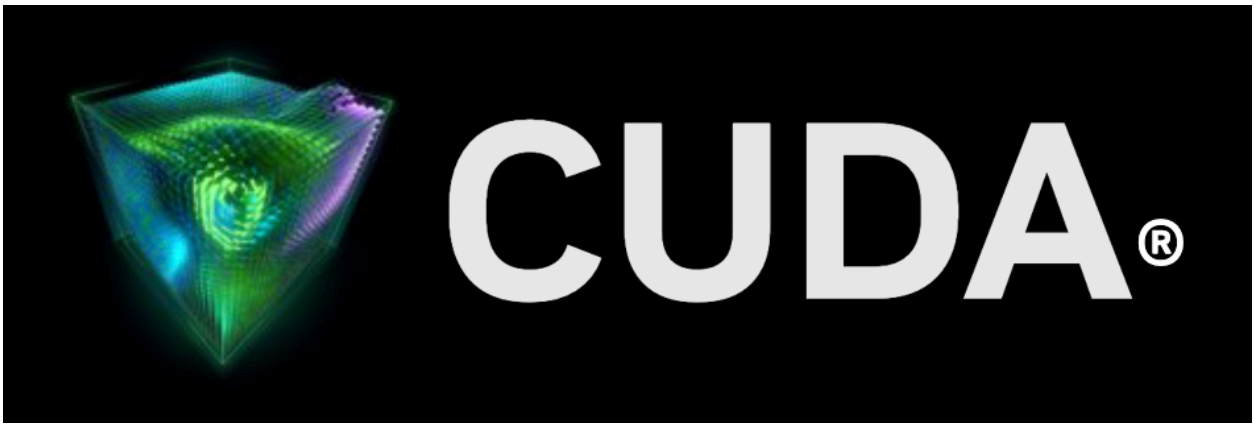

CUDA Features Archive

Release 13.2

NVIDIA Corporation

Mar 05, 2026

Contents

Chapter 1. Overview

CUDA Features Archive

The list of CUDA features by release.

Chapter 2. CUDA 12.9 Features

2.1. 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

2.2. 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 3. CUDA 12.8 Features

3.1. 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

3.2. 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 4. CUDA 12.6 Features

4.1. 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*]

4.2. 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 5. CUDA 12.5 Features

5.1. 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 6. CUDA 12.4 Features

6.1. 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.

6.2. CUDA Compilers

- 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 7. CUDA 12.3 Features

7.1. 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

7.2. CUDA Compilers

- 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 8. CUDA 12.2 Features

8.1. 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`

8.2. CUDA Compilers

- 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 9. CUDA 12.1 Features

9.1. 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.

9.2. CUDA Compilers

- 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 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 10. CUDA 12.0 Features

10.1. 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`.

10.2. CUDA Compilers

- 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 11. CUDA 11.8 Features

11.1. 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 12. CUDA 11.7 Features

12.1. 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.

12.2. CUDA Compilers

- Grid private constants
- NVCC host compiler support for clang13

Chapter 13. CUDA 11.6 Features

13.1. 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++, int 128 data datype is supported by math functions.

Chapter 14. Notices

14.1. 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.

14.2. OpenCL

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

14.3. 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