

NVIDIA CUDA Installation Guide for Linux

Installation and Verification on Linux Systems

DU-05347-001_v11.4 | July 2021

Table of Contents

Chapter 1. Introduction 1
1.1. System Requirements 1
1.2. About This Document 3
Chapter 2. Pre-installation Actions 4
2.1. Verify You Have a CUDA-Capable GPU 4
2.2. Verify You Have a Supported Version of Linux 4
2.3. Verify the System Has gcc Installed 5
2.4. Verify the System has the Correct Kernel Headers and Development Packages Installed 5
2.5. Install MLNX_OFED 7
2.6. Choose an Installation Method 7
2.7. Download the NVIDIA CUDA Toolkit 7
2.8. Handle Conflicting Installation Methods 8
Chapter 3. Package Manager Installation 9
3.1. Overview 9
3.2. RHEL7/CentOS7 10
3.3. RHEL8/CentOS8 11
3.4. Fedora 12
3.5. SLES 13
3.6. OpenSUSE 13
3.7. WSL 14
3.8. Ubuntu 14
3.9. Debian 15
3.10. Additional Package Manager Capabilities 16
3.10.1. Available Packages 16
3.10.2. Package Upgrades 16
3.10.3. Meta Packages 17
Chapter 4. Driver Installation 18
Chapter 5. Precompiled Streams 19
5.1. Precompiled Streams Support Matrix 20
5.2. Modularity Profiles 20
Chapter 6. Kickstart Installation 22
6.1. RHEL8/CentOS8 22
Chapter 7. Runfile Installation 23

7.1. Overview....23
7.2. Installation....23
7.3. Disabling Nouveau....25
7.3.1. Fedora....25
7.3.2. RHEL/CentOS....25
7.3.3. OpenSUSE....25
7.3.4. SLES....26
7.3.5. WSL....26
7.3.6. Ubuntu....26
7.3.7. Debian....26
7.4. Device Node Verification....26
7.5. Advanced Options....27
7.6. Uninstallation....28
Chapter 8. CUDA Cross-Platform Environment....29
8.1. CUDA Cross-Platform Installation....29
8.2. CUDA Cross-Platform Samples....30
TARGET_ARCH....30
TARGET_OS....30
TARGET_FS....30
Cross Compiling to Embedded ARM architectures....31
Copying Libraries....31
Chapter 9. Post-installation Actions....32
9.1. Mandatory Actions....32
9.1.1. Environment Setup....32
9.1.2. POWER9 Setup....32
9.2. Recommended Actions....33
9.2.1. Install Persistence Daemon....33
9.2.2. Install Writable Samples....34
9.2.3. Verify the Installation....34
9.2.3.1. Verify the Driver Version....34
9.2.3.2. Compiling the Examples....34
9.2.3.3. Running the Binaries....34
9.2.4. Install Nsight Eclipse Plugins....36
9.3. Optional Actions....36
9.3.1. Install Third-party Libraries....36
9.3.2. Install the source code for cuda-gdb....37
9.3.3. Select the Active Version of CUDA....37
Chapter 10. Advanced Setup....38

Chapter 11. Frequently Asked Questions..... 41
How do I install the Toolkit in a different location?.....41
Why do I see "nvcc: No such file or directory" when I try to build a CUDA application?.....41
Why do I see "error while loading shared libraries: <lib name>: cannot open shared object file: No such file or directory" when I try to run a CUDA application that uses a CUDA library?..... 42
Why do I see multiple "404 Not Found" errors when updating my repository meta-data on Ubuntu?..... 42
How can I tell X to ignore a GPU for compute-only use?.....42
Why doesn't the cuda-repo package install the CUDA Toolkit and Drivers?..... 42
How do I get CUDA to work on a laptop with an iGPU and a dGPU running Ubuntu14.04?.....43
What do I do if the display does not load, or CUDA does not work, after performing a system update?.....43
How do I install a CUDA driver with a version less than 367 using a network repo?..... 43
How do I install an older CUDA version using a network repo?..... 44
Chapter 12. Additional Considerations..... 45
Chapter 13. Removing CUDA Toolkit and Driver.....46

Chapter 1. Introduction

CUDA® is a parallel computing platform and programming model invented by NVIDIA®. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA was developed with several design goals in mind:

- Provide a small set of extensions to standard programming languages, like C, that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources.

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

This guide will show you how to install and check the correct operation of the CUDA development tools.

1.1. System Requirements

To use CUDA on your system, you will need the following installed:

- CUDA-capable GPU
- A supported version of Linux with a gcc compiler and toolchain
- NVIDIA CUDA Toolkit (available at https://developer.nvidia.com/cuda-downloads)

The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for this CUDA Toolkit release.

The following table lists the supported Linux distributions. Please review the footnotes associated with the table.

Table 1. Native Linux Distribution Support in CUDA 11.4

Distribution	Kernel[1]	Default GCC	GLIBC	GCC[2,3]	ICC[3]	PGI[3]	XLC[3]	CLANG	Arm C/C++
x86_64									
RHEL 8.y (y <= 4)	4.18.0-240	8.3.1	2.28	11	2021	19.x, 20.x	NO	12	NO
CentOS 8.y (y <= 4)	4.18	8.3.1	2.28						
RHEL 7.y (y <= 9)	3.10.0-1136	6.x	2.17						
CentOS 7.y (y <= 9)	3.10.0-1136	6.x	2.17						
OpenSUSE Leap 15.y (y <= 3)	5.3.18-57	7.5.0	2.31						
SUSE SLES 15.y (y <= 3)	5.3.18-57	7.5.0	2.31						
Ubuntu 20.04.2 LTS	5.4.0	9.3.0	2.31						
Ubuntu 18.04.z (z <= 5) LTS	5.3.0	7.5.0	2.27						
Debian 10.y (y<=8)	4.19.0	8.3.0	2.28						
Fedora 34	5.11	11	2.33						
Arm64 sbsa[4]									
RHEL 8.4	4.18.0-305	8.4.1	2.28	11	NO	19.x, 20.x	NO	11.0	21.3
SUSE SLES 15.y (y <= 2)	5.3.18-59.5	7.5.0	2.31						
Ubuntu 20.04.2.LTS	5.4.0-77	9.3.0	2.31						
Arm64 Jetson[4]									
Ubuntu 18.04.z (z <= 4)	4.9.201	7.5.0	2.27	10.2	NO	19.x, 20.x	13.1.x, 16.1.x	11.0	NO
POWER 9[4]									
RHEL 8.y (y <= 4)	4.18.0-305	8.4.1	2.28	11	NO	19.x, 20.x	16.1.x	11.0	NO

(1) The following notes apply to the kernel versions supported by CUDA:

- For specific kernel versions supported on Red Hat Enterprise Linux (RHEL), visit https://access.redhat.com/articles/3078.
- For a list of kernel versions including the release dates for SUSE Linux Enterprise Server (SLES) is available at https://wiki.microfocus.com/index.php/SUSE/SLES/Kernel_versions.
- For Ubuntu LTS on x86-64, both the HWE kernel (e.g. 5.x for 18.04) and the server LTS kernel (e.g. 4.15.x for 18.04) are supported in CUDA 11.4. Visit https://wiki.ubuntu.com/Kernel/Support for more information.

(2) Note that starting with CUDA 11.0, the minimum recommended GCC compiler is at least GCC 6 due to C++11 requirements in CUDA libraries e.g. cuFFT and CUB. On distributions such as RHEL 7 or CentOS 7 that may use an older GCC toolchain by default, it is recommended to use a newer GCC toolchain with CUDA 11.0. Newer GCC toolchains are available with the Red Hat Developer Toolset. For platforms that ship a compiler version older than GCC 6 by default, linking to static cuBLAS and cuDNN using the default compiler is not supported.

(3) Minor versions of the following compilers listed: of GCC, ICC, PGI and XLC, as host compilers for `nvcc` are supported.

(4) Only Tesla V100 and T4 GPUs are supported for CUDA 11.4 on Arm64 (`aarch64`) POWER9 (`ppc64le`).

1.2. About This Document

This document is intended for readers familiar with the Linux environment and the compilation of C programs from the command line. You do not need previous experience with CUDA or experience with parallel computation. Note: This guide covers installation only on systems with X Windows installed.



Note: Many commands in this document might require *superuser* privileges. On most distributions of Linux, this will require you to log in as root. For systems that have enabled the sudo package, use the sudo prefix for all necessary commands.

Chapter 2. Pre-installation Actions

Some actions must be taken before the CUDA Toolkit and Driver can be installed on Linux:

- Verify the system has a CUDA-capable GPU.
- Verify the system is running a supported version of Linux.
- Verify the system has gcc installed.
- Verify the system has the correct kernel headers and development packages installed.
- Download the NVIDIA CUDA Toolkit.
- Handle conflicting installation methods.



Note: You can override the install-time prerequisite checks by running the installer with the -override flag. Remember that the prerequisites will still be required to use the NVIDIA CUDA Toolkit.

2.1. Verify You Have a CUDA-Capable GPU

To verify that your GPU is CUDA-capable, go to your distribution's equivalent of System Properties, or, from the command line, enter:

```
$ lspci | grep -i nvidia
```

If you do not see any settings, update the PCI hardware database that Linux maintains by entering `update-pciids` (generally found in `/sbin`) at the command line and rerun the previous `lspci` command.

If your graphics card is from NVIDIA and it is listed in https://developer.nvidia.com/cuda-gpus, your GPU is CUDA-capable.

The Release Notes for the CUDA Toolkit also contain a list of supported products.

2.2. Verify You Have a Supported Version of Linux

The CUDA Development Tools are only supported on some specific distributions of Linux. These are listed in the CUDA Toolkit release notes.

To determine which distribution and release number you're running, type the following at the command line:

```
$ uname -m && cat /etc/*release
```

You should see output similar to the following, modified for your particular system:

```
x86_64
Red Hat Enterprise Linux Workstation release 6.0 (Santiago)
```

The `x86_64` line indicates you are running on a 64-bit system. The remainder gives information about your distribution.

2.3. Verify the System Has gcc Installed

The `gcc` compiler is required for development using the CUDA Toolkit. It is not required for running CUDA applications. It is generally installed as part of the Linux installation, and in most cases the version of gcc installed with a supported version of Linux will work correctly.

To verify the version of gcc installed on your system, type the following on the command line:

```
$ gcc --version
```

If an error message displays, you need to install the *development tools* from your Linux distribution or obtain a version of `gcc` and its accompanying toolchain from the Web.

2.4. Verify the System has the Correct Kernel Headers and Development Packages Installed

The CUDA Driver requires that the kernel headers and development packages for the running version of the kernel be installed at the time of the driver installation, as well whenever the driver is rebuilt. For example, if your system is running kernel version 3.17.4-301, the 3.17.4-301 kernel headers and development packages must also be installed.

While the Runfile installation performs no package validation, the RPM and Deb installations of the driver will make an attempt to install the kernel header and development packages if no version of these packages is currently installed. However, it will install the latest version of these packages, which may or may not match the version of the kernel your system is using. **Therefore, it is best to manually ensure the correct version of the kernel headers and development packages are installed prior to installing the CUDA Drivers, as well as whenever you change the kernel version.**

The version of the kernel your system is running can be found by running the following command:

```
$ uname -r
```

This is the version of the kernel headers and development packages that must be installed prior to installing the CUDA Drivers. This command will be used multiple times below to specify the version of the packages to install. Note that below are the common-case scenarios

for kernel usage. More advanced cases, such as custom kernel branches, should ensure that their kernel headers and sources match the kernel build they are running.



Note: If you perform a system update which changes the version of the linux kernel being used, make sure to rerun the commands below to ensure you have the correct kernel headers and kernel development packages installed. Otherwise, the CUDA Driver will fail to work with the new kernel.

RHEL7/CentOS7

The kernel headers and development packages for the currently running kernel can be installed with:

```
$ sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r)
```

Fedora/RHEL8/CentOS8

The kernel headers and development packages for the currently running kernel can be installed with:

```
$ sudo dnf install kernel-devel-$(uname -r) kernel-headers-$(uname -r)
```

OpenSUSE/SLES

The kernel development packages for the currently running kernel can be installed with:

```
$ sudo zypper install -y kernel-<variant>-devel=<version>
```

To run the above command, you will need the variant and version of the currently running kernel. Use the output of the `uname` command to determine the currently running kernel's variant and version:

```
$ uname -r
3.16.6-2-default
```

In the above example, the variant is `default` and version is `3.16.6-2`.

The kernel development packages for the default kernel variant can be installed with:

```
$ sudo zypper install -y kernel-default-devel=$(uname -r | sed 's/\-default//')
```

WSL

This section does not need to be performed for WSL.

Ubuntu

The kernel headers and development packages for the currently running kernel can be installed with:

```
$ sudo apt-get install linux-headers-$(uname -r)
```

2.5. Install MLNX_OFED

GPUDirectStorage (GDS) packages can be installed using the CUDA packaging guide. In order to use GDS you must install the CUDA package and MLNX_OFED package. Follow the instructions in MLNX_OFED Requirements and Installation.

GDS is supported in two different modes: GDS (default/full perf mode) and Compatibility mode. Installation instructions for them differ slightly. Compatibility mode is the only mode that is supported on certain distributions due to software dependency limitations.

Full GDS support is restricted to the following Linux distros:

- Ubuntu 18.04, 20.04
- RHEL 8.3, RHEL 8.4

2.6. Choose an Installation Method

The CUDA Toolkit can be installed using either of two different installation mechanisms: distribution-specific packages (RPM and Deb packages), or a distribution-independent package (runfile packages). The distribution-independent package has the advantage of working across a wider set of Linux distributions, but does not update the distribution's native package management system. The distribution-specific packages interface with the distribution's native package management system. It is recommended to use the distribution-specific packages, where possible.



Note: Standalone installers are not provided for architectures other than the x86_64 release. For both native as well as cross development, the toolkit must be installed using the distribution-specific installer. See the CUDA Cross-Platform Installation section for more details.

2.7. Download the NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit is available at https://developer.nvidia.com/cuda-downloads.

Choose the platform you are using and download the NVIDIA CUDA Toolkit

The CUDA Toolkit contains the CUDA driver and tools needed to create, build and run a CUDA application as well as libraries, header files, CUDA samples source code, and other resources.

Download Verification

The download can be verified by comparing the MD5 checksum posted at https://developer.download.nvidia.com/compute/cuda/11.3.1/docs/sidebar/md5sum.txt with that of the downloaded file. If either of the checksums differ, the downloaded file is corrupt and needs to be downloaded again.

To calculate the MD5 checksum of the downloaded file, run the following:

```
$ md5sum <file>
```

2.8. Handle Conflicting Installation Methods

Before installing CUDA, any previously installations that could conflict should be uninstalled. This will not affect systems which have not had CUDA installed previously, or systems where the installation method has been preserved (RPM/Deb vs. Runfile). See the following charts for specifics.

Table 2. CUDA Toolkit Installation Compatibility Matrix

		Installed Toolkit Version == X.Y		Installed Toolkit Version != X.Y	
		RPM/Deb	run	RPM/Deb	run
Installing Toolkit Version X.Y	RPM/Deb	No Action	Uninstall Run	No Action	No Action
	run	Uninstall RPM/Deb	Uninstall Run	No Action	No Action

Table 3. NVIDIA Driver Installation Compatibility Matrix

		Installed Driver Version == X.Y		Installed Driver Version != X.Y	
		RPM/Deb	run	RPM/Deb	run
Installing Driver Version X.Y	RPM/Deb	No Action	Uninstall Run	No Action	Uninstall Run
	run	Uninstall RPM/Deb	No Action	Uninstall RPM/Deb	No Action

Use the following command to uninstall a Toolkit runfile installation:

```
$ sudo /usr/local/cuda-X.Y/bin/cuda-uninstaller
```

Use the following command to uninstall a Driver runfile installation:

```
$ sudo /usr/bin/nvidia-uninstall
```

Use the following commands to uninstall a RPM/Deb installation:

```
$ sudo dnf remove <package_name>                    # RHEL8/CentOS8
$ sudo yum remove <package_name>                    # RHEL7/CentOS7
$ sudo dnf remove <package_name>                    # Fedora
$ sudo zypper remove <package_name>                 # OpenSUSE/SLES
$ sudo apt-get --purge remove <package_name>        # Ubuntu
```

Chapter 3. Package Manager Installation

Basic instructions can be found in the Quick Start Guide. Read on for more detailed instructions.

3.1. Overview

The Package Manager installation interfaces with your system's package management system. When using RPM or Deb, the downloaded package is a repository package. Such a package only informs the package manager where to find the actual installation packages, but will not install them.

If those packages are available in an online repository, they will be automatically downloaded in a later step. Otherwise, the repository package also installs a local repository containing the installation packages on the system. Whether the repository is available online or installed locally, the installation procedure is identical and made of several steps.

Distribution-specific instructions detail how to install CUDA:

- RHEL7/CentOS7
- RHEL8/CentOS8
- Fedora
- SLES
- OpenSUSE
- Ubuntu
- Debian

Finally, some helpful package manager capabilities are detailed.

These instructions are for native development only. For cross-platform development, see the CUDA Cross-Platform Environment section.



Note: The package "cuda-core" has been deprecated in CUDA 9.1. Please use "cuda-compiler" instead.

3.2. RHEL7/CentOS7

1. Perform the pre-installation actions.
2. **Satisfy third-party package dependency**
   - **Satisfy DKMS dependency**: The NVIDIA driver RPM packages depend on other external packages, such as DKMS and `libvdpau`. Those packages are only available on third-party repositories, such as EPEL. Any such third-party repositories must be added to the package manager repository database before installing the NVIDIA driver RPM packages, or missing dependencies will prevent the installation from proceeding.

     To enable EPEL:

     ```
     $ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
     ```
   - **Enable optional repos**:

     On **RHEL 7 Linux** only, execute the following steps to enable optional repositories.

     - **On x86_64 workstation:**

       ```
       $ subscription-manager repos --enable=rhel-7-workstation-optional-rpms
       ```
     - **On POWER9 system:**

       ```
       $ subscription-manager repos --enable=rhel-7-for-power-9-optional-rpms
       ```
     - **On x86_64 server:**

       ```
       $ subscription-manager repos --enable=rhel-7-server-optional-rpms
       ```
3. **Address custom xorg.conf, if applicable**

   The driver relies on an automatically generated `xorg.conf` file at `/etc/X11/xorg.conf`. If a custom-built `xorg.conf` file is present, this functionality will be disabled and the driver may not work. You can try removing the existing `xorg.conf` file, or adding the contents of `/etc/X11/xorg.conf.d/00-nvidia.conf` to the `xorg.conf` file. The `xorg.conf` file will most likely need manual tweaking for systems with a non-trivial GPU configuration.
4. **Install repository meta-data**

   When installing using the local repo:

   ```
   $ sudo rpm --install cuda-repo-<distro>-<version>.<architecture>.rpm
   ```

   When installing using the network repo:

   ```
   $ sudo yum-config-manager --add-repo https://developer.download.nvidia.com/compute/cuda/repos/rhel7/x86_64/cuda-rhel7.repo
   ```
5. **Clean Yum repository cache**

   ```
   $ sudo yum clean expire-cache
   ```
6. **Install CUDA**

   ```
   $ sudo yum install nvidia-driver-latest-dkms
   $ sudo yum install cuda
   $ sudo yum install cuda-drivers
   ```
7. **Add libcuda.so symbolic link, if necessary**

The libcuda.so library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use libcuda.so, it may be useful to add a symbolic link from libcuda.so in the `/usr/lib{,64}` directory.

8. Perform the post-installation actions.

3.3. RHEL8/CentOS8

1. Perform the pre-installation actions.
2. **Satisfy third-party package dependency**

   - **Satisfy DKMS dependency**: The NVIDIA driver RPM packages depend on other external packages, such as DKMS and `libvdpau`. Those packages are only available on third-party repositories, such as EPEL. Any such third-party repositories must be added to the package manager repository database before installing the NVIDIA driver RPM packages, or missing dependencies will prevent the installation from proceeding.

     To enable EPEL:

     ```
     $ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
     ```

   - **Enable optional repos**:

     On **RHEL 8 Linux** only, execute the following steps to enable optional repositories.

     - **On x86_64 systems:**

       ```
       $ subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms
       $ subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
       $ subscription-manager repos --enable=codeready-builder-for-rhel-8-x86_64-rpms
       ```

     - **On POWER9 systems:**

       ```
       $ subscription-manager repos --enable=rhel-8-for-ppc64le-appstream-rpms
       $ subscription-manager repos --enable=rhel-8-for-ppc64le-baseos-rpms
       $ subscription-manager repos --enable=codeready-builder-for-rhel-8-ppc64le-rpms
       ```

3. **Address custom xorg.conf, if applicable**

   The driver relies on an automatically generated `xorg.conf` file at `/etc/X11/xorg.conf`. If a custom-built `xorg.conf` file is present, this functionality will be disabled and the driver may not work. You can try removing the existing `xorg.conf` file, or adding the contents of `/etc/X11/xorg.conf.d/00-nvidia.conf` to the `xorg.conf` file. The `xorg.conf` file will most likely need manual tweaking for systems with a non-trivial GPU configuration.

4. **Install repository meta-data**

   When installing using the local repo:

   ```
   $ sudo rpm --install cuda-repo-<distro>-<version>.<architecture>.rpm
   ```

   When installing using the network repo:

   ```
   $ sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/cuda/repos/rhel8/<architecture>/cuda-rhel8.repo
   ```

5. **Clean Yum repository cache**

   ```
   $ sudo yum clean expire-cache
   ```

   To include all GDS packages:

```
$ sudo dnf install nvidia-gds
```

6. **Install CUDA**

```
$ sudo dnf clean expire-cache
$ sudo dnf module install nvidia-driver:latest-dkms
$ sudo dnf install cuda
```

7. **Add libcuda.so symbolic link, if necessary**

   The libcuda.so library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use libcuda.so, it may be useful to add a symbolic link from libcuda.so in the `/usr/lib{,64}` directory.

8. Perform the post-installation actions.

3.4. Fedora

1. Perform the pre-installation actions.
2. **Address custom xorg.conf, if applicable**

   The driver relies on an automatically generated xorg.conf file at `/etc/X11/xorg.conf`. If a custom-built xorg.conf file is present, this functionality will be disabled and the driver may not work. You can try removing the existing xorg.conf file, or adding the contents of `/etc/X11/xorg.conf.d/00-nvidia.conf` to the xorg.conf file. The xorg.conf file will most likely need manual tweaking for systems with a non-trivial GPU configuration.

3. **Install repository meta-data**

   When installing using the local repo:

```
$ sudo rpm --install cuda-repo-fedora33-<version>.x86_64.rpm
```

   When installing using the network repo:

```
$ sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/cuda/repos/fedora33/x86_64/cuda-fedora33.repo
```

4. **Clean DNF repository cache**

```
$ sudo dnf clean expire-cache
```

5. **Install CUDA**

```
$ sudo dnf module install nvidia-driver:latest-dkms
$ sudo dnf install cuda
```

   The CUDA driver installation may fail if the RPMFusion non-free repository is enabled. In this case, CUDA installations should temporarily disable the RPMFusion non-free repository:

```
$ sudo dnf --disablerepo="rpmfusion-nonfree*" install cuda
```

   It may be necessary to rebuild the grub configuration files, particularly if you use a non-default partition scheme. If so, then run this below command, and reboot the system:

```
$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

   Remember to reboot the system.

6. **Add libcuda.so symbolic link, if necessary**

   The libcuda.so library is installed in the `/usr/lib{,64}/nvidia` directory. For pre-existing projects which use libcuda.so, it may be useful to add a symbolic link from libcuda.so in the `/usr/lib{,64}` directory.

7. Perform the post-installation actions.

3.5. SLES

1. Perform the pre-installation actions.
2. On SLES12 SP4, install the Mesa-libgl-devel Linux packages before proceeding. See Mesa-libGL-devel.
3. **Install repository meta-data**

   When installing using the local repo:

   ```
   $ sudo rpm --install cuda-repo-sles15-<version>.<architecture>.rpm
   ```

   When installing using the network repo:

   ```
   $ sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   sles15/<architecture>/cuda-sles15.repo
   ```
4. **Refresh Zypper repository cache**

   ```
   $ sudo SUSEConnect --product PackageHub/15/<architecture>
   $ sudo zypper refresh
   ```
5. **Install CUDA**

   ```
   $ sudo zypper install cuda
   ```
6. **Add the user to the video group**

   ```
   $ sudo usermod -a -G video <username>
   ```
7. **Install CUDA Samples GL dependencies**

   The CUDA Samples package on SLES does not include dependencies on GL and X11 libraries as these are provided in the SLES SDK. These packages must be installed separately, depending on which samples you want to use.
8. Perform the post-installation actions.

3.6. OpenSUSE

1. Perform the pre-installation actions.
2. **Install repository meta-data**

   When installing using the local repo:

   ```
   $ sudo rpm --install cuda-repo-opensuse15-<version>.x86_64.rpm
   ```

   When installing using the network repo:

   ```
   $ sudo zypper addrepo https://developer.download.nvidia.com/compute/cuda/repos/
   opensuse15/x86_64/cuda-opensuse15.repo
   ```
3. **Refresh Zypper repository cache**

   ```
   $ sudo zypper refresh
   ```
4. **Install CUDA**

   ```
   $ sudo zypper install cuda
   ```
5. **Add the user to the video group**

   ```
   $ sudo usermod -a -G video <username>
   ```

6. Perform the post-installation actions.

3.7. WSL

These instructions must be used if you are installing in a WSL environment. Do not use the Ubuntu instructions in this case.

1. Perform the pre-installation actions.
2. **Install repository meta-data**

   When installing using the local repo:

   ```
   $ sudo dpkg -i cuda-repo-<distro>_<version>_x86_64.deb
   ```

   When installing using the network repo:

   ```
   $ sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/
   cuda/repos/<distro>/x86_64/ /"
   ```

3. **Installing the CUDA public GPG key**

   When installing using the local repo:

   ```
   $ sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub
   ```

   When installing using the network repo:

   ```
   $ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/
   cuda/repos/<distro>/x86_64/7fa2af80.pub
   ```

   Pin file to prioritize CUDA repository:

   ```
   $ wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/x86_64/
   cuda-<distro>.pin
   $ sudo mv cuda-<distro>.pin /etc/apt/preferences.d/cuda-repository-pin-600
   ```

4. **Update the Apt repository cache**

   ```
   $ sudo apt-get update
   ```

5. **Install CUDA**

   ```
   $ sudo apt-get install cuda
   ```

6. Perform the post-installation actions.

3.8. Ubuntu

1. Perform the pre-installation actions.
2. **Install repository meta-data**

   ```
   $ sudo dpkg -i cuda-repo-<distro>_<version>_<architecture>.deb
   ```

3. **Install the CUDA public GPG key**

   When installing using the local repo:

   ```
   $ sudo apt-key add /var/cuda-repo-<distro>-<version>/7fa2af80.pub
   ```

   When installing using network repo on Ubuntu 20.04/18.04:

   ```
   $ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/
   cuda/repos/<distro>/<architecture>/7fa2af80.pub
   ```

When installing using network repo on Ubuntu 16.04:

```
$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/
cuda/repos/<distro>/<architecture>/7fa2af80.pub
```

Pin file to prioritize CUDA repository:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/
<architecture>/cuda-<distro>.pin
$ sudo mv cuda-<distro>.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

4. **Update the Apt repository cache**

```
$ sudo apt-get update
```

5. **Install CUDA**



Note: These two commands must be executed separately.

```
$ sudo apt-get install cuda
```

To include all GDS packages:

```
$ sudo apt-get install nvidia-gds
```

6. Perform the post-installation actions.

3.9. Debian

1. Perform the pre-installation actions.
2. **Install repository meta-data**

When installing using the local repo:

```
$ sudo dpkg -i cuda-repo-debian10_<version>_x86_64.deb
```

When installing using the network repo:

```
sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/cuda/
repos/debian10/x86_64/ /"
```

3. **Installing the CUDA public GPG key**

When installing using the local repo:

```
$ sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub
```

When installing using the network repo:

```
$ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/
cuda/repos/debian10/x86_64/7fa2af80.pub
```

4. **Enable the contrib repository:**

```
$ sudo add-apt-repository contrib
```

5. **Update the Apt repository cache**

```
$ sudo apt-get update
```

6. **Install CUDA**

```
$ sudo apt-get -y install cuda
```

7. Perform the post-installation actions.

3.10. Additional Package Manager Capabilities

Below are some additional capabilities of the package manager that users can take advantage of.

3.10.1. Available Packages

The recommended installation package is the `cuda` package. This package will install the full set of other CUDA packages required for native development and should cover most scenarios.

The `cuda` package installs all the available packages for native developments. That includes the compiler, the debugger, the profiler, the math libraries, and so on. For x86_64 patforms, this also include Nsight Eclipse Edition and the visual profilers. It also includes the NVIDIA driver package.

On supported platforms, the `cuda-cross-aarch64` and `cuda-cross-ppc64el` packages install all the packages required for cross-platform development to ARMv8 and POWER8, respectively. The libraries and header files of the target architecture's display driver package are also installed to enable the cross compilation of driver applications. The `cuda-cross-<arch>` packages do not install the native display driver.

The packages installed by the packages above can also be installed individually by specifying their names explicitly. The list of available packages be can obtained with:

```
$ yum --disablerepo="*" --enablerepo="cuda*" list available     # RedHat
$ dnf --disablerepo="*" --enablerepo="cuda*" list available     # Fedora
$ zypper packages -r cuda                                       # OpenSUSE & SLES
$ cat /var/lib/apt/lists/*cuda*Packages | grep "Package:"       # Ubuntu
```

3.10.2. Package Upgrades

The `cuda` package points to the latest stable release of the CUDA Toolkit. When a new version is available, use the following commands to upgrade the toolkit and driver:

```
$ sudo yum install cuda                                         # RHEL7
$ sudo dnf upgrade cuda                                         # Fedora/RHEL8
$ sudo zypper install cuda                                      # OpenSUSE & SLES
$ sudo apt-get install cuda                                     # Ubuntu
```

The `cuda-cross-<arch>` packages can also be upgraded in the same manner.

The cuda-drivers package points to the latest driver release available in the CUDA repository. When a new version is available, use the following commands to upgrade the driver:

```
$ sudo yum install nvidia-driver-latest-dkms                    # RHEL7
$ sudo yum install cuda-drivers                                 # RHEL7

$ sudo dnf module update nvidia-driver:latest-dkms             # RHEL8/Fedora

$ sudo zypper install cuda-drivers \
                      nvidia-gfxG04-kmp-default                 # OpenSUSE & SLES

$ sudo apt-get install cuda-drivers                             # Ubuntu
```

Some desktop environments, such as GNOME or KDE, will display an notification alert when new packages are available.

To avoid any automatic upgrade, and lock down the toolkit installation to the X.Y release, install the `cuda-X-Y` or `cuda-cross-<arch>-X-Y` package.

Side-by-side installations are supported. For instance, to install both the X.Y CUDA Toolkit and the X.Y+1 CUDA Toolkit, install the `cuda-X.Y` and `cuda-X.Y+1` packages.

3.10.3. Meta Packages

Meta packages are RPM/Deb packages which contain no (or few) files but have multiple dependencies. They are used to install many CUDA packages when you may not know the details of the packages you want. Below is the list of meta packages.

Table 4. Meta Packages Available for CUDA 11.4

Meta Package	Purpose
cuda	Installs all CUDA Toolkit and Driver packages. Handles upgrading to the next version of the `cuda` package when it's released.
cuda-11-4	Installs all CUDA Toolkit and Driver packages. Remains at version 11.4 until an additional version of CUDA is installed.
cuda-toolkit-11-4	Installs all CUDA Toolkit packages required to develop CUDA applications. Does not include the driver.
cuda-tools-11-4	Installs all CUDA command line and visual tools.
cuda-runtime-11-4	Installs all CUDA Toolkit packages required to run CUDA applications, as well as the Driver packages.
cuda-compiler-11-4	Installs all CUDA compiler packages.
cuda-libraries-11-4	Installs all runtime CUDA Library packages.
cuda-libraries-dev-11-4	Installs all development CUDA Library packages.
cuda-drivers	Installs all Driver packages. Handles upgrading to the next version of the Driver packages when they're released.

Chapter 4. Driver Installation

This section is for users who want to install a specific driver version.

For Debian and Ubuntu:

```
$ sudo apt-get install cuda-drivers-<branch>
```

For example:

```
$ sudo apt-get install cuda-drivers-418
```

For OpenSUSE and SLES:

```
$ sudo zypper install cuda-drivers-<branch>
```

For example:

```
$ sudo zypper install cuda-drivers-450
```

This allows you to get the highest version in the specified branch.

For Fedora and RHEL8:

```
$ sudo dnf module install nvidia-driver:<stream>/<profile>
```

where profile by default is "default" and does not need to be specified.

- Example dkms streams: `450-dkms` or `latest-dkms`
- Example precompiled streams: `450` or `latest`



Note: Precompiled streams are only supported on RHEL8 x86_64.

To uninstall or change streams on Fedora and RHEL8:

```
$ sudo dnf remove nvidia-driver
$ sudo dnf module reset nvidia-driver
```

Chapter 5. Precompiled Streams

Precompiled streams offer an optional method of streamlining the installation process.

The advantages of precompiled sreams:

- Precompiled: faster boot up after driver and/or kernel updates
- Pre-tested: kernel and driver combination has been validated
- Removes gcc dependency: no compiler installation required
- Removes dkms dependency: enabling EPEL repository not required
- Removes kernel-devel and kernel-headers dependencies: no black screen if matching packages are missing

When using precompiled drivers, a plugin for the dnf package manager is enabled that cleans up stale .ko files. To prevent system breakages, the NVIDIA dnf plugin also prevents upgrading to a kernel for which no precompiled driver yet exists. This can delay the application of security fixes but ensures that a tested kernel and driver combination is always used. A warning is displayed by `dnf` during that upgrade situation:

```
NOTE:  Skipping kernel installation since no NVIDIA driver kernel module package
 kmod-nvidia-${driver}-${kernel} ... could be found
```

Packaging templates and instructions are provided on GitHub to allow you to maintain your own precompiled kernel module packages for custom kernels and derivative Linux distros:NVIDIA/yum-packaging-precompiled-kmod

To use the new driver packages on RHEL 8:

1. First, ensure that the Red Hat repositories are enabled:

   ```
   $ subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms
   $ subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
   ```

2. Choose **one** of the four options below depending on the desired driver:

   - `latest` always updates to the highest versioned driver (precompiled):

     ```
     $ sudo dnf module install nvidia-driver:latest
     ```

   - `<id>` locks the driver updates to the specified driver branch (precompiled):

     ```
     $ sudo dnf module install nvidia-driver:<id>
     ```

     Note: Replace `<id>` with the appropriate driver branch streams, for example 455, 450, 440, or 418.

   - *latest-dkms* always updates to the highest versioned driver (non-precompiled):

```
$ sudo dnf module install nvidia-driver:latest-dkms
```

Note: This is the default stream.

- `<id>-dkms` locks the driver updates to the specified driver branch (non-precompiled):

```
$ sudo dnf module install nvidia-driver:<id>-dkms
```

Note: Valid streams include `455-dkms`, `450-dkms`, `440-dkms`, and `418-dkms`.

5.1. Precompiled Streams Support Matrix

This table show the supported precompiled and legacy DKMS streams for each driver.

NVIDIA Driver	Precompiled Stream	Legacy DKMS Stream
Highest version	latest	latest-dkms
Locked at 455.x	455	455-dkms
Locked at 450.x	450	450-dkms
Locked at 440.x	440	440-dkms
Locked at 418.x	418	418-dkms

5.2. Modularity Profiles

Modularity profiles work with any supported modularity stream and allow for additional use cases. These modularity profiles are available on RHEL8 and Fedora.

Table 5. List of nvidia-driver Module Profiles

Stream	Profile	Use Case
Default	`/default`	Installs all the driver packages in a stream.
Kickstart	`/ks`	Performs unattended Linux OS installation using a config file.
NVSwitch Fabric	`/fm`	Installs all the driver packages plus components required for bootstrapping an NVSwitch system (including the Fabric Manager and NSCQ telemetry).
Source	`/src`	Source headers for compilation (precompiled streams only).

For example:

```
$ sudo dnf module nvidia-driver:<stream>/default
$ sudo dnf module nvidia-driver:<stream>/ks
$ sudo dnf module nvidia-driver:<stream>/fm
$ sudo dnf module nvidia-driver:<stream>/src
```

You can install multiple modularity profiles using BASH curly brace expansion, for example:

```
$  sudo dnf module install nvidia-driver:latest/{default,src}
```

See https://developer.nvidia.com/blog/streamlining-nvidia-driver-deployment-on-rhel-8-with-modularity-streams in the Developer Blog and https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/precompiled/ for more information.

Chapter 6. Kickstart Installation

6.1. RHEL8/CentOS8

1. Enable the EPEL repository:
   ```
   repo --name=epel --baseurl=http://download.fedoraproject.org/pub/epel/8/
   Everything/x86_64/
   ```
2. Enable the CUDA repository:
   ```
   repo --name=cuda-rhel8 --baseurl=https://developer.download.nvidia.com/compute/
   cuda/repos/rhel8/x86_64/
   ```
3. In the packages section of the `ks.cfg` file, make sure you are using the **/ks** profile and **:latest-dkms** stream:
   ```
   @nvidia-driver:latest-dkms/ks
   ```
4. Perform the post-installation actions.

Chapter 7. Runfile Installation

Basic instructions can be found in the Quick Start Guide. Read on for more detailed instructions.

This section describes the installation and configuration of CUDA when using the standalone installer. The standalone installer is a ".run" file and is completely self-contained.

7.1. Overview

The Runfile installation installs the NVIDIA Driver, CUDA Toolkit, and CUDA Samples via an interactive ncurses-based interface.

The installation steps are listed below. Distribution-specific instructions on disabling the Nouveau drivers as well as steps for verifying device node creation are also provided.

Finally, advanced options for the installer and uninstallation steps are detailed below.

The Runfile installation does not include support for cross-platform development. For cross-platform development, see the CUDA Cross-Platform Environment section.

7.2. Installation

1. Perform the pre-installation actions.
2. Disable the Nouveau drivers.
3. Reboot into text mode (runlevel 3).

   This can usually be accomplished by adding the number "3" to the end of the system's kernel boot parameters.

   Since the NVIDIA drivers are not yet installed, the text terminals may not display correctly. Temporarily adding "nomodeset" to the system's kernel boot parameters may fix this issue.

   Consult your system's bootloader documentation for information on how to make the above boot parameter changes.

   The reboot is required to completely unload the Nouveau drivers and prevent the graphical interface from loading. The CUDA driver cannot be installed while the Nouveau drivers are loaded or while the graphical interface is active.

4. Verify that the Nouveau drivers are not loaded. If the Nouveau drivers are still loaded, consult your distribution's documentation to see if further steps are needed to disable Nouveau.
5. Run the installer and follow the on-screen prompts:

```
$ sudo sh cuda_<version>_linux.run
```

The installer will prompt for the following:

- EULA Acceptance
- CUDA Driver installation
- CUDA Toolkit installation, location, and `/usr/local/cuda` symbolic link
- CUDA Samples installation and location

The default installation locations for the toolkit and samples are:

Component	Default Installation Directory
CUDA Toolkit	/usr/local/cuda-11.4
CUDA Samples	$(HOME)/NVIDIA_CUDA-11.4_Samples

The `/usr/local/cuda` symbolic link points to the location where the CUDA Toolkit was installed. This link allows projects to use the latest CUDA Toolkit without any configuration file update.

The installer must be executed with sufficient privileges to perform some actions. When the current privileges are insufficient to perform an action, the installer will ask for the user's password to attempt to install with root privileges. Actions that cause the installer to attempt to install with root privileges are:

- installing the CUDA Driver
- installing the CUDA Toolkit to a location the user does not have permission to write to
- installing the CUDA Samples to a location the user does not have permission to write to
- creating the `/usr/local/cuda` symbolic link

Running the installer with **sudo**, as shown above, will give permission to install to directories that require root permissions. Directories and files created while running the installer with **sudo** will have root ownership.

If installing the driver, the installer will also ask if the openGL libraries should be installed. If the GPU used for display is not an NVIDIA GPU, the NVIDIA openGL libraries should not be installed. Otherwise, the openGL libraries used by the graphics driver of the non-NVIDIA GPU will be overwritten and the GUI will not work. If performing a silent installation, the `--no-opengl-libs` option should be used to prevent the openGL libraries from being installed. See the Advanced Options section for more details.

If the GPU used for display is an NVIDIA GPU, the X server configuration file, `/etc/X11/xorg.conf`, may need to be modified. In some cases, `nvidia-xconfig` can be used to automatically generate a `xorg.conf` file that works for the system. For non-standard

systems, such as those with more than one GPU, it is recommended to manually edit the `xorg.conf` file. Consult the `xorg.conf` documentation for more information.



Note: Installing Mesa may overwrite the `/usr/lib/libGL.so` that was previously installed by the NVIDIA driver, so a reinstallation of the NVIDIA driver might be required after installing these libraries.

6. Reboot the system to reload the graphical interface.
7. Verify the device nodes are created properly.
8. Perform the post-installation actions.

7.3. Disabling Nouveau

To install the Display Driver, the Nouveau drivers must first be disabled. Each distribution of Linux has a different method for disabling Nouveau.

The Nouveau drivers are loaded if the following command prints anything:

```
$ lsmod | grep nouveau
```

7.3.1. Fedora

1. Create a file at `/usr/lib/modprobe.d/blacklist-nouveau.conf` with the following contents:

   ```
   blacklist nouveau
   options nouveau modeset=0
   ```

2. Regenerate the kernel initramfs:

   ```
   $ sudo dracut --force
   ```

3. Run the below command:

   ```
   $ sudo grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

4. Reboot the system.

7.3.2. RHEL/CentOS

1. Create a file at `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:

   ```
   blacklist nouveau
   options nouveau modeset=0
   ```

2. Regenerate the kernel initramfs:

   ```
   $ sudo dracut --force
   ```

7.3.3. OpenSUSE

1. Create a file at `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:

   ```
   blacklist nouveau
   options nouveau modeset=0
   ```

2. Regenerate the kernel initrd:

   ```
   $ sudo /sbin/mkinitrd
   ```

7.3.4. SLES

No actions to disable Nouveau are required as Nouveau is not installed on SLES.

7.3.5. WSL

No actions to disable Nouveau are required as Nouveau is not installed on WSL.

7.3.6. Ubuntu

1. Create a file at `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:
   ```
   blacklist nouveau
   options nouveau modeset=0
   ```
2. Regenerate the kernel initramfs:
   ```
   $ sudo update-initramfs -u
   ```

7.3.7. Debian

1. Create a file at `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:
   ```
   blacklist nouveau
   options nouveau modeset=0
   ```
2. Regenerate the kernel initramfs:
   ```
   $ sudo update-initramfs -u
   ```

7.4. Device Node Verification

Check that the device files`/dev/nvidia*` exist and have the correct (0666) file permissions. These files are used by the CUDA Driver to communicate with the kernel-mode portion of the NVIDIA Driver. Applications that use the NVIDIA driver, such as a CUDA application or the X server (if any), will normally automatically create these files if they are missing using the *setuid* `nvidia-modprobe` tool that is bundled with the NVIDIA Driver. However, some systems disallow setuid binaries, so if these files do not exist, you can create them manually by using a startup script such as the one below:

```
#!/bin/bash

/sbin/modprobe nvidia

if [ "$?" -eq 0 ]; then
  # Count the number of NVIDIA controllers found.
  NVDEVS=`lspci | grep -i NVIDIA`
  N3D=`echo "$NVDEVS" | grep "3D controller" | wc -l`
  NVGA=`echo "$NVDEVS" | grep "VGA compatible controller" | wc -l`

  N=`expr $N3D + $NVGA - 1`
  for i in `seq 0 $N`; do
    mknod -m 666 /dev/nvidia$i c 195 $i
  done

  mknod -m 666 /dev/nvidiactl c 195 255

else
```

```
  exit 1
fi

/sbin/modprobe nvidia-uvm

if [ "$?" -eq 0 ]; then
  # Find out the major device number used by the nvidia-uvm driver
  D=`grep nvidia-uvm /proc/devices | awk '{print $1}'`

  mknod -m 666 /dev/nvidia-uvm c $D 0
else
  exit 1
fi
```

7.5. Advanced Options

Action	Options Used	Explanation
Silent Installation	--silent	Required for any silent installation. Performs an installation with no further user-input and minimal command-line output based on the options provided below. Silent installations are useful for scripting the installation of CUDA. Using this option implies acceptance of the EULA. The following flags can be used to customize the actions taken during installation. At least one of --driver, --uninstall, --toolkit, and --samples must be passed if running with non-root permissions.
	--driver	Install the CUDA Driver.
	--toolkit	Install the CUDA Toolkit.
	--toolkitpath=<path>	Install the CUDA Toolkit to the <path> directory. If not provided, the default path of `/usr/local/cuda-11.4` is used.
	--samples	Install the CUDA Samples.
	--samplespath=<path>	Install the CUDA Samples to the <path> directory. If not provided, the default path of `$(HOME)/NVIDIA_CUDA-11.4_Samples` is used.
	--defaultroot=<path>	Install libraries to the <path> directory. If the <path> is not provided, then the default path of your distribution is used. *This only applies to the libraries installed outside of the CUDA Toolkit path.*
Extraction	--extract=<path>	Extracts to the <path> the following: the driver runfile, the raw files of the toolkit and samples to <path>. This is especially useful when one wants to install the driver using one or more of the command-line options provided by the driver installer which are not exposed in this installer.
Overriding Installation Checks	--override	Ignores compiler, third-party library, and toolkit detection checks which would prevent the CUDA Toolkit and CUDA Samples from installing.

Action	Options Used	Explanation
No OpenGL Libraries	--no-opengl-libs	Prevents the driver installation from installing NVIDIA's GL libraries. Useful for systems where the display is driven by a non-NVIDIA GPU. In such systems, NVIDIA's GL libraries could prevent X from loading properly.
No man pages	--no-man-page	Do not install the man pages under /usr/share/man.
Overriding Kernel Source	--kernel-source-path=<path>	Tells the driver installation to use <path> as the kernel source directory when building the NVIDIA kernel module. Required for systems where the kernel source is installed to a non-standard location.
Running nvidia-xconfig	--run-nvidia-xconfig	Tells the driver installation to run nvidia-xconfig to update the system X configuration file so that the NVIDIA X driver is used. The pre-existing X configuration file will be backed up.
No nvidia-drm kernel module	--no-drm	Do not install the nvidia-drm kernel module. This option should only be used to work around failures to build or install the nvidia-drm kernel module on systems that do not need the provided features.
Custom Temporary Directory Selection	--tmpdir=<path>	Performs any temporary actions within <path> instead of /tmp. Useful in cases where /tmp cannot be used (doesn't exist, is full, is mounted with 'noexec', etc.).
Show Installer Options	--help	Prints the list of command-line options to stdout.

7.6. Uninstallation

To uninstall the CUDA Toolkit, run the uninstallation script provided in the bin directory of the toolkit. By default, it is located in `/usr/local/cuda-11.4/bin`:

```
$ sudo /usr/local/cuda-11.4/bin/cuda-uninstaller
```

To uninstall the NVIDIA Driver, run `nvidia-uninstall`:

```
$ sudo /usr/bin/nvidia-uninstall
```

To enable the Nouveau drivers, remove the blacklist file created in the Disabling Nouveau section, and regenerate the kernel initramfs/initrd again as described in that section.

Chapter 8. CUDA Cross-Platform Environment

Cross-platform development is only supported on Ubuntu systems, and is only provided via the Package Manager installation process.

We recommend selecting Ubuntu 18.04 as your cross-platform development environment. This selection helps prevent host/target incompatibilities, such as GCC or GLIBC version mismatches.

8.1. CUDA Cross-Platform Installation

Some of the following steps may have already been performed as part of the native Ubuntu installation. Such steps can safely be skipped.

These steps should be performed on the x86_64 host system, rather than the target system. To install the native CUDA Toolkit on the target system, refer to the native Ubuntu installation section.

1. Perform the pre-installation actions.
2. Install repository meta-data package with:
   ```
   $ sudo dpkg -i cuda-repo-cross-<identifier>_all.deb
   ```
   where `<identifier>` indicates the operating system, architecture, and/or the version of the package.
3. Update the Apt repository cache:
   ```
   $ sudo apt-get update
   ```
4. Install the appropriate cross-platform CUDA Toolkit:
   a). For aarch64:
   ```
   $ sudo apt-get install cuda-cross-aarch64
   ```
   b). For QNX:
   ```
   $ sudo apt-get install cuda-cross-qnx
   ```
5. Perform the post-installation actions.

8.2. CUDA Cross-Platform Samples

This section describes the options used to build cross-platform samples. `TARGET_ARCH=<arch>` and `TARGET_OS=<os>` should be chosen based on the supported targets shown below. `TARGET_FS=<path>` can be used to point nvcc to libraries and headers used by the sample.

Table 6. Supported Target Arch/OS Combinations

		TARGET OS		
		linux	android	qnx
TARGET ARCH	x86_64	YES	NO	NO
	aarch64	YES	YES	YES
	sbsa	YES	NO	NO

TARGET_ARCH

The target architecture must be specified when cross-compiling applications. If not specified, it defaults to the host architecture. Allowed architectures are:

- `x86_64` - 64-bit x86 CPU architecture
- `aarch64` - 64-bit ARM CPU architecture, like that found on Jetson TX1 onwards
- `sbsa` - 64-bit ARM Server CPU architecture

TARGET_OS

The target OS must be specified when cross-compiling applications. If not specified, it defaults to the host OS. Allowed OSes are:

- `linux` - for any Linux distributions
- `android` - for any supported device running Android
- `qnx` - for any supported device running QNX

TARGET_FS

The most reliable method to cross-compile the CUDA Samples is to use the TARGET_FS variable. To do so, mount the target's filesystem on the host, say at `/mnt/target`. This is typically done using `exportfs`. In cases where `exportfs` is unavailable, it is sufficient to copy the target's filesystem to `/mnt/target`. To cross-compile a sample, execute:

```
$ make TARGET_ARCH=<arch> TARGET_OS=<os> TARGET_FS=/mnt/target
```

Cross Compiling to Embedded ARM architectures

While cross compiling the samples from x86_64 installation to embedded ARM architectures, that is, `aarch64` or `armv7l`, if you intend to run the executable on tegra GPU then `SMS` variable need to override SM architectures to the tegra GPU through `SMS=<TEGRA_GPU_SM_ARCH>`, where `<TEGRA_GPU_SM_ARCH>` is the SM architecture of tegra GPU on which you want the generated binary to run on. For instance it can be `SMS="53 62 72"`. Note you can also add SM arch of discrete GPU to this list <TEGRA_GPU_SM_ARCH> if you intend to run on embedded board having discrete GPU as well. To cross compile a sample, execute:

```
$ make TARGET_ARCH=<arch> TARGET_OS=<os> SMS=<TEGRA_GPU_SM_ARCHS> TARGET_FS=/mnt/
target
```

Copying Libraries

If the TARGET_FS option is not available, the libraries used should be copied from the target system to the host system, say at `/opt/target/libs`. If the sample uses GL, the GL headers must also be copied, say at `/opt/target/include`. The linker must then be told where the libraries are with the `-rpath-link` and/or `-L` options. To ignore unresolved symbols from some libraries, use the `--unresolved-symbols` option as shown below. `SAMPLE_ENABLED` should be used to force the sample to build. For example, to cross-compile a sample which uses such libraries, execute:

```
$ make TARGET_ARCH=<arch> TARGET_OS=<os> \
           EXTRA_LDFLAGS="-rpath-link=/opt/target/libs -L/opt/target/libs --
unresolved-symbols=ignore-in-shared-libs" \
           EXTRA_CCFLAGS="-I /opt/target/include" \
           SAMPLE_ENABLED=1
```

Chapter 9. Post-installation Actions

The post-installation actions must be manually performed. These actions are split into mandatory, recommended, and optional sections.

9.1. Mandatory Actions

Some actions must be taken after the installation before the CUDA Toolkit and Driver can be used.

9.1.1. Environment Setup

The `PATH` variable needs to include `$ export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}`. Nsight Compute has moved to `/opt/nvidia/nsight-compute/` only in rpm/deb installation method. When using `.run` installer it is still located under `/usr/local/cuda-11.4/`.

To add this path to the `PATH` variable:

```
$ export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}
```

In addition, when using the runfile installation method, the `LD_LIBRARY_PATH` variable needs to contain `/usr/local/cuda-11.4/lib64` on a 64-bit system, or `/usr/local/cuda-11.4/lib` on a 32-bit system

- To change the environment variables for 64-bit operating systems:

  ```
  $ export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib64\
                           ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
  ```

- To change the environment variables for 32-bit operating systems:

  ```
  $ export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib\
                           ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
  ```

Note that the above paths change when using a custom install path with the runfile installation method.

9.1.2. POWER9 Setup

Because of the addition of new features specific to the NVIDIA POWER9 CUDA driver, there are some additional setup requirements in order for the driver to function properly. These additional steps are not handled by the installation of CUDA packages, and failure to ensure these extra requirements are met will result in a non-functional CUDA driver installation.

There are two changes that need to be made manually after installing the NVIDIA CUDA driver to ensure proper operation:

1. The NVIDIA Persistence Daemon should be automatically started for POWER9 installations. Check that it is running with the following command:
   ```
   $ systemctl status nvidia-persistenced
   ```
   If it is not active, run the following command:
   ```
   $ sudo systemctl enable nvidia-persistenced
   ```
2. Disable a udev rule installed by default in some Linux distributions that cause hot-pluggable memory to be automatically onlined when it is physically probed. This behavior prevents NVIDIA software from bringing NVIDIA device memory online with non-default settings. This udev rule must be disabled in order for the NVIDIA CUDA driver to function properly on POWER9 systems.

   On RedHat Enterprise Linux 8.1, this rule can be found in:
   ```
   /lib/udev/rules.d/40-redhat.rules
   ```
   On Ubuntu 18.04, this rule can be found in:
   ```
   /lib/udev/rules.d/40-vm-hotadd.rules
   ```
   The rule generally takes a form where it detects the addition of a memory block and changes the 'state' attribute to online. For example, in RHEL8, the rule looks like this:
   ```
   SUBSYSTEM=="memory", ACTION=="add", PROGRAM="/bin/uname -p", RESULT!="s390*",
    ATTR{state}=="offline", ATTR{state}="online"
   ```
   This rule must be disabled by copying the file to `/etc/udev/rules.d` and commenting out, removing, or changing the hot-pluggable memory rule in the `/etc` copy so that it does not apply to POWER9 NVIDIA systems. For example, on RHEL 7.5 and earlier:
   ```
   $ sudo cp /lib/udev/rules.d/40-redhat.rules /etc/udev/rules.d
   $ sudo sed -i '/SUBSYSTEM=="memory", ACTION=="add"/d' /etc/udev/rules.d/40-
   redhat.rules
   ```
   On RHEL 7.6 and later versions:
   ```
   $ sudo cp /lib/udev/rules.d/40-redhat.rules /etc/udev/rules.d
   $ sudo sed -i 's/SUBSYSTEM!="memory",.*GOTO="memory_hotplug_end"/SUBSYSTEM=="*",
    GOTO="memory_hotplug_end"/' /etc/udev/rules.d/40-redhat.rules
   ```

You will need to reboot the system to initialize the above changes.



Note: For NUMA best practices on IBM Newell POWER9, see NUMA Best Practices.

9.2. Recommended Actions

Other actions are recommended to verify the integrity of the installation.

9.2.1. Install Persistence Daemon

NVIDIA is providing a user-space daemon on Linux to support persistence of driver state across CUDA job runs. The daemon approach provides a more elegant and robust solution to this problem than persistence mode. For more details on the NVIDIA Persistence Daemon, see the documentation here.

The NVIDIA Persistence Daemon can be started as the root user by running:

```
$ /usr/bin/nvidia-persistenced --verbose
```

This command should be run on boot. Consult your Linux distribution's init documentation for details on how to automate this.

9.2.2. Install Writable Samples

In order to modify, compile, and run the samples, the samples must be installed with write permissions. A convenience installation script is provided:

```
$ cuda-install-samples-11.4.sh <dir>
```

This script is installed with the cuda-samples-11-4 package. The cuda-samples-11-4 package installs only a read-only copy in /usr/local/cuda-11.4/samples.

9.2.3. Verify the Installation

Before continuing, it is important to verify that the CUDA toolkit can find and communicate correctly with the CUDA-capable hardware. To do this, you need to compile and run some of the included sample programs.



Note: Ensure the PATH and, if using the runfile installation method, LD_LIBRARY_PATH variables are set correctly.

9.2.3.1. Verify the Driver Version

If you installed the driver, verify that the correct version of it is loaded. If you did not install the driver, or are using an operating system where the driver is not loaded via a kernel module, such as L4T, skip this step.

When the driver is loaded, the driver version can be found by executing the command

```
$ cat /proc/driver/nvidia/version
```

Note that this command will not work on an iGPU/dGPU system.

9.2.3.2. Compiling the Examples

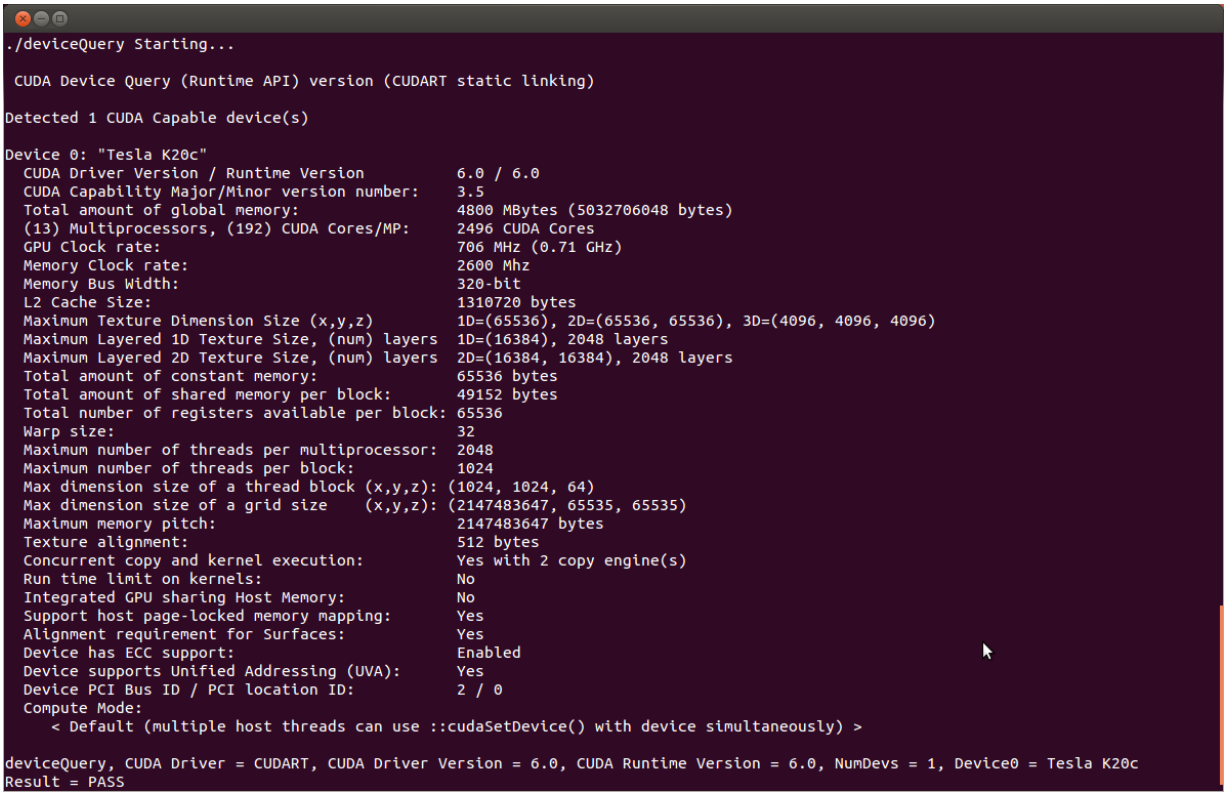
The version of the CUDA Toolkit can be checked by running `nvcc -V` in a terminal window. The `nvcc` command runs the compiler driver that compiles CUDA programs. It calls the `gcc` compiler for C code and the NVIDIA PTX compiler for the CUDA code.

The NVIDIA CUDA Toolkit includes sample programs in source form. You should compile them by changing to `~/NVIDIA_CUDA-11.4_Samples` and typing `make`. The resulting binaries will be placed under `~/NVIDIA_CUDA-11.4_Samples/bin`.

9.2.3.3. Running the Binaries

After compilation, find and run `deviceQuery` under `~/NVIDIA_CUDA-11.4_Samples`. If the CUDA software is installed and configured correctly, the output for `deviceQuery` should look similar to that shown in Figure 1.

Figure 1. Valid Results from deviceQuery CUDA Sample



The exact appearance and the output lines might be different on your system. The important outcomes are that a device was found (the first highlighted line), that the device matches the one on your system (the second highlighted line), and that the test passed (the final highlighted line).

If a CUDA-capable device and the CUDA Driver are installed but `deviceQuery` reports that no CUDA-capable devices are present, this likely means that the `/dev/nvidia*` files are missing or have the wrong permissions.

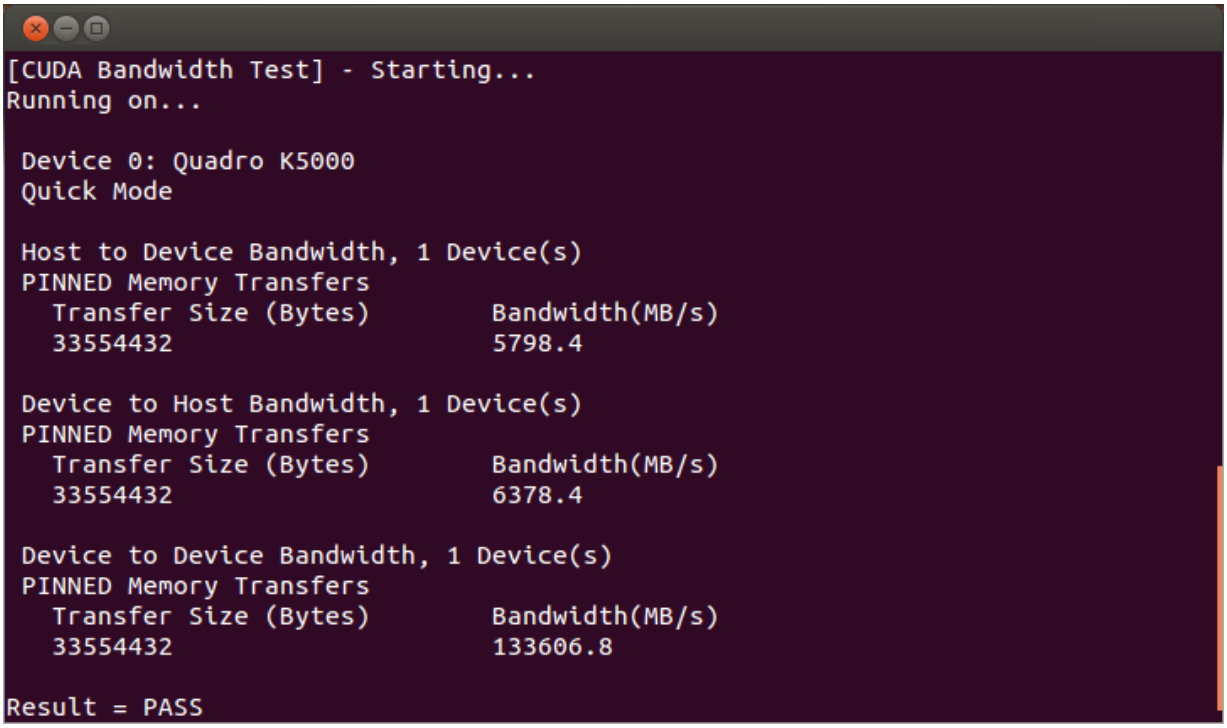
On systems where `SELinux` is enabled, you might need to temporarily disable this security feature to run `deviceQuery`. To do this, type:

```
$ setenforce 0
```

from the command line as the *superuser*.

Running the `bandwidthTest` program ensures that the system and the CUDA-capable device are able to communicate correctly. Its output is shown in Figure 2.

Figure 2. Valid Results from bandwidthTest CUDA Sample



Note that the measurements for your CUDA-capable device description will vary from system to system. The important point is that you obtain measurements, and that the second-to-last line (in Figure 2) confirms that all necessary tests passed.

Should the tests not pass, make sure you have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

If you run into difficulties with the link step (such as libraries not being found), consult the *Linux Release Notes* found in the `doc` folder in the CUDA Samples directory.

9.2.4. Install Nsight Eclipse Plugins

To install Nsight Eclipse plugins, an installation script is provided:

```
$ /usr/local/cuda-11.4/bin/nsight_ee_plugins_manage.sh install <eclipse-dir>
```

Refer to Nsight Eclipse Plugins Installation Guide for more details.

9.3. Optional Actions

Other options are not necessary to use the CUDA Toolkit, but are available to provide additional features.

9.3.1. Install Third-party Libraries

Some CUDA samples use third-party libraries which may not be installed by default on your system. These samples attempt to detect any required libraries when building. If a library is

not detected, it waives itself and warns you which library is missing. To build and run these samples, you must install the missing libraries. These dependencies may be installed if the RPM or Deb `cuda-samples-11-4` package is used. In cases where these dependencies are not installed, follow the instructions below.

RHEL/CentOS

```
$ sudo yum install freeglut-devel libX11-devel libXi-devel libXmu-devel \
    make mesa-libGLU-devel
```

Fedora

```
$ sudo dnf install freeglut-devel libX11-devel libXi-devel libXmu-devel \
    make mesa-libGLU-devel
```

SLES

```
$ sudo zypper install libglut3 libX11 libXi6 libXmu6 libGLU1 make
```

OpenSUSE

```
$ sudo zypper install freeglut-devel libX11-devel libXi-devel libXmu-devel \
    make Mesa-libGL-devel
```

Ubuntu

```
$ sudo apt-get install g++ freeglut3-dev build-essential libx11-dev \
    libxmu-dev libxi-dev libglu1-mesa libglu1-mesa-dev
```

9.3.2. Install the source code for cuda-gdb

The `cuda-gdb` source must be explicitly selected for installation with the runfile installation method. During the installation, in the component selection page, expand the component "CUDA Tools 11.0" and select the `cuda-gdb-src` for installation. It is unchecked by default.

To obtain a copy of the source code for cuda-gdb using the RPM and Debian installation methods, the **cuda-gdb-src** package must be installed.

The source code is installed as a tarball in the `/usr/local/cuda-11.4/extras` directory.

9.3.3. Select the Active Version of CUDA

For applications that rely on the symlinks `/usr/local/cuda` and `/usr/local/cuda-MAJOR`, you may wish to change to a different installed version of CUDA using the provided alternatives.

To show the active version of CUDA and all available versions:

```
$ update-alternatives --display cuda
```

To show the active minor version of a given major CUDA release:

```
$ update-alternatives --display cuda-11
```

To update the active version of CUDA:

```
$ sudo update-alternatives --config cuda
```

Chapter 10. Advanced Setup

Below is information on some advanced setup scenarios which are not covered in the basic instructions above.

Table 7. Advanced Setup Scenarios when Installing CUDA

Scenario	Instructions

Install CUDA using the Package Manager installation method without installing the NVIDIA GL libraries.	**Fedora**
	Install CUDA using the following command:
	`$ sudo dnf install cuda-toolkit-11-4 \`
	`  nvidia-driver-cuda akmod-nvidia`
	Follow the instructions here to ensure that Nouveau is disabled.
	If performing an upgrade over a previous installation, the NVIDIA kernel module may need to be rebuilt by following the instructions here.
	OpenSUSE/SLES
	On some system configurations the NVIDIA GL libraries may need to be locked before installation using:
	`$ sudo zypper addlock nvidia-glG04`
	Install CUDA using the following command:
	`$ sudo zypper install --no-recommends cuda-toolkit-11-4 \`
	`  nvidia-computeG04 \`
	`  nvidia-gfxG04-kmp-default`
	Follow the instructions here to ensure that Nouveau is disabled.
	Ubuntu
	This functionality isn't supported on Ubuntu. Instead, the driver packages integrate with the Bumblebee framework to provide a solution for users who wish to control what applications the NVIDIA drivers are used for. See Ubuntu's Bumblebee wiki for more information.

Upgrade from a RPM/Deb driver installation which includes the diagnostic driver packages to a driver installation which does not include the	**RHEL/CentOS**
	Remove diagnostic packages using the following command:
	`$ sudo yum remove cuda-drivers-diagnostic \`
	`  xorg-x11-drv-nvidia-diagnostic`
	Follow the instructions here to continue installation as normal.
	Fedora

Scenario	Instructions

diagnostic driver packages.	Remove diagnostic packages using the following command:
	`$ sudo dnf remove cuda-drivers-diagnostic \`
	`  xorg-x11-drv-nvidia-diagnostic`
	Follow the instructions here to continue installation as normal.
	OpenSUSE/SLES
	Remove diagnostic packages using the following command:
	`$ sudo zypper remove cuda-drivers-diagnostic \`
	`  nvidia-diagnosticG04`
	Follow the instructions here to continue installation as normal.
	Ubuntu
	Remove diagnostic packages using the following command:
	`$ sudo apt-get purge cuda-drivers-diagnostic \`
	`  nvidia-384-diagnostic`
	Follow the instructions here to continue installation as normal.

Use a specific GPU for rendering the display.	Add or replace a **Device** entry in your xorg.conf file, located at `/etc/X11/xorg.conf`. The **Device** entry should resemble the following:
	`Section "Device"`
	`  Identifier  "Device0"`
	`  Driver  "driver_name"`
	`  VendorName  "vendor_name"`
	`  BusID  "bus_id"`
	`EndSection`
	The details will you will need to add differ on a case-by-case basis. For example, if you have two NVIDIA GPUs and you want the first GPU to be used for display, you would replace "driver_name" with "nvidia", "vendor_name" with "NVIDIA Corporation" and "bus_id" with the Bus ID of the GPU.
	The Bus ID will resemble "PCI:00:02.0" and can be found by running `lspci`.

Install CUDA to a specific directory using the Package Manager installation method.	**RPM**
	The RPM packages don't support custom install locations through the package managers (Yum and Zypper), but it is possible to install the RPM packages to a custom location using rpm's `--relocate` parameter:
	`$ sudo rpm --install --relocate /usr/local/cuda-11.4=/new/toolkit package.rpm`
	You will need to install the packages in the correct dependency order; this task is normally taken care of by the package managers. For example, if package "foo" has a dependency on package "bar", you should install package "bar" first, and package "foo" second. You can check the dependencies of a RPM package as follows:
	`$ rpm -qRp package.rpm`
	Note that the driver packages cannot be relocated.
	Deb
	The Deb packages do not support custom install locations. It is however possible to extract the contents of the Deb packages and move the files to the desired install location. See the next scenario for more details one xtracting Deb packages.

Scenario	Instructions

Extract the contents of the installers.	**Runfile**
	The Runfile can be extracted into the standalone Toolkit, Samples and Driver Runfiles by using the `--extract` parameter. The Toolkit and Samples standalone Runfiles can be further extracted by running:
	`$ ./runfile.run --tar mxvf`
	The Driver Runfile can be extracted by running:
	`$ ./runfile.run -x`
	RPM
	The RPM packages can be extracted by running:
	`$ rpm2cpio package.rpm | cpio -idmv`
	Deb
	The Deb packages can be extracted by running:
	`$ dpkg-deb -x package.deb output_dir`

Modify Ubuntu's apt package manager to query specific architectures for specific repositories.
This is useful when a foreign architecture has been added, causing "404 Not Found" errors to appear when the repository meta-data is updated.
	Each repository you wish to restrict to specific architectures must have its `sources.list` entry modified. This is done by modifying the `/etc/apt/sources.list` file and any files containing repositories you wish to restrict under the `/etc/apt/sources.list.d/` directory. Normally, it is sufficient to modify only the entries in `/etc/apt/sources.list`
	An architecture-restricted repository entry looks like:
	`deb [arch=<arch1>,<arch2>] <url>`
	For example, if you wanted to restrict a repository to only the amd64 and i386 architectures, it would look like:
	`deb [arch=amd64,i386] <url>`
	It is not necessary to restrict the `deb-src` repositories, as these repositories don't provide architecture-specific packages.
	For more details, see the `sources.list` manpage.

The nvidia.ko kernel module fails to load, saying some symbols are unknown. For example: `nvidia: Unknown symbol drm_open (err 0)`	Check to see if there are any optionally installable modules that might provide these symbols which are not currently installed. For the example of the `drm_open` symbol, check to see if there are any packages which provide drm_open and are not already installed. For instance, on Ubuntu 14.04, the `linux-image-extra` package provides the DRM kernel module (which provides drm_open). This package is optional even though the kernel headers reflect the availability of DRM regardless of whether this package is installed or not.
The runfile installer fails to extract due to limited space in the TMP directory.	This can occur on systems with limited storage in the TMP directory (usually /tmp), or on systems which use a tmpfs in memory to handle temporary storage. In this case, the `--tmpdir` command-line option should be used to instruct the runfile to use a directory with sufficient space to extract into. More information on this option can be found here.
Re-enable Wayland after installing the RPM driver on Fedora.	Wayland is disabled during installation of the Fedora driver RPM due to compatability issues. To re-enable wayland, comment out this line in /etc/gdm/custom.conf: `WaylandEnable=false`

Chapter 11. Frequently Asked Questions

How do I install the Toolkit in a different location?

The Runfile installation asks where you wish to install the Toolkit and the Samples during an interactive install. If installing using a non-interactive install, you can use the --toolkitpath and --samplespath parameters to change the install location:

```
$ ./runfile.run --silent \
                --toolkit --toolkitpath=/my/new/toolkit \
                --samples --samplespath=/my/new/samples
```

The RPM and Deb packages cannot be installed to a custom install location directly using the package managers. See the "Install CUDA to a specific directory using the Package Manager installation method" scenario in the Advanced Setup section for more information.

Why do I see "nvcc: No such file or directory" when I try to build a CUDA application?

Your PATH environment variable is not set up correctly. Ensure that your PATH includes the bin directory where you installed the Toolkit, usually /usr/local/cuda-11.4/bin.

```
$ export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}
```

Why do I see "error while loading shared libraries: <lib name>: cannot open shared object file: No such file or directory" when I try to run a CUDA application that uses a CUDA library?

Your LD_LIBRARY_PATH environment variable is not set up correctly. Ensure that your LD_LIBRARY_PATH includes the lib and/or lib64 directory where you installed the Toolkit, usually /usr/local/cuda-11.4/lib{,64}:

```
$ export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib\
                         ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

Why do I see multiple "404 Not Found" errors when updating my repository meta-data on Ubuntu?

These errors occur after adding a foreign architecture because apt is attempting to query for each architecture within each repository listed in the system's sources.list file. Repositories that do not host packages for the newly added architecture will present this error. While noisy, the error itself does no harm. Please see the Advanced Setup section for details on how to modify your sources.list file to prevent these errors.

How can I tell X to ignore a GPU for compute-only use?

To make sure X doesn't use a certain GPU for display, you need to specify which **other** GPU to use for display. For more information, please refer to the "Use a specific GPU for rendering the display" scenario in the Advanced Setup section.

Why doesn't the cuda-repo package install the CUDA Toolkit and Drivers?

When using RPM or Deb, the downloaded package is a repository package. Such a package only informs the package manager where to find the actual installation packages, but will not install them.

See the Package Manager Installation section for more details.

How do I get CUDA to work on a laptop with an iGPU and a dGPU running Ubuntu14.04?

After installing CUDA, set the driver value for the intel device in /etc/X11/xorg.conf to 'modesetting' as shown below:

```
Section "Device"
    Identifier "intel"
    Driver "modesetting"
    ...
EndSection
```

To prevent Ubuntu from reverting the change in xorg.conf, edit /etc/default/grub to add "nogpumanager" to GRUB_CMDLINE_LINUX_DEFAULT.

Run the following command to update grub before rebooting:

```
$ sudo update-grub
```

What do I do if the display does not load, or CUDA does not work, after performing a system update?

System updates may include an updated Linux kernel. In many cases, a new Linux kernel will be installed without properly updating the required Linux kernel headers and development packages. To ensure the CUDA driver continues to work when performing a system update, rerun the commands in the Kernel Headers and Development Packages section.

Additionally, on Fedora, the Akmods framework will sometimes fail to correctly rebuild the NVIDIA kernel module packages when a new Linux kernel is installed. When this happens, it is usually sufficient to invoke Akmods manually and regenerate the module mapping files by running the following commands in a virtual console, and then rebooting:

```
$ sudo akmods --force
$ sudo depmod
```

You can reach a virtual console by hitting `ctrl+alt+f2` at the same time.

How do I install a CUDA driver with a version less than 367 using a network repo?

To install a CUDA driver at a version earlier than 367 using a network repo, the required packages will need to be explicitly installed at the desired version. For example, to install

352.99, instead of installing the cuda-drivers metapackage at version 352.99, you will need to install all required packages of cuda-drivers at version 352.99.

How do I install an older CUDA version using a network repo?

Depending on your system configuration, you may not be able to install old versions of CUDA using the cuda metapackage. In order to install a specific version of CUDA, you may need to specify all of the packages that would normally be installed by the cuda metapackage at the version you want to install.

If you are using yum to install certain packages at an older version, the dependencies may not resolve as expected. In this case you may need to pass "--setopt=obsoletes=0" to yum to allow an install of packages which are obsoleted at a later version than you are trying to install.

Chapter 12. Additional Considerations

Now that you have CUDA-capable hardware and the NVIDIA CUDA Toolkit installed, you can examine and enjoy the numerous included programs. To begin using CUDA to accelerate the performance of your own applications, consult the *CUDA C++ Programming Guide*, located in `/usr/local/cuda-11.4/doc`.

A number of helpful development tools are included in the CUDA Toolkit to assist you as you develop your CUDA programs, such as NVIDIA® Nsight™ Eclipse Edition, NVIDIA Visual Profiler, cuda-gdb, and cuda-memcheck.

For technical support on programming questions, consult and participate in the developer forums at https://developer.nvidia.com/cuda/.

Chapter 13. Removing CUDA Toolkit and Driver

Follow the below steps to properly uninstall the CUDA Toolkit and NVIDIA Drivers from your system. These steps will ensure that the uninstallation will be clean.

RHEL8/CentOS8

To remove CUDA Toolkit:

```
$ sudo dnf remove "cuda*" "*cublas*" "*cufft*" "*curand*" \
 "*cusolver*" "*cusparse*" "*npp*" "*nvjpeg*" "nsight*"
```

To remove NVIDIA Drivers:

```
$ sudo dnf remove nvidia-driver
```

To reset the module stream:

```
$ sudo dnf module reset nvidia-driver
```

RHEL7/CentOS7

To remove CUDA Toolkit:

```
$ sudo yum remove "cuda*" "*cublas*" "*cufft*" "*curand*" \
 "*cusolver*" "*cusparse*" "*npp*" "*nvjpeg*" "nsight*"
```

To remove NVIDIA Drivers:

```
$ sudo yum remove "*nvidia*"
```

Fedora

To remove CUDA Toolkit:

```
$ sudo dnf remove "cuda*" "*cublas*" "*cufft*" "*curand*" \
 "*cusolver*" "*cusparse*" "*npp*" "*nvjpeg*" "nsight*"
```

To remove 3rd party NVIDIA Drivers:

```
$ sudo dnf remove "*nvidia*"
```

To remove NVIDIA Drivers:

```
$ sudo dnf remove nvidia-driver
```

To reset the module stream:

```
$ sudo dnf module reset nvidia-driver
```

OpenSUSE/SLES

To remove CUDA Toolkit:

```
$ sudo zypper remove "cuda*" "*cublas*" "*cufft*" "*curand*" \
```

```
 "*cusolver*" "*cusparse*" "*npp*" "*nvjpeg*" "nsight*"
```

To remove NVIDIA Drivers:

```
$ sudo zypper remove "*nvidia*"
```

Ubuntu and Debian

To remove CUDA Toolkit:

```
$ sudo apt-get --purge remove "*cublas*" "*cufft*" "*curand*" \
 "*cusolver*" "*cusparse*" "*npp*" "*nvjpeg*" "cuda*" "nsight*"
```

To remove NVIDIA Drivers:

```
$ sudo apt-get --purge remove "*nvidia*"
```

To clean up the uninstall:

```
$ sudo apt-get autoremove
```

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.

OpenCL

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

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

© 2009-2021 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

