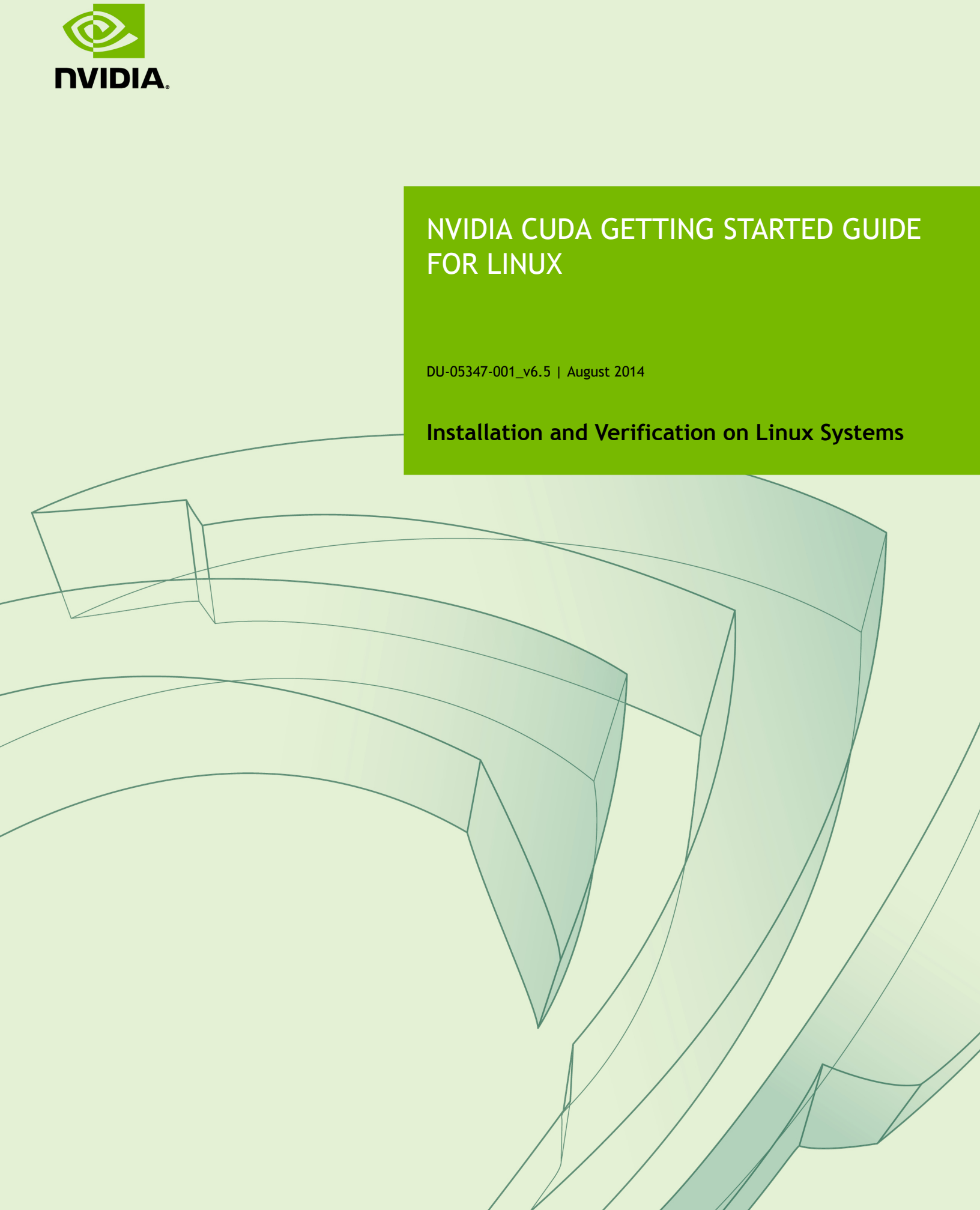

NVIDIA CUDA GETTING STARTED GUIDE FOR LINUX

DU-05347-001_v6.5 | August 2014

Installation and Verification on Linux Systems

TABLE OF CONTENTS

Chapter 1. Introduction 1
1.1. System Requirements 1
1.1.1. x86 32-bit Support 2
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. Choose an Installation Method 5
2.5. Download the NVIDIA CUDA Toolkit 6
2.6. Handle Conflicting Installation Methods 6
Chapter 3. Package Manager Installation 8
3.1. Overview 8
3.2. Redhat/CentOS 8
3.3. Fedora 9
3.4. SLES 10
3.5. OpenSUSE 11
3.6. Ubuntu 12
3.7. L4T 12
3.8. Additional Package Manager Capabilities 12
3.8.1. Available Packages 13
3.8.2. Package Upgrades 13
Chapter 4. Runfile Installation 14
4.1. Pre-installation Setup 14
4.2. Prerequisites 14
4.3. Contents 14
4.4. Graphical Interface Shutdown 15
4.5. Installation 15
4.6. Interaction with Nouveau 15
4.7. Extra Libraries 16
4.8. Verifications 16
4.9. Graphical Interface Restart 17
4.10. Post-installation Setup 17
4.11. Uninstallation 17
Chapter 5. Cross-build Environment for ARM 18
5.1. Cross-build Installation for ARM 18
5.2. Cross Samples 19
TARGET_FS 19
Copying Libraries 19
Ignore Symbol Detection 19

5.3. Nsight Eclipse Edition..... 20
Chapter 6. Post-installation Actions..... 21
6.1. Environment Setup..... 21
6.2. (Optional) Install Writable Samples..... 21
6.3. (Optional) Install Third-party Libraries..... 22
6.4. Verify the Installation..... 22
6.4.1. Verify the Driver Version..... 22
6.4.2. Compiling the Examples..... 22
6.4.3. Running the Binaries..... 22
Chapter 7. Frequently Asked Questions..... 25
How do I install the Toolkit in a different location?..... 25
Why do I see "nvcc: No such file or directory" when I try to build a CUDA application?..... 26
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?.. 26
How can I extract the contents of the installers?..... 26
How can I tell X to ignore a GPU for compute-only use?..... 26
Why doesn't the cuda-repo package install the CUDA Toolkit and Drivers?..... 27
Chapter 8. Additional Considerations..... 28

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 http://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.

Table 1 Native Linux Distribution Support in CUDA 6.5

Distribution	x86_64	x86(*)	ARMv7 (aarch32)	ARMv8 (aarch64)	Kernel	GCC	GLIBC	ICC(**)
Fedora 20	YES	NO	NO	NO	3.12.0	4.8.2	2.18	
CentOS 6.x	YES	NO	NO	NO	2.6.32	4.4.7	2.12	
CentOS 5.5+	DEPR	NO	NO	NO	2.6.18	4.1.2	2.5	
OpenSUSE 13.1	YES	NO	NO	NO	3.11.6	4.8	2.18	
RHEL 6.x	YES	NO	NO	NO	2.6.32	4.4.7	2.12	
RHEL 5.5+	DEPR	NO	NO	NO	2.6.18	4.1.2	2.5	14.0.1
SUSE SLES 11 SP3	YES	NO	NO	NO	3.0.76	4.3.4	2.11.3	
Ubuntu 14.04	YES	DEPR	YES	YES	3.13	4.8.2	2.19	
Ubuntu 12.04	DEPR	DEPR	NO	NO	3.2.0	4.6	2.15	
SteamOS 1.0-beta	YES	NO	NO	NO	3.10.11	4.7.2	2.17	
L4T r21.1	NO	NO	YES	NO	3.10.24	4.8.2	2.19	

Table 2 Cross-build Environment Linux Distribution Support in CUDA 6.5

Host Distribution (x86_64)	Targeting x86(*)	Targeting ARMv7 (aarch32)	Targeting ARMv8 (aarch64)
Ubuntu 14.04	DEPR	YES	YES
Ubuntu 12.04	DEPR	DEPR	NO
SteamOS 1.0-beta	YES	NO	NO

(*) x86 support is limited. See the x86 32-bit Support section for details.

(**) ICC support is limited to x86_64 only

DEPR = Support deprecated

1.1.1. x86 32-bit Support

Support for x86 32-bit applications on x86 and x86_64 Linux is limited to use with:

- GeForce GPUs with Kepler or higher architecture
- CUDA Driver
- CUDA Runtime (cudart)
- CUDA Math Library (math.h)
- CUDA C++ Compiler (nvcc)
- CUDA Development Tools

Support for this configuration is only available in the .run file installer.

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.

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.
- Download the NVIDIA CUDA Toolkit.
- Handle conflicting installation methods.

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 http://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. Choose an Installation Method

The CUDA Toolkit can be installed using either of two different installation mechanisms: distribution-specific packages, or a distribution-independent package. 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.

Distribution-specific packages and repositories are not provided for Redhat 5. For Redhat 5, the stand-alone installer must be used.

Standalone installers are not provided for the ARMv7 release. For both native ARMv7 as well as cross development, the toolkit must be installed using the distribution-specific installer. See the Cross-build Environment for ARM installation section for more details.

2.5. Download the NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit is available at http://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 http://developer.nvidia.com/cuda-downloads/checksums 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>
```

2.6. 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 3 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 4 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/uninstall_cuda_X.Y.pl
```

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 apt-get --purge remove <package_name>          # Ubuntu
$ sudo yum remove <package_name>                      # Fedora/Redhat/CentOS
$ sudo zypper remove <package_name>                   # OpenSUSE/SLES
```

Chapter 3.
PACKAGE MANAGER INSTALLATION

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:

- Redhat/CentOS
- Fedora
- SLES
- OpenSUSE
- Ubuntu
- L4T

Finally, some helpful package manager capabilities are detailed.

These instructions are for native development only. For cross development, see the Cross-build environment for ARM section.

3.2. Redhat/CentOS

1. Perform the pre-installation actions.
2. **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.

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**

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

5. **Clean Yum repository cache**

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

6. **Install CUDA**

   ```
   $ sudo yum install cuda
   ```

   If the i686 libvdpau package dependency fails to install, try using the following steps to fix the issue:

   ```
   $ yumdownloader libvdpau.i686
   $ sudo rpm -U --oldpackage libvdpau*.rpm
   ```

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. 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. **Satisfy Akmods dependency**

   The NVIDIA driver RPM packages depend on the Akmods framework which is provided by the RPMFusion free repository. The RPMFusion free repository 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.

4. **Install repository meta-data**

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

5. **Clean Yum repository cache**

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

6. **Install CUDA**

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

If also installing the **gpu-deployment-kit** package, the **cuda** and **gpu-deployment-kit** packages should be either installed using separate instances of **yum**:

```
$ sudo yum install cuda
$ sudo yum install gpu-deployment-kit
```

Or, installed while also specifying the **cuda-drivers** package:

```
$ sudo yum install cuda cuda-drivers gpu-deployment-kit
```

If a system has installed both packages with the same instance of **yum**, some driver components may be missing. Such an installation can be corrected by running:

```
$ sudo yum install cuda-drivers
```

If the i686 libvdpau package dependency fails to install, try using the following steps to fix the issue:

```
$ yumdownloader libvdpau.i686
$ sudo rpm -U --oldpackage libvdpau*.rpm
```

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. SLES

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

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

3. **Refresh Zypper repository cache**

```
$ sudo zypper refresh
```

4. **Install CUDA**

```
$ sudo zypper install cuda
```

The driver is provided in multiple packages, nvidia-gfxG03-kmp-desktop, nvidia-gfxG03-kmp-default, nvidia-gfxG03-kmp-trace, and their Unified Memory variants. When installing cuda, the correct driver packages should also be specified. Without doing this, zypper will select packages that may not work on the system. Run the following to detect the flavor of kernel and install cuda with the appropriate driver packages:

```
$ uname -r
  3.4.6-2.10-<flavor>
$ sudo zypper install cuda nvidia-gfxG03-kmp-<flavor> \
                       nvidia-uvm-gfxG03-kmp-<flavor>
```

5. **Add the user to the video group**

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

6. **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.

7. Perform the post-installation actions.

3.5. OpenSUSE

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

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

3. **Refresh Zypper repository cache**

```
$ sudo zypper refresh
```

4. **Install CUDA**

```
$ sudo zypper install cuda
```

The driver is provided in multiple packages, nvidia-gfxG03-kmp-desktop, nvidia-gfxG03-kmp-default, nvidia-gfxG03-kmp-trace, and their Unified Memory variants. When installing cuda, the correct driver packages should also be specified. Without doing this, zypper will select packages that may not work on the system. Run the following to detect the flavor of kernel and install cuda with the appropriate driver packages:

```
$ uname -r
  3.4.6-2.10-<flavor>
$ sudo zypper install cuda nvidia-gfxG03-kmp-<flavor> \
                       nvidia-uvm-gfxG03-kmp-<flavor>
```

5. **Add the user to the video group**

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

6. Perform the post-installation actions.

3.6. Ubuntu

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

When using a proxy server with aptitude, ensure that **wget** is set up to use the same proxy settings before installing the cuda-repo package.

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

3. **Update the Apt repository cache**

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

4. **Install CUDA**

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

5. Perform the post-installation actions.

3.7. L4T

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

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

3. **Update the Apt repository cache**

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

4. **Install CUDA Toolkit**

```
$ sudo apt-get install cuda-toolkit-6-5
```

5. **Add the user to the video group**

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

6. Perform the post-installation actions.

3.8. Additional Package Manager Capabilities

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

3.8.1. 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,... For x86_64 patforms, this also include NSight Eclipse Edition and the visual profiler It also includes the NVIDIA driver package.

On supported platforms, the **cuda-cross-armhf** package installs all the packages required for cross-platform development on ARMv7. The libraries and header files of the ARMv7 display driver package are also installed to enable the cross compilation of ARMv7 applications. The **cuda-cross-armhf** package does 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 & Fedora
$ zypper packages -r cuda                                      # OpenSUSE & SLES
$ cat /var/lib/apt/lists/*cuda*Packages | grep "Package:"      # Ubuntu
```

3.8.2. 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                                        # RedHat & Fedora
$ sudo zypper install cuda                                     # OpenSUSE & SLES
$ sudo apt-get install cuda                                    # Ubuntu
```

The **cuda-cross-armhf** package 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 cuda-drivers                                # RedHat & Fedora
$ sudo zypper install cuda-drivers \
                      nvidia-gfxG03-kmp-<flavor> \
                      nvidia-uvm-gfxG03-kmp-<flavor>           # 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-armhf-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.

Chapter 4. RUNFILE INSTALLATION

This section describes the installation and configuration of CUDA when using the standalone installer.

4.1. Pre-installation Setup

Before the stand-alone installation can be run, perform the pre-installation actions.

4.2. Prerequisites

If you have already installed a standalone CUDA driver and desire to keep using it, you need to make sure it meets the minimum version requirement for the toolkit. This requirement can be found in the CUDA Toolkit release notes. With many distributions, the driver version number can be found in the graphical interface menus under **`Applications > System Tools > NVIDIA X Server Settings`**.. On the command line, the driver version number can be found by running **`/usr/bin/nvidia-settings`**.

4.3. Contents

The standalone installer can install any combination of the NVIDIA Driver (that includes the CUDA Driver), the CUDA Toolkit, or the CUDA Samples. If needed, each individual installer can be extracted by using the **`-extract=/absolute/path/to/extract/location/`**. The extraction path must be an absolute path.

The CUDA Toolkit installation includes a read-only copy of the CUDA Samples. The read-only copy is used to create a writable copy of the CUDA Samples at some other location at any point in time. To create this writable copy, use the **`cuda-install-samples-6.5.sh`** script provided with the toolkit. It is equivalent to installing the CUDA Samples with the standalone installer.

4.4. Graphical Interface Shutdown

Exit the GUI if you are in a GUI environment by pressing **Ctrl-Alt-Backspace**. Some distributions require you to press this sequence twice in a row; others have disabled it altogether in favor of a command such as **`sudo service lightdm stop`**. Still others require changing the system runlevel using a command such as **`/sbin/init 3`** Consult your distribution's documentation to find out how to properly exit the GUI. This step is only required in the event that you want to install the NVIDIA Display Driver included in the standalone installer.

4.5. Installation

To install any combination of the driver, toolkit, and the samples, simply execute the .run script. The installation of the driver requires the script to be run with root privileges. Depending on the target location, the toolkit and samples installations may also require root privileges.

By default, the toolkit and the samples will install under **`/usr/local/cuda-6.5`** and **`$(HOME)/NVIDIA_CUDA-6.5_Samples`**, respectively. In addition, a symbolic link is created from **`/usr/local/cuda`** to **`/usr/local/cuda-6.5`**. The symbolic link is created in order for existing projects to automatically make use of the newly installed CUDA Toolkit.

If the target system includes both an integrated GPU (iGPU) and a discrete GPU (dGPU), the --no-opengl-libs option must be used. Otherwise, the openGL library used by the graphics driver of the iGPU will be overwritten and the GUI will not work. In addition, the xorg.conf update at the end of the installation must be declined.

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.

4.6. Interaction with Nouveau

The Nouveau drivers may be installed into your root filesystem (initramfs) and may cause the Display Driver installation to fail. To fix the situation, the initramfs image must be rebuilt with:

```
$ sudo mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r)-
nouveau.img
$ sudo dracut /boot/initramfs-$(uname -r).img $(uname -r)
```

if Grub2 is used as the bootloader, the **`rdblacklist=nouveau nouveau.modeset=0`** line must be added at the end of the GRUB_CMDLINE_LINUX entry in /etc/default/grub. Then, the Grub configuration must be remade by running:

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

Once this is done, the machine must be rebooted and the installation attempted again.

4.7. Extra Libraries

If you wish to build *all* the samples, including those with graphical rather than command-line interfaces, additional system libraries or headers may be required. While every Linux distribution is slightly different with respect to package names and package installation procedures, the libraries and headers most likely to be necessary are OpenGL (e.g., Mesa), GLU, GLUT, and X11 (including Xi, Xmu, and GLX).

On Ubuntu, those can be installed as follows:

```
$ sudo apt-get install freeglut3-dev build-essential libx11-dev libxmu-dev
 libxi-dev libgl1-mesa-glx libglu1-mesa libglu1-mesa-dev
```

4.8. Verifications

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. Some systems disallow setuid binaries, however, so if these files do not exist, you can create them manually either by running the command **nvidia-smi** as root at boot time or 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
```

4.9. Graphical Interface Restart

Restart the GUI environment using the command **startx**, **init 5**, **sudo service lightdm start**, or the equivalent command on your system.

4.10. Post-installation Setup

Once the stand-alone installation is complete, be sure to perform the post-installation actions.

4.11. 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-6.5/bin**:

```
$ sudo /usr/local/cuda-6.5/bin/uninstall_cuda_6.5.pl
```

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

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

Chapter 5.
CROSS-BUILD ENVIRONMENT FOR ARM

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

Due to the supported ARMv7 native platforms being Ubuntu 14.04 based, we recommend selecting Ubuntu 14.04 as your cross development platform. This selection helps prevent host/target incompatibilities, such as GCC or GLIBC version mismatches.

5.1. Cross-build Installation for ARM

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 ARMv7 target system. To install the native CUDA Toolkit on the target ARMv7 system, refer to the native Ubuntu and L4T installation sections.

1. Perform the pre-installation actions.
2. **Enable armhf foreign architecture**

   The armhf foreign architecture must be enabled in order to install the cross-armhf toolkit. To enable armhf as a foreign architecture, the following commands must be executed:

   On Ubuntu 12.04,

   ```
   $ sudo sh -c \
     'echo "foreign-architecture armhf" >> /etc/dpkg/dpkg.cfg.d/multiarch'
   $ sudo apt-get update
   ```

   On Ubuntu 14.04,

   ```
   $ sudo dpkg --add-architecture armhf
   $ sudo apt-get update
   ```
3. **Install repository meta-data**

When using a proxy server with aptitude, ensure that **`wget`** is set up to use the same proxy settings before installing the cuda-repo package.

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

4. **Update the Apt repository cache**

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

5. **Install the cross-ARM CUDA Toolkit**

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

6. Perform the post-installation actions.

5.2. Cross Samples

When cross-compiling an ARM CUDA application, nvcc must be able to find any libraries used, or be told to ignore missing symbols. One of the following methods should be chosen when cross-compiling the CUDA Samples. Regardless of which option is chosen, **`ARMv7=1`** should always be used.

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 ARMv7=1 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. For samples which use GL, HEADER_SEARCH_PATH must be set. For example, to cross-compile a sample which uses GL, execute:

```
$ make ARMv7=1 \
       EXTRA_LDFLAGS="-rpath-link=/opt/target/libs -L/opt/target/libs" \
       GLPATH=/opt/target/libs \
       HEADER_SEARCH_PATH=/opt/target/include
```

Ignore Symbol Detection

If neither of the above options are available, the linker can be told to ignore unresolved symbols. The samples should be forced to build using SAMPLE_ENABLED, and any library inclusion (**`-lfoo`**) should be removed from the Makefiles. To perform such a build, execute:

```
$ make ARMv7=1 \
       EXTRA_LDFLAGS="--unresolved-symbols=ignore-in-object-files" \
       SAMPLE_ENABLED=1
```

5.3. Nsight Eclipse Edition

Nsight Eclipse Edition supports cross-platform development. See the Nsight Eclipse Edition Getting Started Guide for more details.

Chapter 6. POST-INSTALLATION ACTIONS

Some actions must be taken after installing the CUDA Toolkit and Driver before they can be completely used:

- Setup evironment variables.
- Install a writable copy of the CUDA Samples.
- Install third-party libraries used by some of the CUDA Samples.
- Verify the installation.

6.1. Environment Setup

The **PATH** variable needs to include **/usr/local/cuda-6.5/bin**

The **LD_LIBRARY_PATH** variable needs to contain **/usr/local/cuda-6.5/lib64** on a 64-bit system, and **/usr/local/cuda-6.5/lib** on a 32-bit ARM system

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

```
$ export PATH=/usr/local/cuda-6.5/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib64:$LD_LIBRARY_PATH
```

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

```
$ export PATH=/usr/local/cuda-6.5/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib:$LD_LIBRARY_PATH
```

6.2. (Optional) 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-6.5.sh <dir>
```

This script is installed with the cuda-samples-6-5 package. The cuda-samples-6-5 package installs only a read-only copy in /usr/local/cuda-6.5/samples.

6.3. (Optional) 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.

6.4. 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.

Ensure the PATH and LD_LIBRARY_PATH variables are set correctly.

6.4.1. Verify the Driver Version

If you installed the driver, verify that the correct version of it is installed.

This can be done through your System Properties (or equivalent) or by executing the command

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

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

6.4.2. 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-6.5_Samples`** and typing **`make`**. The resulting binaries will be placed under **`~/NVIDIA_CUDA-6.5_Samples/bin`**.

6.4.3. Running the Binaries

After compilation, find and run **`deviceQuery`** under **`~/NVIDIA_CUDA-6.5_Samples`**. If the CUDA software is installed and configured correctly, the output for **`deviceQuery`** should look similar to that shown in Figure 1.

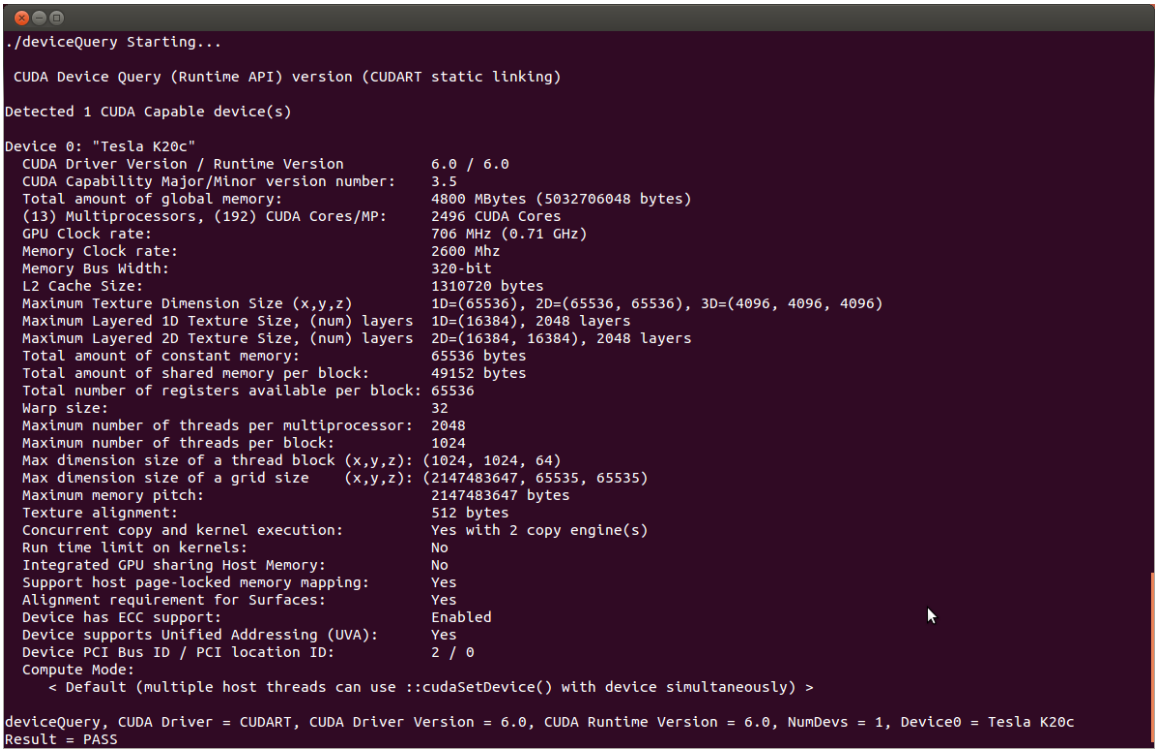

Figure 1 Valid Results from deviceQuery CUDA Sample

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.

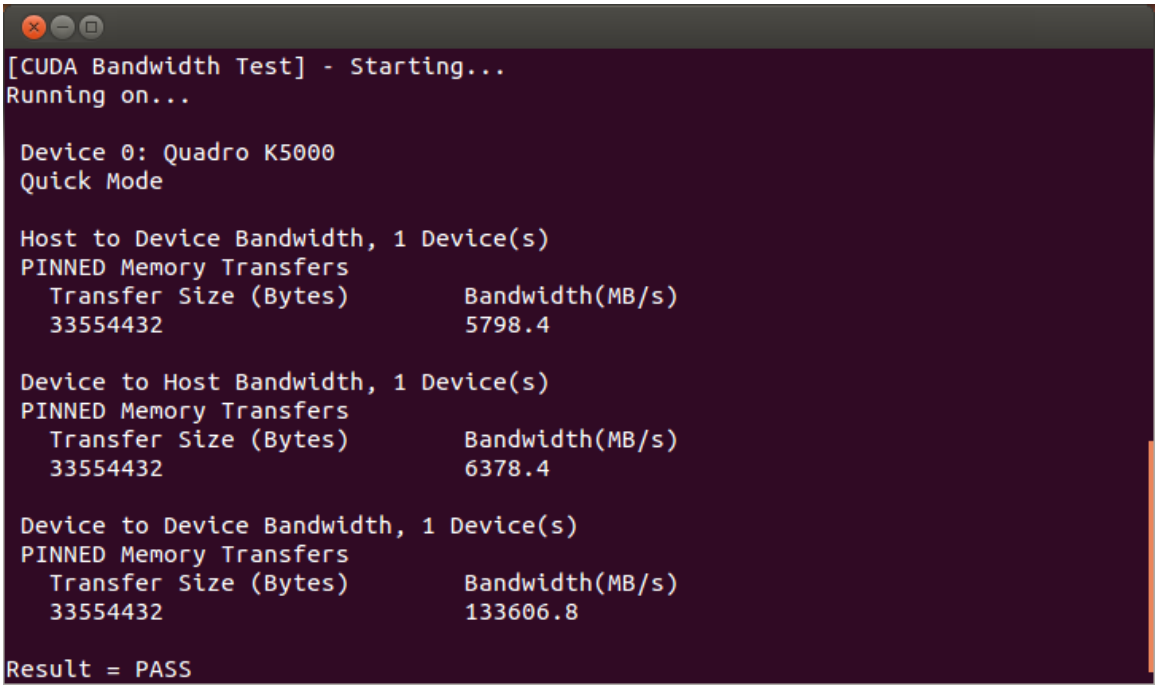

Figure 2 Valid Results from bandwidthTest CUDA Sample

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.

Chapter 7. 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 packages don't support custom install locations though the package managers (Yum and Zypper), but it is possible to install the RPM packages in custom locations using rpm's --relocate parameter:

```
$ rpm --install --relocate /usr/local/cuda-6.5=/my/new/toolkit rpmpackage.rpm
```

The Deb packages don't support custom install locations through the package manager (apt), but it is possible to install the Deb packages in custom locations using dpkg's --instdir parameter:

```
$ dpkg --instdir=/my/new/toolkit --install debpackage.deb
```

For RPM and Deb packages, you will need to install the packages in the correct order of dependency; normally the package managers take care of this automatically. 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 or Deb package as follows:

```
$ rpm -qRp rpmpackage.rpm
$ dpkg -I debpackage.deb | grep Depends
```

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-6.5/bin.

```
$ export PATH=/usr/local/cuda-6.5/bin:$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-6.5/lib{,64}:

```
$ export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib:$LD_LIBRARY_PATH
```

How can I extract the contents of the installers?

The Runfile can be extracted into the standalone Toolkit, Samples and Driver Runfiles by using the --extract parameter. These standalone Runfiles can be further extracted by running:

```
$ ./runfile.run --tar mxvf
```

The RPM packages can be extracted by running:

```
$ rpm2cpio rpm_package.rpm | cpio -idmv
```

The Deb packages can be extracted by running:

```
$ dpkg-deb -x deb_package.deb output_dir
```

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. This is done by editing the xorg.conf file located at /etc/xorg/xorg.conf.

You will need to add a section that resembles the following to your xorg.conf file:

```
Section "Device"
    Identifier     "Device0"
    Driver         "driver_name"
    VendorName     "vendor_name"
    BusID          "bus_id"
EndSection
```

The exact details of what 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`**.

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.

Chapter 8. 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-6.5/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 http://developer.nvidia.com/cuda/.

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

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

© 2009-2014 NVIDIA Corporation. All rights reserved.