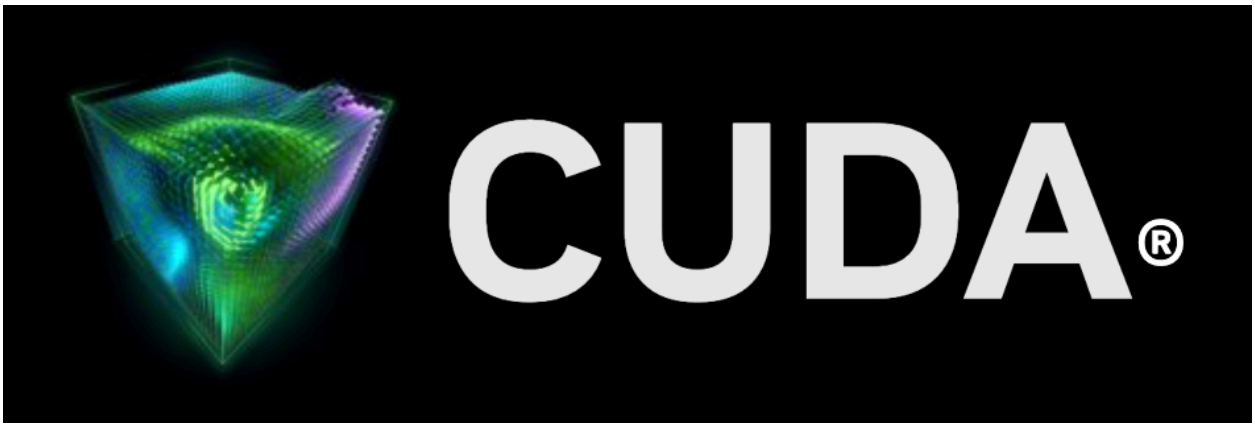

Release Notes

Release 13.3

NVIDIA Corporation

May 27, 2026

Contents

Chapter 1. Overview

CUDA Toolkit 13.3 - Release Notes

Welcome to the release notes for NVIDIA® CUDA® Toolkit 13.3. This release includes enhancements and fixes across the CUDA Toolkit and its libraries.

This documentation is organized into two main sections:

- **General CUDA**

  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. General CUDA

2.1. CUDA Toolkit Major Components

Note: Starting with CUDA 11, individual components within the CUDA Toolkit (for example: compiler, libraries, tools) are versioned independently.

For CUDA 13.3, the table below indicates the versions:

Table 1: CUDA 13.3 Component Versions

Component Name		Version Information	Supported Architectures	Supported Platforms
CUDA C++ Core Compute Libraries	Thrust	3.3.3	x86_64, arm64-sbsa	Linux, Windows
	CUB	3.3.3		
	libcu++	3.3.3		
	Cooperative Groups	13.3.3.3.1		
CUDA Application Compiler (crt)		13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA Compilation Optimizer (ctadvisor)		13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA Runtime (cudart)		13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA culibos		13.3.33	x86_64, arm64-sbsa	Linux
CUDA cuobjdump		13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUPTI		13.3.35	x86_64, arm64-sbsa	Linux, Windows
CUDA cuxxfilt (demangler)		13.3.29	x86_64, arm64-sbsa	Linux, Windows

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA Documentation	13.3.40	x86_64, arm64-sbsa	Linux, Windows
CUDA GDB	13.3.27	x86_64, arm64-sbsa	Linux
CUDA NVCC	13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA nvdisasm	13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA NVML Headers	13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA nvprune	13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA NVRTC	13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA NVTX	13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA OpenCL	13.3.27	x86_64	Linux, Windows
CUDA Profiler API	13.3.27	x86_64, arm64-sbsa	Linux, Windows
CUDA Sandbox dev	13.3.29	x86_64, arm64-sbsa	Linux
CUDA Compute Sanitizer API	13.3.27	x86_64, arm64-sbsa	Linux, Windows
CUDA TILE-IR AS	13.3.36	x86_64, arm64-sbsa	Linux, Windows
CUDA cuBLAS	13.5.1.27	x86_64, arm64-sbsa	Linux, Windows
CUDA cuDLA	13.3.29	x86_64, arm64-sbsa	Linux
CUDA cuFFT	12.3.0.29	x86_64, arm64-sbsa	Linux, Windows
CUDA cuFile	1.18.0.66	x86_64, arm64-sbsa	Linux
CUDA cuobjclient	1.2.0.59	x86_64, arm64-sbsa	Linux
CUDA cuRAND	10.4.3.29	x86_64, arm64-sbsa	Linux, Windows

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA cuSOLVER	12.2.2.18	x86_64, arm64-sbsa	Linux, Windows
CUDA cuSPARSE	12.8.1.7	x86_64, arm64-sbsa	Linux, Windows
CUDA NPP	13.1.2.48	x86_64, arm64-sbsa	Linux, Windows
CUDA nvFatbin	13.3.29	x86_64, arm64-sbsa	Linux, Windows
CUDA nvJitLink	13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA nvJPEG	13.2.0.21	x86_64, arm64-sbsa	Linux, Windows
CUDA nvptxcompiler	13.3.33	x86_64, arm64-sbsa	Linux, Windows
CUDA nvvm	13.3.33	x86_64, arm64-sbsa	Linux, Windows
Nsight Compute	2026.2.0.7	x86_64, arm64-sbsa	Linux, Windows
Nsight Systems	2026.1.3.243	x86_64, arm64-sbsa	Linux, Windows
Nsight Visual Studio Edition (VSE)	2026.2.0.26084	x86_64 (Windows)	Windows
nvidia_fs[1]	2.29.4	x86_64, arm64-sbsa	Linux
nvlsm	2025.10.12	x86_64, arm64-sbsa	Linux
Visual Studio Integration	13.3.27	x86_64 (Windows)	Windows
NVIDIA Linux Driver	610.43.02	x86_64, arm64-sbsa	Linux

[1] Only available on select Linux distros

2.2. CUDA Driver

Running a CUDA application requires the system with at least one CUDA capable GPU and a driver that is compatible with the CUDA Toolkit. See Table 3. For more information various GPU products that are CUDA capable, visit https://developer.nvidia.com/cuda-gpus.

Each release of the CUDA Toolkit requires a minimum version of the CUDA driver. The CUDA driver is backward compatible, meaning that applications compiled against a particular version of the CUDA will continue to work on subsequent (later) driver releases.

More information on compatibility can be found at https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#cuda-compatibility-and-upgrades.

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

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

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

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

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

** Starting with CUDA 13.1, the Windows display driver is **no longer** bundled with the CUDA Toolkit package. Users must download and install the appropriate NVIDIA driver separately from the official driver download page.

For more information on supported driver versions, see the CUDA Compatibility Guide for drivers.

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

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

Table 3: CUDA Toolkit and Corresponding Driver Versions

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 13.3 GA	>=610.43.02	N/A

continues on next page

Table 3 – continued from previous page

CUDA Toolkit	Toolkit Driver Version	
CUDA 13.2 Update 1	>=595.58.03	N/A
CUDA 13.2 GA	>=595.45.04	N/A
CUDA 13.1 Update 1	>=590.48.01	N/A
CUDA 13.1 GA	>=590.44.01	N/A
CUDA 13.0 Update 2	>=580.95.05	N/A
CUDA 13.0 Update 1	>=580.82.07	N/A
CUDA 13.0 GA	>=580.65.06	N/A
CUDA 12.9 Update 1	>=575.57.08	>=576.57
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

continues on next page

Table 3 – continued from previous page

CUDA Toolkit	Toolkit Driver Version	
CUDA 11.6 Update 1	>=510.47.03	>=511.65
CUDA 11.6 GA	>=510.39.01	>=511.23
CUDA 11.5 Update 2	>=495.29.05	>=496.13
CUDA 11.5 Update 1	>=495.29.05	>=496.13
CUDA 11.5 GA	>=495.29.05	>=496.04
CUDA 11.4 Update 4	>=470.82.01	>=472.50
CUDA 11.4 Update 3	>=470.82.01	>=472.50
CUDA 11.4 Update 2	>=470.57.02	>=471.41
CUDA 11.4 Update 1	>=470.57.02	>=471.41
CUDA 11.4.0 GA	>=470.42.01	>=471.11
CUDA 11.3.1 Update 1	>=465.19.01	>=465.89
CUDA 11.3.0 GA	>=465.19.01	>=465.89
CUDA 11.2.2 Update 2	>=460.32.03	>=461.33
CUDA 11.2.1 Update 1	>=460.32.03	>=461.09
CUDA 11.2.0 GA	>=460.27.03	>=460.82
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

- CUDA Toolkit driver bundling (pre-CUDA 13.1):

- The CUDA Toolkit previously included an NVIDIA display driver for convenience.
- This bundled driver was intended only for development purposes.
- It is not recommended for production use, especially with Tesla GPUs.

- Recommended driver for Tesla GPUs:
  - For production environments using Tesla GPUs, download the latest certified driver from the official NVIDIA Driver Downloads site:

    https://www.nvidia.com/drivers
- Optional driver installation during Toolkit setup:
  - During CUDA Toolkit installation, users may choose to skip driver installation:
    - On Windows: via interactive or silent install options.
    - On Linux: by skipping driver meta packages.
- Change in CUDA 13.1 (Windows-specific):
  - Starting with CUDA 13.1, the Windows display driver is **no longer bundled** with the CUDA Toolkit.
  - Windows users must **manually download and install** the appropriate driver from the official NVIDIA site.
- Driver compatibility notes:
  - Some compatibility tables may list “N/A” for Windows driver versions.
  - Users must still ensure the installed driver meets or exceeds the minimum required version for the CUDA Toolkit.
  - For details, refer to the official CUDA Compatibility Guide for Drivers:

    https://docs.nvidia.com/deploy/cuda-compatibility/index.html

2.3. New Features

General CUDA

- Added Event Tracing for Windows (ETW) support for CUDA driver activity reporting.

  ETW is a high-performance, low-overhead logging system built into the Windows operating system. This support enables CUDA driver activity to be reported through ETW-based diagnostics for debugging, performance monitoring, and analysis.
- Added `mmap()` support for DMA-BUF file descriptors exported from CUDA device memory on discrete GPUs.

  This support extends the GPU driver’s existing DMA-BUF `mmap()` support from Tegra system memory cases to discrete GPU video memory. It provides a low-latency CPU mapping of discrete GPU memory in environments where installing GDRCopy kernel drivers might not be desirable.

  To use this support:
  - Allocate CUDA device memory.
  - Export the desired address range as a DMA-BUF file descriptor through the CUDA DMA-BUF export path, for example by using the following driver API:

```
cuMemGetHandleForAddressRange(..., CU_MEM_RANGE_HANDLE_TYPE_DMA_BUF_FD, ...)
```

- Check whether `CU_DEVICE_ATTRIBUTE_DMA_BUF_MMAP_SUPPORTED` is supported.
- Call Linux `mmap()` on the exported DMA-BUF file descriptor.

If `mmap()` fails with `ENOTSUPP`, the driver and kernel combination does not support this path.

After the mapping is created, applications can use the CPU pointer for low-latency access to GPU memory from the CPU. Applications must follow the standard DMA-BUF CPU access protocol, including `SYNC_START` and `SYNC_END`.

Manual synchronization or fencing might be required when the same buffer is also used by the GPU, I/O devices, persistent `nv-p2p` or GDRCopy mappings, or P2PDMA.

- Added the `cuStreamBeginRecaptureToGraph()` API, which allows applications to initiate stream capture into an existing source graph.

  As the graph is recaptured, updated node parameters are applied to the existing nodes. The nodes must be recaptured in the same order as the original source graph. All recaptures to an existing source graph check for a topological match and fail if the recaptured topology diverges from the original graph.

- Updated Green Contexts so that creation of the default, or NULL, stream is no longer required through the `CU_GREEN_CTX_DEFAULT_STREAM` flag.

  Creating the default stream for a Green Context is now optional.

- Added partial support for error isolation when using MPS.

  This feature addresses a long-standing MPS limitation where some fatal GPU errors could only be contained at a broad affected-device scope, causing collateral termination of clients that did not cause the fault. This support is intended for deployments that rely on MPS for high GPU utilization while also requiring stronger service isolation, such as mixed online and offline workloads, many small database or query workloads, or embedded and robotics pipelines with independent processes.

  Partial error isolation relies on exclusive SM ownership. The MPS control daemon creates SM partitions, the MPS server assigns clients to those partitions, and the client enforces the partition through the launch affinity programmed for its work. Because an SM is owned by only one partition, CUDA can attribute SM error state to the faulting partition or client. When a fault is detected, the CUDA driver terminates work for the faulting client and prevents additional work from being submitted. Faulting or terminated operations may report `CUDA_ERROR_LAUNCH_FAILED` or a more specific CUDA error.

  Clients in different SM partitions are isolated from each other's SM-triggered kernel faults. One or more MPS clients may intentionally share the same partition, in which case those clients also share the same fault domain. This feature provides partial isolation and does not guarantee that every possible GPU or system-level failure is isolated per process.

  To use partial error isolation with MPS:

  - Start MPS in static-partition mode:

    ```
    nvidia-cuda-mps-control -d -S
    ```

    Alternatively, use:

    ```
    nvidia-cuda-mps-control -d -static-partition
    ```

  - Create one or more SM partitions:

```
echo "sm_partition add <device UUID> <number of chunks>" | nvidia-cuda-mps-
↪control
```

The command returns the full partition ID.

- Inspect the configured partitions:

```
echo "lspart" | nvidia-cuda-mps-control
```

To view the client PID, partition, and SM count, use:

```
echo "ps -f" | nvidia-cuda-mps-control
```

- Assign a client process to a partition:

```
CUDA_MPS_SM_PARTITION=<device UUID>/<partition ID> ./application
```

- Remove an idle partition:

```
echo "sm_partition rm <device UUID> <partition ID>" | nvidia-cuda-mps-control
```

When static-partition mode is enabled, a client that starts without `CUDA_MPS_SM_PARTITION`, uses an invalid partition ID, or tries to assign more than one partition from the same device fails context creation with `CUDA_ERROR_INVALID_RESOURCE_CONFIGURATION`. Removing a partition fails while clients are still actively using it. Partition IDs are deterministic for the same command sequence, partition sizes, and initial state on matching GPU SKUs.

Partitions are requested in chunks rather than percentages. For dGPU, a chunk is 4 SMs on Ampere-class GPUs and 8 SMs on Hopper and newer GPUs. For iGPU, the chunk size is 2 SMs to better fit lower-SM-count devices.

This mode trades some MPS flexibility for isolation. SMs reserved for a partition remain exclusive to the clients assigned to that partition, and other clients cannot automatically borrow idle SMs from it. This mode is best suited for workloads with known resource needs, workloads that can tolerate strict resource limits, or deployments where fault containment is more important than opportunistic sharing.

- Added the `nvmlDeviceGetRemappedRows_v2` NVML API. In addition to the information returned by `nvmlDeviceGetRemappedRows`, `nvmlDeviceGetRemappedRows_v2` returns the number of inactive row remappings.
- Added support for using both coherent-memory and non-coherent-memory GPUs in the same process on NVIDIA DGX Station systems.
- The accompanying r610 driver branch improves observability and error classification for Blackwell systems.

  Previously, certain error conditions were reported as `XID 119` timeouts, followed by `XID 94`, and then `XID 154` to indicate that the GPU required a reset. This sequence could be misleading because `XID 94` can imply error containment, while in this scenario the GPU is not usable without a reset.

  With the updated behavior, the driver logs `XID 140` for the unrecoverable ECC poison condition, skips the generic `XID 119` timeout report, logs `XID 95` for an uncontained robust-channel error from the fatal GSP poison interrupt path, and marks the GPU for reset with `XID 154`.

  Monitoring rules should treat `XID 140` or `XID 95` followed by `XID 154` as a GSP memory poison and reset-required condition instead of as a general GSP timeout.

- Added Dynamic Boost support for Grace Blackwell systems.

  Note: This feature is available on Grace Blackwell systems and requires the installation of the NVIDIA r610 driver, corresponding to CUDA Toolkit 13.3, or later.

  Dynamic Boost is a Grace CPU frequency optimization feature that runs as a background service. It uses machine learning to model performance and power impact, and balances them based on workload demands. It continuously monitors the system and adjusts the CPU clock speed in real time to match the needs of the running application.

  When the workload enters phases that are less CPU-bound, Dynamic Boost temporarily lowers the CPU frequency to reduce power consumption. The saved power can then be reallocated to the GPU for improved performance. When the workload becomes more CPU-bound, Dynamic Boost can increase the CPU frequency accordingly.

  Dynamic Boost is available on GB200 and GB300 systems. The service can be managed using standard `systemctl` commands:

  - Enable the service:

    ```
    systemctl enable nvidia-powerd.service
    ```

  - Start the service:

    ```
    systemctl start nvidia-powerd.service
    ```

  - Stop the service:

    ```
    systemctl stop nvidia-powerd.service
    ```

  - Disable the service:

    ```
    systemctl disable nvidia-powerd.service
    ```

- Added support for remote CPU-to-GPU mapping of managed memory.

  CDMM-managed memory that is resident on the CPU can now be mapped directly onto a GPU, allowing the GPU to access the memory without an explicit migration. This capability is reported based on GPU and platform support. It is enabled when `cuDeviceGetAttribute()` reports `CU_DEVICE_ATTRIBUTE_DIRECT_MANAGED_MEM_ACCESS_FROM_HOST = 1` for the device.

- Added System-Allocated Memory (SAM) migration support for CDMM.

  CDMM can now migrate system-allocated memory, such as memory allocated with `malloc()`, between the CPU and GPU. This support requires a Linux kernel that includes the necessary HMM fixes. Kernel 6.12 or later is recommended. On older kernels, SAM migration remains disabled and existing behavior is preserved.

CUDA Compiler

- For new features from PTX, refer to PTX ISA version 9.3.
- Introduced CUDA Tile C++, which adds support for tile programming in CUDA C++. This feature is supported in both `nvcc` and NVRTC. For more information about writing tile kernels, see the CUDA Programming Guide and the CUDA Tile C++ API Reference.

  For CUDA Tile C++ API-specific release information, see the CUDA Tile C++ API Reference release notes.

- Added NVRTC support for installing a curated set of CUDA C++ and CCCL headers directly from the NVRTC distribution.

  This support enables runtime compilation of code that uses modern CUDA facilities without requiring a full CUDA Toolkit installation or manual header management.

- Added support for Advanced Control Files (ACFs) in NVIDIA compiler toolchains through the `--apply-controls=<file>` option.

  This option enables compiler binaries to process encrypted control files as part of the build workflow. Documentation updates, including usage examples for `--apply-controls`, are available in the CUDA Programming Guide and the NVIDIA CUDA Compiler Driver, NVCC, documentation.

  ACFs are generated using NVIDIA CompileIQ. For more information, see CompileIQ on GitHub.

  Caveats:

  - This feature is not supported in NVRTC.
  - PTXAS support is available only in offline compilation flows and is not supported when using static libraries. For more information, refer to the PTXAS help documentation.
  - Correct compilation behavior is not guaranteed when using ACFs. For more information, see CompileIQ on GitHub.

- Added official C++23 support in `nvcc` and NVRTC.

  This support enables developers to adopt the latest C++ language features across both ahead-of-time and runtime CUDA compilation workflows. It improves code portability, modernizes GPU application development, and helps align CUDA codebases with contemporary C++ standards.

- Added `nvprune` functionality to `nvcc`.

  This support helps developers streamline deployment artifacts and manage builds for targeted GPU architectures without requiring a separate pruning step to remove unnecessary code from the libraries they use.

2.3.1. 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.

2.3.2. CUDA C++ Core Libraries (CCCL)

- Added tensor interoperability support in libcu++.

  The `cuda::to_device_mdspan()`, `cuda::to_host_mdspan()`, and `cuda::to_managed_mdspan()` APIs convert a DLPack `DLTensor` into a typed `cuda::std::mdspan` view that CUDA kernels can operate on with shape and stride metadata. DLPack is an interchange format used by Python frameworks such as PyTorch, JAX, and CuPy.

  The `cuda::to_dlpack_tensor()` API converts an `mdspan` to a `DLManagedTensor` wrapper.

  The `cuda::shared_memory_mdspan` API provides a multidimensional view over a CUDA shared memory tile. The accessor guarantees shared memory load and store instructions and adds address space safety checks.

- Added expanded random number distribution support in libcu++.

  The `<cuda/std/random>` header now includes 17 host- and device-compatible random number distributions, bringing libcu++ closer to parity with the C++ standard library `<random>` header.

  Supported distributions include:

  - Uniform distributions:
    - `cuda::std::uniform_int_distribution`
    - `cuda::std::uniform_real_distribution`
  - Normal-family distributions:
    - `cuda::std::normal_distribution`
    - `cuda::std::lognormal_distribution`
  - Discrete distributions:
    - `cuda::std::bernoulli_distribution`
    - `cuda::std::binomial_distribution`
    - `cuda::std::negative_binomial_distribution`
    - `cuda::std::geometric_distribution`
    - `cuda::std::poisson_distribution`
  - Continuous distributions:
    - `cuda::std::exponential_distribution`
    - `cuda::std::gamma_distribution`
    - `cuda::std::weibull_distribution`
    - `cuda::std::extreme_value_distribution`
    - `cuda::std::cauchy_distribution`
    - `cuda::std::chi_squared_distribution`
    - `cuda::std::fisher_f_distribution`
    - `cuda::std::student_t_distribution`

CCCL 3.3 also backports the C++26 counter-based engines `cuda::std::philox4x32` and `cuda::std::philox4x64` to C++17. The `<cuda/random>` header adds `cuda::pcg64` as an NVIDIA extension.

- Added new CUB device-wide algorithms.

  New and updated algorithms include:

  - `cub::DeviceFind::FindIf`: Improves performance for device-wide searches for the first element that satisfies a predicate.
  - `cub::DeviceFind::LowerBound` and `cub::DeviceFind::UpperBound`: Add parallel binary search support to locate multiple values in an ordered sequence in a single device-wide call.
  - `cub::DeviceSegmentedScan`: Adds segmented prefix-sum and scan support over multiple independent contiguous segments in a single device-wide call.
  - `cub::DeviceTransform::Transform`: Adds an N-input, M-output form. A single call can read from N input sequences and write to M output sequences, driven by a user-provided operation that returns a `cuda::std::tuple<...>`.
  - `cub::DeviceTopK` and `cub::DeviceSegmentedTopK`: Add device-wide top-k selection support, including a fixed-size segments variant.

- For the complete list of CCCL 3.3.0 changes, see the CCCL 3.3.0 changelog.

2.3.3. CUDA Python

- First stable release of `cuda.core`.

  As of version 1.0.0, all APIs are considered stable and follow Semantic Versioning (SemVer), with appropriate deprecation periods for breaking changes. See the support policy for details.

- Added green context support for CUDA 12.4 and later.

  New types `Context`, `ContextOptions`, `SMResource`, `SMResourceOptions`, `WorkqueueResource`, and `WorkqueueResourceOptions` enable GPU SM and workqueue resource partitioning. Green contexts can be created with `Device.create_context()`. Use `Context.create_stream()` and `Context.resources` to work within the partitioned resources. #1976

- Added the `cuda.core.checkpoint` module for CUDA process checkpointing.

  The module includes string process state queries, lock, checkpoint, restore, and unlock operations, and GPU UUID remapping support for restore. #1343

- Added an optional `cache=` keyword argument to `Program.compile()`.

  This argument helps avoid recompilation of identical source, options, and target combinations. Two concrete implementations of the `ProgramCacheResource` abstract base class are provided:

  - `InMemoryProgramCache`, a thread-safe, single-process LRU cache.
  - `FileStreamProgramCache`, a disk-backed, cross-process safe, LRU-evicting cache.

  A standalone `make_program_cache_key()` function is also exposed for callers who need to include additional content, such as headers or PCH files, in the cache key. #1912

- Added NVIDIA Management Library (NVML) access updates to the `cuda.core.system` module.

  The following APIs are now available:

- `system.Device.mig` for querying and setting MIG mode, enumerating MIG device instances, and navigating parent and child relationships. #1916
- `system.Device.compute_running_processes` for querying running compute processes on a device. This API returns `ProcessInfo` objects with PID, GPU memory usage, and MIG instance IDs. #1917
- `system.Device.get_nvlink()` for querying NVLink version and state per link. `system.Device.utilization` now returns current GPU and memory utilization rates. #1918
- `cuda.core.typing.VirtualMemoryLocationType`.

- For the full `cuda.core` 1.0 changelog, see cuda.core 1.0.0 release notes.
- First stable release of `cuda.compute`.

  As of version 1.0.0, all APIs are considered stable and follow Semantic Versioning (SemVer), with appropriate deprecation periods for breaking changes. See the support policy for details.

  - Added broad algorithm coverage to `cuda.compute`.

    Supported algorithms include `reduce_into`, `inclusive_scan`, `exclusive_scan`, `unary_transform`, `binary_transform`, `lower_bound`, `upper_bound`, `segmented_reduce`, `merge_sort`, `radix_sort`, `histogram`, `select`, `three_way_partition`, and `unique_by_key`.
  - Added support for Python callables, including plain lambda expressions, as device-wide operators across reductions, scans, and transforms.

    The callable is JIT-compiled to LTO-IR through Numba and linked into the underlying CUB kernel at runtime. Stateful operators that capture mutable globals or device-side state are also supported.
  - Added `cuda.compute.lower_bound` and `cuda.compute.upper_bound` for parallel binary search.

    These APIs provide device-wide functionality similar to `numpy.searchsorted`. Both APIs return insertion indices and accept any iterator type.
  - Added `ShuffleIterator` for deterministic pseudorandom permutation of an input range.

    `ShuffleIterator` complements the existing iterators: `CountingIterator`, `ConstantIterator`, `TransformIterator`, `ZipIterator`, and `PermutationIterator`.
  - For the full `cuda.compute` 1.0 changelog, see cuda.compute 1.0 release notes.

2.3.4. CUDA TILE

2.3.4.1 Supported Architectures

- Added support for Hopper `sm_90` architecture.
- Added support for all Ampere and later architectures, `sm_80` and later.

2.3.4.2 New Operations

- Added `alloca` for automatic memory allocation.
- Added `mmaf_scaled` for floating-point matrix-multiply-accumulate with scaled inputs on `sm_100` and later architectures.
- Added `pack` to pack a tile into a byte array.
- Added `unpack` to unpack a byte array into a tile.
- Added `make_strided_view` to create a strided view from a tensor view.
- Added `make_gather_scatter_view` to create a gather/scatter view from a tensor view.
- Added `atomic_red_view_tko` for view-based atomic reduction on global memory.

2.3.4.3 New Types

- Added `strided_view` for strided tile views with configurable traversal strides.
- Added `gather_scatter_view` for gather/scatter access patterns over tensor views.
- Added `i4`, a 4-bit integer type for quantization support. `i4` tiles must be converted to a supported integer type before they are used in operations.
- Added `f4E2M1FN`, a 4-bit floating-point type.

2.3.4.4 Modified Operations

- Modified `entry` to add the `num_worker_warps_per_cta` optimization hint.
- Modified `entry` to add default key support for optimization hints, which provides a fallback when no target-specific hint is provided.
- Modified `mmaf` to add the `fast_acc` attribute for faster but less precise FP8 MMA accumulation on Hopper GPUs.
- Modified `global` to add the `constant` attribute to mark globals as immutable or read-only.
- Modified `global` to add the `symbol_visibility` attribute for public or private visibility.
- Modified `module` to add the `producer` attribute for identifying the generating tool.
- Modified `exp` to add the `rounding_mode` attribute with `approx` and `full` modes.
- Modified `atomic_rmw_tko` to add `bf16` support to `ADDF` mode.
- Modified `load_view_tko` and `store_view_tko` to change index types from scalar-only indices to 1D tensor indices for gather/scatter support.
- Modified `exti`, `trunci`, `pack`, and `unpack` to add `i4` type support.

2.3.4.5 Documentation Improvements

- Added 4-bit memory layout documentation to the tensor view type.
- Added overflow and undefined behavior documentation improvements to `ftoi` and `itof`.

2.3.4.6 Fixed Issues

- Fixed an issue where `atomic_rmw_tko FADD` with `f16` could produce incorrect behavior.

2.3.4.7 Known Issues

- Declaring an `f16` constant, converting it to an FP8 type, and then printing it on `sm_120` can cause a compiler crash.

2.4. Resolved Issues

2.4.1. CUDA Compiler

- Fixed an issue where applications that use NVRTC and are linked with LLVM `lld` or `mold` could fail to initialize the supported architecture list. This issue could cause `nvrtcGetNumSupportedArchs` to return `0` and `nvrtcGetSupportedArchs` to fail with `NVRTC_ERROR_INVALID_INPUT`. [*5020829*]
- Fixed a compiler issue, present since CUDA 12.8, that could cause compiler-inserted thread reconvergence to fail and leave stale or corrupted values in registers, resulting in incorrect program execution. This issue could occur only in kernels that contain two or more nested levels of thread divergence where the compiler elided convergence instructions for one or more divergence levels. Kernels with only a single level of divergence are unaffected.[*6156910*]

  Minimal illustrative example:

```
__global__ void affected(const int* in, int* out) {
    int tid = threadIdx.x;

    if (tid < 16) {             // Level 1 divergence
        if (in[tid] > 0) {      // Level 2 divergence, nested
            // If the compiler elides reconvergence for one of these
            // divergence levels, a register written in this region
            // can carry a stale value across the reconvergence point.
            out[tid] = compute(in[tid]);
        }

        // Implicit reconvergence: failure can manifest here.
    }
}
```

2.4.2. CUDA Tools

- Fixed Data Race in WGMMA A/B Register Copy Propagation

  Symptom:

  Kernels that use `wgmma.mma_async` with `wgmma.wait_group.sync.aligned N`, where `N >= 1`, might produce incorrect numerical results when `mov` instructions are intentionally placed after the wait to protect WGMMA input registers from being overwritten by in-flight WGMMA operations.[*5912730*]

  Problem Description:

  In pipelined warp-group MMA loops, `mov.b32` instructions are sometimes intentionally placed after `wgmma.wait_group.sync.aligned N`, where `N > 0`, to ensure that WGMMA input registers are not overwritten while WGMMA operations from a previous iteration are still in flight. In this specific case, `ptxas` could incorrectly copy-propagate across `wgmma.wait_group.sync.aligned N` and eliminate those `mov` instructions.

  For example, in the following pattern, `ldmatrix.x4` feeds two consecutive `wgmma` calls. The compiler could incorrectly eliminate the second `mov.b32` instruction, exposing the race:

```
$loop:
   ldmatrix.sync.aligned.m8n8.x4.shared.b16 {%r71, %r91, %r81, %r101}, [%r8];

   mov.b32 %r70, %r71;
   wgmma.fence.sync.aligned;
   wgmma.mma_async { }, {%r70, ..}, ...;
   wgmma.commit_group.sync.aligned;
   wgmma.wait_group.sync.aligned 1;

   mov.b32 %r80, %r81;                    // compiler eliminates this mov
   wgmma.fence.sync.aligned;
   wgmma.mma_async { }, {%r80, ..}, ...;  // uses %r81 directly after copy
↪propagation
   wgmma.commit_group.sync.aligned;
   wgmma.wait_group.sync.aligned 1;

   bra $loop;

wgmma.wait_group.sync.aligned 0;
```

  Workaround: For CUDA Toolkits prior to 13.3, use one of the following workarounds:

  - Use `wgmma.wait_group.sync.aligned 0` to wait for all in-flight groups before the next `ldmatrix` instruction.
  - Use individual `ldmatrix.x1` loads directly into WGMMA input registers, eliminating the `mov` instruction that triggers copy propagation.

2.5. Known Issues

2.5.1. General CUDA

- Compute Fabric Transport
  - Enabling support for unicast logical endpoints requires Fabric Manager 610 or later.
  - The performance of the `fabric.try_pullred` instruction is suboptimal. This issue will be addressed in a future CUDA release.
  - If a memory allocation is bound to a logical endpoint using `cuLogicalEndpointBindMem` or `cuLogicalEndpointBindAddr`, pointer access to that allocation from multiple GPUs within the same process might not work correctly. This issue will be addressed in a future CUDA release.

2.5.2. CUDA Compiler

- CUDA Tile C++

  The Tile C++ compiler may incorrectly issue a diagnostic when `printf()` is invoked from a tile function if GLIBC fortification level 2 or higher is enabled by the host compiler. This level of GLIBC fortification is enabled by default on Ubuntu distributions of GCC when `-O1` or higher is present. It can also be enabled explicitly when `-O1` or higher is present and `-D_FORTIFY_SOURCE=2` or higher is specified.

  The following example demonstrates the issue:

```
#include <cstdio>

__tile_global__ void kernel() {
  printf("hello\n");
}
```

  Compile the example with the following command:

```
nvcc --enable-tile -arch sm_80 -std=c++20 -O1 -D_FORTIFY_SOURCE=2 main.cu
```

  The compiler may report the following diagnostic:

```
error: calling a __host__ __device__ function("printf") from a __tile_global__
↪function("kernel") is not allowed
```

  As a workaround, disable GLIBC fortification by adding `-U_FORTIFY_SOURCE` to the `nvcc` command line. A fix for this issue will be available in a future CUDA release.

- For semantic clarifications, refer to Changes in PTX ISA version 9.3.

2.6. Deprecated or Dropped Features

2.6.1. General CUDA

- Legacy Nsight Eclipse Edition plugins are no longer delivered in CUDA Toolkit packages beginning with CUDA 13.3.

2.6.2. Deprecated Architectures

- None

2.6.3. Deprecated or Dropped Operating Systems

- None

2.6.4. Deprecated or Dropped CUDA Toolchains

- None

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 13.3

- **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.

3.1.2. 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.
  - 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.
- **Resolved Issues**
  - Fixed an issue in `cublasLtMatmulAlgoGetHeuristic()` that could result in no algorithm candidates being returned for Grouped GEMM on Blackwell GPUs. [*CUB-9657*]

3.1.3. 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.
  - 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*]
- **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*]

3.1.4. 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*]

- **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*]

3.1.5. 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 SM121 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*]

3.1.6. 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*]

3.1.7. 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*]

3.1.8. 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 of SM110 GPUs for arm64-sbsa on Linux.
- **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]*

- **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 13.3

- **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*]

3.2.2. 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.

3.2.3. 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.

3.2.4. 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.

3.2.5. 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 13.3

- **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*]

3.3.2. 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.

3.3.3. 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*]

3.3.4. cuSOLVER: Release 13.1

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

3.3.5. 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.

3.3.6. 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 13.3

- **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*]

3.4.2. 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.

3.4.3. cuSPARSE: Release 13.2

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

3.4.4. 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*]

3.4.5. 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*]

3.4.6. 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)*.

3.4.7. 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 13.3

- **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*]

3.5.2. 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*]

3.5.3. 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*]

3.5.4. 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 13.3

- **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.

3.6.2. 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.

3.6.3. 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]*

3.6.4. 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*).

3.6.5. 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 13.1 Update 1

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

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 and the usage demonstrated in the StreamContexts example.
- **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