# Contents

1 CUDA Toolkit Major Component Versions 3

2 New Features 9
   2.1 General CUDA ............................................................ 9
   2.2 CUDA Compilers ........................................................... 9
   2.3 CUDA Developer Tools .................................................. 9

3 Known Issues and Limitations 11

4 Deprecated or Dropped Features 13
   4.1 Deprecated or Dropped Architectures ................................. 13
   4.2 Deprecated Operating Systems ......................................... 13
   4.3 Deprecated Toolchains .................................................. 13
   4.4 CUDA Tools ............................................................... 14

5 CUDA Libraries 15
   5.1 cuBLAS Library ............................................................ 15
      5.1.1 cuBLAS: Release 12.5 .............................................. 15
      5.1.2 cuBLAS: Release 12.4 Update 1 .................................. 15
      5.1.3 cuBLAS: Release 12.4 .............................................. 16
      5.1.4 cuBLAS: Release 12.3 Update 1 .................................. 17
      5.1.5 cuBLAS: Release 12.3 .............................................. 17
      5.1.6 cuBLAS: Release 12.2 Update 2 .................................. 18
      5.1.7 cuBLAS: Release 12.2 .............................................. 18
      5.1.8 cuBLAS: Release 12.1 Update 1 .................................. 18
      5.1.9 cuBLAS: Release 12.0 Update 1 .................................. 19
      5.1.10 cuBLAS: Release 12.0 ............................................. 19
   5.2 cuFFT Library ............................................................. 20
      5.2.1 cuFFT: Release 12.5 ................................................ 20
      5.2.2 cuFFT: Release 12.4 Update 1 ................................... 20
      5.2.3 cuFFT: Release 12.4 ................................................ 20
      5.2.4 cuFFT: Release 12.3 Update 1 ................................... 21
      5.2.5 cuFFT: Release 12.3 ................................................ 21
      5.2.6 cuFFT: Release 12.2 ................................................ 21
      5.2.7 cuFFT: Release 12.1 Update 1 ................................... 22
      5.2.8 cuFFT: Release 12.1 ................................................ 22
      5.2.9 cuFFT: Release 12.0 Update 1 ................................... 22
      5.2.10 cuFFT: Release 12.0 .............................................. 23
   5.3 cuSOLVER Library ......................................................... 23
      5.3.1 cuSOLVER: Release 12.5 .......................................... 23
      5.3.2 cuSOLVER: Release 12.4 Update 1 ............................... 24
      5.3.3 cuSOLVER: Release 12.4 .......................................... 24
      5.3.4 cuSOLVER: Release 12.2 Update 2 ............................... 24
      5.3.5 cuSOLVER: Release 12.2 .......................................... 25
5.4 cuSPARSE Library ................................. 25
  5.4.1 cuSPARSE: Release 12.5 .................. 25
  5.4.2 cuSPARSE: Release 12.4 .................. 25
  5.4.3 cuSPARSE: Release 12.3 Update 1 ....... 26
  5.4.4 cuSPARSE: Release 12.3 .................. 26
  5.4.5 cuSPARSE: Release 12.2 Update 1 ....... 26
  5.4.6 cuSPARSE: Release 12.1 Update 1 ....... 27
  5.4.7 cuSPARSE: Release 12.0 Update 1 ....... 27
  5.4.8 cuSPARSE: Release 12.0 .................. 27
  5.5 Math Library .................................. 28
    5.5.1 CUDA Math: Release 12.5 ............... 28
    5.5.2 CUDA Math: Release 12.4 ............... 28
    5.5.3 CUDA Math: Release 12.3 ............... 28
    5.5.4 CUDA Math: Release 12.2 ............... 29
    5.5.5 CUDA Math: Release 12.1 ............... 30
    5.5.6 CUDA Math: Release 12.0 ............... 30
  5.6 NVIDIA Performance Primitives (NPP) ...... 30
    5.6.1 NPP: Release 12.4 ....................... 30
    5.6.2 NPP: Release 12.0 ....................... 30
  5.7 nvJPEG Library ................................ 31
    5.7.1 nvJPEG: Release 12.4 ................... 31
    5.7.2 nvJPEG: Release 12.3 Update 1 .......... 31
    5.7.3 nvJPEG: Release 12.2 ................... 31
    5.7.4 nvJPEG: Release 12.0 ................... 31
  6 Notices ....................................... 33
    6.1 Notice ....................................... 33
    6.2 OpenCL ...................................... 34
    6.3 Trademarks .................................. 34
NVIDIA CUDA Toolkit Release Notes

The Release Notes for the CUDA Toolkit.

The release notes for the NVIDIA® CUDA® Toolkit can be found online at https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html.

**Note:** The release notes have been reorganized into two major sections: the general CUDA release notes, and the CUDA libraries release notes including historical information for 12.x releases.
Chapter 1. CUDA Toolkit Major Component Versions

CUDA Components
Starting with CUDA 11, the various components in the toolkit are versioned independently.
For CUDA 12.5, the table below indicates the versions:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Version Information</th>
<th>Supported Architectures</th>
<th>Supported Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA C++ Core Compute Libraries</td>
<td>Thrust 2.4.0</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td></td>
<td>CUB 2.4.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>libcu++ 2.4.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cooperative Groups 12.5.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUDA Compatibility</td>
<td>12.5.36033601</td>
<td>aarch64-jetson</td>
<td>Linux</td>
</tr>
<tr>
<td>CUDA Runtime (cudart)</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>cuobjdump</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUPTI</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA cuxxfilt (demangler)</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA Demo Suite</td>
<td>12.5.39</td>
<td>x86_64</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA GDB</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, WSL</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Component Name</th>
<th>Version Information</th>
<th>Supported Architectures</th>
<th>Supported Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Nsight Eclipse Plugin</td>
<td>12.5.39</td>
<td>x86_64</td>
<td>Linux</td>
</tr>
<tr>
<td>CUDA NVCC</td>
<td>12.5.40</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA nvdisasm</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA NVML Headers</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA nvprof</td>
<td>12.5.39</td>
<td>x86_64</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA nvprune</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA NVRTC</td>
<td>12.5.40</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>NVTX</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA NVVP</td>
<td>12.5.39</td>
<td>x86_64</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA OpenCL</td>
<td>12.5.39</td>
<td>x86_64</td>
<td>Linux, Windows</td>
</tr>
<tr>
<td>CUDA Profiler API</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA Compute Sanitizer API</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA cuBLAS</td>
<td>12.5.2.13</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>cuDLA</td>
<td>12.5.39</td>
<td>aarch64-jetson</td>
<td>Linux</td>
</tr>
<tr>
<td>CUDA cuFFT</td>
<td>11.2.3.18</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA cuFile</td>
<td>1.10.0.4</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Component Name</th>
<th>Version Information</th>
<th>Supported Architectures</th>
<th>Supported Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA cuRAND</td>
<td>10.3.6.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA cuSOLVER</td>
<td>11.6.2.40</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA cuSPARSE</td>
<td>12.4.1.18</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA NPP</td>
<td>12.3.0.116</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA nvFatbin</td>
<td>12.5.39</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA nvJitLink</td>
<td>12.5.40</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>CUDA nvJPEG</td>
<td>12.3.2.38</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>Nsight Compute</td>
<td>2024.2.0.16</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL, Windows (Windows 11)</td>
</tr>
<tr>
<td>Nsight Systems</td>
<td>2024.2.3.38</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux, Windows, WSL</td>
</tr>
<tr>
<td>Nsight Visual Studio Edition (VSE)</td>
<td>2024.2.0.24102</td>
<td>x86_64 (Windows)</td>
<td>Windows</td>
</tr>
<tr>
<td>nvidia_fs(^1)</td>
<td>2.20.5</td>
<td>x86_64, arm64-sbsa, aarch64-jetson</td>
<td>Linux</td>
</tr>
<tr>
<td>Visual Studio Integration</td>
<td>12.5.39</td>
<td>x86_64 (Windows)</td>
<td>Windows</td>
</tr>
<tr>
<td>NVIDIA Linux Driver</td>
<td>555.42.02</td>
<td>x86_64, arm64-sbsa</td>
<td>Linux</td>
</tr>
<tr>
<td>NVIDIA Windows Driver</td>
<td>555.85</td>
<td>x86_64 (Windows)</td>
<td>Windows, WSL</td>
</tr>
</tbody>
</table>

**CUDA Driver**

Running a CUDA application requires the system with at least one CUDA capable GPU and a driver that is compatible with the CUDA Toolkit. See Table 3. For more information various GPU

\(^1\) Only available on select Linux distros
products that are CUDA capable, visit https://developer.nvidia.com/cuda-gpus.

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


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

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

<table>
<thead>
<tr>
<th>CUDA Toolkit</th>
<th>Minimum Required Driver Version for CUDA Minor Version Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA 12.x</td>
<td>&gt;=525.60.13 &gt;=528.33</td>
</tr>
<tr>
<td>CUDA 11.8.x</td>
<td>CUDA 11.7.x CUDA 11.6.x CUDA 11.5.x</td>
</tr>
<tr>
<td></td>
<td>CUDA 11.4.x CUDA 11.3.x CUDA 11.2.x CUDA 11.1.x</td>
</tr>
<tr>
<td></td>
<td>&gt;=450.80.02 &gt;=452.39</td>
</tr>
<tr>
<td>CUDA 11.0 (11.0.3)</td>
<td>&gt;=450.36.06** &gt;=451.22**</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>CUDA Toolkit</th>
<th>Toolkit Driver Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA 12.5 GA</td>
<td>&gt;=555.42.02</td>
</tr>
<tr>
<td>CUDA 12.4 Update 1</td>
<td>&gt;=550.54.15</td>
</tr>
<tr>
<td>CUDA 12.4 GA</td>
<td>&gt;=550.54.14</td>
</tr>
<tr>
<td>CUDA 12.3 Update 1</td>
<td>&gt;=545.23.08</td>
</tr>
<tr>
<td>CUDA 12.3 GA</td>
<td>&gt;=545.23.06</td>
</tr>
<tr>
<td>CUDA 12.2 Update 2</td>
<td>&gt;=535.104.05</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>CUDA Toolkit</th>
<th>Toolkit Driver Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA 12.2 Update 1</td>
<td>&gt;=535.86.09</td>
</tr>
<tr>
<td>CUDA 12.2 GA</td>
<td>&gt;=535.54.03</td>
</tr>
<tr>
<td>CUDA 12.1 Update 1</td>
<td>&gt;=530.30.02</td>
</tr>
<tr>
<td>CUDA 12.1 GA</td>
<td>&gt;=530.30.02</td>
</tr>
<tr>
<td>CUDA 12.0 Update 1</td>
<td>&gt;=525.85.12</td>
</tr>
<tr>
<td>CUDA 12.0 GA</td>
<td>&gt;=525.60.13</td>
</tr>
<tr>
<td>CUDA 11.8 GA</td>
<td>&gt;=520.61.05</td>
</tr>
<tr>
<td>CUDA 11.7 Update 1</td>
<td>&gt;=515.48.07</td>
</tr>
<tr>
<td>CUDA 11.7 GA</td>
<td>&gt;=515.43.04</td>
</tr>
<tr>
<td>CUDA 11.6 Update 2</td>
<td>&gt;=510.47.03</td>
</tr>
<tr>
<td>CUDA 11.6 Update 1</td>
<td>&gt;=510.47.03</td>
</tr>
<tr>
<td>CUDA 11.6 GA</td>
<td>&gt;=510.39.01</td>
</tr>
<tr>
<td>CUDA 11.5 Update 2</td>
<td>&gt;=495.29.05</td>
</tr>
<tr>
<td>CUDA 11.5 Update 1</td>
<td>&gt;=495.29.05</td>
</tr>
<tr>
<td>CUDA 11.5 GA</td>
<td>&gt;=495.29.05</td>
</tr>
<tr>
<td>CUDA 11.4 Update 4</td>
<td>&gt;=470.82.01</td>
</tr>
<tr>
<td>CUDA 11.4 Update 3</td>
<td>&gt;=470.82.01</td>
</tr>
<tr>
<td>CUDA 11.4 Update 2</td>
<td>&gt;=470.57.02</td>
</tr>
<tr>
<td>CUDA 11.4 Update 1</td>
<td>&gt;=470.57.02</td>
</tr>
<tr>
<td>CUDA 11.4.0 GA</td>
<td>&gt;=470.42.01</td>
</tr>
<tr>
<td>CUDA 11.3.1 Update 1</td>
<td>&gt;=465.19.01</td>
</tr>
<tr>
<td>CUDA 11.3.0 GA</td>
<td>&gt;=465.19.01</td>
</tr>
<tr>
<td>CUDA 11.2.2 Update 2</td>
<td>&gt;=460.32.03</td>
</tr>
<tr>
<td>CUDA 11.2.1 Update 1</td>
<td>&gt;=460.32.03</td>
</tr>
<tr>
<td>CUDA 11.2.0 GA</td>
<td>&gt;=460.27.03</td>
</tr>
<tr>
<td>CUDA 11.1.1 Update 1</td>
<td>&gt;=455.32</td>
</tr>
<tr>
<td>CUDA 11.1 GA</td>
<td>&gt;=455.23</td>
</tr>
<tr>
<td>CUDA 11.0.3 Update 1</td>
<td>&gt;=450.51.06</td>
</tr>
<tr>
<td>CUDA 11.0.2 GA</td>
<td>&gt;=450.51.05</td>
</tr>
<tr>
<td>CUDA 11.0.1 RC</td>
<td>&gt;=450.36.06</td>
</tr>
<tr>
<td>CUDA 10.2.89</td>
<td>&gt;=440.33</td>
</tr>
</tbody>
</table>
For convenience, the NVIDIA driver is installed as part of the CUDA Toolkit installation. Note that this driver is for development purposes and is not recommended for use in production with Tesla GPUs.

For running CUDA applications in production with Tesla GPUs, it is recommended to download the latest driver for Tesla GPUs from the NVIDIA driver downloads site at https://www.nvidia.com/drivers.

During the installation of the CUDA Toolkit, the installation of the NVIDIA driver may be skipped on Windows (when using the interactive or silent installation) or on Linux (by using meta packages).

For more information on customizing the install process on Windows, see https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html#install-cuda-software.


<table>
<thead>
<tr>
<th>CUDA Toolkit</th>
<th>Toolkit Driver Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA 10.1 (10.1.105 general release, and updates)</td>
<td>&gt;= 418.39, &gt;= 418.96</td>
</tr>
<tr>
<td>CUDA 10.0.130</td>
<td>&gt;= 410.48, &gt;= 411.31</td>
</tr>
<tr>
<td>CUDA 9.2 (9.2.148 Update 1)</td>
<td>&gt;= 396.37, &gt;= 398.26</td>
</tr>
<tr>
<td>CUDA 9.2 (9.2.88)</td>
<td>&gt;= 396.26, &gt;= 397.44</td>
</tr>
<tr>
<td>CUDA 9.1 (9.1.85)</td>
<td>&gt;= 390.46, &gt;= 391.29</td>
</tr>
<tr>
<td>CUDA 9.0 (9.0.76)</td>
<td>&gt;= 384.81, &gt;= 385.54</td>
</tr>
<tr>
<td>CUDA 8.0 (8.0.61 GA2)</td>
<td>&gt;= 375.26, &gt;= 376.51</td>
</tr>
<tr>
<td>CUDA 8.0 (8.0.44)</td>
<td>&gt;= 367.48, &gt;= 369.30</td>
</tr>
<tr>
<td>CUDA 7.5 (7.5.16)</td>
<td>&gt;= 352.31, &gt;= 353.66</td>
</tr>
<tr>
<td>CUDA 7.0 (7.0.28)</td>
<td>&gt;= 346.46, &gt;= 347.62</td>
</tr>
</tbody>
</table>
Chapter 2. New Features

This section lists new general CUDA and CUDA compilers features.

2.1. 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.

2.2. CUDA Compilers

▶ For changes to PTX, refer to https://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-isa-version-8-5.

2.3. CUDA Developer Tools

▶ For changes to nvprof and Visual Profiler, see the changelog.
▶ For new features, improvements, and bug fixes in Nsight Systems, see the changelog.
▶ For new features, improvements, and bug fixes in Nsight Visual Studio Edition, see the changelog.
▶ For new features, improvements, and bug fixes in CUPTI, see the changelog.
▶ For new features, improvements, and bug fixes in Nsight Compute, see the changelog.
▶ For new features, improvements, and bug fixes in Compute Sanitizer, see the changelog.
▶ For new features, improvements, and bug fixes in CUDA-GDB, see the changelog.
Chapter 2. New Features
Chapter 3. Known Issues and Limitations

- Runfile will not be supported for Amazon Linux 2023.
- Confidential Computing is not supported on CUDA 12.5. Please continue to use CUDA 12.4 and drivers r550.xx to use these features.
- Launching Cooperative Group kernels with MPS is not supported on Tegra platforms.
Chapter 4. Deprecated or Dropped Features

Features deprecated in the current release of the CUDA software still work in the current release, but their documentation may have been removed, and they will become officially unsupported in a future release. We recommend that developers employ alternative solutions to these features in their software.

4.1. Deprecated or Dropped Architectures

► NVIDIA CUDA support for the PowerPC architecture is removed in CUDA 12.5.

4.2. Deprecated Operating Systems

► NVIDIA CUDA support for Red Hat Enterprise Linux 7 and CentOS 7 is removed in CUDA 12.5.
► CUDA 12.5 is the last release to support Debian 10.
► Support for Microsoft Windows 10 21H2 and Microsoft Windows 10 21H2 (SV1) is deprecated.

4.3. Deprecated Toolchains

CUDA Toolkit 12.4 deprecated support for the following host compilers:
► Microsoft Visual C/C++ (MSVC) 2017
► All GCC versions prior to GCC 7.3
4.4. CUDA Tools

- Support for the macOS host client of CUDA-GDB is deprecated. It will be dropped in an upcoming release.
Chapter 5. CUDA Libraries

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

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

5.1. cuBLAS Library

5.1.1. cuBLAS: Release 12.5

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

▶ Known Issues
  ▶ `cublasLtMatmul` ignores inputs to `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER` and `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_SCALE_POINTER` if the elements of the respective matrix are not of FP8 types.

▶ Resolved Issues
  ▶ `cublasLtMatmul` ignored the mismatch between the provided scale type and the implied by the documentation, assuming the latter. For instance, an unsupported configuration of `cublasLtMatmul` with the scale type being FP32 and all other types being FP16 would run with the implicit assumption that the scale type is FP16 and produce incorrect results.

5.1.2. cuBLAS: Release 12.4 Update 1

▶ Known Issues
  ▶ Setting a cuBLAS handle stream to `cudaStreamPerThread` and setting the workspace via `cublasSetWorkspace` will cause any subsequent `cublasSetWorkspace` calls to fail. This will be fixed in an upcoming release.

  ▶ `cublasLtMatmul` ignores mismatches between the provided scale type and the scale type implied by the documentation and assumes the latter. For example, an unsupported configuration of `cublasLtMatmul` with the scale type being FP32 and all other types being
FP16 would run with the implicit assumption that the scale type is FP16 which can produce incorrect results. This will be fixed in an upcoming release.

**Resolved Issues**

- **cublasLtMatmul** ignored the CUBLASLT_MATMUL_DESC_AMAX_D_POINTER for unsupported configurations instead of returning an error. In particular, computing absolute maximum of D is currently supported only for FP8 Matmul when the output data type is also FP8 (CUDA_R_8F_E4M3 or CUDA_R_8F_E5M2).
- Reduced host-side overheads for some of the cuBLASLt APIs: cublasLtMatmul(), cublasLtMatmulAlgoCheck(), and cublasLtMatmulAlgoGetHeuristic(). The issue was introduced in CUDA Toolkit 12.4.
- cublasLtMatmul() and cublasLtMatmulAlgoGetHeuristic() could have resulted in floating point exceptions (FPE) on some Hopper-based GPUs, including Multi-Instance GPU (MIG). The issue was introduced in cuBLAS 11.8.

5.1.3. cuBLAS: Release 12.4

**New Features**

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

**Known Issues**

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

▶ New Features
▶ Improved performance of heuristics cache for workloads that have a high eviction rate.

▶ Known Issues
▶ BLAS level 2 and 3 functions might not treat alpha in a BLAS compliant manner when alpha is zero and the pointer mode is set to CUBLAS_POINTER_MODE_DEVICE. The expected behavior is that the corresponding computations would be skipped. You may encounter the following issues: (1) HER(2,X,K,2K) may zero the imaginary part on the diagonal elements of the output matrix; and (2) HER(2,X,K,2K), SYR(2,X,K,2K) and others may produce NaN resulting from performing computation on matrices A and B which would otherwise be skipped. If strict compliance with BLAS is required, the user may manually check for alpha value before invoking the functions or switch to CUBLAS_POINTER_MODE_HOST.

▶ Resolved Issues
▶ cuBLASLt matmul operations might have computed the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.

▶ When an application compiled with cuBLASLt from CUDA Toolkit 12.2 update 1 or earlier runs with cuBLASLt from CUDA Toolkit 12.2 update 2 or CUDA Toolkit 12.3, matrix multiply descriptors initialized using cublasLtMatmulDescInit() sometimes did not respect attribute changes using cublasLtMatmulDescSetAttribute(). To workaround this issue, create the matrix multiply descriptor using cublasLtMatmulDescCreate() instead of cublasLtMatmulDescInit(). This will be fixed in an upcoming release.

5.1.5. cuBLAS: Release 12.3

▶ New Features
▶ Improved performance on NVIDIA L40S Ada GPUs.

▶ Known Issues
▶ cuBLASLt matmul operations may compute the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.

▶ When an application compiled with cuBLASLt from CUDA Toolkit 12.2 update 1 or earlier runs with cuBLASLt from CUDA Toolkit 12.2 update 2 or later, matrix multiply descriptors initialized using cublasLtMatmulDescInit() may not respect attribute changes using cublasLtMatmulDescSetAttribute(). To workaround this issue, create the matrix multiply descriptor using cublasLtMatmulDescCreate() instead of cublasLtMatmulDescInit(). This will be fixed in an upcoming release.
5.1.6. cuBLAS: Release 12.2 Update 2

► New Features

► cuBLASLt will now attempt to decompose problems that cannot be run by a single gemm kernel. It does this by partitioning the problem into smaller chunks and executing the gemm kernel multiple times. This improves functional coverage for very large m, n, or batch size cases and makes the transition from the cuBLAS API to the cuBLASLt API more reliable.

► Known Issues

► cuBLASLt matmul operations may compute the output incorrectly under the following conditions: the data type of matrices A and B is FP8, the data type of matrices C and D is FP32, FP16, or BF16, the beta value is 1.0, the C and D matrices are the same, the epilogue contains GELU activation function.

5.1.7. cuBLAS: Release 12.2

► Known Issues

► cuBLAS initialization fails on Hopper architecture GPUs when MPS is in use with CUDA_MPS_ACTIVE_THREAD_PERCENTAGE set to a value less than 100%. There is currently no workaround for this issue.

► Some Hopper kernels produce incorrect results for batched matmuls with CUBLASLT_EPILOGUE_RELU_BIAS or CUBLASLT_EPILOGUE_GELU_BIAS and a non-zero CUBLASLT_MATMUL_DESC_BIAS_BATCH_STRIDE. The kernels apply the first batch’s bias vector to all batches. This will be fixed in a future release.

5.1.8. cuBLAS: Release 12.1 Update 1

► New Features

► Support for FP8 on NVIDIA Ada GPUs.

► Improved performance on NVIDIA L4 Ada GPUs.

► Introduced an API that instructs the cuBLASLt library to not use some CPU instructions. This is useful in some rare cases where certain CPU instructions used by cuBLASLt heuristics negatively impact CPU performance. Refer to https://docs.nvidia.com/cuda/cublas/index.html#disabling-cpu-instructions.

► Known Issues

► When creating a matrix layout using the cublasLtMatrixLayoutCreate() function, the object pointed at by cublasLtMatrixLayoutOpaque_t is smaller than cublasLtMatrixLayoutOpaque_t (but enough to hold the internal structure). As a result, the object should not be dereferenced or copied explicitly, as this might lead to out of bound accesses. If one needs to serialize the layout or copy it, it is recommended to manually allocate an object of size sizeof(cublasLtMatrixLayoutOpaque_t) bytes, and initialize it using cublasLtMatrixLayoutInit() function. The same applies to cublasLtMatmulDesc_t and cublasLtMatrixTransformDesc_t. The issue will be fixed in future releases by ensuring that cublasLtMatrixLayoutCreate() allocates at least sizeof(cublasLtMatrixLayoutOpaque_t) bytes.
5.1.9. cuBLAS: Release 12.0 Update 1

- **New Features**
  - Improved performance on NVIDIA H100 SXM and NVIDIA H100 PCIe GPUs.

- **Known Issues**
  - For optimal performance on NVIDIA Hopper architecture, cuBLAS needs to allocate a bigger internal workspace (64 MiB) than on the previous architectures (8 MiB). In the current and previous releases, cuBLAS allocates 256 MiB. This will be addressed in a future release. A possible workaround is to set the `CUBLAS_WORKSPACE_CONFIG` environment variable to :32768:2 when running cuBLAS on NVIDIA Hopper architecture.

- **Resolved Issues**
  - Reduced cuBLAS host-side overheads caused by not using the cublasLt heuristics cache. This began in the CUDA Toolkit 12.0 release.
  - Added forward compatible single precision complex GEMM that does not require workspace.

5.1.10. cuBLAS: Release 12.0

- **New Features**
  - `cublasLtMatmul` now supports FP8 with a non-zero beta.
  - Added `int64` APIs to enable larger problem sizes; refer to 64-bit integer interface.
  - Added more Hopper-specific kernels for `cublasLtMatmul` with epilogues:
    - `CUBLASLT_EPILOGUE_BGRAD{A,B}`
    - `CUBLASLT_EPILOGUE_{RELU,GELU}_AUX`
    - `CUBLASLT_EPILOGUE_D{RELU,GELU}`
  - Improved Hopper performance on arm64-sbsa by adding Hopper kernels that were previously supported only on the x86_64 architecture for Windows and Linux.

- **Known Issues**
  - There are no forward compatible kernels for single precision complex gemms that do not require workspace. Support will be added in a later release.

- **Resolved Issues**
  - Fixed an issue on NVIDIA Ampere architecture and newer GPUs where `cublasLtMatmul` with epilogue `CUBLASLT_EPILOGUE_BGRAD{A,B}` and a nontrivial reduction scheme (that is, not `CUBLASLT_REDUCTION_SCHEME_NONE`) could return incorrect results for the bias gradient.
  - `cublasLtMatmul` for gemv-like cases (that is, m or n equals 1) might ignore bias with the `CUBLASLT_EPILOGUE_RELU_BIAS` and `CUBLASLT_EPILOGUE_BIAS` epilogues.

**Deprecations**

- Disallow including `cublas.h` and `cublas_v2.h` in the same translation unit.
- Removed:
CUBLAS MATMUL STAGE 16x80 and CUBLAS MATMUL STAGE 64x80 from cublasLtMatmulStages_t. No kernels utilize these stages anymore.

- cublasLt3mMode_t, CUBLASLT_MATMUL_PREF_MATH_MODE_MASK, and CUBLASLT_MATMUL_PREF_GAUSSIAN_MODE_MASK from cublasLtMatmulPref- erenceAttributes_t. Instead, use the corresponding flags from cublasLtNumericalImplFlags_t.

- CUBLASLT_MATMUL_PREF_POINTER_MODE_MASK, CUBLASLT_MATMUL_PREF_EPILOGUE_MASK, and CUBLASLT_MATMUL_PREF_SM_COUNT_TARGET from cublasLtMatmulPref- erenceAttributes_t. The corresponding parameters are taken directly from cublasLtMatmulDesc_t.

- CUBLASLT_POINTER_MODE_MASK_NO_FILTERING from cublasLtPointerMode- Mask_t. This mask was only applicable to CUBLASLT_MATMUL_PREF_MATH_MODE_MASK which was removed.

5.2. cuFFT Library

5.2.1. cuFFT: Release 12.5

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

5.2.2. cuFFT: Release 12.4 Update 1

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

5.2.3. cuFFT: Release 12.4

- **New Features**
  - Added Just-In-Time Link-Time Optimized (JIT LTO) kernels for improved performance in FFTs with 64-bit indexing.
  - Added per-plan properties to the cuFFT API. These new routines can be leveraged to give users more control over the behavior of cuFFT. Currently they can be used to enable JIT LTO kernels for 64-bit FFTs.
  - Improved accuracy for certain single-precision (fp32) FFT cases, especially involving FFTs for larger sizes.
Known Issues

- A routine from the cuFFT LTO EA library was added by mistake to the cuFFT Advanced API header (cufftXt.h). This routine is not supported by cuFFT, and will be removed from the header in a future release.

Resolved Issues

- Fixed an issue that could cause overwriting of user data when performing out-of-place real-to-complex (R2C) transforms with user-specified output strides (i.e. using the `ostride` component of the Advanced Data Layout API).
- Fixed inconsistent behavior between `libcufftw` and FFTW when both `inembed` and `onembed` are `nullptr / NULL`. From now on, as in FFTW, passing `nullptr / NULL` as `inembed` or `onembed` parameter is equivalent to passing `n`, that is, the logical size for that dimension.

5.2.4. cuFFT: Release 12.3 Update 1

Known Issues

- Executing a real-to-complex (R2C) or complex-to-real (C2R) plan in a context different to the one used to create the plan could cause undefined behavior. This issue will be fixed in an upcoming release of cuFFT.

Resolved Issues

- Complex-to-complex (C2C) execution functions (cufftExec and similar) now properly error-out in case of error during kernel launch, for example due to a missing CUDA context.

5.2.5. cuFFT: Release 12.3

New Features

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

5.2.6. cuFFT: Release 12.2

New Features

- `cufftSetStream` can be used in multi-GPU plans with a stream from any GPU context, instead of from the primary context of the first GPU listed in `cufftXtSetGPUs`.
- Improved performance of 1000+ of FFTs of sizes ranging from 62 to 16380. The improved performance spans hundreds of single precision and double precision cases for FFTs with contiguous data layout, across multiple GPU architectures (from Maxwell to Hopper GPUs) via PTX JIT.
- Reduced the size of the static libraries when compared to cuFFT in the 12.1 release.

Resolved Issues
cuFFT no longer exhibits a race condition when threads simultaneously create and access plans with more than 1023 plans alive.

cuFFT no longer exhibits a race condition when multiple threads call cufftXtSetGPUs concurrently.

### 5.2.7. cuFFT: Release 12.1 Update 1

**Known Issues**

- cuFFT exhibits a race condition when one thread calls cufftCreate (or cufftDestroy) and another thread calls any API (except cufftCreate or cufftDestroy), and when the total number of plans alive exceeds 1023.

- cuFFT exhibits a race condition when multiple threads call cufftXtSetGPUs concurrently on different plans.

### 5.2.8. cuFFT: Release 12.1

**New Features**

- Improved performance on Hopper GPUs for hundreds of FFTs of sizes ranging from 14 to 28800. The improved performance spans over 542 cases across single and double precision for FFTs with contiguous data layout.

**Known Issues**

- Starting from CUDA 11.8, CUDA Graphs are no longer supported for callback routines that load data in out-of-place mode transforms. An upcoming release will update the cuFFT callback implementation, removing this limitation. cuFFT deprecated callback functionality based on separate compiled device code in cuFFT 11.4.

**Resolved Issues**

- cuFFT no longer produces errors with compute-sanitizer at program exit if the CUDA context used at plan creation was destroyed prior to program exit.

### 5.2.9. cuFFT: Release 12.0 Update 1

**Resolved Issues**

- Scratch space requirements for multi-GPU, single-batch, 1D FFTs were reduced.
5.2.10. cuFFT: Release 12.0

▶ New Features
  ▶ PTX JIT kernel compilation allowed the addition of many new accelerated cases for Maxwell, Pascal, Volta and Turing architectures.

▶ Known Issues
  ▶ cuFFT plan generation time increases due to PTX JIT compiling. Refer to Plan Initialization Time.

▶ Resolved Issues
  ▶ cuFFT plans had an unintentional small memory overhead (of a few kB) per plan. This is resolved.

5.3. cuSOLVER Library

5.3.1. cuSOLVER: Release 12.5

▶ New Features
  ▶ Performance improvements of cusolverDnXgesvd and cusolverDn<t>gesvd if jobu ! = 'N' or jobvt != 'N'.
  ▶ Performance improvements of cusolverDnXgesvdp if jobz = CU-SOLVER_EIG_MODE_NOVECTOR.

▶ Known Issues
  ▶ With CUDA Toolkit 12.4 Update 1, values ldt > k in calls of cusolverDnXlarft can result in out-of-bound memory accesses on bufferOnDevice. As a workaround it is possible to allocate a larger device workspace buffer of size workspaceInBytesOnDevice=ALIGN_32((ldt*k + n*k)*sizeofCudaDataType(dataTypeT)), with

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

and

```c
auto sizeofCudaDataType=[](cudaDataType dt) {
    if (dt == CUDA_R_32F) return sizeof(float);
    if (dt == CUDA_R_64F) return sizeof(double);
    if (dt == CUDA_C_32F) return sizeof(cuComplex);
    if (dt == CUDA_C_64F) return sizeof(cuDoubleComplex);
};
```
5.3.2. cuSOLVER: Release 12.4 Update 1

► New Features

► The performance of cusolverDnXlarft has been improved. For large matrices, the speedup might exceed 100x. The performance on H100 is now consistently better than on A100. The change in cusolverDnXlarft also results in a modest speedup in cusolverDn<t>ormqr, cusolverDn<t>ormtr, and cusolverDnXsyevd.

► The performance of cusolverDnXgesvd when singular vectors are sought has been improved. The job configuration that computes both left and right singular vectors is up to 1.5x faster.

► Resolved Issues

► cusolverDnXtrtri_bufferSize now returns the correct workspace size in bytes.

► Deprecations

► Using long-deprecated cusolverDnPotrf, cusolverDnPotrs, cusolverDnGeqrf, cusolverDnGetrf, cusolverDnGetrs, cusolverDnSyevd, cusolverDnSyevdx, cusolverDnDgesvd, and their accompanying bufferSize functions will result in a deprecation warning. The warning can be turned off by using the -DDISABLE_CUSOLVER_DEPRECATED flag while compiling; however, users should use cusolverDnXpotrf, cusolverDnXpotrs, cusolverDnXgeqrf, cusolverDnXgetrf, cusolverDnXgetrs, cusolverDnXsyevd, cusolverDnXsyevdx, cusolverDnXgesvd, and the corresponding bufferSize functions instead.

5.3.3. cuSOLVER: Release 12.4

► New Features

► cusolverDnXlarft and cusolverDnXlarft_bufferSize APIs were introduced. cusolverDnXlarft forms the triangular factor of a real block reflector, while cusolverDnXlarft_bufferSize returns its required workspace sizes in bytes.

► Known Issues

► cusolverDnXtrtri_bufferSize` returns an incorrect required device workspace size. As a workaround the returned size can be multiplied by the size of the data type (for example, 8 bytes if matrix A is of type double) to obtain the correct workspace size.

5.3.4. cuSOLVER: Release 12.2 Update 2

► Resolved Issues

► Fixed an issue with cusolverDn<t>gesvd(), cusolverDnGesvd(), and cusolverDnXgesvd(), which could cause wrong results for matrices larger than 18918 if jobu or jobvt was unequal to ‘N’.
5.3.5. cuSOLVER: Release 12.2

▶ New Features
▶ A new API to ensure deterministic results or allow non-deterministic results for improved performance. See cusolverDnSetDeterministicMode() and cusolverDnGetDeterministicMode(). Affected functions are: cusolverDn<t>geqrf(), cusolverDn<t>syevd(), cusolverDn<t>syevdx(), cusolverDn<t>gesvdj(), cusolverDnXgeqrf(), cusolverDnXsyevd(), cusolverDnXsyevdx(), cusolverDnXgesvdj(), and cusolverDnXgesvdx().

▶ Known Issues
▶ Concurrent executions of cusolverDn<t>getrf() or cusolverDnXgetrf() in different non-blocking CUDA streams on the same device might result in a deadlock.

5.4. cuSPARSE Library

5.4.1. cuSPARSE: Release 12.5

▶ New Features
▶ Added support for mixed input types in SpMV: single precision input matrix, double precision input vector, double precision output vector.

▶ Resolved Issues
▶ cusparseSpMV() introduces invalid memory accesses when the output vector is not aligned to 16 bytes.

5.4.2. cuSPARSE: Release 12.4

▶ New Features
▶ Added the preprocessing step for sparse matrix-vector multiplication cusparseSpMV_preprocess().
▶ Added support for mixed real and complex types for cusparseSpMM().
▶ Added a new API cusparseSpSM_updateMatrix() to update the sparse matrix between the analysis and solving phase of cusparseSpSM().

▶ Known Issues
▶ cusparseSpMV() introduces invalid memory accesses when the output vector is not aligned to 16 bytes.

▶ Resolved Issues
▶ cusparseSpVV() provided incorrect results when the sparse vector has many non-zeros.
5.4.3. cuSPARSE: Release 12.3 Update 1

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

5.4.4. cuSPARSE: Release 12.3

▶ New Features
  ▶ The cusparseSpSV_bufferSize() and cusparseSpSV_analysis() routines now accept NULL pointers for the dense vector.
  ▶ The cusparseSpSM_bufferSize() and cusparseSpSM_analysis() routines now accept dense matrix descriptors with NULL pointer for values.

▶ Known Issues
  ▶ The cusparseSpSV_analysis() and cusparseSpSM_analysis() routines are blocking calls/not asynchronous.
  ▶ Wrong results can occur for cusparseSpSV() using sliced ELLPACK format and transpose/conjugate transpose operation on matrix A.

▶ Resolved Issues
  ▶ cusparseSpSV() provided indeterministic results in some cases.
  ▶ Fixed an issue that caused cusparseSpSV_analysis() to hang sometimes in a multi-thread environment.
  ▶ Fixed an issue with cusparseSpSV() and cusparseSpSV() that sometimes yielded wrong output when the output vector/matrix or input matrix contained NaN.

5.4.5. cuSPARSE: Release 12.2 Update 1

▶ New Features
  ▶ The library now provides the opportunity to dump sparse matrices to files during the creation of the descriptor for debugging purposes. See logging API https://docs.nvidia.com/cuda/cusparse/index.html#cusparse-logging-api.

▶ Resolved Issues
  ▶ Removed CUSPARSE_SPMM_CSR_ALG3 fallback to avoid confusion in the algorithm selection process.
  ▶ Clarified the supported operations for cusparseSDDMM().
  ▶ cusparseCreateConstSlicedEll() now uses const pointers.
  ▶ Fixed wrong results in rare edge cases of cusparseCsr2CscEx2() with base 1 indexing.
  ▶ cusparseSpSM_bufferSize() could ask slightly less memory than needed.
 cusparseSpMV() now checks the validity of the buffer pointer only when it is strictly needed.

**Deprecations**

Several legacy APIs have been officially deprecated. A compile-time warning has been added to all of them.

### 5.4.6. cuSPARSE: Release 12.1 Update 1

**New Features**

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

### 5.4.7. cuSPARSE: Release 12.0 Update 1

**New Features**

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

**Resolved Issues**

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

### 5.4.8. cuSPARSE: Release 12.0

**New Features**

- JIT LTO functionalities (cusparseSpMMOp()) switched from driver to nvJitLto library. Starting from CUDA 12.0 the user needs to link to libnvJitLto.so, see cuSPARSE documentation. JIT LTO performance has also been improved for cusparseSpMMOpPlan().
- Introduced const descriptors for the Generic APIs, for example, cusparseConstSpVecGet(). Now the Generic APIs interface clearly declares when a descriptor and its data are modified by the cuSPARSE functions.
Release Notes, Release 12.5

- Added two new algorithms to \texttt{cusparseSpGEMM()} with lower memory utilization. The first algorithm computes a strict bound on the number of intermediate product, while the second one allows partitioning the computation in chunks.
- Added \texttt{int8_t} support to \texttt{cusparseGather()}, \texttt{cusparseScatter()}, and \texttt{cusparseCsr2cscEx2()}.
- Improved \texttt{cusparseSpSV()} performance for both the analysis and the solving phases.
- Improved \texttt{cusparseSpSM()} performance for both the analysis and the solving phases.
- Improved \texttt{cusparseSDDMM()} performance and added support for batch computation.
- Improved \texttt{cusparseCsr2cscEx2()} performance.

Resolved Issues

- \texttt{cusparseSpSV()} and \texttt{cusparseSpSM()} could produce wrong results.
- \texttt{cusparseDnMatGetStridedBatch()} did not accept batchStride $= 0$.

Deprecations

- Removed deprecated CUDA 11.x APIs, enumerators, and descriptors.

5.5. Math Library

5.5.1. CUDA Math: Release 12.5

Known Issues

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

5.5.2. CUDA Math: Release 12.4

Resolved Issues

- Host-specific code in \texttt{cuda_fp16/bf16} headers is now free from type-punning and shall work correctly in the presence of optimizations based on strict-aliasing rules.

5.5.3. CUDA Math: Release 12.3

New Features

- Performance of SIMD Integer CUDA Math APIs was improved.

Resolved Issues
The __hisinf() Math APIs from cuda_fp16.h and cuda_bf16.h headers were silently producing wrong results if compiled with the -std=c++20 compiler option because of an underlying nvcc compiler issue, resolved in version 12.3.

**Known Issues**

Users of cuda_fp16.h and cuda_bf16.h headers are advised to disable host compilers strict aliasing rules based optimizations (e.g. pass -fno-strict-aliasing to host GCC compiler) as these may interfere with the type-punning idioms used in the __half, __half2, __nv_bfloat16, __nv_bfloat162 types implementations and expose the user program to undefined behavior. Note, the headers suppress GCC diagnostics through: #pragma GCC diagnostic ignored -Wstrict-aliasing. This behavior may improve in future versions of the headers.

5.5.4. CUDA Math: Release 12.2

**New Features**

CUDA Math APIs for __half and __nv_bfloat16 types received usability improvements, including host side <emulated> support for many of the arithmetic operations and conversions.

__half and __nv_bfloat16 types have implicit conversions to/from integral types, which are now available with host compilers by default. These may cause build issues due to ambiguous overloads resolution. Users are advised to update their code to select proper overloads. To opt-out user may want to define the following macros (these macros will be removed in the future CUDA release):

- __CUDA_FP16_DISABLE_IMPLICIT_INTEGER_CONVERTS_FOR_HOST_COMPILERS__
- __CUDA_BF16_DISABLE_IMPLICIT_INTEGER_CONVERTS_FOR_HOST_COMPILERS__

**Resolved Issues**

During ongoing testing, NVIDIA identified that due to an algorithm error the results of 64-bit floating-point division in default round-to-nearest-even mode could produce spurious overflow to infinity. NVIDIA recommends that all developers requiring strict IEEE754 compliance update to CUDA Toolkit 12.2 or newer. The affected algorithm was present in both offline compilation as well as just-in-time (JIT) compilation. As JIT compilation is handled by the driver, NVIDIA recommends updating to driver version greater than or equal to R535 (R536 on Windows) when IEEE754 compliance is required and when using JIT. This is a software algorithm fix and is not tied to specific hardware.

Updated the observed worst case error bounds for single precision intrinsic functions __expf(), __exp10f() and double precision functions asinh(), acosh().
5.5.5. CUDA Math: Release 12.1

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

5.5.6. CUDA Math: Release 12.0

- **New Features**

- **Known Issues**
  - Double precision inputs that cause the double precision division algorithm in the default 'round to nearest even mode' produce spurious overflow: an infinite result is delivered where `DBL_MAX 0x7FEF_FFFF_FFFF_FFFF` is expected. Affected CUDA Math APIs: `__ddiv_rn()`. Affected CUDA language operation: double precision / operation in the device code.

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

5.6. NVIDIA Performance Primitives (NPP)

5.6.1. NPP: Release 12.4

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

5.6.2. NPP: Release 12.0

- **Deprecations**
  - Deprecating non-CTX API support from next release.

- **Resolved Issues**
  - A performance issue with the NPP `ResizeSqrPixel` API is now fixed and shows improved performance.
5.7. nvJPEG Library

5.7.1. nvJPEG: Release 12.4

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

5.7.2. nvJPEG: Release 12.3 Update 1

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

5.7.3. nvJPEG: Release 12.2

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

5.7.4. nvJPEG: Release 12.0

- **New Features**
  - Improved the GPU Memory optimisation for the nvJPEG codec.
- **Resolved Issues**
  - An issue that causes runtime failures when nvJPEGDecMultipleInstances was tested with a large number of threads is resolved.
  - An issue with CMYK four component color conversion is now resolved.
- **Known Issues**
  - Backend NVJPEG_BACKEND_GPU_HYBRID - Unable to handle bistreams with extra scans lengths.
- **Deprecations**
  - The reuse of Huffman table in Encoder (nvjpegEncoderParamsCopyHuffmanTables).
Chapter 6. Notices

6.1. 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.

6.2. OpenCL

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

6.3. 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

©2007-2024, NVIDIA Corporation & affiliates. All rights reserved