexing, with the exception of `cusparseSpSV` and `cusparseSpSM`. *[CUSPARSE-2153]*
- Fixed a potential race condition that could occur when dynamically loading driver APIs. *[CUSPARSE-2764]*

3.4.9. cuSPARSE: Release 13.0 Update 1

- **New Features**
  - Added support for the BSR format in the generic SpMV API *(CUSPARSE-2518)*.
- **Deprecation**
  - Deprecated the legacy BSR SpMV API (replaced by the generic SpMV API).
- **Resolved Issues**
  - Enabled all generic APIs to support zero-dimension matrices/vectors (m, n, k = 0) *(CUSPARSE-2378)*.
  - Enabled all generic APIs to support small-dimension matrices/vectors (small m, n, or k) *(CUSPARSE-2379)*.
  - Fixed incorrect results in mixed-precision CSR/COO SpMV computations *(CUSPARSE-2349)*.

3.4.10. cuSPARSE: Release 13.0

- **New Features**
  - Added support for 64-bit index matrices in SpGEMM computation. *(CUSPARSE-2365)*
- **Known Issues**
  - cuSPARSE logging APIs can crash on Windows.
  - `CUSPARSE_SPMM_CSR_ALG3` does not return deterministic results as stated in the documentation.
- **Deprecation**
  - Dropped support for pre-Turing architectures (Maxwell, Volta, and Pascal).
- **Resolved Issues**
  - Fixed a bug in `cusparseSparseToDense_bufferSize` that caused it to request up to 16× more memory than required. *[CUSPARSE-2352]*
  - Fixed unwanted 16-byte alignment requirements on the external buffer. Most routines will now work with any alignment. In the generic API, only `cusparseSpGEMM` routines are still affected. *[CUSPARSE-2352]*
  - Fixed incorrect results from `cusparseCsr2cscEx2` when any of the input matrix dimensions are zero, such as when `m = 0` or `n = 0`. *[CUSPARSE-2319]*
  - Fixed incorrect results from CSR SpMV when any of the input matrix dimensions are zero, such as when `m = 0` or `n = 0`. *[CUSPARSE-1800]*

3.5. Math Library

3.5.1. CUDA Math: Release 13.5

- **New Features**
  - Added the fast-math intrinsics `__sin2pif`, `__cos2pif`, and `__sincos2pif`. *[CTK-11689]*
- **Known Issues**
  - The double-precision Bessel functions `jn(n, x)` and `yn(n, x)` can have correctness issues for arguments around zero for orders `n >= 2`. *[6830848]*
- **Resolved Issues**
  - Updated the CUDA Math documentation for integer SIMD functions to better explain their current behavior. *[6632035]*
  - Fixed the host-side and older-GPU fallback implementations of the integer SIMD functions `__viaddmin_s32_relu`, `__viaddmax_s32_relu`, `__viaddmin_s32`, and `__viaddmax_s32` to prevent undefined behavior due to signed integer overflow.

3.5.2. CUDA Math: Release 13.4

- **New Features**
  - Added new FP8 scaling factor type `__nv_fp8_ue5m3` and corresponding conversion operations in `cuda_fp8.h`. *[5482730]*
- **Resolved Issues**
  - Fixed `fp128` nearest integral functions returning off-by-one results in pathological cases. This fix also delivers performance improvements of 2x–8x for `fp128` operations on GB200 and up to approximately 200x on RTX 5080, covering conversion, nearest integral, `frexp`, `ilogb`, `modf`, and multiplication. Fixed an underflow in `fp128` `hypot`.
  - Fixed `__hmin_nan` and `__hmax_nan` bfloat16 functions ignoring the sign of zero in emulation code paths for devices with compute capability below 8.0 and host CPUs; `-0.0` now correctly compares as less than `+0.0`. *[6182113]*
  - Fixed `__nv_fp8_ue5m3` conversions from long integer types that could produce a result off by one. *[6182039]*

3.5.3. CUDA Math: Release 13.3

- **Resolved Issues**
  - Fixed an issue where silent data corruption could occur when the CUDA Math API `__mul24()` intrinsic was called with compile-time constant inputs due to undefined behavior from compiler optimizations applied to overflowing signed integer multiplication. This issue was introduced in CUDA Toolkit 11.1 and resolved in CUDA Toolkit 13.3. *[5807344]*

3.5.4. CUDA Math: Release 13.2 Update 1

- **Known Issues**
  - Silent data corruption can occur when the CUDA Math API `__mul24()` intrinsic is called with compile-time constant inputs. Compiler optimizations applied to overflowing signed integer multiplication can expose the program to undefined behavior. This issue was introduced in CUDA Toolkit 11.1 and will be fixed in a future release. *[5807344]*

3.5.5. CUDA Math: Release 13.2

- **New Features**
  - Accuracy and performance improvements were made to the following libdevice single-precision math functions:
    - `expm1f()`: up to 20% faster, with minor accuracy improvements.
    - `erff()`: 5% to 10% faster, with minor accuracy improvements.

  These gains come from algorithmic simplifications, reduced branching, and tighter approximations. *[5480287]*
- **Resolved Issues**
  - ACLE extension support for GCC: ARM64 users, notice the behavior change for `__clz` and `__clzll` CUDA Math integer intrinsics: the signatures were updated to match the host compiler's declarations, relevant for GCC 11.4 onwards. The (un-)signedness of the return type of the intrinsic affects the caller code relying on integer types promotions. *[6258270]*

3.5.6. CUDA Math: Release 13.0

- **New Features**
  - Single and double precision math functions received targeted performance and accuracy improvements through algorithmic simplifications, reduced branching, and tighter approximations.
    - `atan2f`, `atan2`: Up to 10% faster with minor improvements in accuracy.
    - `sinhf`, `coshf`, `acoshf`, `asinhf`, `asinh`: Up to 50% speedups with minor improvements in accuracy.
    - `cbrtf`, `rcbrtf`: 15% faster with minor improvements in accuracy.
    - `erfinvf`, `erfcinvf`, `normcdfinvf`: Minor accuracy improvements, performance neutral.
    - `ldexpf`, `ldexp`: Up to 3x faster in single precision and 30% faster in double precision, with no accuracy loss.
    - `modff`, `modf`: Up to 50% faster in single precision and 10% faster in double precision, with no accuracy loss.

3.6. nvJPEG Library

3.6.1. nvJPEG: Release 13.5

- **New Features**
  - Improved decode throughput for the `NVJPEG_BACKEND_GPU_HYBRID` backend by 10% to 600%, depending on the image size and whether the standard Huffman table is used.
  - `NVJPEG_BACKEND_GPU_HYBRID` is now considered during automatic backend selection when using the `nvjpegDecode` API. Previously, only `NVJPEG_BACKEND_HYBRID` and `NVJPEG_BACKEND_HARDWARE` were considered.
- **Resolved Issues**
  - Fixed parsing issues that could cause crashes or hangs when decoding malformed JPEG bitstreams.
  - Fixed various security issues.

3.6.2. nvJPEG: Release 13.4 Update 1

- **Resolved Issues**
  - Fixed parsing issues that could cause crashes or hangs when decoding malformed JPEG bitstreams.

3.6.3. nvJPEG: Release 13.3 Update 1

- **Resolved Issues**
  - Fixed an issue that would cause `nvjpegCreate*` calls to error out on Orin. *[6176492]*

3.6.4. nvJPEG: Release 13.3

- **New Features**
  - Added support for region-of-interest decoding with `nvjpegDecodeBatchedEx` when using the `NVJPEG_BACKEND_LOSSLESS_JPEG` backend.
- **Resolved Issues**
  - Fixed an issue with boundary handling when decoding a region of interest with `NVJPEG_FLAGS_UPSAMPLING_WITH_INTERPOLATION` enabled.

3.6.5. nvJPEG: Release 13.2 Update 2

- **Known Issues**
  - `nvjpegCreate*` calls may error out on Orin due to compatibility issues with the `libnvcuvid.so` driver library. This issue is fixed in CUDA Toolkit 13.3 Update 1. *[6176492]*

3.6.6. nvJPEG: Release 13.2 Update 1

- **New Features**
  - Added the `NVJPEG_OUTPUT_UNCHANGEDI` enum value to `nvjpegOutputFormat_t` for unchanged interleaved output. For chroma subsampling formats other than 4:4:4, chroma values are duplicated so that the chroma and luma dimensions match.

3.6.7. nvJPEG: Release 13.1 Update 1

- **Resolved Issues**
  - Reduced nvJPEG encoder initialization time on Thor. *[5533951]*

3.6.8. nvJPEG: Release 13.1

- **Resolved Issues**
  - nvJPEG's lossless JPEG 92 (lj92) implementation can now correctly handle lj92 files that contain a comment marker in the header. *[5484797]*

3.6.9. nvJPEG: Release 13.0 Update 1

- **Resolved Issues**
  - Fixed a race condition in certain cases during progressive encoding *[5307748]*.
  - Fixed an uninitialized read when encoding images as 4:1:0 JPEG bitstreams *[5308008]*.

3.6.10. nvJPEG: Release 13.0

- **Deprecations**
  - Removed the `nvjpegEncoderParamsCopyHuffmanTables` API.
- **Resolved Issues**
  - nvJPEG is now more robust and no longer crashes or exhibits undefined behavior when decoding malformed or truncated bitstreams. *[5168024, 5133845, 5143450]*

- `nvjpegEncodeYUV` now avoids reading outside of allocated device memory in certain cases. *[5133826]*
- Optimized memory usage when encoding RGB inputs using the hardware encoder.
- Fixed issues related to rounding in various transform, sampling, and conversion steps, improving image quality for both encoder and decoder. *[5064901, 3976092]*
- Various bug fixes for improved security.

3.7. NPP Library

3.7.1. NPP: Release 13.4

- **Resolved Issues**
  - Fixed a regression in `nppiNV12ToRGB_8u_ColorTwist32f_P2C3R_Ctx` introduced in CUDA 12.9 that could render black NV12 frames as blue. *[6229084]*

3.7.2. NPP: Release 13.1

- **Resolved Issues**
  - Fixed an issue in `nppiCFAToRGB_8u_C1C3R()` affecting SSIM validation for `NPPI_BAYER_GBRG` patterns. *[5192648]*

3.7.3. NPP: Release 13.0

- **Deprecations**
  - **Removal of Legacy Non-Context APIs**

    All legacy NPP APIs without the `_Ctx` suffix have been deprecated and are now removed starting with this release. Developers should transition to the context-aware (`_Ctx`) versions to ensure continued support and compatibility with the latest CUDA releases.
  - **Deprecation of ``nppGetStreamContext()``**

    The `nppGetStreamContext()` API has been deprecated and removed. Developers are strongly encouraged to adopt application-managed stream contexts by explicitly managing the `NppStreamContext` structure. For guidance, refer to the NPP Documentation – General Conventions.
- **Resolved Issues**
  - Fixed an issue in `nppiFloodFillRange_8u_C1IR_Ctx` where the flood fill operation did not correctly fill the full target area. *[5141474]*
  - Resolved a bug in the `nppiDebayer()` API that affected proper reconstruction of color data during Bayer pattern conversion. *[5138782]*

Chapter 4. Notices

4.1. 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.

4.2. OpenCL

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

4.3. 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

©2007-2026, NVIDIA Corporation & affiliates. All rights reserved