
NVIDIA CUDA Software Development Kit (CUDA SDK)
Release Notes
Version 1.1 for Linux

Please, also refer to the release notes of version 1.1 of CUDA, installed by the CUDA Toolkit installer.

TABLE OF CONTENTS

- I. Quick Start Installation Instructions
 - II. Detailed Installation Instructions
 - III. Creating Your Own CUDA Program
 - IV. Supported Linux Distributions
 - V. Known Issues
 - VI. Frequently Asked Questions
 - VII. Change Log
-

I. Quick Start Instructions

For more detailed instructions, see section II below.

0. a. Install the NVIDIA Linux display driver by executing the file
NVIDIA-Linux-*-pkg1.run

(Note this is pkg2 for 64-bit linux.)

For information on installing NVIDIA Linux display drivers, please refer to the NVIDIA Accelerated Linux Driver Set README and Installation Guide:

<http://us.download.nvidia.com/XFree86/Linux-x86/1.0-9755/README/index.html>

1. Install version 1.1 of the NVIDIA CUDA Toolkit by executing the file
NVIDIA_CUDA_Toolkit_1.1-*.run corresponding to your Linux distribution

Add the CUDA binaries and lib path to your PATH and LD_LIBRARY_PATH environment variables.

2. Install the NVIDIA CUDA SDK by executing the file
NVIDIA_CUDA_SDK_1.1-*.run

The installer will prompt you to enter an installation path for the SDK or accept the default. We will refer to the path you choose as SDK_INSTALL_PATH.

3. Build the SDK project examples.

```
cd <SDK_INSTALL_PATH>  
make
```

4. Run the examples:

```
cd <SDK_INSTALL_PATH>/bin/linux32/release  
matrixmul
```

(or any of the other executables in that directory)

See the next section for more details on installing, building, and running SDK samples.

II. Detailed Installation Instructions

This package consists of a ".run" file. This is a self-extracting archive that decompresses its contents to a temporary folder and then installs the contents to

a path that you specify. The archive is:

NVIDIA_CUDA_SDK_1.1-*.run : NVIDIA CUDA SDK Installer

In addition, an NVIDIA Linux Display driver and the NVIDIA CUDA Toolkit have been made available along with this package. They are also installers with names like:

NVIDIA-Linux-*-pkg1.run (or pkg2 for 64-bit linux)

To install the driver and the CUDA Toolkit and SDK, follow the following instructions.

0. Install the NVIDIA Linux display driver by executing the file NVIDIA-Linux-*-pkg1.run

For information on installing NVIDIA Linux display drivers, please refer to the NVIDIA Accelerated Linux Driver Set README and Installation Guide: <http://us.download.nvidia.com/XFree86/Linux-x86/1.0-9755/README/index.html>

1. Install version 1.1 of the NVIDIA CUDA Toolkit by executing the file NVIDIA_CUDA_Toolkit_1.1-*.run corresponding to your Linux distribution

To install, run the NVIDIA_CUDA_Toolkit_1.1-*.run script. You will be prompted for the path to where you want to put the CUDA files. In the following we will call this path <CUDA_INSTALL_PATH>. It is recommended that you run the installer as root and use the default install path (/usr/local).

Make sure that you add the location of the CUDA binaries (such as nvcc) to your PATH environment variable and the location of the CUDA libraries (such as libcuda.so) to your LD_LIBRARY_PATH environment variable.

In the bash shell, one way to do this is to add the following lines to the file .bash_profile in your home directory.

```
PATH=$PATH:<CUDA_INSTALL_PATH>/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<CUDA_INSTALL_PATH>/lib
export PATH
export LD_LIBRARY_PATH
```

2. Install the NVIDIA CUDA SDK by executing the file NVIDIA_CUDA_SDK_1.1-*.run

To install, run the NVIDIA_CUDA_SDK_1.1-*.run script. You will be prompted for the path to where you want to put the CUDA SDK. You can regard the CUDA SDK as user code (it is a set of examples), and therefore the default installation is in the current user's home directory (~/.NVIDIA_CUDA_SDK). You must either accept the default or specify a path to which the user has write permissions.

We will refer to the path you choose as SDK_INSTALL_PATH below.

3. Build the SDK project examples.
 - a. Go to <SDK_INSTALL_PATH> ("cd <SDK_INSTALL_PATH>")
 - b. Build:
 - release configuration by typing "make".
 - debug configuration by typing "make dbg=1".
 - emurelease configuration by typing "make emu=1".
 - emudebug configuration by typing "make emu=1 dbg=1".

Running make at the top level first builds libcutil, a utility library used by the SDK examples (libcutil is simply for convenience -- it is not a part of CUDA and is not required for your own CUDA programs). Make then builds each of the projects in the SDK.

NOTES:

- The release and debug configurations require an NVIDIA 8 Series GPU (Such as an NVIDIA GeForce 8600 or 8800 or a Quadro FX 4600 or 5600 GPU) to run properly.
- The emurelease and emudebug configurations run in device emulation mode, and therefore do not require an NVIDIA 8 Series GPU to run properly.
- You can build an individual sample by typing "make" (or "make emu=1", etc.) in that sample's project directory. For example:

```
cd <SDK_INSTALL_PATH>/projects/matrixmul
make emu=1
```

And then execute the sample with:

```
<SDK_INSTALL_PATH>/bin/linux32/emurelease/matrixmul
```

- To build just libcutil, type "make" (or "make dbg=1") in the "common" subdirectory:

```
cd <SDK_INSTALL_PATH>/common
make
```

4. Run the examples from the release, debug, emurelease, or emudebug directories located in /bin/linux32/[release|debug|emurelease|emudebug].

III. Creating Your Own CUDA Program

Creating a new CUDA Program using the NVIDIA CUDA SDK infrastructure is easy. We have provided a "template" project that you can copy and modify to suit your needs. Just follow these steps:

1. Copy the template project

```
cd <SDK_INSTALL_PATH>/projects
cp -r template <myproject>
```

2. Edit the filenames of the project to suit your needs

```
mv template.cu myproject.cu
mv template_kernel.cu myproject_kernel.cu
mv template_gold.cpp myproject_gold.cpp
```

3. Edit the Makefile and source files. Just search and replace all occurrences of "template" with "myproject".

4. Build the project

```
make
```

You can build a debug version with "make dbg=1", an emulation version with "make emu=1", and a debug emulation with "make dbg=1 emu=1".

5. Run the program

```
../../bin/linux32/release/myproject
```

(It should print "Test PASSED")

6. Now modify the code to perform the computation you require. See the CUDA Programming Guide for details of programming in CUDA.

IV. Supported Linux Distributions

NVIDIA CUDA version 1.1 is supported and tested on the following Linux distributions:

32-bit operating systems

- Windows XP
- Red Hat Enterprise Linux 3.8
- Red Hat Enterprise Linux 4.3
- Red Hat Enterprise Linux 4.4
- Red Hat Enterprise Linux 5.0
- SUSE Linux Enterprise Desktop 10.0
- SUSE Linux 10.1

- SUSE Linux 10.2

64-bit operating systems

- Red Hat Enterprise Linux 3.8
 - Red Hat Enterprise Linux 4.3
 - Red Hat Enterprise Linux 4.4
 - Red Hat Enterprise Linux 5.0
 - SUSE Linux Enterprise Desktop 10.0
 - SUSE Linux 10.1
 - SUSE Linux 10.2

CUDA and its libraries are compiled with gcc 3.4.5 and the same tools should be used on other distributions. Alternatively, compatibility packages containing libstdc++.so.6 may be used. These are available for many Linux distributions. For example, on recent Fedora Core releases, the compat-libstdc++-34 is needed.

V. Known Issues

Note: Please see the CUDA Toolkit release notes for additional issues.

1. "ld: cannot find -lglut". On some linux installations (notably default RHEL 4 update 3 installations), building the simpleGL example (and other examples that use OpenGL) can result in a linking error like the following.

```
/usr/bin/ld: cannot find -lglut
```

Typically this is because the SDK makefiles look for libglut.so and not for variants of it (like libglut.so.3). To confirm this is the problem, simply run the following command.

```
ls /usr/lib | grep glut
```

You should see the following (or similar) output.

```
lrwxrwxrwx 1 root root      16 Jan  9 14:06 libglut.so.3 -> libglut.so.3.8.0
-rwxr-xr-x 1 root root 164584 Aug 14  2004 libglut.so.3.8.0
```

If you have libglut.so.3 in /usr/lib, simply run the following command as root.

```
ln -s /usr/lib/libglut.so.3 /usr/lib/libglut.so
```

If you do NOT have libglut.so.3 then you can check whether the glut package is installed on your RHEL system with the following command.

```
rpm -qa | grep glut
```

You should see "freeglut-2.2.0-14" or similar in the output. If not, you or your system administrator should install the package "freeglut-2.2.0-14". Refer to the Red Hat and/or rpm documentation for instructions.

If you have libglut.so.3 but you do not have write access to /usr/lib, you can also fix the problem by creating the soft link in a directory to which you have write permissions and then add that directory to the library search path (-L) in the Makefile.

2. In code sample alignedTypes, the following aligned type does not provide maximum throughput because of a compiler bug:

```
typedef struct __align__(16) {
    unsigned int r, g, b;
} RGB32;
```

The workaround is to use the following type instead:

```
typedef struct __align__(16) {
    unsigned int r, g, b, a;
} RGBA32;
```

as illustrated in the sample.

VI. Frequently Asked Questions

 The Official CUDA FAQ is available online on the NVIDIA CUDA Forums:
<http://forums.nvidia.com/index.php?showtopic=36286>

Note: Please also see the CUDA Toolkit release notes for additional Frequently Asked Questions.

 VII. Change Log

Release 1.1

- * Updated to the 1.1 CUDA Toolkit
- * Fixed several bugs in common/common.mk
- * Removed isInteropSupported() from cutil: OpenGL interop now works on multi-GPU systems
- * MonteCarlo sample: Improved performance. Previously it was very fast for large numbers of paths and options, now it is also very fast for small- and medium-sized runs.
- * Transpose sample: updated kernel to use a 2D shared memory array for clarity, and optimized bank conflicts.
- * 11 new code samples:
 eigenvalues, fastWalshTransform, histogram256, Mandelbrot, MonteCarloMultiGPU nbody, oceanFFT, particles, reduction, simpleAtomics, SobelFilter
- * 13 new code samples:
 asyncAPI, cudaOpenMP, eigenvalues, fastWalshTransform, histogram256, Mandelbrot, MonteCarloMultiGPU, nbody, oceanFFT, particles, reduction, simpleAtomics, and simpleStreams

Release 1.0

- * Updated to the 1.0 CUDA Toolkit.
- * Added 4 new code samples: convolutionTexture, convolutionFFT2D, histogram64, and SobelFilter.
- * All graphics interop samples now call the cutil library function isInteropSupported(), which returns false on machines with multiple CUDA GPUs, currently (see above).
- * When compiling in DEBUG mode, CU_SAFE_CALL() now calls cuCtxSynchronize() and CUDA_SAFE_CALL() and CUDA_CHECK_ERROR() now call cudaThreadSynchronize() in order to return meaningful errors. This means that performance might suffer in DEBUG mode.

Release 0.9

- * Updated to the 0.9 CUDA Toolkit.
- * Added 7 new code samples: MersenneTwister, MonteCarlo, imageDenoising, simpleTemplates, deviceQuery, alignedTypes, and convolutionSeparable
- * Removed 3 old code samples: convolution, vectorLoads and loadUByte. Convolution is replaced by convolutionSeparable and vectorLoads and loadUByte are replaced by alignedTypes.

Release 0.8.1 beta

- * Standardized project and file naming conventions. Several project names changed as a result.
- * cppIntegration output now matches the other samples ("Test PASSED").
- * Modified transpose16 sample to transpose arbitrary matrices efficiently, and renamed it to transpose.
- * Added 11 new code samples: bandwidthTest, binomialOptions, BlackScholes, boxFilter, convolution, dxtc, fluidsGL, multiGPU, postProcessGL, simpleTextureDrv, and vectorLoads.
- * Changed object file output directory to reside within each project's directory rather than in the shared bin directory. This prevents linker errors caused by different projects having files with identical names.
- * Added "make clobber" mode to makefiles to support deleting project obj directories.
- * Consolidated scan_naive, scan_workefficient, and scan_best into a single "scan" sample that runs all three scan kernels and compares performance.

Release 0.8 beta prerelease 2

- * Linux CUDA SDK now uses a self extracting installer (.run)
- * Linux CUDA SDK samples now build correctly if CUDA Toolkit is installed to a path other than /usr/local

- * Added matrixmul_drv sample to Linux SDK
- * Added CUBINFILES mode to project Makefiles
- * simpleCUFFT and simpleCUBLAS samples now work in emulation mode (make emu=1).

Release 0.8 beta prerelease 1

- * Second limited release (RHEL 4 update 3 only)
- * Several new samples, better release notes

Release 0.2 alpha 2

- * First limited release (RHEL 3 update 4 only)