NVIDIA CUDA SAMPLES

v5.0 | October 2012

Release Notes

CUDA SAMPLES 5.0 NOTES

R304 Driver

- Windows XP, Windows Vista, Windows 7, Windows 8 (32/64-bit)
- Windows Server 2003, 2003 R2, 2008, 2008 R2
- Linux OS (32/64-bit)
- Mac OS X (10.7.x Lion 32/64-bit, 10.8 Mountain Lion 32/64-bit)

Note: Please also refer to the release notes of Release Version 5.0 of the CUDA Toolkit, installed by the CUDA Toolkit installer.

TABLE OF CONTENTS

WHAT IS NEW WITH CUDA 5.0 TOOLKIT?

- CUDA 5.0 CUDA Samples release includes a number of new SDK samples. Please refer to the change log under Change Log for complete details and descriptions.
- New CUDA Dynamic Parallelism Samples (requires SM 3.5): `cdpSimplePrint`, `cdpSimpleQuicksort`, `cdpAdvancedQuicksort`, `cdpLUDecomposition`, `cdpQuadtree`, `simpleDevLibCUBLAS`, `simpleHyperQ`
- New CUDA Samples: `simpleCallback`, `simpleD3D10RenderTarget`, `simpleSeparateCompilation`, `simpleIPC`, `stereoDisparity`, `shfl_scan`
- Revised CUDA Samples: `deviceQuery`, `histEqualizationNPP`

Chapter 1. INSTALLATION INSTRUCTIONS

1.1 Windows Installation Instructions

CUDA 5.0 Toolkit Installer includes CUDA Toolkit 5.0 and Version R304 Driver (Windows XP, Vista, Win7, or Windows Server 2008 R8), and CUDA Samples.

1. Uninstall any previous versions of the NVIDIA CUDA Toolkit and NVIDIA GPU Computing SDK:

   You can uninstall the NVIDIA CUDA Toolkit through the **Start** menu: **Start menu** > **All Programs** > **NVIDIA Corporation** > **CUDA Toolkit** > **Uninstall CUDA**

   You can uninstall the NVIDIA GPU Computing SDK through the **Start** menu: **Start menu** > **All Programs** > **NVIDIA Corporation** > **NVIDIA GPU Computing SDK** > **Uninstall NVIDIA GPU Computing SDK**
2. Install version Release 5.0 of the NVIDIA CUDA Toolkit by launching:

   ```
   cuda_5.0.xx_[winxp_general|winvista_win7_win8_general|
   winvista_win7_win8_notebook]_[32|64].exe
   ```

   The filename depends on the Windows operating system being used.

   This installs the Toolkit, CUDA Samples, and Driver. Each of these components can be installed optionally in the installation GUI when launched for the first time. If you install the Driver via silent install, only the display driver and CUDA driver will be included. If you need the full NVIDIA driver to be installed, please uncheck **Silent Driver Install**. The full NVIDIA driver installation will happen after the Toolkit and CUDA Samples are installed.
3. Build the 32-bit and/or 64-bit `release` or `debug` configurations of the project examples using the provided:

   `*_vs2008.sln`
   : solution files for Microsoft Visual Studio 2008

   `*_vs2010.sln`
   : solution files for Microsoft Visual Studio 2010

   You can:

- Use the solution files located in each of the example directories in:

```
CUDA Samples\v5.0\<category>
```

- Use the global solution files located under:

```
CUDA Samples\v5.0
    samples_vs2008.sln
    samples_vs2010.sln
```

Notes:

- The `simpleD3D9` example and many others including CUDA DirectX samples require that Microsoft DirectX SDK (June 2010 or newer) is installed and that the VC++ directory paths are properly set up (located in **Tools** > **Options...**).
- Prior to CUDA 5.0, CUDA Sample projects referenced a utility library with header and source files called `cutil`. This has been removed with the CUDA Samples in CUDA 5.0, and replaced with header files found in `CUDA Samples\v5.0\common\inc`: `helper_cuda.h`, `helper_cuda_gl.h`, `helper_cuda_drvapi.h`, `helper_functions.h`, `helper_image.h`, `helper_math.h`, `helper_string.h`, and `helper_timer.h`

  These files provide utility functions for CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to `cutil`, and will now use these helper functions going forward.

4. Run the examples from the `release` or `debug` directories located in:

```
CUDA Samples\v5.0\bin\win[32|64]\[release|debug]
```

Notes:

- The `release` and `debug` configurations require a CUDA-capable GPU to run properly (see *CUDA-Enabled GPUs* in the *CUDA Programming Guide* for a complete list of CUDA-capable GPUs).

1.2 Linux Installation Instructions

Note: The default installation folder `<SAMPLES_INSTALL_PATH>` is `~/NVIDIA_CUDA_Samples`. Also, a read-only copy of the samples can be found in `/usr/local/cuda-5.0/samples`.

- Before installing the combined installer, you must be in a console mode. Exit the GUI of your Linux environment by pressing **Ctrl+Alt+Backspace**.
- For some Linux distributions, you may need to stop GDM via:

```
> sudo /etc/init.d/gdm stop
```

or

```
> /sbin/init 3
```

Note: It is also possible to extract the individual packages for separate installation. Please refer to the *Getting Started Guide for Linux* for more details.

1. Install the CUDA 5.0 Toolkit with one of the following commands:

   - For 32-bit Linux distributions:

     ```
     > sudo sh cuda_5.0.xx_linux_32_[distro].run
     ```

   - For 64-bit Linux distributions:

     ```
     > sudo sh cuda_5.0.xx_linux_64_[distro].run
     ```

   Note: For optimus configurations, you may need to add `--optimus` 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.

   You are prompted for the path where you want to put the CUDA Toolkit (`/usr/local/cuda-5.0` is the default) and CUDA Samples (`~/NVIDIA_CUDA-5.0` is the default). CUDA Samples are treated like user development code (it is a collection of CUDA examples). During installation, the prompt is to accept the default or override it with a specified path to which the user has write permissions.

   After installation, you can find the location of the files here:

   CUDA Toolkit: `/usr/local/cuda-5.0` with a symbolic link `/usr/local/cuda` point to this folder.
   CUDA Samples: `$(HOME)/NVIDIA_CUDA-5.0`

   Note: In addition, a pristine read-only version of the samples can also be found in `/usr/local/cuda-5.0`

2. Set up environment variables for CUDA Development.

   You may want to add this to your `~/.bash_profile`:

   - Add the following to your system `PATH`:

     ```
     export PATH=/usr/local/cuda-5.0/bin:$PATH
     ```

   - Add the following to your `LD_LIBRARY_PATH` (if running on a 32-bit OS)

     ```
     export LD_LIBRARY_PATH=/usr/local/cuda-5.0/lib:$LD_LIBRARY_PATH
     ```

   - Add the following to your `LD_LIBRARY_PATH` (if running on a 64-bit OS)

     ```
     export LD_LIBRARY_PATH=/usr/local/cuda-5.0/lib64:$LD_LIBRARY_PATH
     ```

3. Build the CUDA Samples projects:

   ```
   cd <SAMPLES_INSTALL_PATH>
   make
   ```

   Note: Adding the following in `make` builds for specific targets:

   `make x86_64=1`
   for 64-bit targets

   `make i386=1`
   for 32-bit targets

   `make`
   for the `release` configuration

   `make dbg=1`
   for the `debug` configuration

Note: Prior to CUDA 5.0, CUDA Sample projects referenced a utility library with header and source files called CUTIL. Also many of the `Makefile` projects have been rewritten to be self contained and no longer depend on `common.mk`. CUTIL has been removed with the CUDA Samples in CUDA 5.0, and replaced with helper functions found in `NVIDIA_CUDA-5.0/common/inc`: `helper_cuda.h`, `helper_cuda_gl.h`, `helper_cuda_drvapi.h`, `helper_functions.h`, `helper_image.h`, `helper_math.h`, `helper_string.h`, `helper_timer.h`

These helper functions handle CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to CUTIL, and now use these helper functions going forward.

4. Run the CUDA examples (32-bit or 64-bit Linux):

```
cd <SAMPLES_INSTALL_PATH>/bin/linux/release
matrixmul
```

(or any of the other executables in that directory)

1.3 Mac OS X Installation Instructions

Note: The default installation folder `<SAMPLES_INSTALL_PATH>` is:

```
/Developer/NVIDIA/CUDA-5.0/samples
```

Note: For Snow Leopard (10.6), Lion (10.7), and Mountain Lion (10.8):

To boot up in 32-bit kernel mode, after Power-On (and hearing the boot up sound), hit keys **3** and **2** at the same time immediately after the startup sound. The OS will startup in a 32-bit kernel mode.

To boot up with a 64-bit kernel, during Power-On, hit keys **6** and **4** at the same time.

Please install the packages in this order.

1. Install the NVIDIA CUDA Toolkit Installer Package (Mac OSX Leopard)
   - Do you have a Quadro 4000 for Mac and/or recently updated to the Mac OSX 10.7.x? If so, please first install the release 256 driver for Mac. You can download the package from here:

     http://www.nvidia.com/object/quadro-macosx-256.01.00f03-driver.html
   - For NVIDIA GeForce GPU or Quadro GPUs, install this package:

     ```
     cuda_5.0.xx_macos.pkg
     ```
2. Install version 5.0 Release of the CUDA 5.0 Toolkit installer by executing the file:

   ```
   cuda_5.0.xx_macos.pkg
   ```

   This package will work Mac OS X running 32/64-bit. CUDA applications built in 32/64-bit (CUDA Driver API) are supported. CUDA applications built as 32/64 bit (CUDA Runtime API) are supported. (10.6 SnowLeopard, 10.7 Lion, and 10.8 Mountain Lion)

   You are now able to pick which packages you wish to install

- CUDA Driver is installed to `/Library/Frameworks/CUDA.framework`
- CUDA Toolkit is installed to `/Developer/NVIDIA/CUDA-5.0` (previous toolkit installations will automatically be moved to `/Developer/NVIDIA/CUDA-#.#`)
- CUDA Samples will be installed to `/Developer/NVIDIA/CUDA-5.0/samples`

After installation, you may want to add the following paths to your environment:

```
> export PATH=/Developer/NVIDIA/CUDA-5.0/bin:$PATH
           > export DYLD_LIBRARY_PATH=/Developer/NVIDIA/
CUDA-5.0/lib:$DYLD_LIBRARY_PATH
```

To make these settings permanent, place them in `~/.bash_profile`

3. Build the SDK project examples:

- Go to `<SAMPLES_INSTALL_PATH>` (`cd <SAMPLES_INSTALL_PATH>`)
- Build:

  `make x86_64=1`
    for 64-bit targets

  `make i386=1`
    for 32-bit targets

  `make`
    for the `release` configuration

  `make dbg=1`
    for the `debug` configuration

  Note: Prior to CUDA 5.0, CUDA Sample projects referenced a utility library with header and source files called CUTIL. Also many of the `Makefile` projects have been rewritten to be self contained and no longer depend on `common.mk`. CUTIL has been removed with the CUDA Samples in CUDA 5.0, and replaced with helper functions found in `NVIDIA_CUDA-5.0/common/inc`: `helper_cuda.h`, `helper_cuda_gl.h`, `helper_cuda_drvapi.h`, `helper_functions.h`, `helper_image.h`, `helper_math.h`, `helper_string.h`, `helper_timer.h`

  These helper functions handle CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. The CUDA Samples projects no longer have references and dependencies to CUTIL, and now use these helper functions going forward.

4. Run the CUDA examples:

```
cd <SAMPLES_INSTALL_PATH>/bin/darwin/[release|debug]
./matrixmul
```

(or any of the other executables in that directory)

Chapter 2. CREATING YOUR OWN CUDA PROJECTS

2.1 Creating CUDA Projects for Windows

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

(`<category>` refers to one of the following folders: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, `7_CUDALibraries`.)

1. Copy the content of:

   ```
   CUDA Samples\v5.0\<category>\template
   ```

   or

   ```
   CUDA Samples\v5.0\<category>\template_runtime
   ```

   to a directory of your own:

   ```
   CUDA Samples\v5.0\<category>\myproject
   ```
2. Edit the filenames of the project to suit your needs.
3. Edit the `*.sln`, `*.vcproj` and source files.

   Just search and replace all occurrences of `template` or `template_runtime` with `myproject`.
4. Build the 32-bit and/or 64-bit, release or debug configurations using:

   ```
   myproject_vs2008.sln
   myproject_vs2010.sln
   ```
5. Run `myproject.exe` from the `release` or `debug` directories located in:

   ```
   CUDA Samples\v5.0\bin\win[32|64]\[release|debug]
   ```
6. Now modify the code to perform the computation you require.

   See the *CUDA Programming Guide* for details of programming in CUDA.

2.2 Creating CUDA Projects for Linux

Note: The default installation folder `<SAMPLES_INSTALL_PATH>` is `NVIDIA_CUDA_5.0_Samples` and <category> is one of the following: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, `7_CUDALibraries`.

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

1. Copy the `template` or `template_runtime` project:

   ```
   cd <SAMPLES_INSTALL_PATH>/<category>
   cp -r template <myproject>
   ```

   or (using `template_runtime`):

   ```
   cd <SAMPLES_INSTALL_PATH>/<category>
   cp -r template_runtime <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
   ```

   or (using `template_runtime`):

   ```
   mv main.cu  myproject.cu
   ```
3. Edit the `Makefile` and source files.

   Just search and replace all occurrences of `template` or `template_runtime` with `myproject`.
4. Build the project:

   ```
   make
   ```

   You can build a debug version with:

   ```
   make dbg=1
   ```
5. Run the program:

   ```
   ../../bin/linux/release/myproject
   ```
6. Now modify the code to perform the computation you require.

   See the *CUDA Programming Guide* for details of programming in CUDA.

2.3 Creating CUDA Projects for Mac OS X

Note: The default installation folder `<SAMPLES_INSTALL_PATH>` is: `/Developer/NVIDIA/CUDA-5.0/samples`

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

(<category> is one of the following: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, `7_CUDALibraries`.)

1. Copy the template project:
   ```
   cd <SAMPLES_INSTALL_PATH>/<category>
   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
   ```
5. Run the program:
   ```
   ../../bin/darwin/release/myproject
   ```
   (It should print `PASSED`.)
6. Now modify the code to perform the computation you require.

   See the *CUDA Programming Guide* for details of programming in CUDA.

Chapter 3. KNOWN ISSUES ON CUDA

3.1 Known Issues in CUDA Samples for Windows

Note: Please see the *CUDA Toolkit Release Notes* for additional issues.

- 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.
- By default the CUDA Samples 5.0 will be installed to:

  ```
  ProgramData\NVIDIA Corporation\CUDA Samples\v5.0
  ```

  so it will not have conflicts with Vista with UAC.

  By default, UAC is enabled for Vista. If UAC is disabled, the user is free to install the samples in other folders.

  Before CUDA 2.1, the samples installation path would be under:

  ```
  Program Files\NVIDIA Corporation\NVIDIA CUDA SDK
  ```

  Starting with CUDA 2.1, the new default installation folder was:

  ```
  Application Data\NVIDIA Corporation\NVIDIA CUDA SDK
  ```

  residing under `All Users` or `Current`.

  For NVIDIA GPU Computing 4.2 Release, the installation path was under:

  ```
  ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.2
  ```

  Starting with NVIDIA CUDA Samples 5.0 Release, the new default installation folder is:

  ```
  ProgramData\NVIDIA Corporation\CUDA Samples\v5.0
  ```

residing under `All Users` or `Current`.

- There are number of samples that are not pre-built with the CUDA Samples. Why are these samples not pre-built?

  `cudaOpenMP`, `simpleMPI`, `ExcelCUDA2007`, `ExcelCUDA2010`

  The samples may depend on other header and library packages to be installed on the development machine. These are not distributed with the CUDA Samples, hence these are not pre-built.
- The following Direct3D samples are not officially supported on Telsa GPU:

  `cudaDecodeD3D9`, `fluidsD3D9`, `simpleD3D9`, `simpleD3D9Texture`, `simpleD3D10`, `simpleD3D10Texture`, `simpleD3D11Texture`, `vFlockingD3D10`

  These samples will not run and report that a Direct3D device is not available.

3.2 Known Issues in CUDA Samples for Linux

Note: Please see the *CUDA Toolkit Release Notes* for additional issues.

- The samples that make use of OpenGL fail to build or link. This is because many of the default installations for many Linux distributions do not include the necessary OpenGL, GLUT, GLU, GLEW, X11, Xi, Xlib, or Xmi headers or libraries. Here are some general and specific solutions:
  - Redhat 4 Linux Distributions

    ```
    ld: cannot find -lglut
    ```

    On some Linux installations, building the `simpleGL` example shows the following linking error:

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

    Typically this is because the 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
    ls /usr/lib64 | 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` and/or `/usr/lib64`, simply run the following command as root:

    ```
    ln -s /usr/lib/libglut.so.3 /usr/lib/libglut.so
    ln -s /usr/lib64/libglut.so.3 /usr/lib64/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-2.2.2-14` or similar in the output. If not, you or your system administrator should install the package `freeglut-2.2.2-14`. 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`.

- Some Linux distributions (i.e., Redhat or Fedora) do not include the GLU library. For the latest packages download this file from this website. Please make sure you match the correct Linux distribution.

  http://fr.rpmfind.net/linux/rpm2html/search.php?query=libGLU.so.1&submit=Search+...
- (SLED11) SUSE Linux 11 is missing:

  `libGLU`, `libX11`, `libXi`, `libXm`, `libXmu`

  This particular version of SUSE Linux Enterprise Edition 11 (SLED11) does not have the proper symbolic links for the following libraries:

  - `libGLU`

    ```
    ls /usr/lib | grep GLU
    ls /usr/lib64 | grep GLU
    ```

    ```
    libGLU.so.1
    libGLU.so.1.3.0370300
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libGLU.so.1 /usr/lib/libGLU.so
    ln -s /usr/lib64/libGLU.so.1 /usr/lib64/libGLU.so
    ```
  - `libX11`

    ```
    ls /usr/lib | grep X11
    ls /usr/lib64 | grep X11
    ```

    ```
    libX11.so.6
    libX11.so.6.2.0
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libX11.so.6 /usr/lib/libX11.so
    ln -s /usr/lib64/libX11.so.6 /usr/lib64/libX11.so
    ```
  - `libXi`

    ```
    ls /usr/lib | grep Xi
    ls /usr/lib64 | grep Xi
    ```

    ```
    libXi.so.6
    libXi.so.6.0.0
    ```

    To create the proper symbolic links (32-bit and 64-bit OS):

    ```
    ln -s /usr/lib/libXi.so.6  /usr/lib/libXi.so
    ```

```
ln -s /usr/lib64/libXi.so.6  /usr/lib64/libXi.so
```

- libXm

  ```
  ls /usr/lib | grep Xm
  ls /usr/lib64 | grep Xm
  ```

  ```
  libXm.so.6
  libXm.so.6.0.0
  ```

  To create the proper symbolic links (32-bit and 64-bit OS):

  ```
  ln -s /usr/lib/libXm.so.6  /usr/lib/libXm.so
  ln -s /usr/lib64/libXm.so.6  /usr/lib64/libXm.so
  ```

- libXmu

  ```
  ls /usr/lib | grep Xmu
  ls /usr/lib64 | grep Xmu
  ```

  ```
  libXmu.so.6
  libXmu.so.6.0.0
  ```

  To create the proper symbolic links (32-bit and 64-bit OS):

  ```
  ln -s /usr/lib/libXmu.so.6  /usr/lib/libXmu.so
  ln -s /usr/lib64/libXmu.so.6  /usr/lib64/libXmu.so
  ```

- Ubuntu Linux unable to build these samples that use OpenGL

  The default Ubuntu distribution is missing many libraries.

  - What is missing are the GLUT, Xi, Xmu, GL, and X11 headers. To add these headers and libraries to your distribution, type the following in at the command line:

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

  - Note, by installing Mesa, you may see linking errors against `libGL`. This can be solved below:

    ```
    cd /usr/lib/
    sudo rm libGL.so
    sudo ln -s libGL.so.1 libGL.so
    ```

- 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.

- Unable to build `simpleMPI` sample on Linux Distros

  ```
  simpleMPI.cpp:35:17: error: mpi.h: No such file or directory
  ```

  The Linux system is missing the libraries and headers for MPI.

  - For OpenSuSE or RedHat distributions: Search http://www.rpmfind.net for `openmpi-devel` for your specific distribution

For Ubuntu or Debian distributions, using `apt-get`:

```
sudo apt-get
install build-essential openmpi-bin openmpi-dev
```

- For 32-bit Linux distributions:

```
ln -s /usr/lib/mpi/gcc/openmpi/lib/libmpi_cxx.so.0 /usr/lib/
libmpi_cxx.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libmpi.so.0 /usr/lib/libmpi.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libopen-rte.so.0 /usr/lib/libopen-
rte.so
ln -s /usr/lib/mpi/gcc/openmpi/lib/libopen-pal.so.0 /usr/lib/libopen-
pal.so
```

- For 64-bit Linux distributions:

```
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libmpi_cxx.so.0 /usr/lib64/
libmpi_cxx.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libmpi.so.0 /usr/lib64/libmpi.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libopen-rte.so.0 /usr/lib64/
libopen-rte.so
ln -s /usr/lib64/mpi/gcc/openmpi/lib64/libopen-pal.so.0 /usr/lib64/
libopen-pal.so
```

- Fedora 13 or 14 has linking error when building the following samples:

  `MonteCarloMultiGPU`, `simpleMultiGPU`, `threadMigration`

  The following error is seen:

```
make -C 6_Advanced/threadMigration/
make[1]: Entering directory `/root/sdk/C/6_Advanced/
threadMigration'
/usr/bin/ld: obj/i386/release/threadMigration.cpp.o:
 undefined reference to symbol 'pthread_create@@GLIBC_2.1'
/usr/bin/ld: note: 'pthread_create@@GLIBC_2.1' is defined
 in DSO /lib/libpthread.so.0 so try adding it to the linker
 command line
/lib/libpthread.so.0: could not read symbols: Invalid
 operation
collect2: ld returned 1 exit status
make[1]: *** [../../bin/linux/release/threadMigration] Error
 1
make[1]: Leaving directory `/root/sdk/C/6_Advanced/
threadMigration'
make: *** [6_Advanced/threadMigration/Makefile.ph_build]
 Error 2
```

  For these Linux distributions: Fedora 13 or 14, symbolic links are missing from the following libraries:

  `libpthread`

  To create the proper symbolic links (32-bit OS and 64-bit OS) type this:

```
ln -s /usr/lib/libpthread.so.0 /usr/lib/libpthread.so
ln -s /usr/lib64/libpthread.so.0 /usr/lib64/libpthread.so
```

3.3 Known Issues in CUDA Samples for Mac OS X

Note: In addition, please look at the *CUDA Toolkit 5.0 Release Notes* for additional issues.

- With release CUDA 5.0, support for Mac OS X 10.8.x (Mountain Lion) is added
- With release CUDA 4.0, support for Mac OS X 10.7.x (Lion) is added
- With release CUDA 3.1, Mac OS X now supports CUDA Runtime API (with 64-bit applications)
- CUDA 3.1 Beta and newer now supports 10.6.3 (Snow Leopard) 64-bit Runtime API.
- For CUDA 3.0, Note on CUDA Mac 10.5.x (Leopard) or 10.6.x (Snow Leopard). CUDA applications built with the CUDA driver API can run as either 32-bit or 64-bit applications. CUDA applications using CUDA Runtime APIs can only be built on 32-bit applications.

Chapter 4. FREQUENTLY ASKED QUESTIONS

The Official CUDA FAQ is available online on the NVIDIA CUDA Forums: http://forums.nvidia.com/index.php?showtopic=84440

Note: Please also see the *CUDA Toolkit Release Notes* for additional Frequently Asked Questions.

Chapter 5. ADDENDUM

Important note about the use of *volatile* in warp-synchronous code: The optimized code in the samples `reduction`, `radixSort`, and `scan` uses a technique known as *warp-synchronous* programming, which relies on the fact that within a warp of threads running on a CUDA GPU, all threads execute instructions synchronously. The code uses this to avoid `__syncthreads()` when threads within a warp are sharing data via `__shared__` memory. It is important to note that for this to work correctly without race conditions on all GPUs, the shared memory used in these warp-synchronous expressions must be declared volatile. If it is not declared volatile, then in the absence of `__syncthreads()`, the compiler is free to delay stores to `__shared__` memory and keep the data in registers (an optimization technique), which will result in incorrect execution. So please heed the use of volatile in these samples and use it in the same way in any code you derive from them.

Chapter 6. CHANGE LOG

Note: These Change Logs apply to all OS platforms unless the description states otherwise.

6.1 Release 5.0 (R304 Release Driver)

- Added new CDP samples (CDP = CUDA Dynamic Parallelism)

  `cdpSimplePrint`
  : sample that demonstrates how to use CDP with CUDA printf

  `cdpSimpleQuicksort`
  : sample that demonstrates a simple QuickSort implemented with CDP.

  `cdpAdvancedQuicksort`
  : sample that demonstrates an advanced QuickSort implemented with CDP.

  `cdpLUDecomposition`
  : sample that demonstrates an implementation LU Decomposition with CDP.

  `cdpQuadTree`
  : sample that demonatrates a Quad Tree implementation with CDP.

  `simpleDevLibCUBLAS`
  : sample that demonstrates how to link with the GPU Device CUBLAS library.

6.2 Release 5.0 RC (R304 Release Beta Driver)

- Added new SDK samples

  `bindlessTexture`
  : sample that demonstrates how to use bindless texture (`cudaSurfaceObject`, `cudaTexture`, and MipMap objects)

  `simpleCallback`
  : sample that demonstrates how to use device linking to implement callback functions

  `simpleD3D10RenderTarget`
  : sample that demonstrates D3D10 render target interop

`simpleSeparateCompilation`
: sample that demonstrates how to create a project with static GPU Device libraries and also link to it

`simpleIPC`
: sample that demonstrates Inter Process Communication running CUDA kernels.

`stereoDisparity`
: sample code that makes use of Video SIMD intrinsics to compute the stereo disparity between two images.

`shfl_scan`
: advanced sample that demonstrates a more performant scan that makes use of the `shfl` intrinsic.

- Revisions

  `deviceQuery`
  : this sample has been updated with CUDA Runtime API functions for querying Memory clock, Memory bus width, and L2 Cache Size

  `histEqualizationNPP`
  : `nppiLUT_Linear_8u_C1R` now requires GPU device pointers as input parameters.

- `MonteCarloCURAND` samples folder structure has been flattened (`MonteCarloCURAND` is no long present) and samples underneath have been moved to `7_CUDALibraries` top folder and also renamed.

```
MonteCarloCURAND/EstimatePiInlineP  -> MC_EstimatePiInlineP
MonteCarloCURAND/EstimatePiInlineQ  -> MC_EstimatePiInlineQ
MonteCarloCURAND/EstimateP          -> MC_EstimateP
MonteCarloCURAND/EstimateQ          -> MC_EstimateQ
MonteCarloCURAND/SingleAsianOptionP -> MC_SingleAsianOptionP
```

6.3 Release 5.0 EA (R304 Release Beta Driver)

- Removed all references to CUTIL and SHRUTIL.

  Note: Prior to CUDA 5.0, CUDA Sample projects referenced a CUDA and SHARED utility library that included header and source files. CUDA Samples in CUDA 5.0 no longer depend on these. There is replacement functionality as part of new header files found in `CUDA Samples\v5.0\common\inc`: `helper_cuda.h`, `helper_cuda_gl.h`, `helper_cuda_drvapi.h`, `helper_functions.h`, `helper_image.h`, `helper_math.h`, `helper_string.h`, `helper_timer.h`

  These files provide utility functions for CUDA device initialization, CUDA error checking, string parsing, image file loading and saving, and timing functions. These header files in CUDA Sample projects no longer have references and dependencies to `cutil` or `shrutil` and will now use these helper functions going forward.

- Directories are now restructured according to category (i.e., `0_Simple`, `2_Graphics, 6_Advanced`, ...).
- Samples revised to improve readability and usability for developers.
- All Linux/Mac `Makefiles` are self-contained. `Makefiles` and `common.mk` have been removed.

6.4 Release 4.2 (R295 and R300 Release Driver Update)

All OSs

- Added `segmentationTreeThrust`
- Updated `Makefiles` and `Projects` to enable native support for Kepler GPU architectures (SM 3.0)

6.5 Release 4.1 (R285 Release Driver Update)

All OSs

- Moved `EstimatePiP`, `EstimatePiQ`, `EstimatePiInlineP`, `EstimatePiInlineQ`, and `SingleAsianOptionP` from CUDA C to the CUDALibraries
- `MersenneTwisterGP11213`(computation of `MersenneTwister` using CURAND), added to the `CUDALibraries`
- Removed older CUDA SDK sample `MerseeneTwister` (MT607)
- Moved randomFog from CUDA C src folder to `CUDALibraries`. This sample uses CURAND
- Several of the CUDA C SDK samples restructured so that they are self contained and no longer depend nor link with CUTIL or SHRUTIL libraries

6.6 Release 4.1 RC1 (R285 Beta Driver)

All OSs

- Added `HSOpticalFlow` (Horn-Schunck Method for Optical Flow)
- Added `volumeFiltering` (CUDA 3D Volume Filtering)
- Added `simpleCubemapTexture` (SDK sample demonstrating how to use texcubemap fetch instruction in CUDA C)
- Added `simpleAssert` (SDK Sample demonstrating how to use the GPU assert function) Added `grabcutNPP` (SDK Sample that demonstrates how to use the Graph Cut functions within the NPP Library)

6.7 Release 4.0 Final (R270 Release Driver Update)

All OSs

- `nBody` SDK sample can now support multiple GPUs through CUDA 4.0's API for multi-GPU support.
- Added 2 new documents to the Documentation:

  CUDA_SDK_New_Features_Guide.pdf
  Getting_Started_With_CUDA_SDK_Samples.pdf
- Updated CUDA and CUDALibraries individual SDK sample solution files (`*.sln`) for VS2008, and VS2010. Now the individual solutions include project dependencies with SDK helper libraries. If an SDK sample depends on linking with `shrUtils` or `cutil`, the individual SDK samples can now be built without using `Release_vs20??.sln` to properly build its dependencies.

Windows OS

- Visual Studio 2010 projects for CUDA C, CUDALibraries, and OpenCL are now included with the GPU Computing SDK.

6.8 Release 4.0 RC2 (R270 Release Driver Update)

All OS

- Added `simpleP2P` SDK sample to illustrate how to use P2P and UVA. This requires a Tesla GPU with SM 2.0 (Tesla C2050/C2070). For Windows Vista/Win7, it must be running 64-bit and two or more GPUs must be running the TCC driver. With linux, the standard driver will work on a Tesla GPU with SM 2.0 capabilities.
- CUDA SDK samples that use the Driver API now use the new `cuLaunchKernel` API.

Windows OS

- 49 of the CUDA C SDK samples include VS2010 projects. All of the remaining samples will support Visual Studio 2010

6.9 Release 4.0 RC1 (R270 Release Driver Update)

All OS

- Changes to `lineOfSight`, `marchingCubes`, `radixSort`, `particles`, `smokeParticles` to use Thrust library (now included as part of the CUDA 4.0 Toolkit). The SDK sample `radixSort` has been renamed to `radixSortThrust` with this change to use Thrust. CUDPP libraries, headers, have been removed from the SDK. Thrust headers are now part of the CUDA Toolkit and are used by these SDK samples.
- Updated `MonteCarloMultiGPU` SDK sample to illlustrate the new CUDA 4.0 method for Multi-GPU programming. The sample has the ability to launch kernels on multiple GPUs through a single CPU thread.
- Updated Simple Multi-GPU SDK sample to illustrate how to the new CUDA 4.0 method for CUDA context management and multi-threaded access to CUDA contexts.
- Updated `matrixMul`, `simpleCUBLAS`, `batchCUBLAS` SDK Sample to illustrate how to use the new CUDA 4.0 CUBLAS API interface.
- Added `template_runtime`, simple project that shows how to develop a CUDA project without the use of helper libraries or functions (i.e., `cutil`, `shrUtil`).
- Added `conjugateGradientPrecond` (sample showing how to implement a preconditioned conjugate gradient solver with CUBLAS and CUSPARSE).
- Added `newdelete` (sample that demonstrates GPU device memory `alloc/free` using C++).
- Added NPP SDK samples: `boxFilterNPP`, `freeImageInteropNPP`, `histEqualizationNPP`, `imageSegmentationNPP`
- Updated `simpleSTreams` SDK sample that demonstrates CUDA 4.0 capability of supporting generic pinning of system memory
- Added `simpleLayeredTexture` (sample that demonstrates how to use a new CUDA 4.0 feature to support texture arrays in CUDA).
- Updated `ExcelCUDA2007` and `ExcelCUDA2010` projects so they will build out of the box. `ExcelCUDA2007` includes the necessary header/library files to build. `ExcelCUDA2010` requires the Microsoft Excel 2010 SDK which can be downloaded from the Microsoft developer website.

Windows

- Updated `cudaEncode` (H.264 Encode Sample) to support true 64-bit support of Device Pointers (`CUdevicePtr`)
- Updated `cudaDecodeD3D9` and `cudaDecodeGL` samples to support 64-bit support of Device Pointers (`CUdevicePtr`)

6.10 Release 3.2 Release (R260 Release Driver Update)

All OS

- Added `randomFog`, which uses CURAND to generate random values for `Fog`
- Added `simpleSurfaceWrite`, which demonstrates how to write a CUDA kernel that handles Surface Writes

Windows

- The `cudaDecodeGL` handles multi-GPU configurations and can enumerate which one is a WDDM and TCC device. Support to explicitly specify a GPU for decoding has also been added.
- The `cudaEncode` sample has been wrapped into a `VideoEncoder` class for ease of use. GPU Device Memory Input support has also been added, as well as support for other YUV formats for input (YUY2, UYVY, YV12, NV12, IYUV).
- Fixes `cudaDecodeGL` and `cudaDecodeD3D9` sample so the 64-bit builds work with visualization (using 32-bit builds/device pointers). Added support for better handling of multiple-GPU devices. Developers can now specify which GPU to run the decoding on.

6.11 Release 3.2 Beta (R260 Beta Driver Update)

Windows

- CUDA SDK Visual Studio projects have been revised to use environment variables `$(CUDA_PATH)` to reference the CUDA Toolkit Installation path, includes, and compiler. The CUDA Tookit also installs `NvCudaDriverAPI.rules` and `NvCudaRuntimeApi.rules` to refer to rules files which are versioned: `NvCudaDriverApi.v3.2.rules` and `NvCudaRuntimeApi.v.3.2.rules`. To help with development migration from CUDA 3.1 to CUDA 3.2, the environment paths `CUDA_BIN_PATH`, `CUDA_INC_PATH`, and `CUDA_LIB_PATH` are defined and set by the CUDA 3.2 toolkit.

  Please refer to the *Getting_Started_Windows.pdf* on windows and the CUDA toolkit release notes for more details in the CUDA Toolkit release notes.
- Added `cudaEncode` sample which demonstrates CUDA GPU accelerated video encoding of YUV to H.264 surface
- Added SLI D3D10 Texture sample that demonstrates improved performance for multi-GPU configurations.

- Added Interval Computing sample (illustrates how to use Recursion on Fermi Architecture)
- Added `VFlocking` (not pre-built) sample demonstrating a CUDA simulation of bird flocking behavior.

All OS

- Added `MonteCarloCURAND` samples for estimation of MonteCarlo Simulation with the NIVIDA CURAND libraries

```
MonteCarloCURAND\EstimatePiInlineP
MonteCarloCURAND\EstimatePiInlineQ
MonteCarloCURAND\EstimatePiP
MonteCarloCURAND\EstimatePiQ
MonteCarloCURAND\SingleAsianOptionP
```

- Add `bilateralFiltering` sample
- Add `conjugateGradient` solver on the GPU using CUBLAS and CUSPARSE libraries
- Add `simplePrintf` (shows how to call cuprintf through device code)
- Updated `DeviceQuery/DeviceQueryDrv` samples to support determine if ECC/TCC is enabled for Tesla devices
- Added `FunctionPointers` (version of `SobelFilter`) sample; requires GPU based on Fermi architecture
- Updated `BicubicTexture` sample to include Catmull Rom

6.12 Release 3.1 Final (R256 Driver Update)

- Added support for 64-bit CUDA runtime for Mac OS X 10.6.3
- Added `FunctionPointers` (version of `SobelFilter`) sample; requires GPU based on Fermi architecture
- Updated `BicubicTexture` sample to include Catmull Rom
- Added `excelCUDA` - demonstrates how to create an Excel plugin that can run CUDA kernels

6.13 Release 3.0 Final

- Replaced `3dfd` sample with `FDTD3d` (Finite Difference sample has been updated)
- Added support for Fermi Architecture (Compute 2.0 profile) to the SDK samples
- Updated `Graphics/CUDA` samples to use the new unified graphics interop
- Several samples with Device Emulation have been removed. Device Emulation is deprecated for CUDA 3.0, and will be removed with CUDA 3.1.
- Added new samples:

`concurrentKernels` (Fermi feature)
how to run more than 1 CUDA kernel simultaneously
`simpleMultiCopy` (Fermi feature)

- exercises simulteaneous copy, compute, readback.
- GeForce has 1 copy engine, and Quadro/Tesla have 2 copy engines

`simpleD3D11Texture`
demonstrates Direct3D11 and CUDA interop

- Bug Fixes

6.14 Release 3.0 (R195 Beta 1 Release Update)

- Minor updates to the CUDA SDK samples
- Support for `shrUtils` (shared utilities, useful for logging information)

6.15 Release 3.0 Beta 1

- New Samples
    - `3DFD` - 3D Finite Difference sample demonstrates 3DFD stencil computation on a regular grid.
    - `matrixMulDynlinkJIT`
    - Matrix Multiply sample that uses PTXJIT (inlined) and also supports dynamic linking to `nvcuda.dll`
    - `bitonicSort` has been renamed to `sortingNetworks`
    - `histogram64` and `histogram256` have been combined to a single project histogram
- Removed
    - `scanLargeArray` removed from the SDK, consolidated to scan

6.16 Release 3.0 Beta 1 Beta

- Added PTXJIT
    - New SDK sample that illustrates how to use `cuModuleLoadDataEx`
    - Loads a PTX source file from memory instead of file.
- Windows SDK only
    - Changing the name **NVIDIA CUDA SDK** > **NVIDIA GPU Computing SDK**
    - Added `cudaDecodeD3D9` and `cudaDecodeGL`
        - NVCUVID now support for OpenGL interop, these two new SDK samples illustrates how to use NVCUVID to decode MPEG-2, VC-1, or H.264 content.
        - The samples show how to decode video and pass to D3D9 or OpenGL, however these samples do not display the decoded video frames.

6.17 Release 2.3 (R190 Release Update)

- New SDK Samples
  - `vectorAdd`, `vectorAddDrv` - Two samples (Runtime and Driver API) which implements element by element vector addition. These simple samples do not use the CUTIL library.
- Linux SDK Support for Cross Compilation
  - Adding the following during `Make` will build for specific targets `x86_64=1` (for 64-bit targets) `i386=1` (for 32-bit targets)
  - Note on Linux, the 190.18 or newer driver installs the necessary `libcuda.so` for both 32-bit and 64-bit targets.

6.18 Release 2.2.1

Windows SDK

- Updated `cuda.rules` files (no long uses `-m32`) to generate `CUBIN/PTX` output,
- `cuda.rules` option to generate PTX and to inline CUDA source with PTX generated assembly

Mac and Linux SDK

- Updated `common.mk` file to removed `-m32` when generating CUBIN output
- Support for PTX output has been added to `common.mk`

All SDK Packages

- CUDA Driver API samples: `simpleTextureDrv`, `matrixMulDrv`, and `threadMigration` have been updated to reflect changes:
  - Previously when compiling these CUDA SDK samples, `gcc` would generate a compilation error when building on a 64-bit Linux OS if the 32-bit `glibc` compatibility libraries were not previously installed. This SDK release addresses this problem. The CUDA Driver API samples have been modified and solve this problem by casting device pointers correctly before being passed to CUDA kernels.
  - When setting parameters for CUDA kernel functions, the address offset calculation is now properly aligned so that CUDA code and applications will be compatible on 32-bit and 64-bit Linux platforms.
  - The new CUDA Driver API samples by default build CUDA kernels with the output as PTX instead of CUBIN. The CUDA Driver API samples now use PTXJIT to load the CUDA kernels and launch them.

- Added sample `pitchLinearTexture` that shows how to texture from pitch linear memory

6.19 Release 2.2 Final

Windows and Linux SDK

- Supports CUDA Event Blocking Stream Synchronization (`CU_CTX_BLOCKING_SYNC`) on Linux and Windows

All SDK Packages

- Added Mandelbrot (Julia Set), `deviceQueryDrv`, `radixSort`, `SobolQRNG`, `threadFenceReduction`
- New CUDA 2.2 capabilities:
    - supports zero-memory copy (GT200, MCP79) `* simpleZeroCopy` SDK sample
    - supports OS allocated pinned memory (write combined memory). Test this by:

    ```
    > bandwidthTest -memory=PINNED -wc
    ```

6.20 Release 2.1

Windows SDK

- Projects that depend on `paramGL` now build the `paramGL` source files instead of statically linking with `paramGL*.lib`.

All SDK Packages

- CUDA samples that use OpenGL interop now call `cudaGLSetGLDevice` after the GL context is created. This ensures that `OpenGL/CUDA` interop gets the best possible performance possible.
- Bug fixes

6.21 Release 2.1 Beta

Windows SDK

- Now supports Visual Studio 2008 projects, all samples also include VS2008
- Removed Visual Studio 2003.NET projects
- Added Visual Studio CUDA.rules to support `*.cu files`. Most projects now use this rule with VS2008 projects.

- Default CUDA SDK installation folder is under `All Users` or `Current User` in a sub-folder `Application Data\NVIDIA Corporation\NVIDIA GPU Computing SDK`. See Known Issues in CUDA Samples for Windows for more details.

Mac and Linux SDK

- For CUDA samples that use the Driver API, you must install the Linux 32-bit compatibility (`glibc`) binaries on Linux 64-bit Platforms. See Known Issues in CUDA Samples for Linux on how to do this.

All SDK Packages

- Added CUDA `smokeParticles` (volumetric particle shadows samples)
- Note: added `cutil_inline.h` for CUDA functions as an alternative to using the `cutil.h` macro definitions

6.22 Release 2.0 Beta2 Windows

- 2 new code samples: `cudaVideoDecode` and `simpleVoteIntrinsics`

6.23 Release 2.0 Beta

All SDK Packages

- Updated to the 2.0 CUDA Toolkit
- `CUT_DEVICE_INIT` macro modified to take command line arguments. All samples now support specifying the CUDA device to run on from the command line (`–device=n`).
- `deviceQuery` sample: Updated to query number of multiprocessors and overlap flag.
- `multiGPU` sample: Renamed to `simpleMultiGPU`.
- `reduction`, `MonteCarlo`, and `binomialOptions` samples: updated with optional double precision support for upcoming hardware.
- `simpleAtomics` sample: Renamed to `simpleAtomicIntrinsics`.
- 7 new code samples: `dct8x8`, `quasirandomGenerator`, `recursiveGaussian`, `simpleD3D9Texture`, `simpleTexture3D`, `threadMigration`, and `volumeRender`

Windows SDK

- `simpleD3D` sample: Renamed to `simpleD3D9` and updated to the new Direct3D interoperability API.

- `fluidsD3D` sample: Renamed to `fluidsD3D9` and updated to the new Direct3D interoperability API.

6.24 Release 1.1

- Updated to the 1.1 CUDA Toolkit
- Removed `isInteropSupported()` from `cutil`: graphics interoperability 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 2D shared memory array for clarity, and optimized bank conflicts.
- 15 new code samples: `asyncAPI`, `cudaOpenMP`, `eigenvalues`, `fastWalshTransform`, `histogram256`, `lineOfSight`, `Mandelbrot`, `marchingCubes`, `MonteCarloMultiGPU`, `nbody`, `oceanFFT`, `particles`, `reduction`, `simpleAtomics`, and `simpleStreams`

6.25 Release 1.0

- Added support for CUDA on the MAC
- 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.

6.26 Release 0.9

- Updated to version 0.9 of the CUDA Toolkit.
- Added 6 new code samples: `MersenneTwister`, `MonteCarlo`, `imageDenoising`, `simpleTemplates`, `deviceQuery`, `alignedTypes`, and `convolutionSeparable`.
- Removed 3 old code samples:
  - `vectorLoads` and `loadUByte` replaced by `alignedTypes`;
  - convolution replaced by `convolutionSeparable`

6.27 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`.

6.28 Release 0.8 beta

- First public release.

Chapter 7. OS PLATFORMS AND COMPILERS SUPPORTED

7.1 Windows Platforms Supported

OS Platform Support with CUDA 5.0

- Added support for Windows 8
- Removed support for Visual Studio 2005

OS Platform Support with CUDA 4.2 and 4.1

- No changes

OS Platform Support with CUDA 4.0

- New compilers supported

  Visual Studio 10 (2010)
- Continued supported compilers

  Visual Studio 9 (2008)
- Continued supported OS

  Windows XP, Windows Vista, Windows 7
  Windows Server 2008 and 2008 R2

OS Platform Support added to CUDA 3.0 Release

- Windows 7 32 and 64
- Windows Server 2008 and 2008 R2

OS Platform Support to CUDA 2.2

- Vista 32 and 64bit, WinXP 32 and 64-bit

  Visual Studio 9 (2008)

7.2 Linux Platforms Supported

OS Platform Support with CUDA 5.0

- New OS Platforms added

  Ubuntu 11.10 (gcc 4.6.2, glibc 2.13)
  Fedora16 (gcc 4.6.1, glibc 2.12.90)
  RHEL 5.5+ 64-bit (gcc 4.1.2, glibc 2.5)
  RHEL 6.X (gcc 4.4.5, glibc 2.12)
  OpenSUSE-11.2 (gcc 4.5.1, glibc 2.11.3)
  OpenSUSE-12.1 (gcc 4.6.2, glibc 2.13)
  ICC Compiler 12.1 64-bit
- Platforms no longer supported

  ICC Compiler 11.1 64-bit
  RHEL 5.5+ 32-bit (gcc 4.1.2, glibc 2.5)
  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)
  SLES-11.1 (gcc 4.3.4, glibc 2.11.1)
  Fedora14 (gcc 4.5.1, glibc 2.12.90)
  Ubuntu-11.04 (gcc 4.5.2, glibc 2.13)

OS Platform Support with CUDA 4.2

- New OS Platforms added

  OpenSUSE-11.2 (gcc 4.5.1, glibc 2.11.3)
- Platforms no longer supported

  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)

OS Platform Support with CUDA 4.1

- New OS Platforms added

  Ubuntu 11.04,
  Fedora 14,
  RHEL-5.5, 5.6, 5.7 (32-bit and 64-bit)
  RHEL-6.X (6.0, 6.1) (64-bit only),

ICC Compiler 11.1 (32-bit and 64-bit) Linux

- Continued OS Platforms

  SLES 11.1,
  Ubuntu 10.04,
  OpenSUSE-11.2 (gcc 4.4.1, glibc 2.10.1)

- Platforms no longer supported

  Ubuntu 10.10,
  Fedora 13,
  RHEL-4.8

OS Platform Support with CUDA 4.0

- New OS Platforms added

  SLES11-SP1,
  RHEL-6.0 (64-bit only),
  Ubuntu 10.10

- Continued OS Platforms

  OpenSUSE-11.2
  Fedora 13,
  RHEL-4.8 (64-bit only),
  RHEL-5.5

- Platforms no longer supported

  RHEL-4.8 (32-bit only)
  Ubuntu 10.04,
  SLED11-SP1

OS Platform Support added to CUDA 3.2

- Additional Platform Support Linux 32 and 64:

  Fedora 13,
  Ubuntu 10.04,
  RHEL-5.5,
  SLED-11SP1,
  ICC (64-bit Linux only?)

- Platforms no longer supported

  Fedora 12,
  Ubuntu 9.10
  RHEL-5.4,

SLED11

OS Platform Support added to CUDA 3.1

- Additional Platform Support Linux 32 and 64:

  Fedora 12,
  OpenSUSE-11.2,
  Ubuntu 9.10
  RHEL-5.4
- Platforms no longer supported

  Fedora 10,
  OpenSUSE-11.1,
  Ubuntu 9.04

OS Platform Support added to CUDA 3.0

- Linux Distributions 32 and 64:

  RHEL-4.x (4.8),
  RHEL-5.x (5.3),
  SLED-11
  Fedora10,
  Ubuntu 9.04,
  OpenSUSE 11.1 (gcc 3.4, gcc 4)

7.3 Mac Platforms Supported

OS Platform Support with CUDA 5.0

- Added support for Mac OS X 10.8.x
- Added support for Mac OS X 10.7.4
- Removed support for Mac OS X 10.6.8

OS Platform Support with CUDA 4.2

- Official support for Mac OS X 10.7.3

OS Platform Support with CUDA 4.1

- No changes

OS Platform Support with CUDA 4.0

- New OS Platforms added

  Mac OS X Lion 10.7.x
- Continued OS Platforms

  Mac OS X Snow Leopard 10.6.x
- Platforms no longer supported ?

OS Platform Support added to CUDA 3.2

- Mac OS X Snow Leopard 10.6.4
- Mac OS X Snow Leopard 10.6.5

OS Platform Support added to CUDA 3.1 Beta

- Mac OS X Snow Leopard 10.6.3

  32/64-bit for CUDA Driver API
  32/64-bit for CUDA Runtime API

OS Platform Support added to CUDA 3.0 Release

- Mac OS X Snow Leopard 10.6.x

  32/64-bit for CUDA Driver API
  32-bit for CUDA Runtime API

OS Platform Support added to CUDA 3.0 Beta 1

- Mac OS X Snow Leopard 10.6 (32-bit)

OS Platform Support added to CUDA 2.2

- Mac OS X Leopard 10.5.6+ (32-bit)

  (llvm-)gcc 5.0 Apple