



Getting Started

OpenCL for NVIDIA GPUs

Installation and Verification
on Linux

Table of Contents

Getting Started	i
OpenCL for NVIDIA GPUs	i
Installation and Verification on Linux	i
Table of Contents	iii
Chapter 1. Introduction.....	1
OpenCL Supercomputing with CUDA architecture GPUs	1
System Requirements	2
About This Document	2
Chapter 2. Installing OpenCL	3
Verify You Have a CUDA-Capable System	3
Verify You Have a Supported Version of Linux.....	4
Verify that gcc Is Installed	4
Download the NVIDIA Driver.....	5
Install the NVIDIA Driver	5
Installing the SDK.....	7
Verify the Installation	8
Compiling the Examples.....	8
Running the Binaries	8
Chapter 3. A simple OpenCL Example	11
Source Code Tips	13
What's Next?.....	14



Chapter 1. Introduction

OpenCL Supercomputing with CUDA architecture GPUs

NVIDIA® CUDA™ is a general purpose parallel computing architecture introduced by NVIDIA. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU. To program to the CUDA architecture, developers can choose between C for CUDA, the original and most widely used language, and OpenCL, the new open heterogeneous computing API proposed by Khronos.

The new OpenCL standard 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 OpenCL for CUDA, 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. OpenCL was designed to allow write once run anywhere algorithms. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on both the CPU and GPU without contention for memory resources.

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. Each core has shared resources, including registers and 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, check the correct operation of, and begin developing with, OpenCL for CUDA GPUs.

System Requirements

To use OpenGL on your system, you will need the following installed:

- ❑ NVIDIA GPU with CUDA architecture
- ❑ A supported version of Linux with a gcc compiler and toolchain
- ❑ NVIDIA OpenCL compatible drivers for CUDA GPUs
- ❑ NVIDIA GPU Computing SDK (OpenCL r190 Update Release)

About This Document

This document is intended for readers familiar with the Linux environment and the compilation of OpenCL programs from the command line. You do not need previous experience with OpenCL or experience with parallel computation. Note: This guide covers installation only on systems running X Windows.

Note: Many commands in this document might require superuser privileges. On most distributions of Linux, this will require you to log in as root. For systems that have enabled the sudo package, use the sudo prefix for all necessary commands. We will no longer remark on the matter of user privilege for the installation process except where critical to correct operation.

Chapter 2. Installing OpenCL

The installation of OpenCL on a system running the appropriate version of Linux consists of these simple steps:

- ❑ Verify the system has a CUDA-capable GPU
- ❑ Verify the system has a supported version of Linux.
- ❑ Download and install the NVIDIA driver
- ❑ Download and install the NVIDIA GPU Computing SDK
- ❑ Build the SDK “super-makefile” (Makefile)

Test your installation by compiling and running one or more of the OpenCL sample programs in the SDK to validate that the hardware and software are running correctly and communicating with each other.

Verify You Have a CUDA-Capable System

Many NVIDIA products today contain CUDA-enabled GPUs. These include:

- ❑ NVIDIA GeForce® 8, 9, and 200 series GPUs
- ❑ NVIDIA Tesla™ computing solutions
- ❑ Many of the NVIDIA Quadro® products

An up-to-date list of CUDA-enabled GPUs can be found on the NVIDIA CUDA Web site at http://www.nvidia.com/object/cuda_learn_products.html.

To verify which video adapter your system uses, find the model number by going to your distribution’s equivalent of System Properties, as shown in Figure 1. 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.

Verify You Have a Supported Version of Linux

OpenCL requires an x86-based distribution of Linux. To determine which distribution and release number you're running, type the following at the command line:

```
uname -i && cat /etc/*release
```

You should see output similar to the following, modified for your particular system:

```
i386
Fedora release 8 (Werewolf)
```

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

As of OpenCL for CUDA v1.0, your Linux distribution must be a kernel 2.6 distribution of one of the following versions:

- ☐ Red Hat Enterprise Linux 5.3
- ☐ Ubuntu 8.10
- ☐ Fedora 8

Note: Subsequent updates of OpenCL will support other versions of Linux, so check the download page for the latest supported platforms.

Verify that gcc Is Installed

The `gcc` compiler and toolchain generally are installed as part of the Linux installation, and in most cases the version of `gcc` installed with a supported version of Linux will work correctly. Currently, OpenCL supports `gcc` version 3.4 as well as versions 4.x through 4.2. To verify the version of `gcc` installed on your system, type the following on the command line:

```
gcc --version
```

If an error message appears, you need to install the “development tools” from your Linux distribution or obtain a version of `gcc` and its accompanying toolchain from the Web.

Download the NVIDIA Driver

Once you have verified that you have a supported NVIDIA processor and a supported version of Linux, you need to make sure you have the version of the NVIDIA display driver indicated in the OpenCL Release Notes. This driver **MUST** be installed for the updated SDK and OpenCL applications to operate properly.

Note that drivers recommended with the OpenCL r190 Update Release are compatible with both OpenCL and CUDA 2.3.

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

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

Figure 1 shows the resulting screen. The “NVIDIA Driver Version” is listed in the right top portion of the dialog box under “System Information”, underneath the “Operating System” information

Install the NVIDIA Driver

After you’ve downloaded the NVIDIA driver and software, you will need to install the driver. If you’re in a GUI environment, exit the GUI (ctl-alt-backspace). At the command line, turn off X Windows via **/sbin/init 3**. Then run the driver package from the command line as a superuser. Restart the GUI environment (**startx** or **init 5**, or the equivalent command on your system). In your System Properties (or equivalent), verify that the driver is installed as shown in Figure 1.

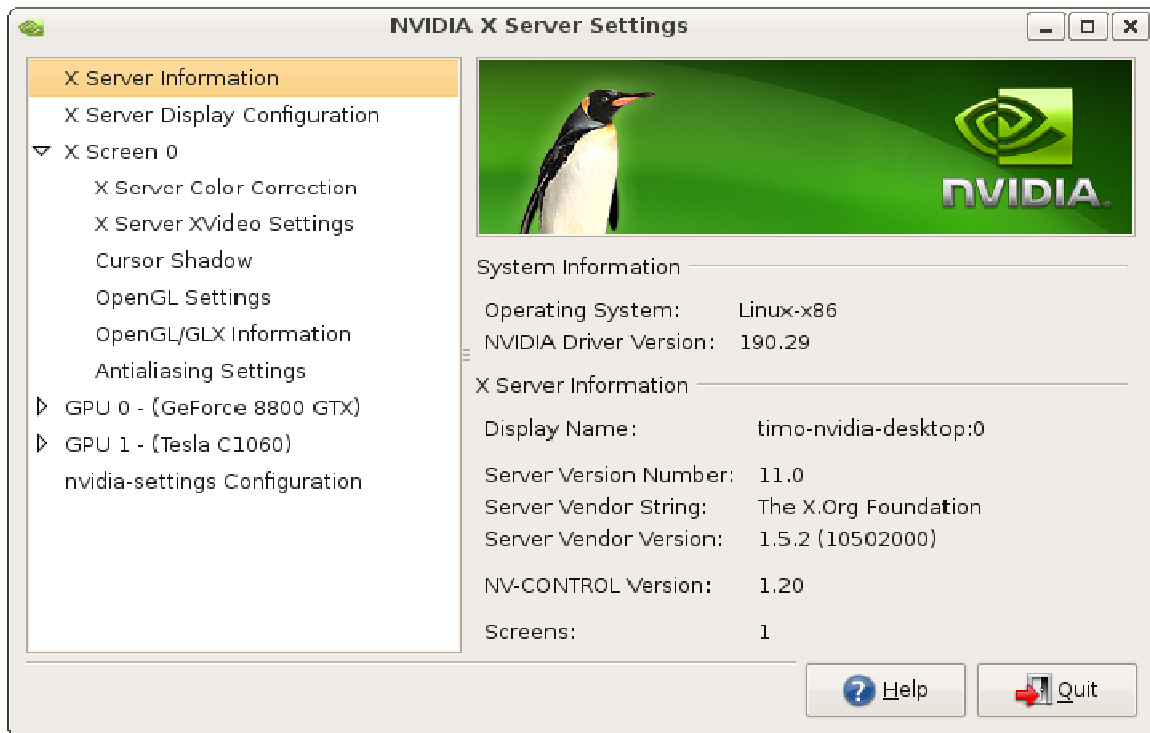


Figure 1. The NVIDIA Driver Information Window

More information on installing the driver is available at:

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

Note: New versions of OpenCL can require later versions of Linux and of the NVIDIA driver, so always verify that you are running the right release for the version of OpenCL you are using.

Installing the SDK

The following section describes the installation and configuration of the NVIDIA GPU Computing SDK, which you downloaded in a previous step.

Before installing the SDK, read the Release Notes, as those notes provide important details on installation and software functionality.

Then, follow these two steps for a successful installation:

Install the SDK by running the installer for your OS in the shell:

```
sh nvcomputesdk_openc11.0-r190-update_linux.run
```

Throughout this document the assumed SDK installation is in the default path:

```
$ (HOME) /NVIDIA_GPU_Computing_SDK/.
```

Best practice for a multiuser Linux system is to also install a version as root that is accessible to users on a read-only basis. This pristine copy can then be copied to a user directory in the event users corrupt their copy of the source code.

Verify the Installation

Before proceeding, it's important to verify that the OpenCL programs can find and communicate correctly with the CUDA-enabled hardware. To do this, you will need to compile and run some of the included sample programs.

Compiling the Examples

NVIDIA includes sample OpenCL programs in source form in the SDK. You should compile them all by changing to **NVIDIA_GPU_Computing_SDK/OpenCL/** in the user's home directory and typing **make**. The resulting binaries will be installed under the home directory in **NVIDIA_GPU_Computing_SDK/OpenCL/bin/linux/release**.

Running the Binaries

After compilation, go to **NVIDIA_GPU_Computing_SDK/OpenCL/bin/linux/release** in the user's home directory and run **oclDeviceQuery**. If OpenCL is installed and configured correctly, the output for **oclDeviceQuery** should look similar to Figure 2.

The exact appearance and the output lines might be different on your system. The important outcomes are that OpenCL v 1.0 was detected (the first highlighted line) at least one OpenCL capable device was found, that matches the one on your system (the second highlighted line), and that the test passed (the final highlighted line).

On systems where SELinux is enabled, you might need to temporarily disable this security feature to run **oclDeviceQuery**. To do this, type:

```
#setenforce 0
```

from the command line as the superuser.

Note: On multiuser systems, access to NVIDIA devices must be enabled for remote users. To do this, enable read-write privileges for all users on **/dev/nv*** devices.

```

oclDeviceQuery
File Edit View Terminal Tabs Help

OpenCL SW Info:
CL_PLATFORM_VERSION:  OpenCL 1.0
OpenCL SDK Version:  $Datetime: 2009/08/16 12:12:38 $  $Change: 4511/61 $

OpenCL Device Info:
# of devices supporting OpenCL - 2:

CL_DEVICE_VENDOR:      NVIDIA Corporation
CL_DEVICE_NAME:         Tesla C1060
CL_DRIVER_VERSION:      190.23
CL_DEVICE_TYPE:         CL_DEVICE_TYPE_GPU
CL_DEVICE_MAX_COMPUTE_UNITS: 30
CL_DEVICE_MAX_WORK_ITEM_SIZES: 512 / 512 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE: 512
CL_DEVICE_MAX_CLOCK_FREQUENCY: 1296 Mhz
CL_DEVICE_IMAGE_SUPPORT: 1
CL_DEVICE_GLOBAL_MEM_SIZE: 4095 MByte
CL_DEVICE_LOCAL_MEM_SIZE: 16 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64 KByte
CL_DEVICE_QUEUE_PROPERTIES: CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES: CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_EXTENSIONS:
    cl_khr_byte_addressable_store
    cl_nv_compiler_options

    cl_khr_local_int32_base_atomics
    cl_khr_global_int32_base_atomics
    cl_khr_global_int32_extended_atomics
    cl_khr_local_int32_extended_atomics

CL_DEVICE_VENDOR:      NVIDIA Corporation
CL_DEVICE_NAME:         GeForce 8800 GTX
CL_DRIVER_VERSION:      190.23
CL_DEVICE_TYPE:         CL_DEVICE_TYPE_GPU
CL_DEVICE_MAX_COMPUTE_UNITS: 16
CL_DEVICE_MAX_WORK_ITEM_SIZES: 512 / 512 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE: 512
CL_DEVICE_MAX_CLOCK_FREQUENCY: 1350 Mhz
CL_DEVICE_IMAGE_SUPPORT: 1
CL_DEVICE_GLOBAL_MEM_SIZE: 767 MByte
CL_DEVICE_LOCAL_MEM_SIZE: 16 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64 KByte
CL_DEVICE_QUEUE_PROPERTIES: CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES: CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_EXTENSIONS:
    cl_khr_byte_addressable_store
    cl_nv_compiler_options

System Info:
Local Time/Date = 12:05:12, 08/18/2009
CPU Name: Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz
# of CPU processors: 2
Linux version 2.6.27-9-generic (build@rothera) (gcc version 4.3.2 (Ubuntu 4.3.2-1ubuntu1)) #1 SMP Thu Nov 20 21:57:00 UTC 2008

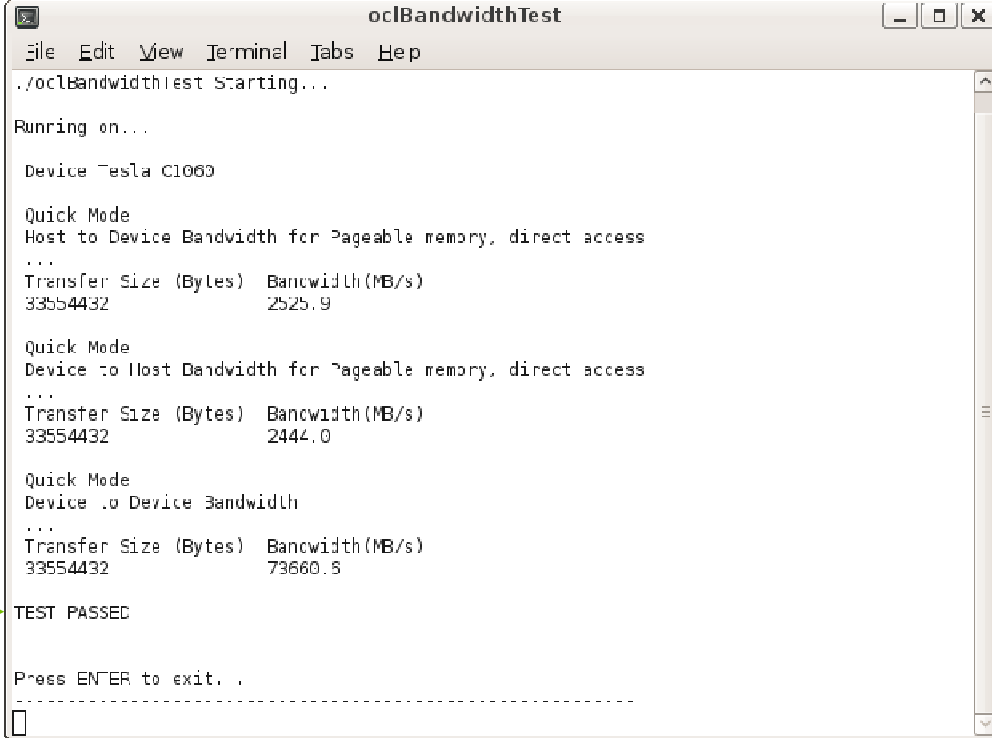
TEST PASSED...

Press ENTER to exit...
-----

```

Figure 2. Valid Results from Sample oclDeviceQuery Program

Running the **oclBandwidthTest** program ensures that the system and the OpenCL device are able to communicate correctly. Its output is shown in Figure 3.



```
oclBandwidthTest
File Edit View Terminal Tabs Help
./oclBandwidthTest Starting...
Running on...
Device Tesla C1060
Quick Mode
Host to Device Bandwidth for Pageable memory, direct access
...
Transfer Size (Bytes)  Bandwidth(MB/s)
33554432                2525.9
Quick Mode
Device to Host Bandwidth for Pageable memory, direct access
...
Transfer Size (Bytes)  Bandwidth(MB/s)
33554432                2444.0
Quick Mode
Device to Device Bandwidth
...
Transfer Size (Bytes)  Bandwidth(MB/s)
33554432                73660.8
TEST PASSED
Press ENTER to exit. .
-----
□
```

Figure 3. Valid Results from Sample **oclBandwidthTest** Program

Note that the measurements for your OpenCL device description will vary from system to system. The important point is that you obtain measurements, and that the second-to-last line (highlighted) confirms that all necessary tests passed.

Should the tests not pass, make sure you have an NVIDIA GPU on your system that supports CUDA 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 SDK directory.

Chapter 3.

A simple OpenCL Example

To build the sample application, save the following source code as **vectoradd.cpp**. Then in the same directory type:

```
> g++ -I ~/NVIDIA_GPU_Computing_SDK/OpenCL/common/inc/ -lOpenCL -D
vectoradd.cpp -o vectoradd
> ./vectoradd
```

```
/* *****
// Demo OpenCL application to compute a simple vector addition
// computation between 2 arrays on the GPU
// *****
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>

// OpenCL source code
const char* OpenCLSource[] = {
    "__kernel void VectorAdd(__global int* c, __global int* a,__global int* b)",
    "{",
    "    // Index of the elements to add \n",
    "    unsigned int n = get_global_id(0);",
    "    // Sum the n'th element of vectors a and b and store in c \n",
    "    c[n] = a[n] + b[n];",
    "}"
};

// Some interesting data for the vectors
int InitialData1[20] = {37,50,54,50,56,0,43,43,74,71,32,36,16,43,56,100,50,25,15,17};
int InitialData2[20] = {35,51,54,58,55,32,36,69,27,39,35,40,16,44,55,14,58,75,18,15};

// Number of elements in the vectors to be added
#define SIZE 2048

// Main function
// *****
int main(int argc, char **argv)
{
    // Two integer source vectors in Host memory
    int HostVector1[SIZE], HostVector2[SIZE];

    // Initialize with some interesting repeating data
    for(int c = 0; c < SIZE; c++)
    {
        HostVector1[c] = InitialData1[c%20];
        HostVector2[c] = InitialData2[c%20];
    }

    // Create a context to run OpenCL on our CUDA-enabled NVIDIA GPU
    cl_context GPUContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU,
                                                    NULL, NULL, NULL);
```

```

// Get the list of GPU devices associated with this context
size_t ParmDataBytes;
clGetContextInfo(GPUContext, CL_CONTEXT_DEVICES, 0, NULL, &ParmDataBytes);
cl_device_id* GPUDevices = (cl_device_id*)malloc(ParmDataBytes);
clGetContextInfo(GPUContext, CL_CONTEXT_DEVICES, ParmDataBytes, GPUDevices, NULL);
// Create a command-queue on the first GPU device
cl_command_queue GPUCommandQueue = clCreateCommandQueue(GPUContext,
                                                         GPUDevices[0], 0, NULL);

// Allocate GPU memory for source vectors AND initialize from CPU memory
cl_mem GPUVector1 = clCreateBuffer(GPUContext, CL_MEM_READ_ONLY |
                                     CL_MEM_COPY_HOST_PTR, sizeof(int) * SIZE, HostVector1, NULL);
cl_mem GPUVector2 = clCreateBuffer(GPUContext, CL_MEM_READ_ONLY |
                                     CL_MEM_COPY_HOST_PTR, sizeof(int) * SIZE, HostVector2, NULL);

// Allocate output memory on GPU
cl_mem GPUOutputVector = clCreateBuffer(GPUContext, CL_MEM_WRITE_ONLY,
                                         sizeof(int) * SIZE, NULL, NULL);

// Create OpenCL program with source code
cl_program OpenCLProgram = clCreateProgramWithSource(GPUContext, 7,
                                                     OpenCLSource, NULL, NULL);

// Build the program (OpenCL JIT compilation)
clBuildProgram(OpenCLProgram, 0, NULL, NULL, NULL, NULL);

// Create a handle to the compiled OpenCL function (Kernel)
cl_kernel OpenCLVectorAdd = clCreateKernel(OpenCLProgram, "VectorAdd", NULL);

// In the next step we associate the GPU memory with the Kernel arguments
clSetKernelArg(OpenCLVectorAdd, 0, sizeof(cl_mem), (void*)&GPUOutputVector);
clSetKernelArg(OpenCLVectorAdd, 1, sizeof(cl_mem), (void*)&GPUVector1);
clSetKernelArg(OpenCLVectorAdd, 2, sizeof(cl_mem), (void*)&GPUVector2);

// Launch the Kernel on the GPU
size_t WorkSize[1] = {SIZE}; // one dimensional Range
clEnqueueNDRangeKernel(GPUCommandQueue, OpenCLVectorAdd, 1, NULL,
                      WorkSize, NULL, 0, NULL, NULL);

// Copy the output in GPU memory back to CPU memory
int HostOutputVector[SIZE];
clEnqueueReadBuffer(GPUCommandQueue, GPUOutputVector, CL_TRUE, 0,
                   SIZE * sizeof(int), HostOutputVector, 0, NULL, NULL);

// Cleanup
free(GPUDevices);
clReleaseKernel(OpenCLVectorAdd);
clReleaseProgram(OpenCLProgram);
clReleaseCommandQueue(GPUCommandQueue);
clReleaseContext(GPUContext);
clReleaseMemObject(GPUVector1);
clReleaseMemObject(GPUVector2);
clReleaseMemObject(GPUOutputVector);

// Print out the results
for (int Rows = 0; Rows < (SIZE/20); Rows++, printf("\n")){
    for(int c = 0; c <20; c++){
        printf("%c", (char)HostOutputVector[Rows * 20 + c]);
    }
}
return 0;
}

```

Source Code Tips

- ❑ The source code presented in the OpenCL SDK is intended to facilitate learning the OpenCL API as applicable to the heterogeneous, massively parallel SPMD programming model appropriate to NVIDIA GPU's.
- ❑ Some application samples in the SDK are very simple and focused upon a particular functional aspect of the OpenCL API. Such examples are an excellent starting point for developers just beginning OpenCL programming and/or GPU programming and include: **oclDeviceQuery**, **oclBandwidthTest**, **oclVectorAdd** and **oclDotProduct**.
- ❑ Some application samples in the SDK are more complex, incorporating graphics output, user input and a wider variety of OpenCL and GPU capabilities. These samples, such as **oclBoxFilter**, **oclRecursiveGaussian**, **oclVolumeRender** and **oclNbody** are more representative of full applications, but beginners may want to skip them until the simpler examples have been fully understood.
- ❑ The source code presented in the OpenCL SDK is intended to facilitate learning the OpenCL API as applicable to the heterogeneous, massively parallel SPMD programming model appropriate to NVIDIA GPU's.
- ❑ Some application samples in the SDK are very simple and focused upon a particular functional aspect of the OpenCL API. Such examples are an excellent starting point for developers just beginning OpenCL programming and/or GPU programming and include: **oclDeviceQuery**, **oclBandwidthTest**, **oclVectorAdd** and **oclDotProduct**.
- ❑ Some application samples in the SDK are more complex, incorporating graphics output, user input and a wider variety of OpenCL and GPU capabilities. These samples, such as **oclBoxFilter**, **oclMedianFilter**, **oclVolumeRender**, **oclParticles** and **oclNbody** are more representative of full applications, but beginners may want to skip them until the simpler examples have been fully understood.
- ❑ An effort has been made to present useful and relevant application samples from a variety of domains. But the source code presented has been tailored for approachability and is generally *not* intended to represent the best production code techniques per se.
 - ❑ For the sake of clarity and emphasis of the most unique aspects of OpenCL and GPU programming, code that might be important for production use has been abbreviated or omitted in some places. For example, teardown/cleanup and error handling code has been intentionally de-emphasized in some portions of the SDK.
 - ❑ Many of the SDK applications present status and timing information useful for a overall perspective of program structure, flow, and a basic awareness of the time required for execution of significant functions. SDK examples have generally been simplified for instructional purposes, however, and are generally *not* highly optimized, except where clearly marked. Advanced optimization techniques are beyond the scope of this SDK. Any timing information presented by the samples is *not* intended for such usage as standardized benchmarking.
- ❑ For convenience, most of the applications additionally log all the console information to a log file in the same directory as the .exe and named after the name of the sample application. For example, the log file for **oclVectorAdd.exe** is **oclVectorAdd.txt**.

- ❑ The oclUtils library (oclUtils.cpp, oclUtils.h, oclUtils.a) contains simple helper functions and a common header used throughout the OpenCL SDK.
 - ❑ These utility functions are host/CPU based C++ code that are mostly specific to the OpenCL API, but are not part of the OpenCL API.
 - ❑ This utility library is used in the OpenCL SDK for convenience, but is not needed for developers to write their own OpenCL applications for NVIDIA GPU's.
- ❑ The shrUtils library (shrUtils.cpp, shrUtils.h and shrUtils.a and other platform & configuration specific binaries) contains simple helper functions and a common header used in many places throughout the NVIDIA GPU Computing SDK (including some samples in the OpenCL SDK).
 - ❑ These utility functions consist of host/CPU based C++ code generic to all of the NVIDIA GPU Computing SDK API's.
 - ❑ This utility library is used in the NVIDIA GPU Computing SDK for convenience, but is not needed for developers to write their own applications for NVIDIA GPU's.

What's Next?

Now that you have CUDA-capable hardware and the OpenCL software installed, you can examine and enjoy the numerous included programs. To begin using OpenCL to accelerate the performance of your own applications, consult the *OpenCL Programming Guide* and the *OpenCL API Specifications*, located in **/NVIDIA_GPU_Computing_SDK/OpenCL/doc**.

For tech support on programming questions, consult and participate in the bulletin board at <http://forums.nvidia.com/index.php?showforum=134>.

Notice

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

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

Trademarks

NVIDIA, the NVIDIA logo, CUDA, GeForce, NVIDIA Quadro, and Tesla are trademarks or registered trademarks of NVIDIA Corporation. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2009 NVIDIA Corporation. All rights reserved.



NVIDIA.