



NVIDIA CUDA GETTING STARTED GUIDE FOR LINUX

DU-05347-001_v5.5 | July 2013

Installation and Verification on Linux Systems



TABLE OF CONTENTS

Chapter 1. Introduction.....	1
1.1. System Requirements.....	1
1.2. About This Document.....	2
Chapter 2. Installing CUDA Development Tools.....	3
2.1. Verify You Have a CUDA-Capable GPU.....	3
2.2. Verify You Have a Supported Version of Linux.....	4
2.3. Verify the System Has gcc Installed.....	4
2.4. Download the NVIDIA CUDA Toolkit.....	4
2.5. Install the NVIDIA CUDA Toolkit.....	5
2.5.1. Package Manager Installation.....	5
2.5.2. Runfile Installation.....	8
2.6. Verify the Installation.....	11
2.6.1. Verify the Driver Version.....	11
2.6.2. Compiling the Examples.....	11
2.6.3. Running the Binaries.....	12
Chapter 3. Additional Considerations.....	14

LIST OF FIGURES

Figure 1 Valid Results from deviceQuery CUDA Sample	12
Figure 2 Valid Results from bandwidthTest CUDA Sample	13

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>)

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.

INSTALLING CUDA DEVELOPMENT TOOLS

The setup of CUDA development tools on a system running the appropriate version of Linux consists of a few simple steps:

- ▶ 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.
- ▶ Install the NVIDIA CUDA Toolkit.
- ▶ Test that the installed software runs correctly and communicates with the hardware.



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:

```
i386 Red Hat Enterprise Linux WS release 4 (Nahant Update 6)
```

The **i386** line indicates you are running on a 32-bit system. On 64-bit systems running in 64-bit mode, this line will generally read: **x86_64**. The second line gives the version number of the operating system.

2.3. Verify the System Has gcc Installed

The **gcc** compiler and toolchain generally are 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.

For ARMv7 cross development, a suitable cross compiler is required. For example, performing the following on Ubuntu 12.04:

```
sudo apt-get install g++-4.6-arm-linux-gnueabi
```

will install the gcc 4.6 cross compiler on your system which will be used by **nvcc**. Please refer to the NVCC manual on how to use **nvcc** to cross compile to the ARMv7 architecture

2.4. 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.

2.5. Install the NVIDIA CUDA Toolkit

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 and Ubuntu 10.04. For those two Linux distributions, 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.

2.5.1. Package Manager Installation

The installation is a two-step process. First the small repository configuration package must be downloaded from the NVIDIA CUDA download page, and installed manually. The package sets the package manager database to include the CUDA repository. Then the CUDA Toolkit is installed using the package manager software.

Prerequisites

- ▶ The package manager installations (RPM/DEB packages) and the stand-alone installer installations (.run file) of the NVIDIA driver are incompatible. Before using the distribution-specific packages, uninstall the NVIDIA driver with the following command:

```
/usr/bin/nvidia-uninstall
```

- ▶ On Redhat, 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.
- ▶ For x86 to ARMv7 cross development, the host system must be an Ubuntu 12.04 system.
- ▶ On Ubuntu 12.04, to enable armhf as a foreign architecture, the following commands must be executed first:

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

Installation Instructions

► Redhat & Fedora

```
$ sudo rpm --install cuda-repo-<distro>-<version>.<architecture>.rpm
$ sudo yum clean expire-cache
$ sudo yum install cuda
```

► SLES & OpenSUSE

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

► Ubuntu

```
$ sudo dpkg -i cuda-repo-<distro>-<version>-<architecture>.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

Environment Setup

The **PATH** variable needs to include **/usr/local/cuda-5.5/bin**

The **LD_LIBRARY_PATH** variable needs to contain **/usr/local/cuda-5.5/lib** on a 32-bit system, and **/usr/local/cuda-5.5/lib64** on a 64-bit system

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

```
$ export PATH=/usr/local/cuda-5.5/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib:$LD_LIBRARY_PATH
```

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

```
$ export PATH=/usr/local/cuda-5.5/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib64:$LD_LIBRARY_PATH
```

(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-5.5.sh <dir>
```

This script is installed with the **cuda-samples-5-5** package. The **cuda-samples-5-5** package installs only a read-only copy in **/usr/local/cuda-5.5/samples**.

Available packages

The recommended installation packages are **cuda** and **cuda-cross**. Those two packages will install the full set of other CUDA packages required for 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 platforms, this also include NSight Eclipse Edition and the visual profiler It also includes the NVIDIA driver package.

The **cuda-cross** package installs all the packages required for cross-platform developments, such as the i386 and x86_64 CUDA libraries. 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** and **cuda-cross-armhf** 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 & Fedora
$ zypper pa -r cuda                                           # OpenSUSE & SLES
$ cat /var/lib/apt/lists/*cuda*Packages | grep "Package:"     # Ubuntu
```

Package Upgrades

The **cuda** and **cuda-cross** packages points to the latest stable release of the CUDA Toolkit. When a new version is available, the standard package managers upgrade command will install the latest CUDA Toolkit.

```
$ yum --disablerepo="*" --enablerepo="cuda" update           # RedHat & Fedora
$ zypper up -r cuda                                           # OpenSUSE & SLES
$ apt-get install cuda nvidia-current nvidia-current-dev \    # Ubuntu
    nvidia-modprobe nvidia-settings
```

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 installation to the X.Y release, install the **cuda-X-Y** or **cuda-cross-X-Y**.

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.

Distribution-Specific Quirks

RHEL and Fedora

- ▶ 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 remove the existing xorg.conf file, or add the contents of /etc/X11/xorg.conf.d/00-nvidia.conf to the xorg.conf file.
- ▶ 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.
- ▶ If the i686 libvdpau package dependency fails to install, try using the following steps to fix the issue:

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

Ubuntu

- ▶ When using a proxy server with aptitude, **wget** must be set up to use the same proxy settings before installing the cuda-repo package.
- ▶ The libcuda.so library is installed in the /usr/lib/nvidia-current 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 directory.

2.5.2. Runfile Installation



This section does not apply for ARMv7 development. The ARMv7 release is only available through the DEB package installer.

This section describes the installation and configuration of the CUDA Toolkit runfile, which you previously downloaded.

The toolkit installation can install any combination of the driver, toolkit and samples.

The toolkit also installs a pristine copy of the samples within the toolkit directory.

This directory is used by the Samples Browser, and can be used to replace the writable directory, should it become corrupt.

The samples installs and a writable copy of the samples where you should build and run the samples.

Before installing the CUDA software packages, you should read the bundled *Release Notes*, as the notes provide important details on installation and software functionality.

Then, follow these few steps for a successful installation.



If you wish to have a version of the Cuda Toolkit installed earlier than v5.5, it must be installed prior to installing a Cuda Toolkit v5.5 or greater.



If you've already installed a stand-alone driver, 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.

On many distributions, the driver version number can be found in the graphical interface menus under **Applications > System Tools > NVIDIA X Server Settings**. Or, from the command line, run:

```
/usr/bin/nvidia-settings
```



You can extract individual installers for each component by running (xx in 5.5.xx is the minor version of the installation package):

```
sh cuda_5.5.xx_linux_32_rhel5.x.run -extract=/path/to/extract/dir/
```

Keep in mind that the extraction path must be an absolute path.

1. 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 /etc/init.d/gdm 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.

2. If you are running an Ubuntu distro which installs the `nvidia-current` Debian package by default, and want to install the NVIDIA Display Driver, remove this package by running:

```
sudo apt-get --purge remove nvidia-current
```

3.  The driver and toolkit must be installed for CUDA to function. If you have not installed a stand-alone driver, install the driver from the NVIDIA CUDA Toolkit.



If the CUDA Samples are installed as a different user, others can install a writable copy of the samples by running `cuda-install-samples-5.5.sh <dir>`

Install the CUDA Toolkit (xx in 5.5.xx is the minor version of the installation package) by running the downloaded `.run` file as a *superuser*.

```
sudo sh cuda_5.5.xx_linux_32_rhel5.x.run
```

If you are using an Optimus system and are installing the driver, you must pass the `--optimus` option to the CUDA Toolkit installer. If you are instead installing a stand alone driver on an optimus system, you must pass `--no-opengl-files` to the installer and decline the `xorg.conf` update at the end of the installation.

If the Display Driver component fails to install, you may have the Nouveau drivers worked into your root filesystem (initramfs). To remove this, you will have to rebuild your initramfs image:

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

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

If you are using Grub2 as a bootloader, you may also need to edit its config to prevent Nouveau from loading. Add `"rdblacklist=nouveau nouveau.modeset=0"` to the end of the `GRUB_CMDLINE_LINUX` entry in `/etc/default/grub`. Then, remake your Grub configuration by running:

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

. Once this is done, reboot your machine and attempt the install again.

You can select which packages you wish to install at the start of the installation.

The CUDA Toolkit installation defaults to `/usr/local/cuda-5.5`. In addition, a symbolic link is created from `/usr/local/cuda` to `/usr/local/cuda-5.5` in order for existing projects to make use of the new CUDA Toolkit. A pristine copy of the samples is installed within the toolkit directory.

The CUDA Samples are installed to a user folder which defaults to `$(HOME) / NVIDIA_CUDA-5.5_Samples`.

4. Define the environment variables.

- ▶ The **PATH** variable needs to include `/usr/local/cuda-5.5/bin`
- ▶ **LD_LIBRARY_PATH** needs to contain `/usr/local/cuda-5.5/lib` on a 32-bit system, and both `/usr/local/cuda-5.5/lib` and `/usr/local/cuda-5.5/lib64` on a 64-bit system

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

```
export PATH=/usr/local/cuda-5.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib:$LD_LIBRARY_PATH
```

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

```
export PATH=/usr/local/cuda-5.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib64:$LD_LIBRARY_PATH
```

5. If you wish to build *all* of the samples, including those with graphical rather than command-line interfaces, you may need to install additional system libraries or headers if you have not done so before.

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). These can be installed on Ubuntu as follows, for example:

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

6. 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

```

7. Restart the GUI environment (using the command **startx** or **init 5** or **sudo /etc/init.d/gdm start** or the equivalent command on your system).

More information on installing the driver is available at http://us.download.nvidia.com/XFree86/Linux-x86_64/319.32/README/index.html.



New versions of CUDA software can require later versions of Linux and of the NVIDIA driver, so always verify that you are running the correct driver release and version of Linux for the version of the CUDA toolkit you are using.



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.

2.6. 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](#).

2.6.1. 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 Optimus system.

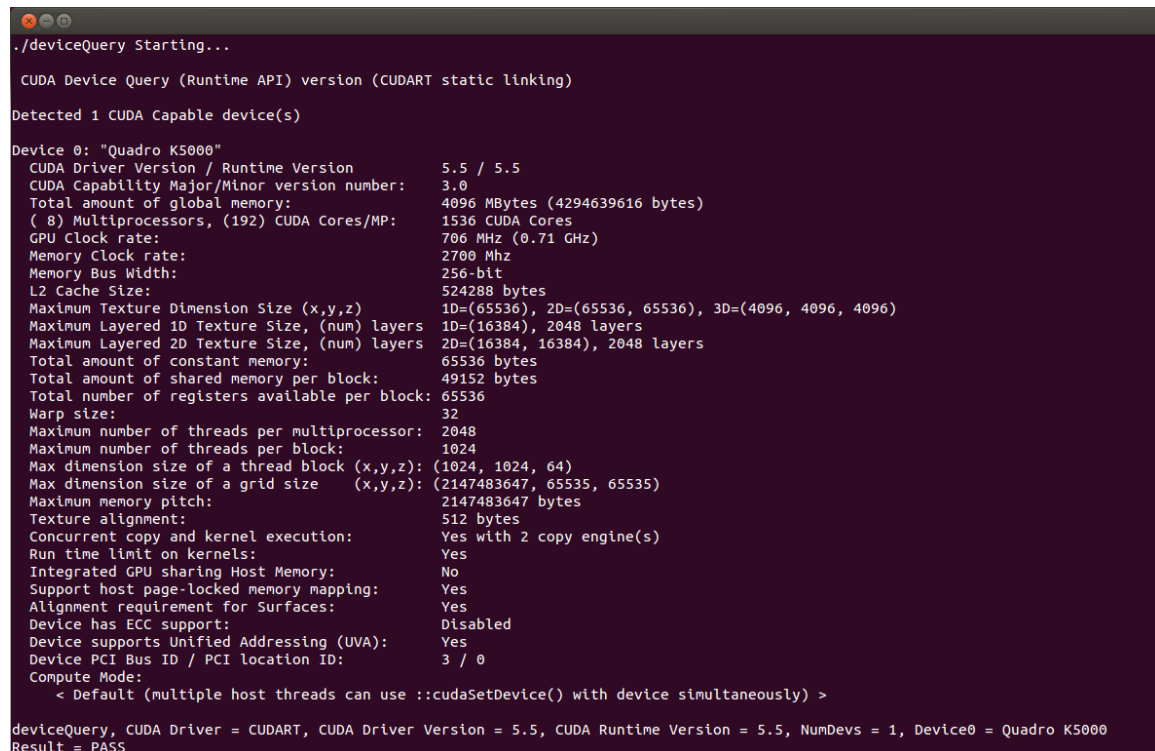
2.6.2. Compiling the Examples

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-5.5_Samples` and typing `make`. The resulting binaries will be placed in `~/NVIDIA_CUDA-5.5_Samples/bin/x86_64/linux/release`.

2.6.3. Running the Binaries

After compilation, go to `~/NVIDIA_CUDA-5.5_Samples/bin/linux/release` and run `deviceQuery`. If the CUDA software is installed and configured correctly, the output for `deviceQuery` should look similar to that shown in Figure 1.



```

./deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro K5000"
  CUDA Driver Version / Runtime Version      5.5 / 5.5
  CUDA Capability Major/Minor version number: 3.0
  Total amount of global memory:             4096 MBytes (4294639616 bytes)
  ( 8) Multiprocessors, (192) CUDA Cores/MP: 1536 CUDA Cores
  GPU Clock rate:                           706 MHz (0.71 GHz)
  Memory Clock rate:                        2700 Mhz
  Memory Bus Width:                         256-bit
  L2 Cache Size:                            524288 bytes
  Maximum Texture Dimension Size (x,y,z)    1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:           65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                     2147483647 bytes
  Texture alignment:                        512 bytes
  Concurrent copy and kernel execution:     Yes with 2 copy engine(s)
  Run time limit on kernels:                 Yes
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                    Disabled
  Device supports Unified Addressing (UVA):   Yes
  Device PCI Bus ID / PCI location ID:       3 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.5, CUDA Runtime Version = 5.5, NumDevs = 1, Device0 = Quadro K5000
Result = PASS

```

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).

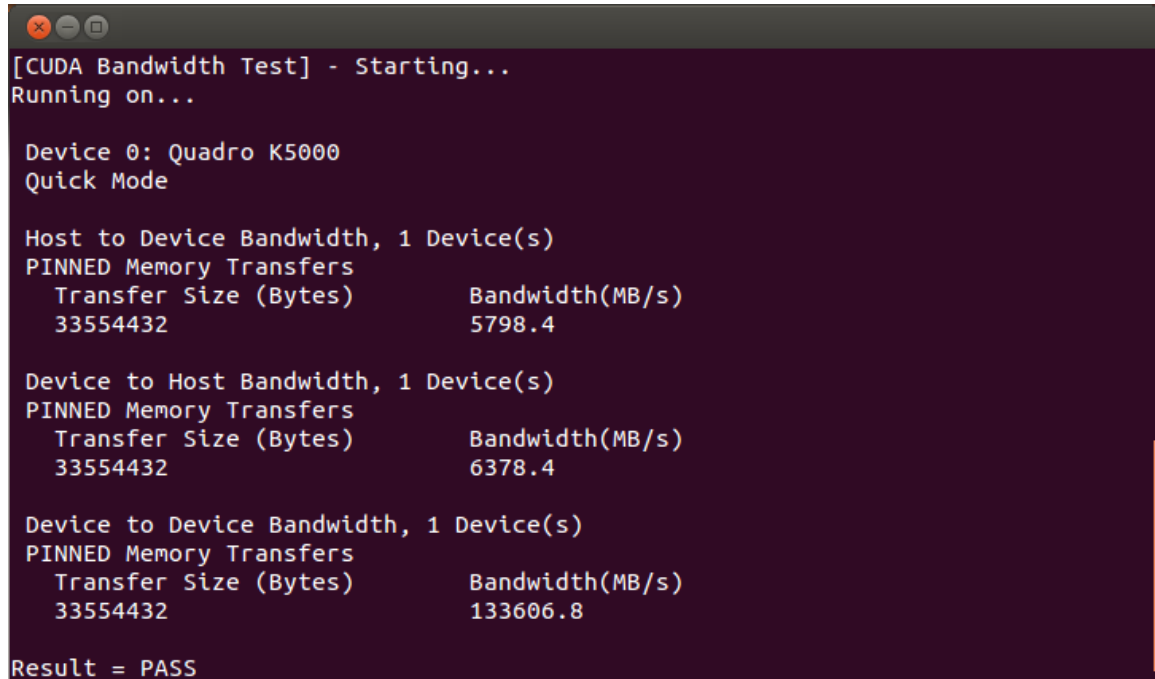
If a CUDA-capable device and the CUDA Driver are installed but `deviceQuery` reports that no CUDA-capable devices are present, this likely means that the `/dev/nvidia*` files are missing or have the wrong permissions.

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](#).



```
[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: Quadro K5000
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  5798.4

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  6378.4

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  133606.8

Result = PASS
```

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 3.

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-5.5/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

© 2009-2013 NVIDIA Corporation. All rights reserved.