**Setting up the GPGPU Environment on Ubuntu Linux 14.04 (with NVIDIA GeForce 1080)**

This article explains how to install CUDA Toolkits and set up the Ubuntu Linux environment. It is assumed that Ubuntu Linux and the graphics card. The purpose is to use graphics card for the GPGPU (General Purpose GPU) programming. The article consists of excerpts from NVIDIA's documents CUDA QUICK START GUIDE [1], NVIDIA CUDA INSTALLATION GUIDE FOR LINUX [2], and my own comments.

Step 1. Install the CUDA-capable hardware

Step 2. Install the OS

Step 3. Verify some prerequisites

$ lspci | grep -i nvidia

01:00.0 VGA compatible controller: NVIDIA Corporation Device 1b80 (rev a1)

01:00.1 Audio device: NVIDIA Corporation Device 10f0 (rev a1)

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

Step 4. Install gcc, the kernel headers and development packages

$ sudo apt-get install g++

$ gcc --version

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

Step 5. Download the NVIDIA CUDA Toolkit at <http://developer.nvidia.com/cuda-downloads>

Step 6. Install the repository meta-data, update the apt-get cache, and install CUDA:

$ sudo dpkg --install cuda-repo-<distro>-<version>.<architecture>.deb

$ sudo apt-get update

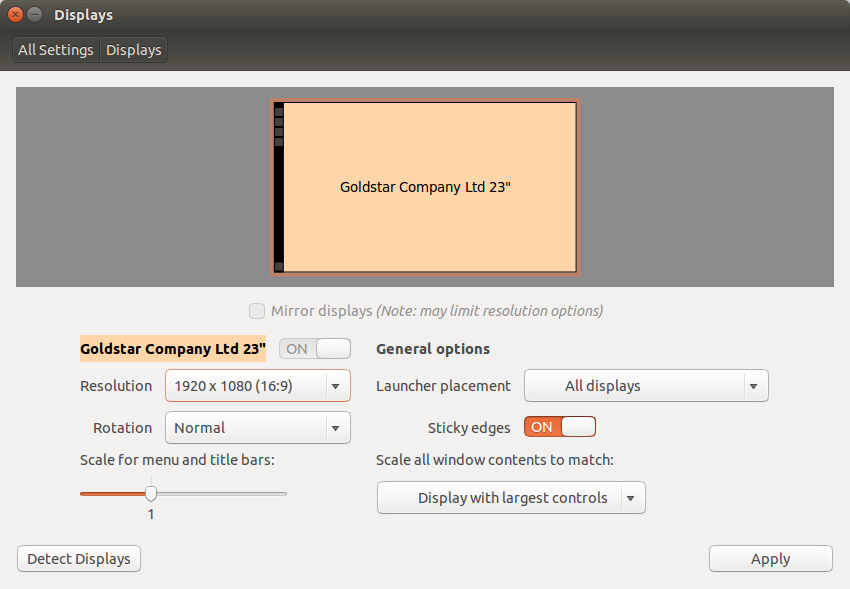
$ sudo apt-get install cuda

For example,

$ sudo dpkg -i cuda-repo-ubuntu1404-8-0-local-ga2\_8.0.61-1\_amd64.deb

Step 7. Reboot the system to load the NVIDIA drivers.

Before the reboot, the screen resolution was limited to 1024x768. Afterward, various display options are available.



Step 8. Set up the development environment by modifying the PATH and LD\_LIBRARY\_PATH variables:

$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

Step 9 . Check that the device files /dev/nvidia\* exist and have the correct (0666) file

permissions.

$ ls -al /dev/nvidia\*

crw-rw-rw- 1 root root 195, 0 2월 11 14:17 /dev/nvidia0

crw-rw-rw- 1 root root 195, 255 2월 11 14:17 /dev/nvidiactl

crw-rw-rw- 1 root root 195, 254 2월 11 14:17 /dev/nvidia-modeset

crw-rw-rw- 1 root root 240, 0 2월 11 14:17 /dev/nvidia-uvm

And check the NVIDIA driver version.

$ cat /proc/driver/nvidia/version

Step 10. Verify the installation by installing a writable copy of the samples then build and run some samples.

$ cuda-install-samples-8.0.sh <dir>

$ cd <dir>/NVIDIA\_CUDA-8.0\_Samples/<sample>

$ make

$ ./<cmd\_sample>

For example, compile and run the **nbody** sample.

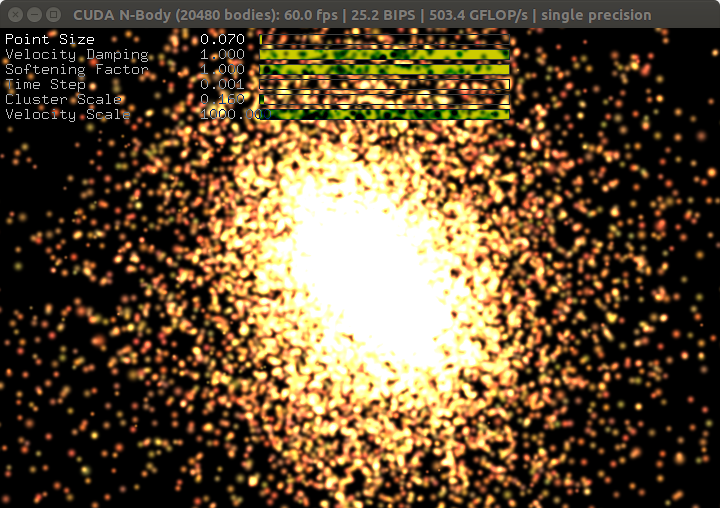
$ cuda-install-samples-8.0.sh ~

$ cd ~/NVIDIA\_CUDA-8.0\_Samples/5\_Simulations/nbody

$ make

$ ./nbody

The figure below is a screen capture of nbody result.



At this point, the basic GPGPU environment is set up and verified. As the next step, you may see the following topics (cuda-gdb/the cuda-gdb-src package, NVIDIA ® Nsight TM Eclipse Edition, NVIDIA Visual Profiler, cuda-memcheck) or read the CUDA C Programming Guide. For technical support on programming questions, consult and participate in the developer forums at <http://developer.nvidia.com/cuda/>.

**1. INTRODUCTION**

(Chapter 1 in [1] and a part of Chapter 1 in [2])

This guide covers the basic instructions needed to install CUDA and verify that a CUDA application can run on each supported platform [1]. 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) [2].

These instructions are intended to be used on a clean installation of a supported platform. For questions which are not answered in this document, please refer to the Windows Installation Guide, Mac Installation Guide, and Linux Installation Guide. The CUDA installation packages can be found on the CUDA Downloads Page [1].

**1.1. System Requirements**

(Chapter 1.1 in [2])

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 [2].

**2. PRE-INSTALLATION ACTIONS**

(A part of Chapter 2 in [2])

**2.1. Verify You Have a CUDA-Capable GPU**

(Chapter 2.1 in [2])

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.

**2.2. Verify You Have a Supported Version of Linux**

(Chapter 2.2 in [2])

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**

(Chapter 2.3 in [2])

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**

(Chapter 2.4 in [2])

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:

**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. Choose an Installation Method**

(Chapter 2.5 in [2])

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.

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.

**2.6. Download the NVIDIA CUDA Toolkit**

(Chapter 2.6 in [2])

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].

Among a few options below, say the Debian local installer is downloaded (cuda-repo-ubuntu1404-8-0-local-ga2\_8.0.61-1\_amd64.deb, as of Feb. 10, 2017)

|  |  |  |
| --- | --- | --- |
|  | Runfile Installer | Debian Installer |
| Local Installer | O | O |
| Network Installer | X | O |

**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>

**3. INSTALLTION (ON UBUNTU LINUX)**

(A part of Chapter 4 in [1])

CUDA on Linux can be installed using an RPM, Debian, or Runfile package, depending on the platform being installed on.

**3.1. Linux x86\_64**

(Chapter 4.1 in [1])

For development on the x86\_64 architecture. In some cases, x86\_64 systems may act as host platforms targeting other architectures. See the Linux Installation Guide for more details

**3.2. Ubuntu**

(Chapter 4.1.5 in [1])

When installing CUDA on Ubuntu, you can choose between the Runfile Installer and the Debian Installer. The Runfile Installer is only available as a Local Installer. The Debian Installer is available as both a Local Installer and a Network Installer. The Network Installer allows you to download only the files you need. The Local Installer is a standalone installer with a large initial download. In the case of the Debian installers, the instructions for the Local and Network variants are the same. For more details, refer to the [Linux Installation Guide](http://docs.nvidia.com/cuda/cuda-installation-guide-linux/).

**3.3. Debian Installer**

(Chapter 4.1.5.1 in [1])

Perform the following steps to install CUDA and verify the installation.

1. Install the repository meta-data, update the apt-get cache, and install CUDA:

$ sudo dpkg --install cuda-repo-<distro>-<version>.<architecture>.deb

$ sudo apt-get update

$ sudo apt-get install cuda

2. Reboot the system to load the NVIDIA drivers.

3. Set up the development environment by modifying the PATH and LD\_LIBRARY\_PATH variables:

$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

4. Install a writable copy of the samples then build and run the nbody sample:

$ cuda-install-samples-8.0.sh ~

$ cd ~/NVIDIA\_CUDA-8.0\_Samples/5\_Simulations/nbody

$ make

$ ./nbody

**3.4. Runfile Installer**

(Chapter 4.1.5.2 in [1])

Perform the following steps to install CUDA and verify the installation.

1. Disable the Nouveau drivers:

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

2. Reboot into runlevel 3 by temporarily adding the number "3" and the word "nomodeset" to the end of the system's kernel boot parameters.

3. Run the installer silently to install with the default selections (implies acceptance of the EULA):

$ sudo sh cuda\_<version>\_linux.run --silent

4. Create an xorg.conf file to use the NVIDIA GPU for display:

$ sudo nvidia-xconfig

5. Reboot the system to load the graphical interface.

6. Set up the development environment by modifying the PATH and

LD\_LIBRARY\_PATH variables:

$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

7. Install a writable copy of the samples then build and run the nbody sample:

$ cuda-install-samples-8.0.sh ~

$ cd ~/NVIDIA\_CUDA-8.0\_Samples/5\_Simulations/nbody

$ make

$ ./nbody

**3.5. Device Node Verification**

(Chapter 4.4 in [1])

Check that the device files /dev/nvidia\* exist and have the correct (0666) file

permissions.

$ ls -al /dev/nvidia\*

crw-rw-rw- 1 root root 195, 0 2월 11 14:17 /dev/nvidia0

crw-rw-rw- 1 root root 195, 255 2월 11 14:17 /dev/nvidiactl

crw-rw-rw- 1 root root 195, 254 2월 11 14:17 /dev/nvidia-modeset

crw-rw-rw- 1 root root 240, 0 2월 11 14:17 /dev/nvidia-uvm

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

**4. POST-INSTALLATION ACTIONS**

(Chapter 6 in [1])

The post-installation actions must be manually performed. These actions are split into

mandatory, recommended, and optional sections.

**4.1. Mandatory Actions**

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

can be used.

**4.1.1. Environment Setup**

The PATH variable needs to include /usr/local/cuda-8.0/bin

To add this path to the PATH variable:

$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

In addition, when using the runfile installation method, the LD\_LIBRARY\_PATH variable

needs to contain /usr/local/cuda-8.0/lib64 on a 64-bit system, or /usr/local/

cuda-8.0/lib on a 32-bit system

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

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

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

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

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

installation method.

**4.2. Recommended Actions**

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

**4.2.1. 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-8.0.sh <dir>

This script is installed with the cuda-samples-8-0 package. The cuda-samples-8-0

package installs only a read-only copy in /usr/local/cuda-8.0/samples.

**4.2.2. 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, if using the runfile installation method, LD\_LIBRARY\_PATH

variables are set correctly.

**4.2.2.1. 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.

**4.2.2.2. 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-8.0\_Samples and typing make . The

resulting binaries will be placed under ~/NVIDIA\_CUDA-8.0\_Samples/bin .

**4.2.2.3. Running the Binaries**

After compilation, find and run deviceQuery under ~/NVIDIA\_CUDA-8.0\_Samples .

If the CUDA software is installed and configured correctly, the output for deviceQuery

should look similar to that shown in Figure 1.

$ ./deviceQuery

./deviceQuery Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1080"

CUDA Driver Version / Runtime Version 8.0 / 8.0

CUDA Capability Major/Minor version number: 6.1

Total amount of global memory: 8110 MBytes (8504279040 bytes)

(20) Multiprocessors, (128) CUDA Cores/MP: 2560 CUDA Cores

GPU Max Clock rate: 1835 MHz (1.84 GHz)

Memory Clock rate: 5005 Mhz

Memory Bus Width: 256-bit

L2 Cache Size: 2097152 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)

Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers

Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 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 Domain ID / Bus ID / location ID: 0 / 1 / 0

Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GeForce GTX 1080

Result = PASS

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$

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 is missing. Screencapture the figure if necessary]

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.

**4.3. Optional Actions**

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

additional features.

**4.3.1. 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.

6.3.2. Install the source code for cuda-gdb

The cuda-gdb source is automatically installed with the runfile installation method.

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-8.0/extras directory.

**5. ADDITIONAL CONSIDERATIONS**

(Chapter 9 in [1])

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-8.0/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 TM 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/>.

**6. UNINSTALLATION**

(Chapter 4.6 in [1])

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-8.0/bin :

$ sudo /usr/local/cuda-8.0/bin/uninstall\_cuda\_8.0.pl

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.

**7. INTRODUCTION – REVISITED**

(A part of Chapter 1 in [2])

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.

**8. SOURCE OF THE EXCERPTS IN THIS DOCUMENT**

[1] CUDA QUICK START GUIDE, DU-050347-301\_V8.0, January 2017, available at <http://developer.download.nvidia.com/compute/cuda/8.0/secure/Prod2/docs/sidebar/CUDA_Quick_Start_Guide.pdf?autho=1486717650_74afbff8e39d37023433c8a354c73320&file=CUDA_Quick_Start_Guide.pdf>

[2] NVIDIA CUDA INSTALLATION GUIDE FOR LINUX - Installation and Verification on Linux Systems, DU-05347-001\_v8.0, January 2017, available at <http://developer.download.nvidia.com/compute/cuda/8.0/secure/Prod2/docs/sidebar/CUDA_Installation_Guide_Linux.pdf?autho=1486717649_18386f2906c46653ff81ad2feec88594&file=CUDA_Installation_Guide_Linux.pdf>

**9. APPENDIX**

**A. Example command results**

Step 1. Install the CUDA-capable hardware

Step 2. Install the OS

Step 3. Verify some prerequisites

$ lspci | grep -i nvidia

01:00.0 VGA compatible controller: NVIDIA Corporation Device 1b80 (rev a1)

01:00.1 Audio device: NVIDIA Corporation Device 10f0 (rev a1)

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

x86\_64

DISTRIB\_ID=Ubuntu

DISTRIB\_RELEASE=16.04

DISTRIB\_CODENAME=xenial

DISTRIB\_DESCRIPTION="Ubuntu 16.04.1 LTS"

NAME="Ubuntu"

VERSION="16.04.1 LTS (Xenial Xerus)"

ID=ubuntu

ID\_LIKE=debian

PRETTY\_NAME="Ubuntu 16.04.1 LTS"

VERSION\_ID="16.04"

HOME\_URL="http://www.ubuntu.com/"

SUPPORT\_URL="http://help.ubuntu.com/"

BUG\_REPORT\_URL="http://bugs.launchpad.net/ubuntu/"

UBUNTU\_CODENAME=xenial

Step 4. Install gcc, the kernel headers and development packages

$ sudo apt-get install g++

$ gcc --version

gcc (Ubuntu 5.4.0-6ubuntu1~16.04.1) 5.4.0 20160609

Copyright (C) 2015 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO

warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

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

[sudo] password for thkim:

Reading package lists... Done

Building dependency tree

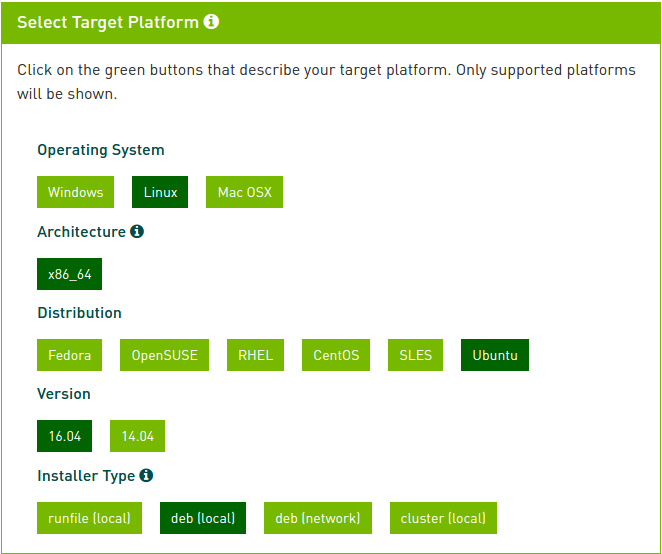
Reading state information... Done

linux-headers-4.4.0-31-generic is already the newest version (4.4.0-31.50).

linux-headers-4.4.0-31-generic set to manually installed.

0 upgraded, 0 newly installed, 0 to remove and 324 not upgraded.

Step 5. Download the NVIDIA CUDA Toolkit at <http://developer.nvidia.com/cuda-downloads>



cuda-repo-ubuntu1604-8-0-local-ga2\_8.0.61-1\_amd64.deb

Step 6. Install the repository meta-data, update the apt-get cache, and install CUDA:

$ sudo dpkg --install cuda-repo-<distro>-<version>.<architecture>.deb

$ sudo apt-get update

$ sudo apt-get install cuda

For example,

$ sudo dpkg -i cuda-repo-ubuntu1604-8-0-local-ga2\_8.0.61-1\_amd64.deb

Selecting previously unselected package cuda-repo-ubuntu1604-8-0-local-ga2.

(Reading database ... 176186 files and directories currently installed.)

Preparing to unpack cuda-repo-ubuntu1604-8-0-local-ga2\_8.0.61-1\_amd64.deb ...

Unpacking cuda-repo-ubuntu1604-8-0-local-ga2 (8.0.61-1) ...

Setting up cuda-repo-ubuntu1604-8-0-local-ga2 (8.0.61-1) ...

OK

$ sudo apt-get update

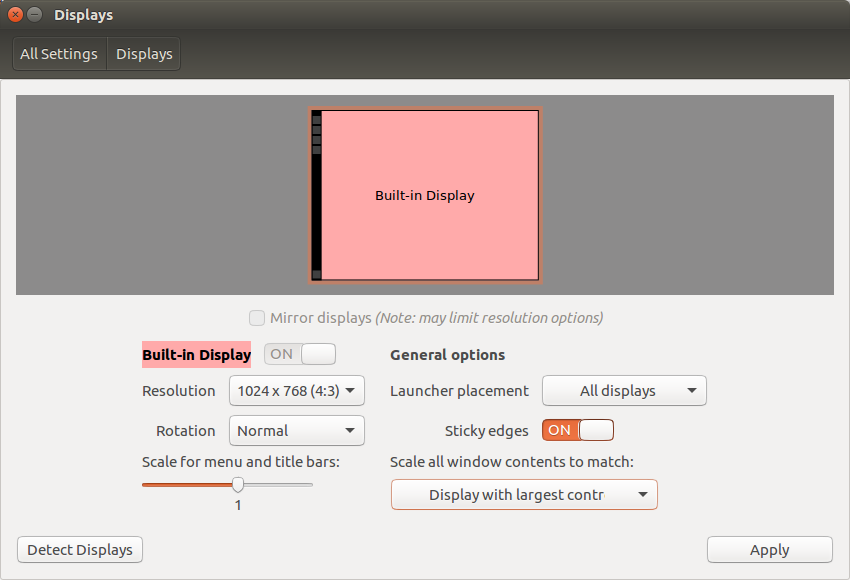
[many lines of commands]

$ sudo apt-get install cuda

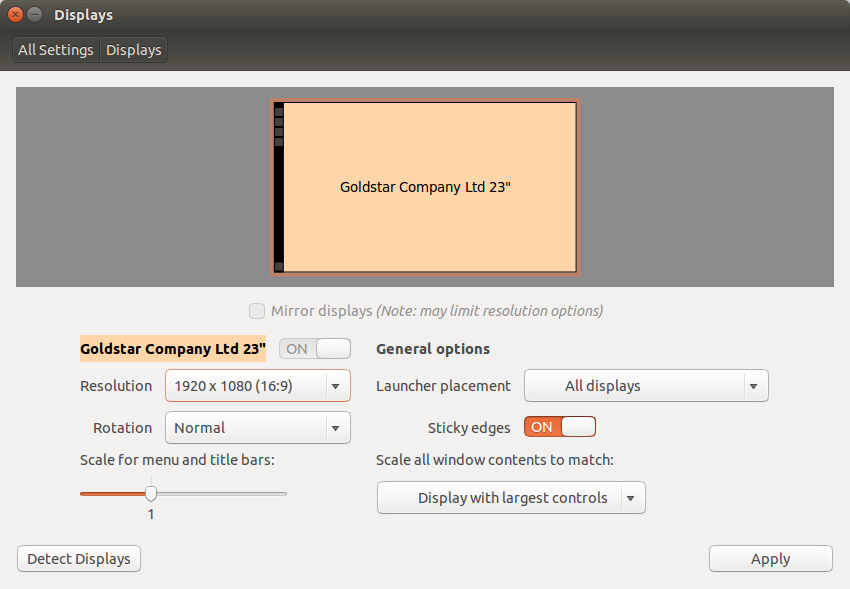
[many lines of commands]

Step 7. Reboot the system to load the NVIDIA drivers.

Before the reboot, the screen resolution was limited to 1024x768.



Afterward, various display options are available.



Step 8. Set up the development environment by modifying the PATH and LD\_LIBRARY\_PATH variables:

$ export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

$ export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64\

${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

Step 9 . Check that the device files /dev/nvidia\* exist and have the correct (0666) file

permissions.

$ ls -al /dev/nvidia\*

crw-rw-rw- 1 root root 195, 0 2월 11 14:17 /dev/nvidia0

crw-rw-rw- 1 root root 195, 255 2월 11 14:17 /dev/nvidiactl

crw-rw-rw- 1 root root 195, 254 2월 11 14:17 /dev/nvidia-modeset

crw-rw-rw- 1 root root 240, 0 2월 11 14:17 /dev/nvidia-uvm

And check the NVIDIA driver version.

$ cat /proc/driver/nvidia/version

NVRM version: NVIDIA UNIX x86\_64 Kernel Module 375.26 Thu Dec 8 18:36:43 PST 2016

GCC version: gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.1)

Step 10. Verify the installation by installing a writable copy of the samples then build and run some samples.

$ cuda-install-samples-8.0.sh <dir>

$ cd <dir>/NVIDIA\_CUDA-8.0\_Samples/<sample>

$ make

$ ./<cmd\_sample>

For example, compile and run the **nbody** sample.

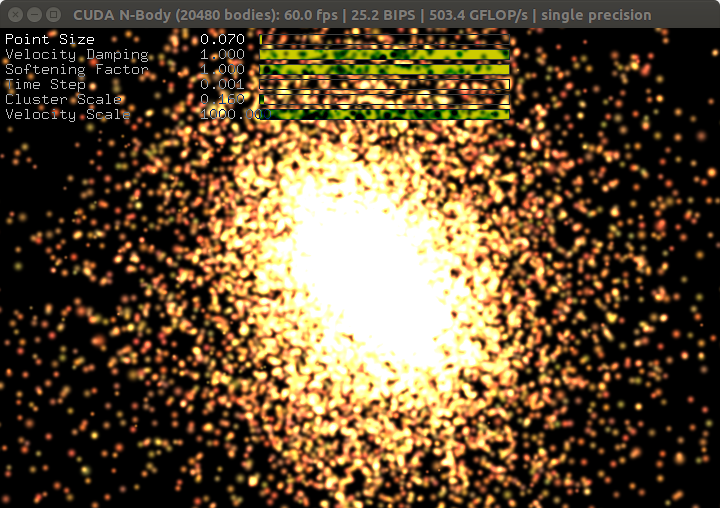
$ cuda-install-samples-8.0.sh ~

$ cd ~/NVIDIA\_CUDA-8.0\_Samples/5\_Simulations/nbody

$ make

$ ./nbody

The figure below is a screen capture of nbody result.



At this point, the basic GPGPU environment is set up and verified. As the next step, you may see the following topics (cuda-gdb/the cuda-gdb-src package, NVIDIA ® Nsight TM Eclipse Edition, NVIDIA Visual Profiler, cuda-memcheck) or read the CUDA C Programming Guide. For technical support on programming questions, consult and participate in the developer forums at <http://developer.nvidia.com/cuda/>.

**B. Test results**

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries$ cd ..

~/NVIDIA\_CUDA-8.0\_Samples$ ls

0\_Simple 1\_Utilities 2\_Graphics 3\_Imaging 4\_Finance 5\_Simulations 6\_Advanced 7\_CUDALibraries bin common EULA.txt Makefile

~/NVIDIA\_CUDA-8.0\_Samples$ find -name "deviceQuery"

./1\_Utilities/deviceQuery

~/NVIDIA\_CUDA-8.0\_Samples$ cd 1\_Utilities/deviceQuery

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ ls

deviceQuery.cpp Makefile NsightEclipse.xml readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ make

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -I../../common/inc -m64 -gencode arch=compute\_20,code=sm\_20 -gencode arch=compute\_30,code=sm\_30 -gencode arch=compute\_35,code=sm\_35 -gencode arch=compute\_37,code=sm\_37 -gencode arch=compute\_50,code=sm\_50 -gencode arch=compute\_52,code=sm\_52 -gencode arch=compute\_60,code=sm\_60 -gencode arch=compute\_60,code=compute\_60 -o deviceQuery.o -c deviceQuery.cpp

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -m64 -gencode arch=compute\_20,code=sm\_20 -gencode arch=compute\_30,code=sm\_30 -gencode arch=compute\_35,code=sm\_35 -gencode arch=compute\_37,code=sm\_37 -gencode arch=compute\_50,code=sm\_50 -gencode arch=compute\_52,code=sm\_52 -gencode arch=compute\_60,code=sm\_60 -gencode arch=compute\_60,code=compute\_60 -o deviceQuery deviceQuery.o

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

mkdir -p ../../bin/x86\_64/linux/release

cp deviceQuery ../../bin/x86\_64/linux/release

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ ls

deviceQuery deviceQuery.cpp deviceQuery.o Makefile NsightEclipse.xml readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ ./deviceQuery

./deviceQuery Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1080"

CUDA Driver Version / Runtime Version 8.0 / 8.0

CUDA Capability Major/Minor version number: 6.1

Total amount of global memory: 8110 MBytes (8504279040 bytes)

(20) Multiprocessors, (128) CUDA Cores/MP: 2560 CUDA Cores

GPU Max Clock rate: 1835 MHz (1.84 GHz)

Memory Clock rate: 5005 Mhz

Memory Bus Width: 256-bit

L2 Cache Size: 2097152 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)

Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers

Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 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 Domain ID / Bus ID / location ID: 0 / 1 / 0

Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GeForce GTX 1080

Result = PASS

~/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$

~$ ls

Desktop Downloads Music Pictures Templates

Documents examples.desktop NVIDIA\_CUDA-8.0\_Samples Public Videos

~$ cd NVIDIA\_CUDA-8.0\_Samples/

~/NVIDIA\_CUDA-8.0\_Samples

$ ls

0\_Simple 2\_Graphics 4\_Finance 6\_Advanced bin EULA.txt

1\_Utilities 3\_Imaging 5\_Simulations 7\_CUDALibraries common Makefile

**7\_CUDALibraries**

~/NVIDIA\_CUDA-8.0\_Samples$ cd 7\_CUDALibraries/

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries$ ls

batchCUBLAS cuSolverSp\_LowlevelCholesky nvgraph\_Pagerank

BiCGStab cuSolverSp\_LowlevelQR nvgraph\_SemiRingSpMV

boxFilterNPP FilterBorderControlNPP nvgraph\_SSSP

cannyEdgeDetectorNPP freeImageInteropNPP randomFog

common histEqualizationNPP simpleCUBLAS

conjugateGradient jpegNPP simpleCUBLASXT

conjugateGradientPrecond MC\_EstimatePiInlineP simpleCUFFT

conjugateGradientUM MC\_EstimatePiInlineQ simpleCUFFT\_2d\_MGPU

cuHook MC\_EstimatePiP simpleCUFFT\_callback

cuSolverDn\_LinearSolver MC\_EstimatePiQ simpleCUFFT\_MGPU

cuSolverRf MC\_SingleAsianOptionP simpleDevLibCUBLAS

cuSolverSp\_LinearSolver MersenneTwisterGP11213

**batchCUBLAS**

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries$ cd batchCUBLAS/

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS$ ls

batchCUBLAS.cpp batchCUBLAS.h Makefile NsightEclipse.xml readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS**$ make**

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -I../../common/inc -m64 -gencode arch=compute\_20,code=compute\_20 -o batchCUBLAS.o -c batchCUBLAS.cpp

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -m64 -gencode arch=compute\_20,code=compute\_20 -o batchCUBLAS batchCUBLAS.o -lcublas

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

mkdir -p ../../bin/x86\_64/linux/release

cp batchCUBLAS ../../bin/x86\_64/linux/release

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS$ ls

batchCUBLAS batchCUBLAS.h Makefile readme.txt

batchCUBLAS.cpp batchCUBLAS.o NsightEclipse.xml

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS**$ ./batchCUBLAS**

batchCUBLAS Starting...

GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

==== Running single kernels ====

Testing sgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0xbf800000, -1) beta= (0x40000000, 2)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00004196 sec GFLOPS=99.9556

@@@@ sgemm test OK

Testing dgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0x0000000000000000, 0) beta= (0x0000000000000000, 0)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00004601 sec GFLOPS=91.1512

@@@@ dgemm test OK

==== Running N=10 without streams ====

Testing sgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0xbf800000, -1) beta= (0x00000000, 0)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00015807 sec GFLOPS=265.342

@@@@ sgemm test OK

Testing dgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0xbff0000000000000, -1) beta= (0x0000000000000000, 0)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00031114 sec GFLOPS=134.806

@@@@ dgemm test OK

==== Running N=10 with streams ====

Testing sgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0x40000000, 2) beta= (0x40000000, 2)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00006294 sec GFLOPS=666.371

@@@@ sgemm test OK

Testing dgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0xbff0000000000000, -1) beta= (0x0000000000000000, 0)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00022602 sec GFLOPS=185.572

@@@@ dgemm test OK

==== Running N=10 batched ====

Testing sgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0x3f800000, 1) beta= (0xbf800000, -1)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00004005 sec GFLOPS=1047.15

@@@@ sgemm test OK

Testing dgemm

#### args: ta=0 tb=0 m=128 n=128 k=128 alpha = (0xbff0000000000000, -1) beta= (0x4000000000000000, 2)

#### args: lda=128 ldb=128 ldc=128

^^^^ elapsed = 0.00023293 sec GFLOPS=180.063

@@@@ dgemm test OK

Test Summary

0 error(s)

**batchCUBLAS**

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS$

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/batchCUBLAS$ cd ..

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries$ ls

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries$ cd BiCGStab/

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab$ ls

gr\_900\_900\_crg.mtx lap2D\_5pt\_n100.mtx Makefile mmio.c mmio.h mmio\_wrapper.h NsightEclipse.xml pbicgstab.cpp readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab**$ make**

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -I../../common/inc -m64 -gencode arch=compute\_20,code=sm\_20 -gencode arch=compute\_30,code=sm\_30 -gencode arch=compute\_35,code=sm\_35 -gencode arch=compute\_37,code=sm\_37 -gencode arch=compute\_50,code=sm\_50 -gencode arch=compute\_52,code=sm\_52 -gencode arch=compute\_60,code=sm\_60 -gencode arch=compute\_60,code=compute\_60 -o mmio.c.o -c mmio.c

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -I../../common/inc -m64 -gencode arch=compute\_20,code=sm\_20 -gencode arch=compute\_30,code=sm\_30 -gencode arch=compute\_35,code=sm\_35 -gencode arch=compute\_37,code=sm\_37 -gencode arch=compute\_50,code=sm\_50 -gencode arch=compute\_52,code=sm\_52 -gencode arch=compute\_60,code=sm\_60 -gencode arch=compute\_60,code=compute\_60 -o pbicgstab.o -c pbicgstab.cpp

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

pbicgstab.cpp: In function ‘int main(int, char\*\*)’:

pbicgstab.cpp:594:80: warning: deprecated conversion from string constant to ‘char\*’ [-Wwrite-strings]

DBICGSTAB\_MAX\_ULP\_ERR, DBICGSTAB\_EPS);

^

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -m64 -gencode arch=compute\_20,code=sm\_20 -gencode arch=compute\_30,code=sm\_30 -gencode arch=compute\_35,code=sm\_35 -gencode arch=compute\_37,code=sm\_37 -gencode arch=compute\_50,code=sm\_50 -gencode arch=compute\_52,code=sm\_52 -gencode arch=compute\_60,code=sm\_60 -gencode arch=compute\_60,code=compute\_60 -o BiCGStab mmio.c.o pbicgstab.o -lcusparse -lcublas

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

mkdir -p ../../bin/x86\_64/linux/release

cp BiCGStab ../../bin/x86\_64/linux/release

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab$ ls

BiCGStab gr\_900\_900\_crg.mtx lap2D\_5pt\_n100.mtx Makefile mmio.c mmio.c.o mmio.h mmio\_wrapper.h NsightEclipse.xml pbicgstab.cpp pbicgstab.o readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab$ ./BiCGStab

WARNING: it is assumed that the matrices are stores in Matrix Market format with double as elementtype

Usage: ./BiCGStab -F[matrix.mtx] [-E] [-D]

Starting [./BiCGStab]

argv[0] = ./BiCGStabUsing default input file [./gr\_900\_900\_crg.mtx]

GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Testing dbicgstab

^^^^ M=900, N=900, nnz=4322

Copy matrix from CPU to GPU, time(s) = 0.00003195

analysis lower 0.001365 (s), upper 0.000408 (s)

CUSPARSE csrilu0 time(s) = 0.00063205

gpu total solve time 0.524424 (s), matvec time 0.051965 (s)

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab$ ls ..

batchCUBLAS conjugateGradient cuSolverRf freeImageInteropNPP MC\_EstimatePiP nvgraph\_SemiRingSpMV simpleCUFFT

BiCGStab conjugateGradientPrecond cuSolverSp\_LinearSolver histEqualizationNPP MC\_EstimatePiQ nvgraph\_SSSP simpleCUFFT\_2d\_MGPU

boxFilterNPP conjugateGradientUM cuSolverSp\_LowlevelCholesky jpegNPP MC\_SingleAsianOptionP randomFog simpleCUFFT\_callback

cannyEdgeDetectorNPP cuHook cuSolverSp\_LowlevelQR MC\_EstimatePiInlineP MersenneTwisterGP11213 simpleCUBLAS simpleCUFFT\_MGPU

common cuSolverDn\_LinearSolver FilterBorderControlNPP MC\_EstimatePiInlineQ nvgraph\_Pagerank simpleCUBLASXT simpleDevLibCUBLAS

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/BiCGStab$ cd ../boxFilterNPP/

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/boxFilterNPP$ ls

boxFilterNPP.cpp Makefile NsightEclipse.xml readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/boxFilterNPP**$ make**

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -I../../common/inc -I../common/UtilNPP -I../common/FreeImage/include -m64 -gencode arch=compute\_20,code=compute\_20 -o boxFilterNPP.o -c boxFilterNPP.cpp

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

/usr/local/cuda-8.0/bin/nvcc -ccbin g++ -m64 -gencode arch=compute\_20,code=compute\_20 -o boxFilterNPP boxFilterNPP.o -L../common/FreeImage/lib/ -L../common/FreeImage/lib/linux -L../common/FreeImage/lib/linux/x86\_64 -lnppi\_static -lnppc\_static -lculibos -lfreeimage

nvcc warning : The 'compute\_20', 'sm\_20', and 'sm\_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).

mkdir -p ../../bin/x86\_64/linux/release 21

cp boxFilterNPP ../../bin/x86\_64/linux/release

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/boxFilterNPP$ ls

boxFilterNPP boxFilterNPP.cpp boxFilterNPP.o Makefile NsightEclipse.xml readme.txt

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/boxFilterNPP**$ ./boxFilterNPP**

./boxFilterNPP Starting...

GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

cudaSetDevice GPU0 = GeForce GTX 1080

NPP Library Version 8.0.61

CUDA Driver Version: 8.0

CUDA Runtime Version: 8.0

Device 0: <GeForce GTX 1080 >, Compute SM 6.1 detected

boxFilterNPP opened: <../../common/data/Lena.pgm> successfully!

Saved image: ../../common/data/Lena\_boxFilter.pgm

~/NVIDIA\_CUDA-8.0\_Samples/7\_CUDALibraries/boxFilterNPP$