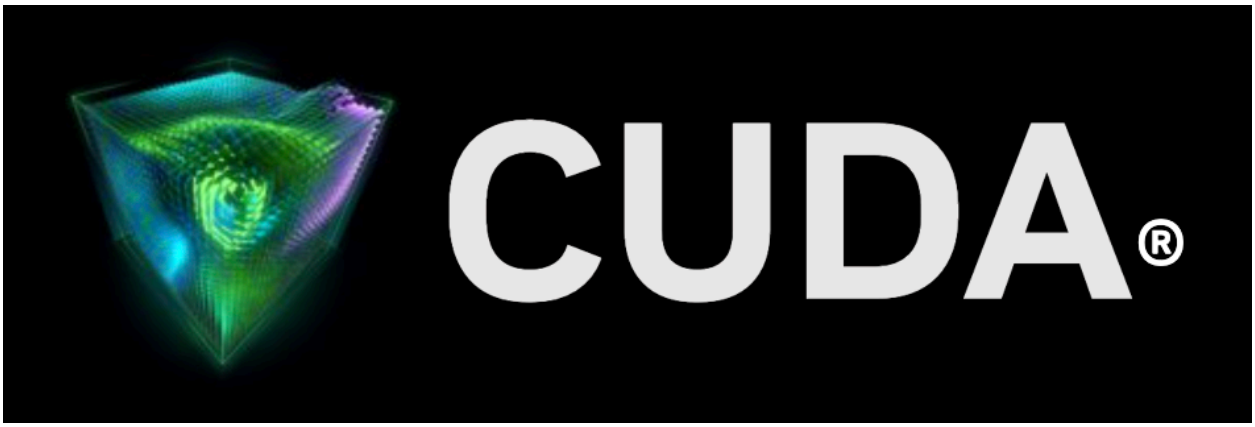


CUDA Features Archive

Release 13.5

NVIDIA Corporation

Nov 02, 2026

Contents

1 CUDA 13.4 **3**
1.1 Features . . . 3
1.1.1 CUDA Platform . . . 3
1.1.2 CUDA Developer Tools . . . 4
1.1.3 CUDA Compiler . . . 5
1.1.4 CUDA C++ Core Libraries (CCCL) . . . 5
1.1.5 CUDA Python . . . 5
1.1.6 CUDA Tile . . . 6
1.1.7 CUDA Tile IR . . . 7
1.2 Resolved Issues . . . 9
1.2.1 General CUDA . . . 9
1.2.2 CUDA Compiler . . . 10
1.2.3 CUDA Tools . . . 11
1.3 Known Issues . . . 12
1.3.1 CUDA Platform . . . 12
1.3.2 CUDA Compiler . . . 12
1.4 Deprecated or Dropped Features . . . 12
1.4.1 CUDA Platform . . . 12
1.4.2 Architectures . . . 12

2 CUDA 13.3 **13**
2.1 Features . . . 13
2.1.1 General CUDA . . . 13
2.1.2 CUDA Compiler . . . 16
2.1.3 CUDA Developer Tools . . . 17
2.1.4 CUDA C++ Core Libraries (CCCL) . . . 17
2.1.5 CUDA Python . . . 19
2.1.6 CUDA Tile . . . 20
2.2 Resolved Issues . . . 21
2.2.1 CUDA Compiler . . . 21
2.2.2 CUDA Tools . . . 22
2.3 Known Issues . . . 23
2.3.1 General CUDA . . . 23
2.3.2 CUDA Compiler . . . 23
2.4 Deprecated or Dropped Features . . . 24
2.4.1 General CUDA . . . 24

3 CUDA 13.2 **25**
3.1 Features . . . 25
3.1.1 General CUDA . . . 25
3.1.2 CUDA Compiler . . . 25
3.1.3 CUDA Developer Tools . . . 25
3.2 Resolved Issues . . . 26

3.2.1 CUDA Compiler . . . 26
3.3 Known Issues . . . 26
3.3.1 General CUDA . . . 26
3.3.2 CUDA Compiler . . . 27
3.4 Deprecated or Dropped Features . . . 27
3.4.1 General CUDA . . . 27
3.4.2 Deprecated Architectures . . . 29
3.4.3 Deprecated or Dropped Operating Systems . . . 29
3.4.4 Deprecated or Dropped CUDA Toolchains . . . 29

4 CUDA 13.1 **31**
4.1 Features . . . 31
4.1.1 General CUDA . . . 31
4.1.2 CUDA Developer Tools . . . 31
4.2 Resolved Issues . . . 31
4.2.1 CUDA Compiler . . . 31
4.3 Known Issues . . . 32
4.3.1 General CUDA . . . 32
4.3.2 CUDA Compiler . . . 32
4.4 Deprecated or Dropped Features . . . 33
4.4.1 General CUDA . . . 33
4.4.2 Deprecated Architectures . . . 34
4.4.3 Deprecated or Dropped Operating Systems . . . 34
4.4.4 Deprecated or Dropped CUDA Toolchains . . . 35

5 CUDA 13.0 **37**
5.1 Features . . . 37
5.1.1 General CUDA . . . 37
5.1.2 CUDA Compiler . . . 37
5.1.3 CUDA Developer Tools . . . 37
5.2 Known Issues . . . 38
5.2.1 General CUDA . . . 38
5.3 Deprecated or Dropped Features . . . 38
5.3.1 General CUDA . . . 38
5.3.2 Deprecated Architectures . . . 40
5.3.3 Deprecated or Dropped Operating Systems . . . 40
5.3.4 Deprecated or Dropped CUDA Toolchains . . . 40

6 CUDA 12.9 **43**
6.1 Features . . . 43
6.1.1 General CUDA . . . 43
6.1.2 CUDA Compiler . . . 44

7 CUDA 12.8 **45**
7.1 Features . . . 45
7.1.1 General CUDA . . . 45
7.1.2 CUDA Compiler . . . 46

8 CUDA 12.6 **47**
8.1 Features . . . 47
8.1.1 General CUDA . . . 47
8.1.2 CUDA Compiler . . . 47

9 CUDA 12.5 **49**
9.1 Features . . . 49

9.1.1 General CUDA 49

10 CUDA 12.4 **51**
10.1 Features 51
10.1.1 General CUDA 51
10.1.2 CUDA Compiler 52

11 CUDA 12.3 **53**
11.1 Features 53
11.1.1 General CUDA 53
11.1.2 CUDA Compiler 54

12 CUDA 12.2 **55**
12.1 Features 55
12.1.1 General CUDA 55
12.1.2 CUDA Compiler 56

13 CUDA 12.1 **57**
13.1 Features 57
13.1.1 General CUDA 57
13.1.2 CUDA Compiler 57

14 CUDA 12.0 **59**
14.1 Features 59
14.1.1 General CUDA 59
14.1.2 CUDA Compiler 60

15 CUDA 11.8 **61**
15.1 Features 61
15.1.1 General CUDA 61

16 CUDA 11.7 **63**
16.1 Features 63
16.1.1 General CUDA 63
16.1.2 CUDA Compiler 63

17 CUDA 11.6 **65**
17.1 Features 65
17.1.1 CUDA Compiler 65

18 Notices **67**
18.1 Notice 67
18.2 OpenCL 68
18.3 Trademarks 68

CUDA Features Archive

This page lists CUDA features and related release information by release.

Chapter 1. CUDA 13.4

1.1. Features

1.1.1. CUDA Platform

- CUDA 13.4 adds support for Windows on Arm on RTX Spark devices.
- Vera Rubin platform support is available in CUDA 13.4 as a developer preview. Vera Rubin support in CUDA Toolkit 13.4 is not intended for benchmarking, performance analysis, or production deployment.
- Separately released R615 driver packages no longer include the proprietary kernel modules; supported Linux systems use the NVIDIA open kernel modules.
- **Note:** The following change requires the R615 driver (corresponding to the CUDA 13.4 Toolkit) or later.

  On all NVIDIA hardware-coherent platforms, the NVIDIA driver now defaults to Coherent Driver-based Memory Management (CDMM) instead of onlining GPU memory to the operating system as a NUMA node. NUMA mode remains fully supported. The mode is a node-wide setting controlled by a kernel module parameter and takes effect after a driver reload or reboot, so it should be selected before upgrading.

  To select CDMM mode:

  ```
  echo 'options nvidia NVreg_CoherentGPUMemoryMode=driver' | sudo tee /etc/modprobe.
  ↪d/nvidia-openrm.conf
  ```

  To select NUMA mode:

  ```
  # Loading NVIDIA module instructions
  echo 'options nvidia NVreg_CoherentGPUMemoryMode=numa' \
  | sudo tee /etc/modprobe.d/nvidia-openrm.conf

  # NVIDIA-UVM module parameter
  echo 'options nvidia-uvm uvm_disable_sam_migration=false' \
  | sudo tee -a /etc/modprobe.d/nvidia-openrm.conf
  ```

  Reload the driver or reboot for the change to take effect. To verify the active mode:

  ```
  grep Coherent /proc/driver/nvidia/params

  # Check for SAM migration
  cat /sys/module/nvidia_uvm/parameters/uvm_disable_sam_migration
  ```

For a comparison of both modes, see the CUDA Memory Management blog post.

- Added `cuMemGetLocationInfo` and `cudaMemGetLocationInfo` APIs for querying residency information for Unified Memory allocations, enabling libraries and runtimes to select compute and communication resources based on current data locality.
- CUDA memory pool IPC is now supported on Windows WDDM systems for allocations created with the stream-ordered allocator.
- CUDA MPS adds a new command-line interface, file-based configuration, process namespaces, cgroup-integrated device-memory limits, and per-container time slicing for GPU fractionalization. For more information, see the CUDA Toolkit 13.4 release blog.
- `cuda-checkpoint` can now write checkpoint state to persistent storage instead of retaining it only in process memory.
- CUDA and NVML add cgroup-based GPU-memory limits. Allocation and memory-reporting APIs now honor the configured limit, and NVML and `nvidia-smi` expose corresponding management support.
- CUDA Graphs now support associating Green Contexts with event-record, host, external-semaphore, and kernel nodes. User-specified contexts on kernel nodes are now honored consistently.
- On supported multi-chip GPUs, CUDA APIs expose GPU topology and allow applications to select compute and memory resources according to locality.
- CUDA Compute Fabric Transport (CFT) is enhanced for handle-based fabric programming. For more information, see the CUDA Toolkit 13.4 release blog.
- CUDA device attribute query introduces the new attribute `CU_DEVICE_ATTRIBUTE_LOGICAL_ENDPOINT_SUPPO` to indicate the supported IPC handle types for logical endpoints.
- Structured CPER (Common Platform Error Record) delivery for GPU errors is available as a preview.

1.1.2. 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.

1.1.3. CUDA Compiler

- PTX ISA 9.4 is supported. For new PTX features, see the PTX ISA 9.4 documentation.
- Added support for GCC 16 and Clang 22 as host-side compilers with NVCC.
- Added support for libc++ 22 as a C++ standard library with NVCC.
- Added support for concatenated-entry fatbins in `nvFatbin` and `fatbinary` for cross-entry similarity compression.

1.1.4. CUDA C++ Core Libraries (CCCL)

- CCCL 3.4 significantly improves `cub::DeviceScan` performance on Blackwell with a warp-specialized implementation using the Tensor Memory Accelerator (TMA).
- CUB now provides single-call, environment-based overloads for all device-wide algorithms, removing the required temporary-storage query phase while retaining the traditional two-phase APIs.
- Added `cub::WarpReduceBatched` for reducing multiple independent batches across a warp.
- Added `cuda::std` parallel algorithms selected with the `cuda::execution::gpu` execution policy.
- Deprecated Thrust constant and strided iterators in favor of their `cuda::` equivalents, along with default-alignment overloads on type-erased memory-resource wrappers and several legacy helper APIs.
- CCCL 3.4.1 and 3.4.2 add scan and reduction fixes, updated tuning, OpenMP scan bounds fixes, and disable Tile compilation mode.
- For the full changelog, see the CCCL 3.4.0 release notes.

1.1.5. CUDA Python

- cuda-bindings 13.4.0 adds the new CUDA Toolkit 13.4 Driver, Runtime, cuFile, and NVML APIs, including fabric-cluster and clique queries, memory-location information, graph node v3 APIs, checkpoint completion, vectored cuFile I/O, adaptive TGP, memory-limit, GPU-fabric, performance-metric, NVLink-telemetry, and operational-event interfaces.
- Added an experimental `cuda.bindings._v2.nvrtc` API with exception-based error handling, PEP 8-compliant names, and improved performance.
- cuda-bindings now raises `OverflowError` for out-of-range `c_int` and `c_byte` kernel arguments, adds linked-LTOIR support for `nvJitLink`, improves thread safety, and adds Windows Arm64 build support.
- cuda.core 1.1.1 adds `ObjectCode.get_module()` for legacy `CUmodule` interoperability, ships C++ headers in wheels and source distributions, fixes graph-node resource lifetimes and DLPack handling, hardens program-cache permissions, and supports prerelease cuda-bindings version strings such as `13.4.0b1`.
- cuda-pathfinder 1.6.0 improves CUDA Toolkit binary discovery and adds library and header discovery for cuQuantum libraries, cuTENSORMp, and Windows Arm64 cuDLA.

- cuda.compute 1.1 adds serialization and deserialization of algorithm objects, ahead-of-time compilation for multiple compute capabilities and GPU-less systems, and build-cache fixes. Version 1.1.1 fixes warpspeed scan on Windows with `sm_120`.
- For more information, see the CUDA Python 13.4 release notes and the CCCL Python 1.1 release notes.

1.1.6. CUDA Tile

Tile C++ Features

- Added `ct::strided_view` type for performing stencil operations.
- Added `ct::gather_scatter_view` type for sparse memory access patterns.
- For additional features and improvements, see the CUDA Tile C++ API Reference Release Notes.

Tile Python Features

- **CUDA Tile Python 1.5.0** adds the following:
  - `ct.assume_divisible_by()` to communicate scalar-divisibility and alignment assumptions to the compiler.
  - Compile-time specialization of selected array-shape dimensions through `ct.ArrayAnnotation(static_shape_dims=...)`.
  - Tuple-valued kernel arguments, including nested tuples and per-element constant annotations.
  - `single_run_timeout_sec` for `ct.tune.exhaustive_search()` so that hanging kernels do not stall an entire tuning run.
  - Improved autotuning performance by stopping slow configurations early.
  - Dynamic start and step arguments for `ct.arange()`.
  - Tuple comprehensions and tuple `in`/`not in` operations.
  - Limited support for dictionaries, `**kwargs`, and dictionary unpacking.
  - `enum.Enum` construction and comparison inside kernels.
  - Printing of dataclass instances, and support for frozen dataclasses as device-code globals.
  - Corrected conversion of strictly typed numeric constants, including integer wrapping and floating-point rounding or clamping.
  - Support for wrappers and decorators applied to user-defined functions.
  - Rejection of repeated axes in `load()`, `store()`, and `num_tiles()` ordering arguments.
  - Calling-convention v2 for tuple arguments and static-shape annotations.
- **CUDA Tile Python 1.6.0** adds the following:
  - A `cutile-cache log` command for inspecting compilation history, including compiler versions, compilation time, kernel names, and TileIR 13.4 compiler remarks.
  - `check_bounds=False` for `ct.load()`, `ct.store()`, and tiled-view load/store methods to omit bounds checks when the programmer guarantees accesses are valid. Requires TileIR 13.4 or later.

- Programmatic Dependent Launch support: `ct.grid_dependency_control_launch_dependents()`, `ct.grid_dependency_control_wait()`, and a `programmatic_dependent_launch` option for `ct.launch()`.
- `float8_e5m3fnu` support for FP4 block-scaled MMA on Rubin `sm_107`.
- Improved floating-point powers with integer exponents using FPowI on TileIR 13.4 or later.
- IEEE rounding modes for float-to-float `astype()` conversions.
- A `propagate_nan` option for min/max and argmin/argmax operations.
- `ct.insert()` and `Tile.insert()` for inserting a subtile into a larger tile.
- Static specialization of selected array-stride dimensions.
- `ct.ensure_constant()` for asserting that a value is compile-time constant.
- `ct.divmod()` and support for Python's built-in integer `divmod()`.
- User-defined context-manager support.
- User-defined dataclass methods, including `__post_init__`, `__call__`, indexing, and string representations.
- `break` support in non-static loops.
- Autotuning can now select a different kernel for each configuration.
- `export_kernel()` can now export portable TileIR bytecode without GPU code.
- Kernels are now compiled for the actual launch device rather than always targeting device 0.
- Tuple parameters in the JAX integration.
- Fixed runtime-loop scalar updates and CUDA crash-dump handling.

1.1.7. CUDA Tile IR

- **Supported Architectures**
  - (Developer Preview) Added support for the `sm_107` (Rubin) architecture.
- **New Operations**
  - Added op `cuda_tile.fpowi` for element-wise floating-point exponentiation with a signed-integer exponent.
  - Added op `cuda_tile.insert` to insert a source subtile into a destination tile at a given subtile index. The source shape must evenly divide the destination shape.
  - Added op `cuda_tile.memory_fence_alias_tko`, a token-ordered fence that orders operations accessing the same physical memory through different virtual aliases.
  - Added op `cuda_tile.gdc_launch_dependents_tko` for grid dependency control, signaling that programmatic-dependent-launch (PDL) dependent kernels may begin. It is a no-op below `sm_90`.
  - Added op `cuda_tile.gdc_wait_tko` for grid dependency control, waiting for predecessor-kernel completion with acquire semantics. It is a no-op below `sm_90` and pairs with `cuda_tile.gdc_launch_dependents_tko`.
- **New Types and Attributes**

- (Developer Preview) Added the `fnv8E5M3FNU` (8-bit floating-point) type. It is an alternative floating-point type intended for use as a block-scale type in `cuda_tile.mmaf_scaled` with `f4E2M1FN` (fp4) elements.

- **Modified Operations**
  - (Developer Preview) Modified op `cuda_tile.mmaf_scaled` to add support for `f4E2M1FN` inputs with `fnv8E5M3FNU` scale factors accumulating to `f32`.
  - Modified op `cuda_tile.loop` to support function return: a `cuda_tile.return` may appear inside a loop body to return from the enclosing function.
  - Modified ops `cuda_tile.load_view_tko` and `cuda_tile.store_view_tko` to add a new `inbounds` attribute so that programs can convey that accesses along specified dimensions are statically known to be in-bounds, allowing the compiler to skip bounds checking for those dimensions. Dimensions left unmarked default to conservative bounds checking, which matches the behavior of prior versions.
  - Modified op `cuda_tile.ftoi` to add an optional saturating modifier that clamps out-of-range values and converts NaN to 0.
  - Modified op `ftof` to support a richer rounding-mode matrix, including `nearest_away` for `f32` to `tf32`.
  - Renamed op `cuda_tile.pow` to `cuda_tile.fpowf`, complementing the new integer-exponent `cuda_tile.fpowi`.

- **Compiler Improvements**
  - Added opt-in compiler optimization remarks that identify selected load and store instructions and explain TMA instruction-selection failures. Use `tileiras --remarks=all` to enable all remarks; see the documentation for further details.
  - `tileiras` and the CUDA driver JIT compiler flow now re-use the Tile IR-level optimization pipelines from `cuda-tile-optimize`, applying canonicalization, common-subexpression elimination, and loop-invariant code motion according to the selected optimization level. See the documentation for further details.

- **Documentation Improvements**
  - Documented that the combiner function for `cuda_tile.reduce` and `cuda_tile.scan` must be commutative as well as associative. This was already required by the implementation, which may reorder the combine freely.
  - Added memory-alignment guidance for pointer loads, stores, atomic operations, and view accesses, including byte-alignment requirements for sub-byte element types.
  - Clarified `cuda_tile.partition_view` and `cuda_tile.strided_view` out-of-bounds semantics, distinguishing in-bounds view indices from partially out-of-bounds tiles and documenting load padding and store masking.
  - Clarified the `cuda_tile.tensor_view` memory contract for 4-bit elements, including dense packing, byte-alignment requirements, and little-endian nibble order.
  - Expanded `cuda_tile.gather_scatter_view` documentation with sparse-dimension indexing rules and multidimensional gather, scatter, padding, and out-of-bounds examples.
  - Expanded optimization-hint reference documentation with per-operation and architecture-specific constraints.

- **Fixed Issues**

- Fixed an `sm_120` compiler crash when an `f16` constant was converted to an FP8 type with `cuda_tile.ftof` and the result was passed to `cuda_tile.print_tko`.

- **Known Issues**
  - 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.

1.2. Resolved Issues

1.2.1. General CUDA

- Fixed CUDA Graphs containing more than approximately 2.3 million kernel nodes that could instantiate successfully but crash, hang, or report an illegal-address error when launched. *[6432056]*
- Fixed updating the dynamic shared-memory size of a cooperative kernel node in an instantiated graph, which could cause a hang or illegal-memory access. *[6090062]*
- Fixed a use-after-free in `cuGraphNodeSetParams` when used with `CU_GRAPH_CHILD_GRAPH_OWNERSHIP_MOVE`. *[6288631]*
- Fixed Xid 32 errors and application crashes when `CUDA_SCALE_LAUNCH_QUEUES` was used with CUDA Graphs containing long chains of kernel nodes. *[5686696]*
- Fixed an HMM/UVM GPU-unregistration issue that could cause a kernel panic or system restart during CUDA or DCGM workloads on Red Hat Enterprise Linux. *[6185077]*
- Fixed a Linux driver deadlock that could leave the GPU unusable in rare situations. *[6095595]*
- Fixed an issue that could cause a GPU to disappear after an in-progress GPU reset was interrupted in a virtual machine. *[5967535]*
- Fixed `libnvidia-ml.a` deadlocking when `nvmlInit_v2()` was invoked from a shared library. *[6174166]*
- Fixed NVML and `nvidia-smi` on Windows Arm64 exposing unsupported TCC-mode controls that could leave the system unstable. *[6225285]*

1.2.2. CUDA Compiler

- Fixed `ptxas` scheduling that could move a predicated `REDUX.SUM` operation above its predicate definition, producing incorrect results on Blackwell GPUs. *[6260886]*
- Fixed a CUDA C++ compiler issue that could produce incorrect results for floating-point comparisons inside a function. *[6235207]*
- Fixed `ptxas` incorrectly dropping `createpolicy.fractional.L2::evict_first`, which could leave a dependent `cp.async` cache-hint descriptor uninitialized and cause an illegal-instruction error on `sm_90`. *[6132797]*
- Fixed incorrect SASS generation for `tensormap.cp_fenceproxy` with `.sys` scope. *[5891184]*
- Fixed missing ordering between `UTMALDG` and `UTMACMDFLUSH` operations that could cause silent data corruption. *[5996051]*
- Fixed an optimization regression introduced in CUDA 13.1 that could cause illegal memory accesses in cuSOLVERDx HTEV and BDSVD operations on `sm_90`. *[5914878]*
- Fixed an `nvcc` 13.3 regression that prevented applications using Abseil `flat_hash_map`, including some ONNX configurations, from compiling. *[6302392]*
- Fixed CUDA C++ frontend handling of hidden friend functions with constrained auto parameters. *[6047481]*
- Fixed a template-substitution issue whose result could change when an otherwise-unused template declaration was present. *[5818596]*
- Fixed an internal compiler assertion triggered by valid CUDA C++ source. *[5933457]*
- Fixed multiple issues found through fuzz testing. *[6194108]*
  - Fixed an issue where the integer constant `0x7fffffff` (`INT_MAX`) reaching a negation or subtraction could produce an incorrect result.
  - Fixed an issue where a `prmt.b32` using a sign-replicating selector, followed by a byte or halfword mask (`and.b32` or `cvt.u32.u16`), could incorrectly produce zero.
  - Fixed cases where extended-precision carry chains (`add.cc`/`addc`, `sub.cc`/`subc`) whose inputs were produced by shifts or multiplies could consume an incorrect carry or borrow, giving a result off by one.
  - Fixed an issue where adding a value back after a saturating subtraction of that same value (`sub.sat.s32`) could be simplified in a way that ignored saturation.
  - Fixed an issue where the bits of a value produced by a multiply or an arithmetic right shift could be incorrectly inferred, causing a subsequent mask or unsigned comparison to fold to an incorrect constant.
  - Fixed cases where a bit-field insert – either `bfi.b32` with pos + len > 32, or one formed by the compiler from a masked shift – could write the field at the wrong bit position or leave part of the field unwritten.
  - Fixed an issue where a shift or funnel shift (`shf.l.wrap.b32`, `shf.r.wrap.b32`, `shr`) whose source was negated or complemented (for example by `neg` or `cnot`) and whose result immediately fed an add could produce an incorrect result.
  - Fixed an issue where using a complemented or negated value inside a branch that pins that value to a known constant could drop the complement or negation.
  - Fixed an issue where two comparisons of the same operands differing only in signedness could be treated as equivalent, causing one of the branches to be dropped.

- Fixed an issue where a min/max clamp pair against zero that mixed signed and unsigned forms could be incorrectly folded to zero.
- Fixed cases where an unsigned comparison against zero of an all-ones boolean (as produced by `set`) or of a negated `abs` value could be folded to the wrong result, taking the wrong branch or `selp` arm.
- Fixed an issue where `slct` with a large constant selector could select the wrong operand.
- Fixed an issue where a `lop3.b32` that reads the same register in more than one input, combined with an adjacent bitwise operation, could produce an incorrect result.
- Fixed an issue where an unrolled loop containing repeated high multiplies (`mul.hi`, `mad.hi.cc`) could have some of those computations incorrectly eliminated.
- Fixed an issue where packed 16-bit operations (`add.u16x2`, `min.u16x2`, `max.u16x2`) with an operand produced by a 16-bit load or by a scalar value could compute an incorrect high lane.
- Fixed an issue where a 16-bit min/max followed by a conversion and an equality test against the second operand could produce the opposite result when the two operands are equal.
- Fixed an issue where a conditional assignment could be incorrectly removed when the register's prior value had the same logical truth value but a different bit pattern.
- Fixed an internal compiler error when compiling a warp-uniform `mul.hi` for Blackwell targets.
- Fixed an internal compiler error when a `cvt.pack.sat` instruction's merge operand became a compile-time constant.
- Fixed an internal compiler error on an `addc`/`madc` whose carry-in was provably zero.
- Fixed an internal compiler error when a kernel reads a `.param` variable with `ld.param` that is never used as a call argument or return value. `ptxas` now reports an error for this construct, which is not defined by the PTX ISA.

1.2.3. CUDA Tools

- Fixed malformed `graphNodeId` values in CUDA Graph memcpy activity records when CUPTI hardware-event sampling was enabled. *[6395204]*
- Fixed CUPTI runtime- and driver-API callback filtering not taking effect when user-defined records were enabled. *[6342060]*
- Fixed host heap corruption when CUPTI hardware tracing and activity collection were enabled together on GB200. *[6274883]*
- Fixed a Compute Sanitizer false positive for `tensormap.cp_fenceproxy`. *[6090556]*
- Fixed a performance regression that made `ncu --import` substantially slower in Nsight Compute 2026.3.0.5. *[6412841]*
- Fixed Nsight Compute failing after profiling with an `unknown encoding: utf-8-sig` error. *[5872516]*
- Fixed a CUDA-GDB regression that prevented standard C++ pretty-printers from working. *[5503986]*

1.3. Known Issues

1.3.1. CUDA Platform

- Applications statically built with, or dynamically linking to, an older CUDA Runtime with the R615 driver may report benign public error messages indicating that IMEX channels are incorrectly set up. These can be ignored if fabric handle support is not required.
- Device attribute queries for `CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_FABRIC_SUPPORTED`, `CU_DEVICE_ATTRIBUTE_MEMPOOL_SUPPORTED_HANDLE_TYPES`, `CU_DEVICE_ATTRIBUTE_HOST_NUMA_MULTIN` and `CU_DEVICE_ATTRIBUTE_LOGICAL_ENDPOINT_SUPPORTED_HANDLE_TYPES` may report that fabric handles are unsupported if IMEX channels are incorrectly configured. If fabric usage reports as unsupported, check the public CUDA error logs to verify whether IMEX setup was the cause.

1.3.2. CUDA Compiler

- There is an open issue in the PTX compiler for the `tcgen05.alloc` instruction when used with `-g` or `-g-tmem-access-check`. This manifests when `tcgen05.alloc` is used with the `.exclusive` qualifier and `nCols` is explicitly set to 576. Mitigation: use a value smaller than 576 for `nCols` with `.exclusive`, or avoid using `-g` or `-g-tmem-access-check` with `tcgen05.alloc`.

1.4. Deprecated or Dropped Features

1.4.1. CUDA Platform

- Legacy Nsight Eclipse Edition plugins are no longer delivered in CUDA Toolkit packages beginning with CUDA 13.3.
- Python 3.10 support is deprecated across the CUDA Python 13.4 packages.

1.4.2. Architectures

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

Chapter 2. CUDA 13.3

2.1. Features

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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.

2.1.5. 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.

2.1.6. CUDA Tile

Supported Architectures

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

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.

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.

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.

Documentation Improvements

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

Fixed Issues

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

Known Issues

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

2.2. Resolved Issues

2.2.1. 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.
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.2.2. 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.

  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.

2.3. Known Issues

2.3.1. 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.

2.3.2. 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.

2.4. Deprecated or Dropped Features

2.4.1. General CUDA

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

Chapter 3. CUDA 13.2

3.1. Features

3.1.1. General CUDA

- Host tasks now support a new spin-wait dispatch mode, which can reduce execution latency, in addition to the existing mode that blocks on an interrupt from the GPU.

  This feature is available via `cudaLaunchHostFunc()` and graph host nodes created with `cudaGraphAddNode()`. For more details, see the corresponding Driver API and Runtime API reference documentation.

3.1.2. CUDA Compiler

- For new features from PTX, refer to PTX ISA version 9.2.
- Added support for new host compilers, including Microsoft Visual Studio 2026.
- Added support for ACLE extensions when using GCC.
- Delivered a single unified CUDA Toolkit for Tegra and desktop GPUs, reducing overhead for containers and libraries.

3.1.3. 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.
- CUDA Programming Guide.

3.2. Resolved Issues

3.2.1. CUDA Compiler

- Improved C++20 standards conformance in `nvcc`, including fixes for constraints, requires-expressions, lambdas, and `noexcept` specifications in templated and inherited code.
- Resolved multiple compiler correctness issues related to the C++20 `[[no_unique_address]]` attribute, including internal compiler errors, incorrect structure layout, and runtime miscompilations in device code.
- Improved `nvcc` host compilation support on `aarch64` systems, including fixes for ARM Neon intrinsics when using newer GCC versions.

3.3. Known Issues

3.3.1. General CUDA

- On SLES 16 systems used for NVLink 5 testing, NVIDIA Fabric Manager may fail to start when DOCA OFED is installed.

  Symptom:

  NVIDIA Fabric Manager fails to start on SLES 16 systems used for NVLink 5 testing when DOCA OFED is installed.

  Problem Description:

  This issue can occur because the `ib_core` module is provided by both the SLES 16 kernel and DOCA OFED, which can lead to a module conflict.

  Workaround:

  Unload the SLES 16 kernel-provided `ib_core` module and load the DOCA OFED-provided `ib_core` module instead. After loading the DOCA OFED-provided module, NVIDIA Fabric Manager starts and the system operates correctly.
- CUDA initialization can fail on certain Linux kernels with KASLR enabled.

  Symptom:

  CUDA initialization fails. This issue is indicated by the following debug message:

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

  Problem Description:

  Certain Linux kernels have a known issue in HMM initialization when KASLR is enabled.

  Workaround:

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

- Option 1: Disable KASLR (preferred option)

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

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

  Then update GRUB and reboot:

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

- Option 2: Disable HMM for UVM

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

     ```
     options nvidia_uvm uvm_disable_hmm=1
     ```

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

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

3.3.2. CUDA Compiler

- For semantic clarifications, refer to Changes in PTX ISA version 9.2.

3.4. Deprecated or Dropped Features

3.4.1. General CUDA

- CUDA 13.0 deprecates the following legacy vector types:
  - `double4`
  - `long4`
  - `ulong4`
  - `longlong4`
  - `ulonglong4`

  These types are being replaced by new aligned variants:
  - `*_16a` and `*_32a` (e.g., `double4_16a`, `double4_32a`)

  Deprecation warnings can be managed as follows:
  - Globally silenced by defining `__NV_NO_VECTOR_DEPRECATION_DIAG`
  - Locally suppressed using the macro pair `__NV_SILENCE_HOST_DEPRECATION_BEGIN` / `__NV_SILENCE_HOST_DEPRECATION_END`

These legacy types are planned for removal in CUDA 14.0.

- The **CUDA installer** for Windows **no longer** bundles the display driver. Users must install the display driver separately, either before or after installing the CUDA Toolkit.
- **Multi-device launch APIs** and related references for Cooperative Groups **have been removed**. These APIs were previously marked as deprecated in CUDA 12.x:
  - `cudaLaunchCooperativeKernelMultiDevice` has been removed from `cuda_runtime_api.h`.
  - The accompanying parameter struct `cudaLaunchParam` has been removed from `driver_types.h`.
  - `this_multi_grid` and `multi_grid_group` have been removed from `cooperative_groups.h`.
- **Changes to cudaDeviceProperties structure**:

  In CUDA 13.0, several deprecated fields have been removed from the `cudaDeviceProperties` structure. To ensure forward compatibility, use the recommended replacement APIs listed below:

  Removed Fields and Their Replacements

Removed Field	Replacement API
clockRate	`cudaDeviceGetAttribute(cudaDevAttrClockRate)`
deviceOverlap	Use the `asyncEngineCount` field
kernelExecTimeoutEnabled	`cudaDeviceGetAttribute(cudaDevAttrKernelExecTimeout)`
computeMode	`cudaDeviceGetAttribute(cudaDevAttrComputeMode)`
maxTexture1DLinear	`cudaDeviceGetTexture1DLinearMaxWidth()`
memoryClockRate	`cudaDeviceGetAttribute(cudaDevAttrMemoryClockRate)`
singleToDoublePrecisionPerfRatio	`cudaDeviceGetAttribute(cudaDevAttrSingleToDoublePrecisionPerfRat`
cooperativeMultiDeviceLaunch	*No replacement available*

  Removed cudaDeviceAttr Types (No Replacement Available)

  - `cudaDevAttrCooperativeMultiDeviceLaunch`
  - `cudaDevAttrMaxTimelineSemaphoreInteropSupported`
- The following legacy header files related to deprecated texture and surface references have been removed from the CUDA 13.0 runtime:
  - cuda_surface_types.h
  - cuda_texture_types.h
  - surface_functions.h
  - texture_fetch_functions.h

3.4.2. Deprecated Architectures

- Architecture support for Maxwell, Pascal, and Volta is considered feature-complete. Offline compilation and library support for these architectures have been removed in CUDA Toolkit 13.0 major version release. The use of CUDA Toolkits through the 12.x series to build applications for these architectures will continue to be supported, but newer toolkits will be unable to target these architectures.

3.4.3. Deprecated or Dropped Operating Systems

- Support for Ubuntu 20.04 has been dropped starting with this release. Users are advised to migrate to Ubuntu 22.04 LTS or later.

3.4.4. Deprecated or Dropped CUDA Toolchains

CUDA Tools

- As of CUDA 13.1, support for Nsight Eclipse Edition plugins is deprecated, and will be dropped in a future CUDA release.

Chapter 4. CUDA 13.1

4.1. Features

4.1.1. General CUDA

- No new features were introduced in CUDA 13.1 Update 1. For features introduced in CUDA 13.1, see the CUDA 13.1 release notes.

4.1.2. 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.
- CUDA Programming Guide.

4.2. Resolved Issues

4.2.1. CUDA Compiler

- Fixed an issue where CUDA 13.0–13.1 could generate illegal instructions when compiling a kernel with FFMA2 enabled.
- Fixed an issue where cuFFTDx FFT computations could produce incorrect results when built with NVCC 12.8 or NVCC 13.
- Fixed a bug where NVCC 13.0 could silently corrupt data in CuTe GEMM on GB202 GPUs.

4.3. Known Issues

4.3.1. General CUDA

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

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

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

  - Option 1: Disable KASLR (Preferred option)

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

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

    Then, update GRUB and reboot:

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

  - Option 2: Disable HMM for UVM

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

```
options nvidia_uvm uvm_disable_hmm=1
```

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

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

4.3.2. CUDA Compiler

- The Tile-IR AS compiler currently supports only Blackwell-class devices and has limited low-precision support. These limitations will be removed in a future release.

4.4. Deprecated or Dropped Features

4.4.1. General CUDA

- CUDA 13.0 deprecates the following legacy vector types:
  - `double4`
  - `long4`
  - `ulong4`
  - `longlong4`
  - `ulonglong4`

  These types are being replaced by new aligned variants:
  - `*_16a` and `*_32a` (e.g., `double4_16a`, `double4_32a`)

  Deprecation warnings can be managed as follows:
  - Globally silenced by defining `__NV_NO_VECTOR_DEPRECATION_DIAG`
  - Locally suppressed using the macro pair `__NV_SILENCE_HOST_DEPRECATION_BEGIN` / `__NV_SILENCE_HOST_DEPRECATION_END`

  These legacy types are planned for removal in CUDA 14.0.
- The **CUDA installer** for Windows **no longer** bundles the display driver. Users must install the display driver separately, either before or after installing the CUDA Toolkit.
- **Multi-device launch APIs** and related references for Cooperative Groups **have been removed**. These APIs were previously marked as deprecated in CUDA 12.x:
  - `cudaLaunchCooperativeKernelMultiDevice` has been removed from `cuda_runtime_api.h`.
  - The accompanying parameter struct `cudaLaunchParam` has been removed from `driver_types.h`.
  - `this_multi_grid` and `multi_grid_group` have been removed from `cooperative_groups.h`.
- **Changes to cudaDeviceProperties structure**:

  In CUDA 13.0, several deprecated fields have been removed from the `cudaDeviceProperties` structure. To ensure forward compatibility, use the recommended replacement APIs listed below:

  Removed Fields and Their Replacements

Removed Field	Replacement API
clockRate	`cudaDeviceGetAttribute(cudaDevAttrClockRate)`
deviceOverlap	Use the `asyncEngineCount` field
kernelExecTimeoutEnabled	`cudaDeviceGetAttribute(cudaDevAttrKernelExecTimeout)`
computeMode	`cudaDeviceGetAttribute(cudaDevAttrComputeMode)`
maxTexture1DLinear	`cudaDeviceGetTexture1DLinearMaxWidth()`
memoryClockRate	`cudaDeviceGetAttribute(cudaDevAttrMemoryClockRate)`
singleToDoublePrecisionPerfRatio	`cudaDeviceGetAttribute(cudaDevAttrSingleToDoublePrecisionPerfRat`
cooperativeMultiDeviceLaunch	*No replacement available*

Removed cudaDeviceAttr Types (No Replacement Available)

- `cudaDevAttrCooperativeMultiDeviceLaunch`
- `cudaDevAttrMaxTimelineSemaphoreInteropSupported`

- The following legacy header files related to deprecated texture and surface references have been removed from the CUDA 13.0 runtime:
  - cuda_surface_types.h
  - cuda_texture_types.h
  - surface_functions.h
  - texture_fetch_functions.h

4.4.2. Deprecated Architectures

- Architecture support for Maxwell, Pascal, and Volta is considered feature-complete. Offline compilation and library support for these architectures have been removed in CUDA Toolkit 13.0 major version release. The use of CUDA Toolkits through the 12.x series to build applications for these architectures will continue to be supported, but newer toolkits will be unable to target these architectures.

4.4.3. Deprecated or Dropped Operating Systems

- Support for Ubuntu 20.04 has been dropped starting with this release. Users are advised to migrate to Ubuntu 22.04 LTS or later.

4.4.4. Deprecated or Dropped CUDA Toolchains

CUDA Tools

- As of CUDA 13.1, support for Nsight Eclipse Edition plugins is deprecated, and will be dropped in a future CUDA release.

Chapter 5. CUDA 13.0

5.1. Features

5.1.1. General CUDA

- 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.

5.1.2. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-9-0.

5.1.3. 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.
- CUDA Programming Guide.

5.2. Known Issues

5.2.1. General CUDA

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

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

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

  - Option 1: Disable KASLR (Preferred option)

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

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

    Then, update GRUB and reboot:

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

  - Option 2: Disable HMM for UVM

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

```
options nvidia_uvm uvm_disable_hmm=1
```

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

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

5.3. Deprecated or Dropped Features

5.3.1. General CUDA

- CUDA 13.0 deprecates the following legacy vector types:
  - `double4`
  - `long4`
  - `ulong4`
  - `longlong4`
  - `ulonglong4`

These types are being replaced by new aligned variants:

- `*_16a` and `*_32a` (e.g., `double4_16a`, `double4_32a`)

Deprecation warnings can be managed as follows:

- Globally silenced by defining `__NV_NO_VECTOR_DEPRECATION_DIAG`
- Locally suppressed using the macro pair `__NV_SILENCE_HOST_DEPRECATION_BEGIN` / `__NV_SILENCE_HOST_DEPRECATION_END`

These legacy types are planned for removal in CUDA 14.0.

- The **CUDA installer** for Windows **no longer** bundles the display driver. Users must install the display driver separately, either before or after installing the CUDA Toolkit.
- **Multi-device launch APIs** and related references for Cooperative Groups **have been removed**. These APIs were previously marked as deprecated in CUDA 12.x:
  - `cudaLaunchCooperativeKernelMultiDevice` has been removed from `cuda_runtime_api.h`.
  - The accompanying parameter struct `cudaLaunchParam` has been removed from `driver_types.h`.
  - `this_multi_grid` and `multi_grid_group` have been removed from `cooperative_groups.h`.
- **Changes to cudaDeviceProperties structure**:

  In CUDA 13.0, several deprecated fields have been removed from the `cudaDeviceProperties` structure. To ensure forward compatibility, use the recommended replacement APIs listed below:

  Removed Fields and Their Replacements

Removed Field	Replacement API
clockRate	`cudaDeviceGetAttribute(cudaDevAttrClockRate)`
deviceOverlap	Use the `asyncEngineCount` field
kernelExecTimeoutEnabled	`cudaDeviceGetAttribute(cudaDevAttrKernelExecTimeout)`
computeMode	`cudaDeviceGetAttribute(cudaDevAttrComputeMode)`
maxTexture1DLinear	`cudaDeviceGetTexture1DLinearMaxWidth()`
memoryClockRate	`cudaDeviceGetAttribute(cudaDevAttrMemoryClockRate)`
singleToDoublePrecisionPerfRatio	`cudaDeviceGetAttribute(cudaDevAttrSingleToDoublePrecisionPerfRat`
cooperativeMultiDeviceLaunch	*No replacement available*

  Removed cudaDeviceAttr Types (No Replacement Available)

  - `cudaDevAttrCooperativeMultiDeviceLaunch`
  - `cudaDevAttrMaxTimelineSemaphoreInteropSupported`
- The following legacy header files related to deprecated texture and surface references have been removed from the CUDA 13.0 runtime:

- cuda_surface_types.h
- cuda_texture_types.h
- surface_functions.h
- texture_fetch_functions.h

5.3.2. Deprecated Architectures

- Architecture support for Maxwell, Pascal, and Volta is considered feature-complete. Offline compilation and library support for these architectures have been removed in CUDA Toolkit 13.0 major version release. The use of CUDA Toolkits through the 12.x series to build applications for these architectures will continue to be supported, but newer toolkits will be unable to target these architectures.

5.3.3. Deprecated or Dropped Operating Systems

- Support for Ubuntu 20.04 has been dropped starting with this release. Users are advised to migrate to Ubuntu 22.04 LTS or later.

5.3.4. Deprecated or Dropped CUDA Toolchains

CUDA Tools

- The profiling tools **NVIDIA Visual Profiler** and **nvprof** have been removed in CUDA 13.0. Users are encouraged to migrate to the next-generation tools:
  - NVIDIA Nsight Systems for GPU and CPU sampling and tracing
  - NVIDIA Nsight Compute for detailed GPU kernel profiling
- The pre-built CUDA Demo Suite applications have been removed in CUDA 13.0. Users are encouraged to build CUDA Samples from source, available on GitHub.
- The `nv-p2p` APIs are deprecated starting with CUDA 13.0 and will be removed in CUDA 14.0. These APIs remain functional in CUDA 13.0 but will no longer receive enhancements or bug fixes.
- The following CUPTI APIs and features have been removed in CUDA 13.0:
  - CUPTI Event API (`cupti_events.h`)
  - CUPTI Metric API (`cupti_metrics.h`)
  - PC Sampling Activity API (`cupti_activity.h`)
  - Source/SASS-level metrics (`cupti_activity.h`)
- The following APIs have been deprecated in CUDA 13.0:
  - CUPTI Profiling API (`cupti_profiler_target.h`)
  - Perfworks Metric API (`nvperf_host.h`)

  Users are encouraged to transition to the following replacements:
  - CUPTI Range Profiling API (`cupti_range_profiler.h`)

- CUPTI Profiler Host API (`cupti_profiler_host.h`)

CUDA Compiler

- The following host compilers are no longer supported as of CUDA 13.0:
  - Intel ICC 2021.7
  - Microsoft Visual Studio 2017 (MSVC 2017)

Chapter 6. CUDA 12.9

6.1. Features

6.1.1. General CUDA

- MPS client termination is now supported on Tegra platforms (For L4T users - starting JetPack 7.0 only). More details can be found here.
- Extended CUDA in Graphics (CIG) mode now supports Vulkan, expanding beyond the previous DirectX-only implementation.
- CPU NUMA allocation support through `cuMemCreate` and `cuMemAllocAsync` is now available on Windows when using the driver in WDDM and MCDM modes, expanding this previously Linux-only feature.
- CUDA Graphs functionality has been enhanced to support the inclusion of memory nodes in child graphs.
- CUDA Toolkit 12.9 adds compiler target support for SM architecture 10.3 (`sm_103`, `sm_103f`, `sm_103a`) and 12.1 (`sm_121`), enabling development for the latest GPU architectures with specific optimizations for each variant.
- CUDA Toolkit 12.9 introduces compiler support for a new target architecture class: family-specific architectures. Learn more: NVIDIA Blog: Family-Specific Architecture Features
- Multiple enhancements to NVML and nvidia-smi:
  - Added counters (in microseconds) for the throttling time for the following reasons:
    - `nvmlClocksEventReasonGpuIdle`
    - `nvmlClocksEventReasonApplicationsClocksSetting`
    - `nvmlClocksEventReasonSwPowerCap`
    - `nvmlClocksThrottleReasonHwSlowdown`
    - `nvmlClocksEventReasonSyncBoost`
    - `nvmlClocksEventReasonSwThermalSlowdown`
    - `nvmlClocksThrottleReasonHwThermalSlowdown`
    - `nvmlClocksThrottleReasonHwPowerBrakeSlowdown`
    - `nvmlClocksEventReasonDisplayClockSetting`
  - Improved consistency for device identification between CUDA and NVML

- Added NVML chip-to-chip (C2C) telemetry APIs
- Added CTXSW metrics
- Implemented GPU average power counters
- Added PCIe bind/unbind events

6.1.2. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-8.
- Added a new compiler option `--Ofast-compile=<level>`, supported in nvcc, nvlink, nvrtc, and ptxas. This option prioritizes faster compilation over optimizations at varying levels, helping to accelerate development cycles. Refer to the fast-compile documentation for more details.
- Added a new compiler option `--frandom-seed=<seed>`, supported in nvcc and nvrtc. The user specified random seed will be used to replace random numbers used in generating symbol names and variable names. The option can be used to generate deterministically identical ptx and object files. If the input value is a valid number (decimal, octal, or hex), it will be used directly as the random seed. Otherwise, the CRC value of the passed string will be used instead. NVCC will also pass the option, as well as the user specified value to host compilers, if the host compiler is either GCC or Clang, since they support -frandom-seed option as well. Users are responsible for assigning different seeds to different files.

Chapter 7. CUDA 12.8

7.1. Features

7.1.1. General CUDA

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

- Added PCIe device ID to CUDA device properties
- Added cudaStreamGetDevice and cuStreamGetDevice APIs to retrieve the device associated with a CUDA stream
- Added CUDA support for INT101010 texture/surface format
- Added batch CUDA asynchronous memory copy APIs (cuMemcpyBatchAsync and cuMemcpyBatch3DAsync) for variable-sized transfers between multiple source and destination buffers

- Userspace Checkpoint and Restore:
  - Added new driver API for checkpoint/restore operations

7.1.2. CUDA Compiler

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

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

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

  `Nvdisasm` now supports emitting JSON formatted SASS disassembly.

Chapter 8. CUDA 12.6

8.1. Features

8.1.1. General CUDA

- The default Linux driver installation changes in this release, preferring NVIDIA GPU Open Kernel Modules to proprietary drivers. The open source drivers are now the default and recommended installation option.

  Important: The GPU Open Kernel Modules drivers are only compatible with Turing and newer GPUs. If your GPU is from an older family (Maxwell, Pascal, or Volta) you must continue to use the proprietary drivers.

  For additional information, refer to this blog post: https://developer.nvidia.com/blog/nvidia-transitions-fully-towards-open-source-gpu-kernel-modules/.

  And, for full details, the CUDA Installation Guide for Linux: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

- New `nvidia-open` meta-packages are available to improve driver installation of NVIDIA Open GPU kernel modules. [*4752203*]

8.1.2. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-5.
- Latest host compiler Clang-18 support.
- Support for Stack Canaries in device code. CUDA compilers can now insert stack canaries in device code. The NVCC flag `--device-stack-protector=true` enables this feature. Stack canaries make it more difficult to exploit certain types of memory safety bugs involving stack-local variables. The compiler uses heuristics to assess the risk of such a bug in each function. Only those functions which are deemed high-risk make use of a stack canary.
- Added a new compiler option `-forward-slash-prefix-opts` (Windows only).

  If this flag is specified, and forwarding unknown options to host toolchain is enabled (`-forward-unknown-opts` or `-forward-unknown-to-host-linker` or `-forward-unknown-to-host-compiler`), then a command line argument beginning with '/' is forwarded to the host toolchain. For example:

```
nvcc -forward-slash-prefix-opts -forward-unknown-opts /T foo.cu
```

will forward the flag `/T` to the host compiler and linker. When this flag is not specified, a command line argument beginning with `/` is treated as an input file. For example, `nvcc /T foo.cu` will treat `/T` as an input file, and the Windows API function `GetFullPathName()` is used to determine the full path name.

Note: This flag is only supported on Windows.

For more details, refer to `nvcc-help`.

- An environment variable `NVCC_CCBIN` is introduced for NVCC: Users can set `NVCC_CCBIN` to specify the host compiler, but it has lower priority than command-line option `-ccbin`. If `NVCC_CCBIN` and `-ccbin` are both set, NVCC uses the host compiler specified by `-ccbin`.

Chapter 9. CUDA 12.5

9.1. Features

9.1.1. General CUDA

- In an upcoming CUDA release the NVIDIA Open GPU kernel module flavor will be the default and recommended installation option. End-users with Maxwell, Pascal, or Volta GPUs may need to take action to install the NVIDIA proprietary kernel modules.
- MPS (Multi-process service) is now supported on L4T and embedded-Linux Tegra platforms. More details can be found here.

Chapter 10. CUDA 12.4

10.1. Features

10.1.1. General CUDA

- Green contexts are a lightweight alternative to traditional contexts, with the ability to pass in a set of resources that they should be initialized with. This allows the developer to represent distinct spatial partitions of the GPU, provision resources for them, and target them via the same programming model that CUDA exposes (streams, kernel launches, etc.). For detail, refer to https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__GREEN__CONTEXTS.html.
- Access-counter-based memory migration for Grace Hopper systems is now enabled by default. As this is the first release with the capability enabled, developers may find that applications that had been optimized for earlier memory migration algorithms may see a performance regression if optimized for the earlier behaviors. Should this occur, we introduce a supported but temporary flag to opt out of this behavior. You can control the enablement of this feature by unloading and reloading the NVIDIA UVM driver, as follows:

```
# modprobe -r nvidia_uvm

# modprobe nvidia_uvm uvm_perf_access_counter_mimc_migration_enable=0
```

- This release introduces support for the following new features in CUDA graphs:
  - Graph conditional nodes (enhanced from 12.3)
  - Device-side node parameter update for device graphs
  - Updatable graph node priorities without recompilation
- Enhanced monitoring capabilities through NVML and nvidia-smi:
  - NVJPG and NVOFA utilization percentage
  - PCIe class and subclass reporting
  - dmon reports are now available in CSV format
  - More descriptive error codes returned from NVML
  - dmon now reports gpm-metrics for MIG (that is, `nvidia-smi dmon --gpm-metrics` runs in MIG mode)
  - NVML running against older drivers will report `FUNCTION_NOT_FOUND` in some cases, failing gracefully if NVML is newer than the driver

- NVML APIs to query protected memory information for Hopper Confidential Computing

- This release introduces nvFatbin, a new library to create CUDA fat binary files at runtime. For more details, please visit https://docs.nvidia.com/cuda/nvfatbin/index.html.

10.1.2. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-4.
- Added the `__maxnreg__` kernel function qualifier to allow users to directly specify the maximum number of registers to be allocated to a single thread in a thread block in CUDA C++.
- Added a new flag `-fdevice-syntax-only` that ends device compilation after front-end syntax checking. This option can provide rapid feedback (warnings and errors) of source code changes as it will not invoke the optimizer. Note: this option will not generate valid object code.
- Add a new flag `-minimal` for NVRTC compilation. The `-minimal` flag omits certain language features to reduce compile time for small programs. In particular, the following are omitted:
  - Texture and surface functions and associated types (for example, `cudaTextureObject_t`).
  - CUDA Runtime Functions that are provided by the cudadevrt device code library, typically named with prefix "cuda", for example, `cudaMalloc`.
  - Kernel launch from device code.
  - Types and macros associated with CUDA Runtime and Driver APIs, provided by `cuda/tools/cudart/driver_types.h`, typically named with the prefix "cuda" for example, `cudaError_t`.
- Starting in CUDA 12.4, PTXAS enables position independent code (`-pic`) as default when the compilation mode is whole program compilation. Users can opt out by specifying the `-pic=false` option to PTXAS. Debug compilation and separate compilation continue to have position independent code disabled by default. In future, position independent code will allow the CUDA Driver to share a single copy of text section across contexts and reduce resident memory usage.

Chapter 11. CUDA 12.3

11.1. Features

11.1.1. General CUDA

- CUDA User Mode Driver, CUDA Runtime libraries and CUBLAS now come with obfuscated symbol names and with frame pointers enabled.
- Frame Pointers are enabled for other libraries in the CUDA Toolkit: NVIDIA Management Library, CUDA Profiling Tools Interface, cuBLAS, Compiler libraries – NVRTC, PTXJIT compiler, nvJitLink, and libnvvm.
  - Allows better runtime visibility and traceability, and allows easier exchange of runtime information with NVIDIA when needed for debugging purposes.
  - See https://developer.nvidia.com/blog/cuda-toolkit-symbol-server/ for information on how to use obfuscated symbols.
  - Symbol server address is: https://cudatoolkit-symbols.nvidia.com/.
- Lazy loading default enablement for Windows:
  - Brings the significant memory savings and load-time reductions of lazy loading to Windows by default. Additionally, makes the behavior equivalent between Linux and Windows.
- Single-step CUDA uninstall for Windows:
  - It is no longer necessary to uninstall multiple components of the CUDA Toolkit individually to upgrade or uninstall CUDA. This can now be done in a single step.
- CUDA Graphs:
  - Graph edge data, allowing modified dependencies between nodes. Programmatic Dependent Launch may now be described natively in CUDA Graphs.
- Launch completion events:
  - Allows a dependency on scheduling, but not completion, of all blocks in a kernel, enabling tighter control of scheduling.
- MPS:
  - Added a CUDA API to query whether or not MPS is running.
- Added a driver API to return the name of a kernel function.
- Added an API to libnvJitLink to return the nvJitLink version.

- Added support for reading kernel parameters in device functions.
- Enable querying the return type of __device__ lambdas with trailing return type. Fixes uncommon failures when using device-side lambdas.
- NVML / nvidia-smi:
  - Metric for front-end context switch utilization (FECS)
  - Added metrics for Ada Lovelace AV1 codec utilization
  - Support GPU monitoring on Tegra
  - Added an NVML API to expose H100 PCIe counters and corresponding PCIe section in nvidia-smi

11.1.2. CUDA Compiler

- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-3.
- Enhanced thread support when using the libNVVM API. Clients can take advantage of improved compilation speeds by spawning multiple compilation threads concurrently.
- Improved compile time in some common scenarios:
  - Extended split compilation to cubin for LTO.
  - Turned on concurrent NVVM processing by default, with documented fallback to serialized compilation.
  - Reduced NVRTC compile time for small programs via moving CUDA C++ builtin function declarations into compiler.
  - Moved `cuda_fp16.h` and `cuda_bf16.h` into compiler bitcode.
- Added new keyword ``__inline_hint__`` to specify device functions in a different `.cu` file to be inlined during LTO.
- Enabled querying return type of __device__ lambdas with trailing return type.
- Provided information about unused bytes to compute-sanitizer for better diagnostics.

Chapter 12. CUDA 12.2

12.1. Features

12.1.1. General CUDA

- This release introduces Heterogeneous Memory Management (HMM), allowing seamless sharing of data between host memory and accelerator devices. HMM is supported on Linux only and requires a recent kernel (6.1.24+ or 6.2.11+).

  HMM requires the use of NVIDIA's GPU Open Kernel Modules driver.

  As this is the first release of HMM, some limitations exist:

  - GPU atomic operations on file-backed memory are not yet supported.
  - Arm CPUs are not yet supported.
  - HugeTLBfs pages are not yet supported on HMM (this is an uncommon scenario).
  - The `fork()` system call is not fully supported yet when attempting to share GPU-accessible memory between parent and child processes.
  - HMM is not yet fully optimized, and may perform slower than programs using `cudaMalloc()`, `cudaMallocManaged()`, or other existing CUDA memory management APIs. The performance of programs not using HMM will not be affected.
- The Lazy Loading feature (introduced in CUDA 11.7) is now enabled by default on Linux with the 535 driver. To disable this feature on Linux, set the environment variable `CUDA_MODULE_LOADING=EAGER` before launch. Default enablement for Windows will happen in a future CUDA driver release. To enable this feature on Windows, set the environment variable `CUDA_MODULE_LOADING=LAZY` before launch.
- Host NUMA memory allocation: Allocate a CPU memory targeting a specific NUMA node using either the CUDA virtual memory management APIs or the CUDA stream-ordered memory allocator. Applications must ensure device accesses to pointer backed by HOST allocations from these APIs are performed only after they have explicitly requested accessibility for the memory on the accessing device. It is undefined behavior to access these host allocations from a device without accessibility for the address range, regardless of whether the device supports pageable memory access or not.
- Added per-client priority mapping at runtime for CUDA Multi-Process Service (MPS). This allows multiple processes running under MPS to arbitrate priority at a coarse-grained level between multiple processes without changing the application code.

We introduce a new environment variable CUDA_MPS_CLIENT_PRIORITY, which accepts two values: NORMAL priority, 0, and BELOW_NORMAL priority, 1.

For example, given two clients, a potential configuration is as follows:

`// Client 1's Environment`
`export CUDA_MPS_CLIENT_PRIORITY=0`
`// NORMAL`
	`// Client 2's Environment`
	`export CUDA_MPS_CLIENT_PRIORITY=1`
	`// BELOW NORMAL`

12.1.2. CUDA Compiler

- libNVVM samples have been moved out of the toolkit and made publicly available on GitHub as part of the NVIDIA/cuda-samples project. Similarly, the nvvmir-samples have been moved from the nvidia-compiler-sdk project on GitHub to the new location of the libNVVM samples in the NVIDIA/cuda-samples project.
- For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-2.

Chapter 13. CUDA 12.1

13.1. Features

13.1.1. General CUDA

- New meta-packages for Linux installation.
  - `cuda-toolkit`
    - Installs all CUDA Toolkit packages required to develop CUDA applications.
    - Handles upgrading to the latest version of CUDA when it's released.
    - Does not include the driver.
  - `cuda-toolkit-12`
    - Installs all CUDA Toolkit packages required to develop CUDA applications.
    - Handles upgrading to the next 12.x version of CUDA when it's released.
    - Does not include the driver.
- New CUDA API to enable mini core dump programmatically is now available. Refer to https://docs.nvidia.com/cuda/cuda-gdb/index.html#gpu-core-dump-support and https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__COREDUMP.html#group__CUDA__COREDUMP for more information.

13.1.2. CUDA Compiler

- NVCC has added support for host compiler: GCC 12.2, NVC++ 22.11, Clang 15.0, VS2022 17.4
- Breakpoint and single stepping behavior for a multi-line statement in device code has been improved, when code is compiled with nvcc using gcc/clang host compiler or when compiled with NVRTC on non-Windows platforms. The debugger will now correctly breakpoint and single-step on each source line of the multiline source code statement.
- PTX has exposed a new special register in the public ISA, which can be used to query total size of shared memory which includes user shared memory and SW reserved shared memory.
- NVCC and NVRTC now show preprocessed source line and column info in a diagnostic to help users to understand the message and identify the issue causing the diagnostic. The source line and column info can be turned off with `--brief-diagnostics=true`.

Chapter 14. CUDA 12.0

14.1. Features

14.1.1. General CUDA

- CUDA 12.0 exposes programmable functionality for many features of the Hopper and Ada Lovelace architectures:
  - Many tensor operations now available via public PTX:
    - TMA operations
    - TMA bulk operations
    - 32x Ultra xMMA (including FP8/FP16)
  - Membar domains in Hopper, controlled via launch parameters
  - Support Hopper asynchronous transaction barrier in C++ and PTX
  - Introduced C intrinsics for Cooperative Grid Array (CGA) relaxed barrier support
  - Programmatic L2 Cache to SM multicast (Hopper-only)
  - Public PTX for SIMT collectives - elect_one
  - Genomics/DPX instructions now available for Hopper GPUs to provide faster combined-math arithmetic operations (three-way max, fused add+max, etc.)
- Enhancements to the CUDA graphs API:
  - You can now schedule graph launches from GPU device-side kernels by calling built-in functions. With this ability, user code in kernels can dynamically schedule graph launches, greatly increasing the flexibility of CUDA graphs.
  - The `cudaGraphInstantiate()` API has been refactored to remove unused parameters.
- Added the ability to use virtual memory management (VMM) APIs such as `cuMemCreate()` with GPUs masked by `CUDA_VISIBLE_DEVICES`.
- Application and library developers can now programmatically update the priority of CUDA streams.
- CUDA 12.0 adds support for revamped CUDA Dynamic Parallelism APIs, offering substantial performance improvements vs. the legacy CUDA Dynamic Parallelism APIs.
- Added new APIs to obtain unique stream and context IDs from user-provided objects:

- `cuStreamGetId(CUstream hStream, unsigned long long *streamId)`
- `cuCtxGetId(CUcontext ctx, unsigned long long *ctxId)`

- Added support for read-only `cuMemSetAccess()` flag `CU_MEM_ACCESS_FLAGS_PROT_READ`.

14.1.2. CUDA Compiler

- JIT LTO support is now officially part of the CUDA Toolkit through a separate nvJitLink library. A technical deep dive blog will go into more details. Note that the earlier implementation of this feature has been deprecated. Refer to the Deprecation/Dropped Features section below for details.
- New host compiler support:
  - GCC 12.1 (Official) and 12.2.1 (Experimental)
  - VS 2022 17.4 Preview 3 fixes compiler errors mentioning an internal function `std::_Bit_cast` by using CUDA's support for `__builtin_bit_cast`.
- NVCC and NVRTC now support the c++20 dialect. Most of the language features are available in host and device code; some such as coroutines are not supported in device code. Modules are not supported for both host and device code. Host Compiler Minimum Versions: GCC 10, Clang 11, VS2022, Arm C/C++ 22.x. Refer to the individual Host Compiler documentation for other feature limitations. Note that a compilation issue in C++20 mode with <complex> header mentioning an internal function `std::_Bit_cast` is resolved in VS2022 17.4.
- NVRTC default C++ dialect changed from C++14 to C++17. Refer to the ISO C++ standard for reference on the feature set and compatibility between the dialects.
- NVVM IR Update: with CUDA 12.0 we are releasing NVVM IR 2.0 which is incompatible with NVVM IR 1.x accepted by the libNVVM compiler in prior CUDA toolkit releases. Linking of NVVM IR Version 1.11 with 2.0 will result in a compiler error. Users of the libNVVM compiler in CUDA 12.0 toolkit must generate NVVM IR 2.0.

Chapter 15. CUDA 11.8

15.1. Features

15.1.1. General CUDA

- This release introduces support for both the Hopper and Ada Lovelace GPU families.
- Added support for Rocky Linux 9.
- Added support for Kylin OS.
- Package upgradable CUDA is now available starting CUDA 11.8 for Jetson devices. Refer to https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/index.html#upgradable-package-for-jetson for details on how to upgrade to the latest CUDA version on Jetson and the supported JetPack versions.

Chapter 16. CUDA 11.7

16.1. Features

16.1.1. General CUDA

- To best ensure the security and reliability of our RPM and Debian package repositories, NVIDIA is updating and rotating the signing keys used by apt, dnf/yum, and zypper package managers beginning April 27, 2022. Failure to update your repository signing keys will result in package management errors when attempting to access or install packages from CUDA repositories. To ensure continued access to the latest NVIDIA software, please follow the instructions here: https://developer.nvidia.com/blog/updating-the-cuda-linux-gpg-repository-key/.
- NVIDIA Open GPU Kernel Modules: With CUDA 11.7 and R515 driver, NVIDIA is open sourcing the GPU kernel mode driver under dual GPL/MIT license. Refer to https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#open-gpu-kernel-modules for more information.
- Lazy Loading: Delay kernel loading from host to GPU to the point where the kernel is called. This also only loads used kernels, which may result in a significant device-side memory savings. This also defers load latency from the beginning of the application to the point where a kernel is first called—overall binary load latency is usually significantly reduced, but is also shifted to later points in the application. To enable this feature, set the environment variable CUDA_MODULE_LOADING=LAZY before launching your process. Note that this feature is only compatible with libraries compiled with CUDA versions >= 11.7.

16.1.2. CUDA Compiler

- Grid private constants
- NVCC host compiler support for clang13

Chapter 17. CUDA 11.6

17.1. Features

17.1.1. CUDA Compiler

- VS2022 Support: CUDA 11.6 officially supports the latest VS2022 as host compiler. A separate Nsight Visual Studio installer 2022.1.1 must be downloaded from here. A future CUDA release will have the Nsight Visual Studio installer with VS2022 support integrated into it.
- New instructions in public PTX: New instructions for bit mask creation—BMSK, and sign extension—SZEXT, are added to the public PTX ISA. You can find documentation for these instructions in the PTX ISA guide: BMSK and SZEXT.
- Unused Kernel Optimization: In CUDA 11.5, unused kernel pruning was introduced with the potential benefits of reducing binary size and improving performance through more efficient optimizations. This was an opt-in feature but in 11.6, this feature is enabled by default. As mentioned in the 11.5 blog, there is an opt-out flag that can be used in case it becomes necessary for debug purposes or for other special situations.

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

- New -arch=native option: In addition to the `-arch=all` and `-arch=all-major` options added in CUDA 11.5, NVCC introduced `-arch= native` in CUDA 11.5 update 1. This -arch=native option is a convenient way for users to let NVCC determine the right target architecture to compile the CUDA device code to based on the GPU installed on the system. This can be particularly helpful for testing when applications are run on the same system they are compiled in.
- Generate PTX from nvlink: Using the following command line, device linker, nvlink will produce PTX as an output in addition to CUBIN:

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

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

- With the option for nvlink that performs LTO to generate the output in PTX, customer applications that require forward compatibility across GPU architectures can span across multiple files and can also take advantage of Link Time Optimization.
- Bullseye support: NVCC compiled source code now works with the code coverage tool Bullseye. The code coverage is only for the CPU or the host functions. Code coverage for device function is not supported through bullseye.
- INT128 developer tool support: In 11.5, CUDA C++ support for 128 bit was added. In 11.6, developer tools support the datatype as well. With the latest version of libcu++, int128 data type is supported by math functions.

Chapter 18. Notices

18.1. 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.

18.2. OpenCL

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

18.3. 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