CUDA COMPATIBILITY

vR450 | May 2020

CUDA Compatibility

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. OVERVIEW

The CUDA toolkit is transitioning to a faster release cadence to deliver new features, performance improvements, and critical bug fixes. However, the tight coupling of the CUDA runtime with the display driver (specifically **`libcuda.so`**—the CUDA driver on Linux systems), means that the customer is required to update the entire driver stack to use the latest CUDA software (including the compiler, libraries, and tools).

The new CUDA Compatibility Platform enables the use of new CUDA toolkit components on systems with older base installations.

Chapter 2. COMPATIBILITY

The current upgrade path for CUDA is shown in Figure 1.

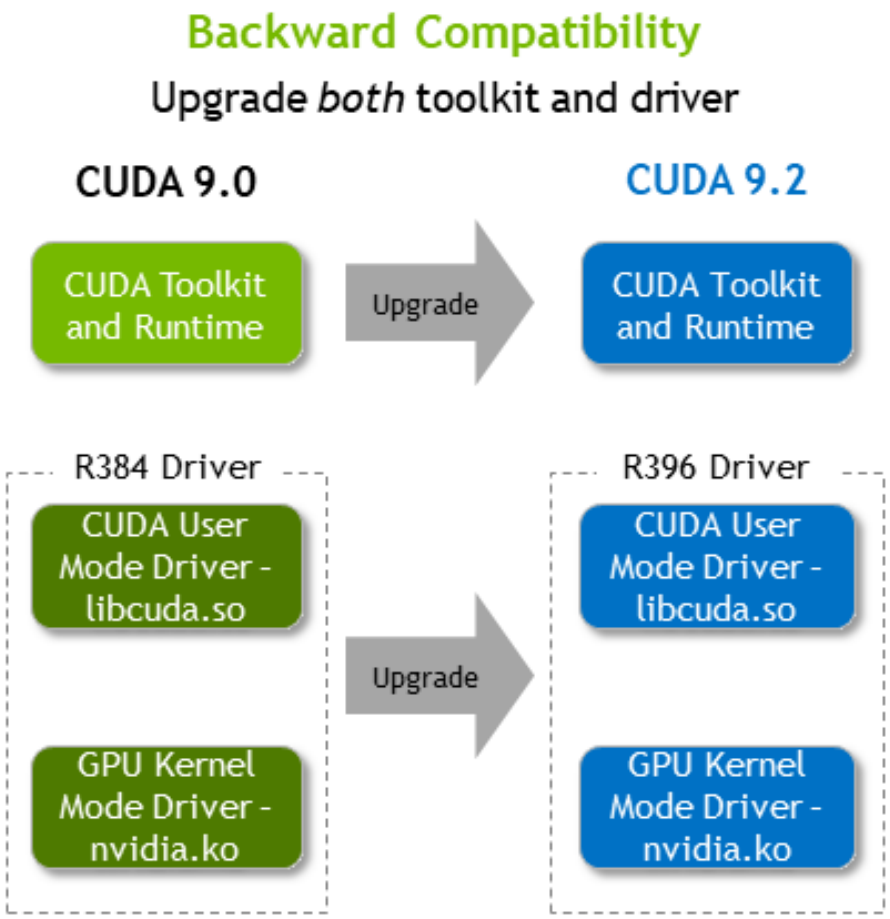

Figure 1 Backward Compatibility

The CUDA toolkit consists of two main components:

- the development libraries (including the CUDA runtime), and
- the driver components. The driver components are further separated into two categories:
  - the kernel mode components (the 'display' driver), and
  - the user mode components (the CUDA driver, the OpenGL driver, etc.).

Starting with **CUDA 10.0**, NVIDIA introduced a new forward-compatible upgrade path that allows the kernel mode components on the system to remain untouched, while the CUDA driver is upgraded. See Figure 2. This allows the use of newer toolkits on existing system installations, providing improvements and features of the latest CUDA while minimizing the risks associated with new driver deployments.

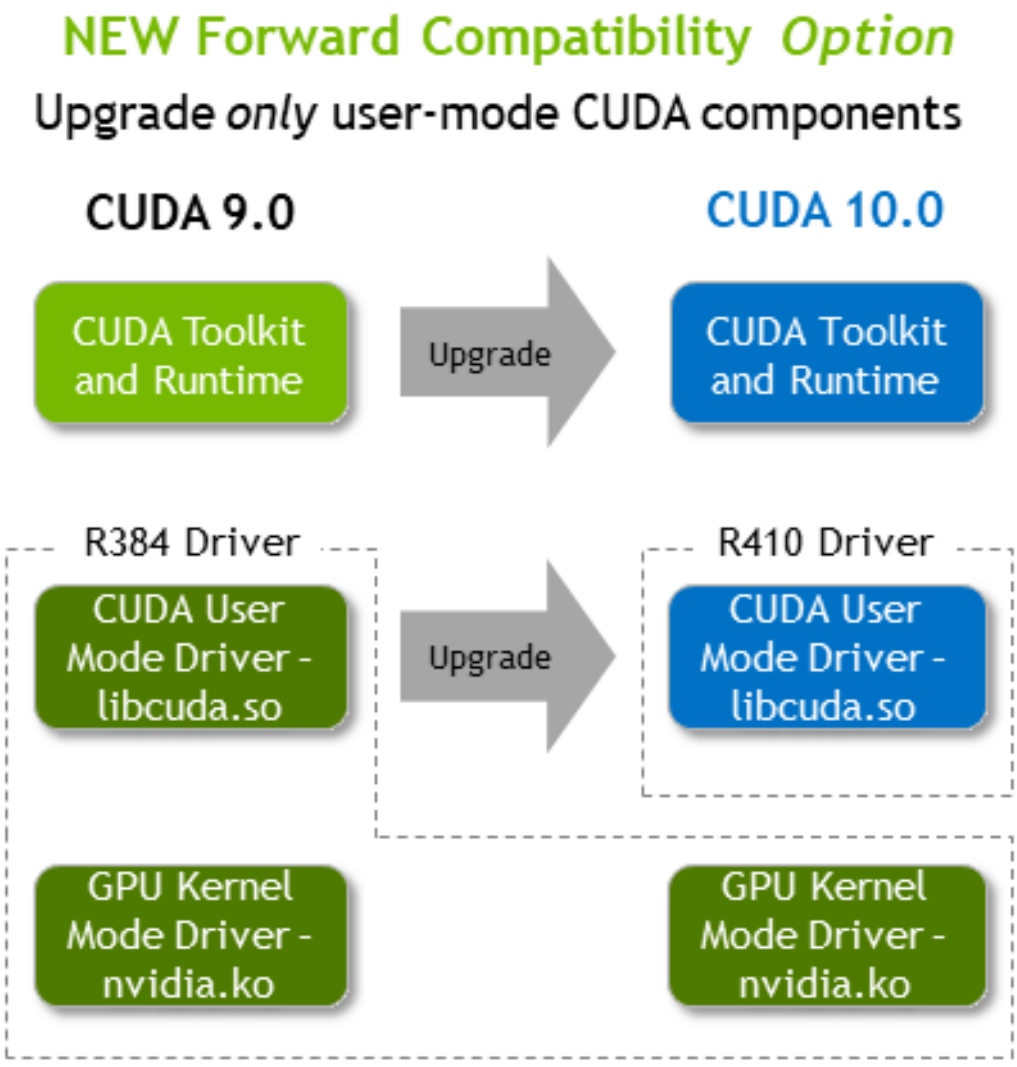

Figure 2 Forward Compatibility

2.1. Source Compatibility

We define source compatibility as a set of guarantees provided by the library, where a well-formed application built against a specific version of the library (using the SDK) will continue to build and run without errors when a newer version of the SDK is installed.

Both the CUDA driver and the CUDA runtime are **not** source compatible across the different SDK releases. APIs can be deprecated and removed, requiring changes to the application. Developers are notified through deprecation and documentation mechanisms of any current or upcoming changes. Although the driver APIs can change, they are versioned, and their symbols persist across releases to maintain binary compatibility.

2.2. Binary Compatibility

We define binary compatibility as a set of guarantees provided by the library, where an application targeting the said library will continue to work when dynamically linked against a different version of the library.

The CUDA driver (`libcuda.so`) provides binary backward compatibility. For example, an application built against the CUDA 3.2 SDK will continue to function even on today's driver stack. On the other hand, the CUDA runtime does **not** provide these guarantees. If your application dynamically links against the CUDA 9.2 runtime, it will only work in the presence of the dynamic 9.2 CUDA runtime on the target system. If the runtime was statically linked into the application, it will function on a minimum supported driver, and any driver beyond.

The Table 1 below summarizes the minimum driver required for a specific version of the CUDA runtime/toolkit. For convenience, today the NVIDIA driver can be installed as part of the CUDA Toolkit installation.

Table 1 CUDA Toolkit and Compatible Driver Versions

CUDA Toolkit	Linux x86_64 Driver Version
CUDA 11.0 (11.0.171)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81
CUDA 8.0 (8.0.61 GA2)	>= 375.26
CUDA 8.0 (8.0.44)	>= 367.48

CUDA 7.5 (7.5.16)	>= 352.31
CUDA 7.0 (7.0.28)	>= 346.46

Chapter 3. SUPPORT

3.1. Hardware Support

The current hardware support is shown in Table 2.

Table 2 Compute Capability Support

Hardware Generation	Compute Capability	Driver 384.111+	Driver 410.48+	Driver 418.40.04+	Driver 440.33.01+	Driver 450.36.06+
Ampere	8.0	No	No	No	No	Yes
Turing	7.5	No	Yes	Yes	Yes	Yes
Volta	7.x	Yes	Yes	Yes	Yes	Yes
Pascal	6.x	Yes	Yes	Yes	Yes	Yes
Maxwell	5.x	Yes	Yes	Yes	Yes	Yes
Kepler	3.x	Yes	Yes	Yes	Yes	Yes
Fermi	2.x	No	No	No	No	No

3.2. Forward-Compatible Upgrade Path

The new upgrade path for the CUDA driver is meant to ease the management of large production systems for enterprise customers. As such, the supported HW (hardware) for this new upgrade path is limited to **Tesla GPU products**. It's important to note that HW support is defined by the kernel mode driver and as such, newer CUDA drivers on their own will not enable new HW support. Refer to Hardware Support for which hardware is supported by your system.

3.3. CUDA Application Compatibility

With the CUDA compatibility platform, applications built with newer CUDA toolkits can be supported on specific enterprise Tesla driver branches. The Table 3 below shows the support matrix when using the CUDA compatibility platform.

Table 3 CUDA Application Compatibility Support Matrix

Linux x86_64 Tesla driver	CUDA 10.0 Compatibility	CUDA 10.1 Compatibility	CUDA 10.2 Compatibility	CUDA 11.0 Compatibility
450.36.06+	N/A	N/A	N/A	Compatible
440.33.01+	N/A	N/A	Compatible	Compatible
418.40.04+	N/A	Compatible	Compatible	Compatible
410.72+	Compatible	Compatible	Compatible	Not Compatible
396.26+	Not Compatible	Compatible	Compatible	Not Compatible
390.46+	Not Compatible	Not Compatible	Not Compatible	Not Compatible
384.111+	Compatible	Compatible	Not Compatible	Not Compatible

3.4. Feature Support

There are specific features in the CUDA driver that require kernel-mode support and will only work with a newer kernel mode driver. A few features depend on other user-mode components and are therefore also unsupported. See Table 4.

Table 4 Forward-Compatible Feature-Driver Support Matrix

CUDA Compatibility Driver	CUDA - OpenGL/Vulkan Interop	POWER9 ATS	cuMemMap* set of functionalities
System Base Installation: 440 (>=.33.01) Driver			
450 (>=.36.06)	No	Yes [2]	Yes [1]
440 (>=.33.01)	No	No	Yes [1]
418 (>=.40.04)	No	No	No
System Base Installation: 418 (>=.XX) Driver			
450 (>=.36.06)	No	Yes [2]	Yes [1]
440 (>=.33.01)	No	No	Yes [1]
418 (>=.40.04)	No	No	No
System Base Installation: 384 (>=.111) Driver			
440 (>=.33.01)	No	No	Yes [1]
418 (>=.40.04)	No	No	No

410 (>=.48)	No	No	No
384 (>=.111)	N/A	N/A	No

[1] This relies on CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_POSIX_FILE_DESCRIPTOR_SUPPORTED and CU_DEVICE_ATTRIBUTE_VIRTUAL_ADDRESS_MANAGEMENT_SUPPORTED, which should be queried if you intend to use the full range of this functionality.

[2] Supported on Red Hat Enterprise Linux operating system version 8.1 or higher.

In addition to the CUDA driver and certain compiler components, there are other drivers in the system installation stack (e.g. OpenCL) that remain on the old version. The forward-compatible upgrade path is for CUDA only.

Chapter 4.
CUDA COMPATIBILITY PLATFORM

The CUDA Compatibility Platform files are meant as additions to the existing system installation and not replacements for those files. The platform consists of:

- **`libcuda.so.*`** - the CUDA Driver
- **`libnvidia-ptxjitcompiler.so.*`** - the JIT (just-in-time) compiler for PTX files

For ease of deployment, a new package is available in the local installers or the CUDA network repositories provided by NVIDIA:

- **`cuda-compat-11.0`**

This package provides the CUDA Compatibility Platform files necessary to run 11.0 CUDA APIs. The package can be installed using Linux package managers such as **`apt`** or **`yum`**. For example, on an Ubuntu 16.04 system, run the following command when using the **`apt`** package manager:

```
$ sudo apt-get install cuda-compat-11-0
```

The package is installed to the versioned toolkit location typically found in the **`/usr/local/cuda-11.0/`** directory (or replace 11.0 with the appropriate version).

This package only provides the files, and does not configure the system. These files should be kept together as the CUDA driver depends on the fatbinary loader that is of the same version.

These files can also be extracted from the appropriate Tesla Recommended Driver installer (.run) available in NVIDIA driver downloads. To do this:

1. Download the latest Tesla Recommended Driver, and extract the .run file using option **`-x`**.
2. Copy the three CUDA Compatibility Platform files, listed at the start of this section, into a user- or root-created directory.
3. Then follow your system's guidelines for making sure that the system linker picks up the new libraries.

Chapter 5. DEPLOYMENT CONSIDERATIONS

Consider a cluster of 500+ multi-gpu servers running bare-metal in support of 50-1500 users, running a variety of DL and HPC workloads. This system is scheduled in a classical manner (for example, using SLURM or LSF) with resources being allocated within a **`cgroup`**, sometimes in exclusive mode. The cluster can be homogeneous with respect to node configurations, or heterogeneous, including some nodes of each of a few generations of Tesla GPUs - V100 16GB or 32GB, P100, P40, K80, and/or K40.

With the introduction of the CUDA 10 Compatibility Platform (regardless of how the files are packaged - run/deb/rpm/etc.), the site operator must pay close attention to where these libraries are placed. It could potentially be part of the disk image (i.e., more closely tied to the kernel and driver installations), or part of one of the modules (i.e., more closely tied to the toolkit and the libraries).

5.1. Disk Image

In this case the compatibility files are located somewhere on the boot image alongside the existing system files. The exact path is not important, but the files should remain together, and be resolvable by the dynamic loader. This could be accomplished by having the system's loader automatically pick them up (e.g. **`ld.so.conf`**), through the use of RPATH, or through a more manual process of documentation (users of the new toolkit must manually set **`LD_LIBRARY_PATH`**).

5.2. Modules System

It is common for the users to request any of the several CUDA Toolkit versions in the same way they might request any of several versions of numerous other system libraries or compiler toolchains.

A common mechanism to enable this is the Modules System, wherein the admin, or the user, sets up 'module' scripts for each version of each package, and then the user can execute a command like '**`module load cuda/10.0`**'.

Often the loading of various module versions will be scripted with the application such that each application picks up exactly the versions of its dependencies that it needs, even if other versions would have been available for other applications to choose from.

If the components from the CUDA Compatibility Platform are placed such that they are chosen by the module load system, it is important to note the limitations of this new path – namely, only certain major versions of the system driver stack, only NVIDIA Tesla GPUs are supported, and only in a forward compatible manner (i.e. an older **`libcuda.so`** will not work on newer base systems).

It is therefore recommended that the module load script be aware of these limitations, and proactively query the system for whether the compatibility platform can be used. In the cases where it cannot use the compatibility platform, a fallback path to the default system's installed CUDA driver can provide a more consistent experience. See Miscellaneous for performing this task with RPATH.

5.3. Miscellaneous

After the system is fully upgraded (the display driver and the CUDA driver) to a newer base installation, the compatibility platform files should be removed as they are no longer necessary and will not function.

A common way of ensuring that the new CUDA Compatibility Platform files are picked up, is by using RPATH. This ensures that during the compilation process of the application, the runtime search path is hard-coded into the executable.

For example, the RPATH can be set to **`/usr/compat`** and this setting is used throughout the cluster. Each individual system then configures **`/usr/compat`** to the appropriate directory – if forward compatibility is desired/allowed, this can contain the (latest) CUDA Compatibility Platform files.

In order to workaround the lack of automatic fallback when the system driver leapfrogs the CUDA Compatibility Platform files, or in case of a non-supported HW configuration, the **`/usr/compat`** can point to the Tesla Recommended Driver for those systems instead. This way a single, consistent, path is used throughout the entire cluster.