**Getting Started**

# OpenCL for NVIDIA GPUs

## Installation and Verification on Windows

February 2, 2010

# Table of Contents

# 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 the OpenCL portions of the NVIDIA GPU Computing SDK on your system, you will need the following:

- NVIDIA GPU with CUDA architecture
- A supported version of Microsoft Windows and Microsoft Visual Studio
- NVIDIA OpenCL compatible drivers for CUDA GPUs
- NVIDIA GPU Computing SDK

## About This Document

This document is intended as a basic start up guide for readers familiar with the Windows environment and compilation of C/C++ programs from Visual Studio. You do not need previous experience with OpenCL parallel computation.

**Note:** Some operations in this document might require Administrator privileges. 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 Windows consists of these simple steps:

- ❑ Verify the system has a NVIDIA GPU with CUDA architecture
- ❑ Verify the system has a supported version of the Microsoft Windows OS.
- ❑ Verify the system has a supported version of the Microsoft Visual Studio.
- ❑ Install the NVIDIA GPU driver with the executable driver installer.
- ❑ Install the NVIDIA GPU Computing SDK with the executable SDK installer.
- ❑ Review OpenCL application samples as desired in the OpenCL tab of the SDK Browser
- ❑ Build the "super-solution" (oclRelease.sln or oclRelease_vc9.sln)

## Verify You Have a CUDA-Capable GPU

Most NVIDIA desktop, workstation and notebook 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

## Verify that you have a supported Microsoft Windows OS

The NVIDIA GPU Computing SDK (OpenCL r190 Update Release) for Windows requires a 32 bit or 64 bit installation of the Microsoft Windows XP, Windows Vista or Windows 7 operating system.

**Note:** Check the download page for information on the latest supported platforms.

## Verify that you have a supported installation of Microsoft Visual Studio

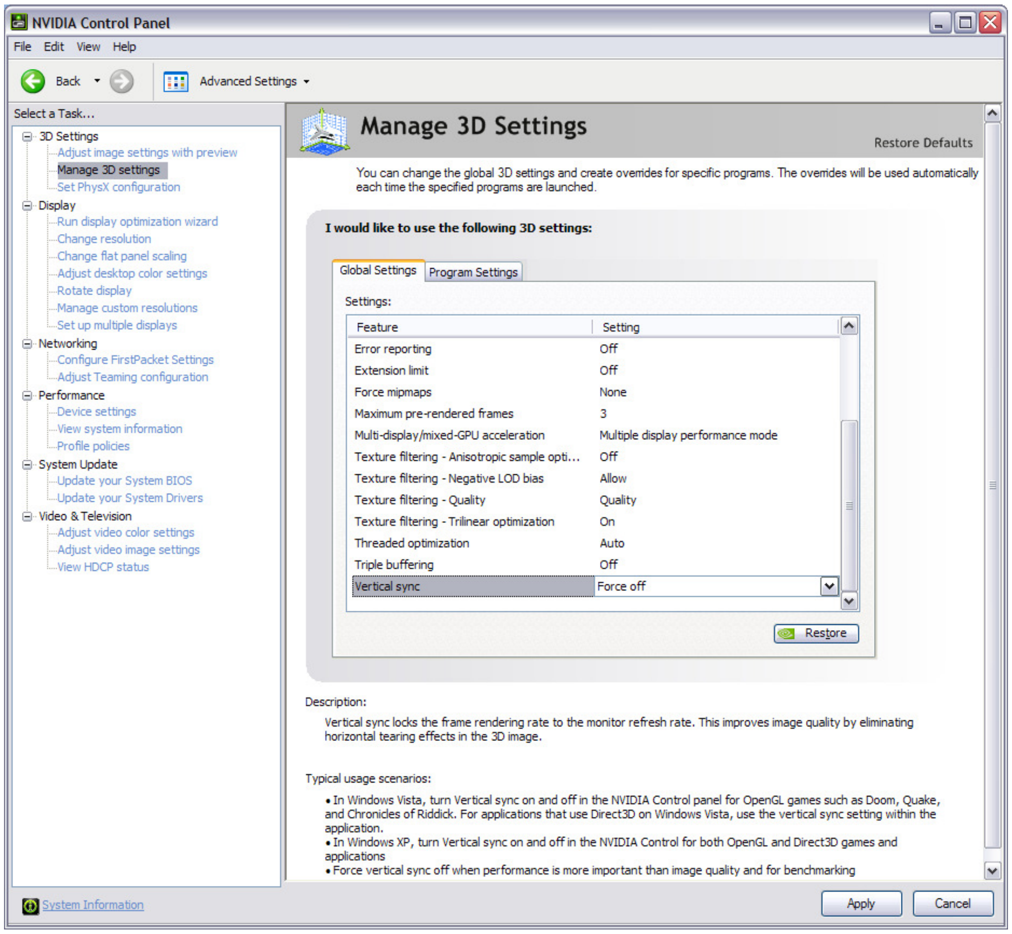
Building the OpenCL samples in the SDK on Windows requires Microsoft Visual Studio 8 (2005) or Visual Studio 9 (2008).

## Install the NVIDIA GPU Driver for Microsoft Windows

- For Windows, the NVIDIA GPU Computing SDK requires that your system run NVIDIA display driver version indicated in the OpenCL Release Notes. This driver MUST be installed for the updated SDK and OpenCL applications to operate properly.

## GPU Driver Settings

- If you have a multiple-GPU configuration, including -GX2 cards, for best OpenCL performance you should disable SLI.
- Alternatively some newer versions of the Forceware drivers offer the option to select secondary GPU as a dedicated GPU for NVIDIA PhysX – which also enables it for use for applications using OpenCL.
- To achieve highest frame rates for SDK samples with OpenGL windows, you can disable the vertical sync option for using the NVIDIA Control Panel as shown here:



Manage 3D Settings

You can change the global 3D settings and create overrides for specific programs. The overrides will be used automatically each time the specified programs are launched.

I would like to use the following 3D settings:

Settings:

| Feature | Setting |
|---|---|
| Error reporting | Off |
| Extension limit | Off |
| Force mipmaps | None |
| Maximum pre-rendered frames | 3 |
| Multi-display/mixed-GPU acceleration | Multiple display performance mode |
| Texture filtering - Anisotropic sample opti... | Off |
| Texture filtering - Negative LOD bias | Allow |
| Texture filtering - Quality | Quality |
| Texture filtering - Trilinear optimization | On |
| Threaded optimization | Auto |
| Triple buffering | Off |
| Vertical sync | Force off |

Description:

Vertical sync locks the frame rendering rate to the monitor refresh rate. This improves image quality by eliminating horizontal tearing effects in the 3D image.

Typical usage scenarios:

- In Windows Vista, turn Vertical sync on and off in the NVIDIA Control panel for OpenGL games such as Doom, Quake, and Chronicles of Riddick. For applications that use Direct3D on Windows Vista, use the vertical sync setting within the application.
- In Windows XP, turn Vertical sync on and off in the NVIDIA Control for both OpenGL and Direct3D games and applications
- Force vertical sync off when performance is more important than image quality and for benchmarking

## Installing the SDK

- To run OpenCL programs for NVIDIA CUDA GPUs, install the NVIDIA GPU Computing SDK. This contains sample OpenCL applications projects illustrating various capabilities of the OpenCL API for use with NVIDIA GPU's.

- The SDK installs pre-built executables and a browser for previewing them, and also includes all of the source code, project and solution files needed to build the sample applications using Microsoft Visual Studio.

- The NVIDIA GPU Computing SDK is available for 32 or 64 bit Windows XP, Vista and Win7. Before installing the SDK, read the Release Notes for details on installation and software functionality, known issues, etc.

- Obtain the NVIDIA GPU Computing SDK executable installer file for your Windows operating system, run it and answer the prompts as appropriate.

- The default installation folder for the OpenCL SDK is:

  **Windows XP**

  **C:\Documents and Settings\All Users\Application Data\NVIDIA Corporation\NVIDIA GPU Computing SDK\OpenCL**

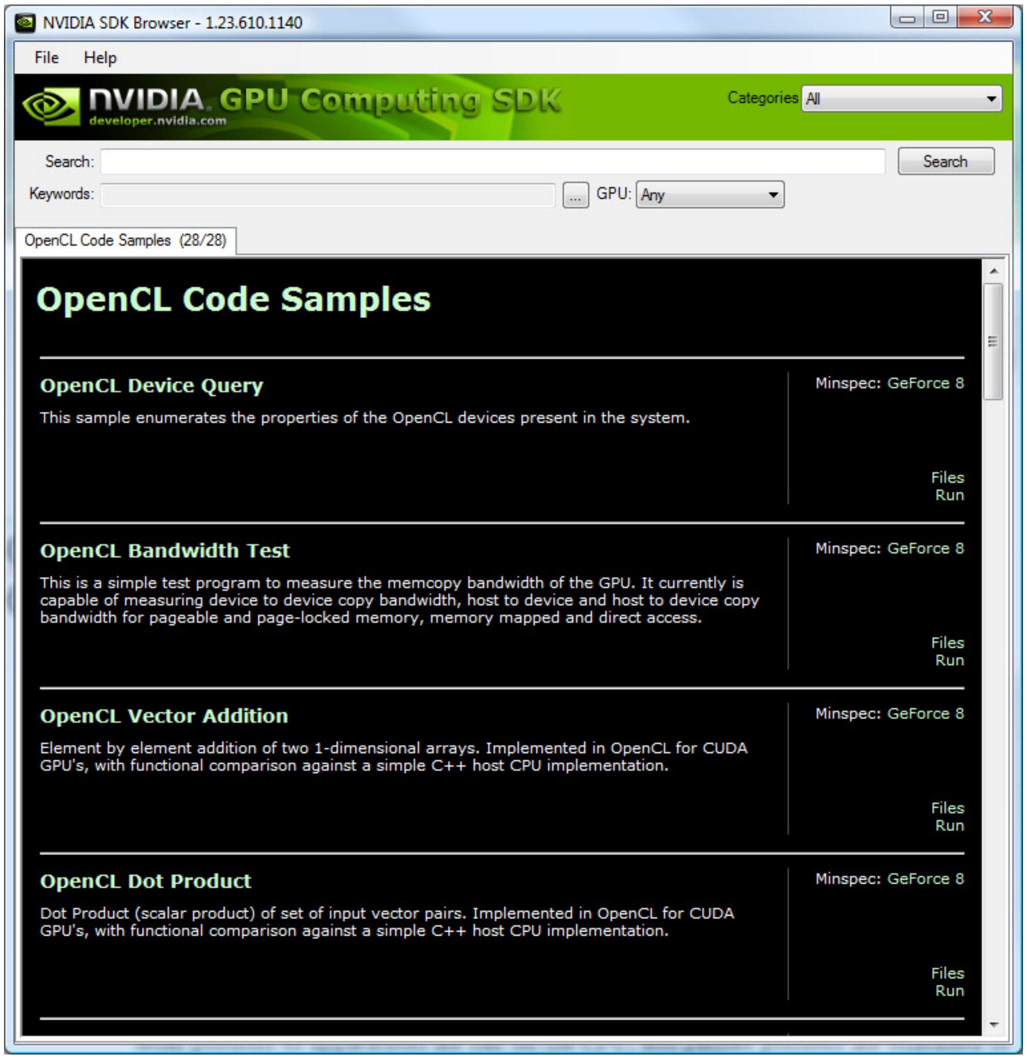
  **Windows Vista / Windows 7**

  **C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK\OpenCL**

**Note:** The "Application Data" and "ProgramData" folders are hidden by default on many Windows installations, but they can be made visible if desired by changing the settings in "Folder Options" in the "Tools" menu in the Windows File Explorer.

## Previewing OpenCL Sample Programs

- After installing the SDK and rebooting, open the SDK Browser by clicking the "NVIDIA GPU Computing SDK Browser" link in the NVIDIA GPU Computing folder within the NVIDIA Corporation program group installed in the Windows Start Menu. Select the "OpenCL Code samples tab".
- As can be seen in figure 1, a description of each installed SDK sample program is featured, along with links for running the executable and viewing the source code files. Some of the samples additionally present a link to a Whitepaper describing the sample in detail.
- The samples are presented within the SDK browser in approximate order of complexity, from the least complex projects at the top to the most complex projects at the bottom.



OpenCL Code Samples

OpenCL Device Query
This sample enumerates the properties of the OpenCL devices present in the system.
Minspec: GeForce 8
Files
Run

OpenCL Bandwidth Test
This is a simple test program to measure the memcopy bandwidth of the GPU. It currently is capable of measuring device to device copy bandwidth, host to device and host to device copy bandwidth for pageable and page-locked memory, memory mapped and direct access.
Minspec: GeForce 8
Files
Run

OpenCL Vector Addition
Element by element addition of two 1-dimensional arrays. Implemented in OpenCL for CUDA GPU's, with functional comparison against a simple C++ host CPU implementation.
Minspec: GeForce 8
Files
Run

OpenCL Dot Product
Dot Product (scalar product) of set of input vector pairs. Implemented in OpenCL for CUDA GPU's, with functional comparison against a simple C++ host CPU implementation.
Minspec: GeForce 8
Files
Run

Figure 1. OpenCL tab of NVIDIA GPU Computing SDK Browser

## Building and Testing SDK Sample Source Code

- After successfully installing the SDK installer, open oclRelease.sln (or oclRelease_vc9.sln) in Visual Studio 2005 or 2008, respectively, and rebuild all the projects presented for the debug and release configuration for the Win32 or x64 platform (as appropriate for the platform of the current installation).

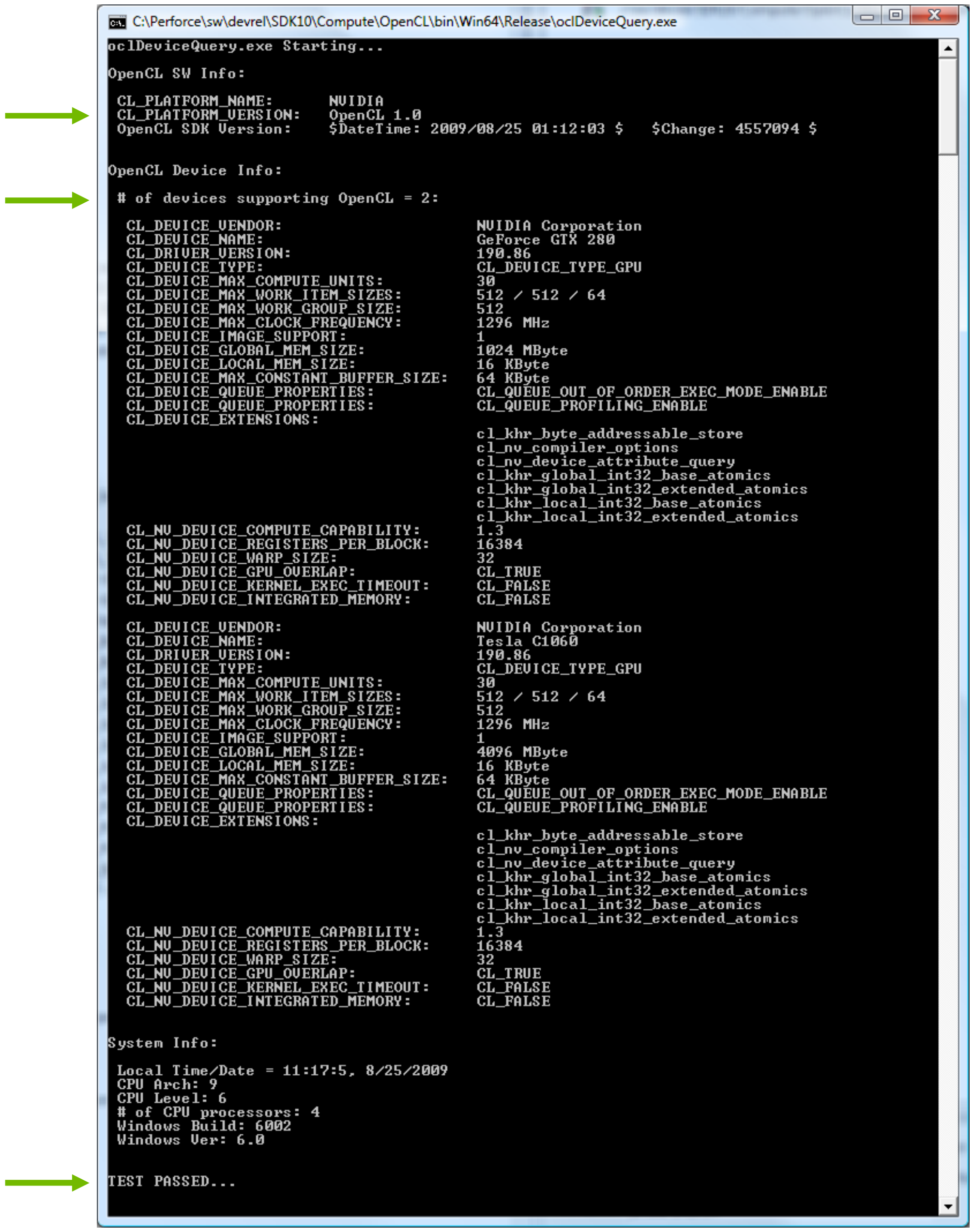
  This will build and/or properly place all library dependencies needed for the current platform, in addition to building all of the SDK samples into executables using the relevant platform-specific dependencies. These executables and other necessary files will be placed within:

  **\NVIDIA GPU Computing SDK\OpenCL\bin\<platform>\<configuration>**

- Each OpenCL sample application executable, such as **oclVectorAdd.exe** may be individually launched from the Windows file explorer by simply double-clicking on the file. Alternately, they may be launched from the command line by opening a console, changing directories to the one containing.exe desired, entering the name of the .exe and pressing enter.
  - OpenCL SDK sample console applications will run until the prompt "Press Enter to exit"
  - OpenCL SDK sample applications that present an OpenGL graphics window will run indefinitely until the operator presses one of: the <Enter>, <Escape>, or "Q" keys
  - Command line options may additionally apply to each application (these are best determined by reviewing the source code).
  - Each OpenCL SDK application creates a session log file storing the same information as the console. These files are found in the same directory as the .exe and named the same as the .exe, except with a .txt extension. The session log file for **oclVectorAdd**, for example, is named "oclVectorAdd.txt".
- A post-build event from build of oclRelease.sln (or oclRelease_vc9.sln) will additionally place the convenient batch file **oclSDK.bat** within the same directory as the sample program executables noted above.
  - **oclSDK.bat** runs ***all*** of the OpenCL SDK sample executables built by oclRelease.sln (or oclRelease_vc9.lsn) in succession.
  - Execution of each sample pauses only briefly before the next application starts, but the session log files generated by each application (as noted above) may be viewed after completion of all the samples for the user's convenience.
  - For convenience, **oclSDK.bat** additionally creates the file oclSDK.txt, which is a comprehensive session log containing the console output from all samples run by oclSDK.bat. This file is equivalent to all the individual sample session logs concatenated together in the order the samples are executed.
- To verify correct hardware and software configuration, reviewing the results from execution of the oclDeviceQuery program is recommended. If this program (or the "super-solution" for the OpenCL SDK, oclRelease.sln or oclRelase_vc9.sln as noted previously) has been built successfully, oclDeviceQuery.exe can be found in:

  **\NVIDIA GPU Computing SDK\OpenCL\bin\<platform>\<configuration>**

- If the SDK is installed and configured correctly, the output from execution of oclDeviceQuery should be a console window that looks similar to Figure 2.



C:\Perforce\sw\devrel\SDK10\Compute\OpenCL\bin\Win64\Release\oclDeviceQuery.exe

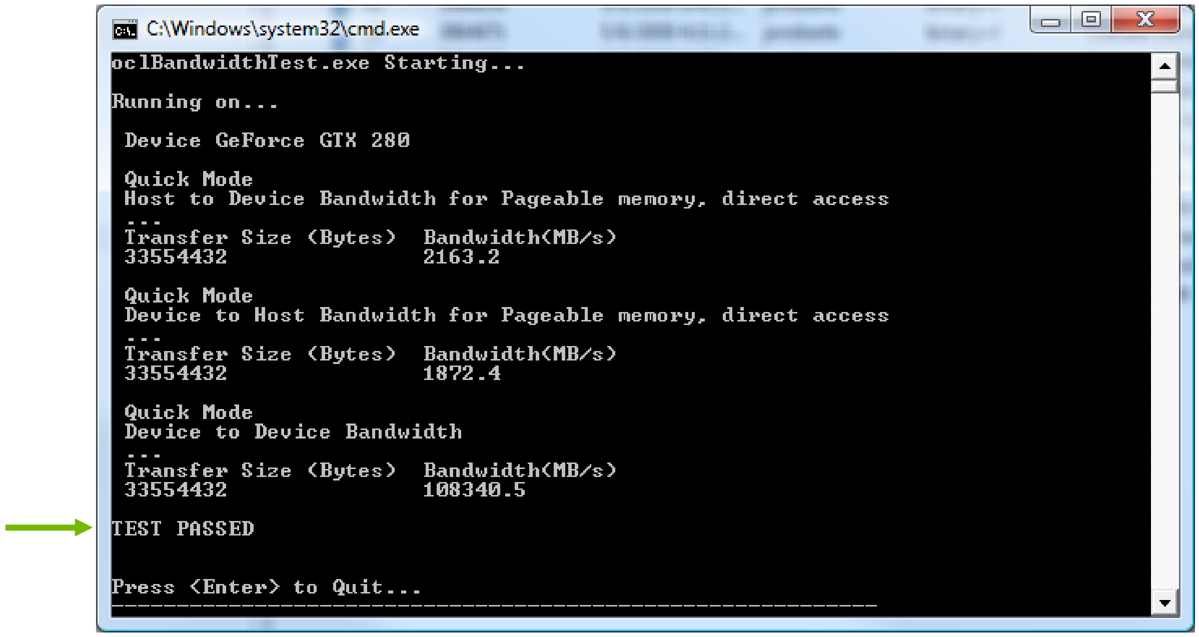
```
oclDeviceQuery.exe Starting...

OpenCL SW Info:

 CL_PLATFORM_NAME:      NVIDIA
 CL_PLATFORM_VERSION:   OpenCL 1.0
 OpenCL SDK Version:    $DateTime: 2009/08/25 01:12:03 $   $Change: 4557094 $


OpenCL Device Info:

 # of devices supporting OpenCL = 2:

  CL_DEVICE_VENDOR:                     NVIDIA Corporation
  CL_DEVICE_NAME:                       GeForce GTX 280
  CL_DRIVER_VERSION:                    190.86
  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:            1024 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_nv_device_attribute_query
                                        cl_khr_global_int32_base_atomics
                                        cl_khr_global_int32_extended_atomics
                                        cl_khr_local_int32_base_atomics
                                        cl_khr_local_int32_extended_atomics
  CL_NV_DEVICE_COMPUTE_CAPABILITY:      1.3
  CL_NV_DEVICE_REGISTERS_PER_BLOCK:     16384
  CL_NV_DEVICE_WARP_SIZE:               32
  CL_NV_DEVICE_GPU_OVERLAP:             CL_TRUE
  CL_NV_DEVICE_KERNEL_EXEC_TIMEOUT:     CL_FALSE
  CL_NV_DEVICE_INTEGRATED_MEMORY:       CL_FALSE

  CL_DEVICE_VENDOR:                     NVIDIA Corporation
  CL_DEVICE_NAME:                       Tesla C1060
  CL_DRIVER_VERSION:                    190.86
  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:            4096 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_nv_device_attribute_query
                                        cl_khr_global_int32_base_atomics
                                        cl_khr_global_int32_extended_atomics
                                        cl_khr_local_int32_base_atomics
                                        cl_khr_local_int32_extended_atomics
  CL_NV_DEVICE_COMPUTE_CAPABILITY:      1.3
  CL_NV_DEVICE_REGISTERS_PER_BLOCK:     16384
  CL_NV_DEVICE_WARP_SIZE:               32
  CL_NV_DEVICE_GPU_OVERLAP:             CL_TRUE
  CL_NV_DEVICE_KERNEL_EXEC_TIMEOUT:     CL_FALSE
  CL_NV_DEVICE_INTEGRATED_MEMORY:       CL_FALSE


System Info:

 Local Time/Date = 11:17:5, 8/25/2009
 CPU Arch: 9
 CPU Level: 6
 # of CPU processors: 4
 Windows Build: 6002
 Windows Ver: 6.0


TEST PASSED...
```

Figure 2. Example output from **oclDeviceQuery** sample

- The device name(s) and other details will vary from system to system (the system shown contains 2 GPU's, for example). The important outcomes are that:
  - OpenCL v 1.0 is detected (the 1st highlighted line),
  - At least one OpenCL capable device that matches one installed on your system is found (the 2nd highlighted line),

- ❑ "PASSED" is displayed at completion of the sample (the 3rd highlighted line).

❑ As an additional check, running the **oclBandwidthTest.exe** program ensures that the system and the CUDA device are able to communicate correctly. Its output is shown in Figure 3.



```
oclBandwidthTest.exe Starting...

Running on...

 Device GeForce GTX 280

 Quick Mode
 Host to Device Bandwidth for Pageable memory, direct access
 ...
 Transfer Size (Bytes)  Bandwidth(MB/s)
 33554432               2163.2

 Quick Mode
 Device to Host Bandwidth for Pageable memory, direct access
 ...
 Transfer Size (Bytes)  Bandwidth(MB/s)
 33554432               1872.4

 Quick Mode
 Device to Device Bandwidth
 ...
 Transfer Size (Bytes)  Bandwidth(MB/s)
 33554432               108340.5

TEST PASSED


Press <Enter> to Quit...
-----------------------------------------------------------
```

Figure 3. Example of console output from **oclBandwidthTest** sample

❑ The oclBandwidthTest results for your OpenCL device will vary from system to system. The important outcomes are that:

- ❑ Some measurements are obtained
- ❑ The output confirms "PASSED" (as shown at the highlighted line).

**Note:** Should these tests not succeed, make sure you do have a supported NVIDIA GPU and Microsoft Windows operating system, that the proper GPU/Display drivers have been installed on your system, and that the SDK has been properly installed.

# Chapter 3.
# A simple OpenCL Example

```c
//*******************************************************************
// 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;
}
```

# Chapter 4. SDK Programming Notes

## A bit of Visual Studio Integration

For an optimal OpenCL development experience, you may want to set up "Syntax Highlighting" for Visual Studio. This will allow keywords such as OpenCL data type names to be highlighted in color just like Visual Studio does by default for native types within .c, .h and.cpp files. Setting this up is simple, as follows:

- The NVIDIA OpenCL SDK provides an OpenCL specific usertype.dat file containing a formatted list of OpenCL API data types. Adding this file to the proper directory (or pasting its contents into any pre-existing copy of this file) prior to starting Visual Studio will provide highlighting of the OpenCL specific data types. By default, the OpenCL SDK installs this file for your use to:

  **C:\Documents and Settings\All Users\Application Data\NVIDIA Corporation\NVIDIA GPU Computing SDK\OpenCL**

- Check to see if your system already has a file called "usertype.dat" in your Visual Studio install directory within the \Common7\IDE folder. The default locations are:
  - 32 bit Windows:

    **(VC8) C:\Program Files\Microsoft Visual Studio 8\Common7\IDE**

    **(VC9) C:\Program Files\Microsoft Visual Studio 9\Common7\IDE**
  - 64 bit Windows:

    **(VC8) C:\Program Files (x86)\Microsoft Visual Studio 8\Common7\IDE**

    **(VC9) C:\Program Files (x86)\Microsoft Visual Studio 9\Common7\IDE**

- If you **do not** have a pre-existing "usertype.dat" file in the Visual Studio subdirectory, then copy the OpenCL-specific usertype.dat file provided in this SDK the **\Common 7\IDE** folder in the Visual Studio installation directory.

- If you **do** have a pre-existing "usertype.dat" file in the Visual Studio subdirectory, open it using Notepad, UltraEdit, etc. Then also open the OpenCL-specific usertype.dat file provided in this SDK, copy the contents of the OpenCL-specific file and paste the contents into your pre-existing "usertype.dat" file. Then save and close the updated usertype.dat file.

- Run Visual Studio and perform these simple steps.
  - Select "Tools-> Options" from the center of the main menu bar at the top of the Visual Studio window. The ***Options*** dialog box will appear.
  - Expand the "Text Editor" node from the tree view at the left and select "File Extension".
  - At the top left of the dialog in the box labeled "Extension", enter ".cl". This is the recommended file extension for OpenCL kernel code. (Another extension may be used, but whatever extension is used should be used consistently and should be entered here).

  - Select "Microsoft Visual C++" in the drop down menu to the immediate right, and then click the "Add" button further to the right and the "OK" button at the bottom right of the dialog.
- Close and restart Visual Studio. Syntax highlighting for OpenCL keywords will be operative for .cl files, in addition to the normal file types such as .c, .cpp, .h, etc.

## 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, 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 to be approachable 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 an 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**.

# Utilities

- 2 utility libraries (**shrUtils**.lib and **oclUtils**.lib) are provided and used with the NVIDIA GPU Computing SDK. These are conventional host C/C++ function libraries provided for convenience only. They include functionality that helps reduce code bloat in the SDK but are considered secondary to the main concern of the SDK, i.e. exposition of the GPU computing API's (e.g. OpenCL, CUDA-C, Direct Compute…) and of GPU computing techniques.

  **These utility libraries are not required for developers to write their own custom OpenCL applications.**

- The source code for oclUtils and shrUtils is provided with the NVIDIA GPU Computing SDK.
  - The header and implementation files contain comment blocks and descriptive code useful in understanding the structure and operation of each function. So after reading this information, it may be useful to review the header files and/or main implementation files for these 2 libraries.
  - Since the source code is provided, developers can set breakpoints in the OpenCL application samples and step into these utility functions to follow the program flow, interact with local variables, etc. Modifications or substitutions can also be easily be made if additional or alternate utility functionality is needed.

- The **oclUtils** library contains simple helper functions, constants and macros specific to the OpenCL samples in the NVIDIA GPU Computing SDK. oclUtils.h also serves as a common header--by including oclUtils.h, a code page also includes shrUtils, cl.h, cl_ext.h and other headers within those files.
  - oclUtils functions are specific to the OpenCL API and OpenCL samples in the SDK, but are **not** part of the OpenCLAPI or OpenCL C language extensions.

  - The primary oclUtils files are:

| | |
|---|---|
| oclUtils.h | \NVIDIA GPU Computing SDK\OpenCL\common\inc\ |
| oclUtils.cpp | \NVIDIA GPU Computing SDK\OpenCL\common\src\ |
| oclUtils[32][64].lib | \NVIDIA GPU Computing SDK\OpenCL\common\lib\ |
| oclUtils[32][64]D.lib | \NVIDIA GPU Computing SDK\OpenCL\common\lib\ |
| oclUtils.sln | \NVIDIA GPU Computing SDK\OpenCL\common |
| oclUtils_vc9.sln | \NVIDIA GPU Computing SDK\OpenCL\common |

  - oclUtils functions are named starting with the letters "**ocl**…" so they are easy to identify within samples. Some of the most commonly used oclUtils functions are:

    oclGetPlatformID(), oclPrintDevName(), oclPrintDevInfo(), oclLoadProgSource(), oclLogPtx(), oclLogBuildInfo(), oclGetMaxFlopsDev()

- The **shrUtils** library contains simple helper functions, constants and macros generic to samples in any portion of the NVIDIA GPU Computing SDK (not just OpenCL, but also CUDA-C and Direct Compute). shrUtils.h also serves as a common header for standard system includes like stdio.h and math.h and other OS-dependent headers (like windows.h for Windows targets).
  - The primary shrUtils files are:

| | |
|---|---|
| shrUtils.h | \NVIDIA GPU Computing SDK\shared\inc |
| shrUtils.cpp | \NVIDIA GPU Computing SDK\shared\src |
| rendercheckGL.cpp | \NVIDIA GPU Computing SDK\shared\src |
| cmd_arg_reader.cpp | \NVIDIA GPU Computing SDK\shared\src |
| shrUtils[32][64].lib | \NVIDIA GPU Computing SDK\shared\lib |
| shrUtils[32][64]D.lib | \NVIDIA GPU Computing SDK\shared\lib |
| shrUtils.sln | \NVIDIA GPU Computing SDK\shared |
| shrUtils_vc9.sln | \NVIDIA GPU Computing SDK\shared |

  - shrUtils functions are named starting with the letters "**shr**…" so they are easy to identify within samples. Some of the most commonly used shrUtils functions are:

    shrLog(), shrDeltaT(), shrFindFilePath(), shrCheckCmdLineFlag(), shrGetCmdLineArgumenti(), shrCheckError(), shrCheckErrorEX() and shrEXIT().

    Note: To enable performance timing and logging code built into many SDK samples, uncomment the following line found near the top of shrUtils.h before rebuilding the SDK.

    ```
    //#define GPU_PROFILING
    ```

- oclRelease.sln and oclRelease_vc9.sln are set up to first automatically build shrUtils.lib and oclUtils.lib so that OpenCL samples in those solutions and depending upon those libraries can statically link to them.
- oclUtils.sln and oclUtils_vc9.sln are set up to first automatically build shrUtils.lib so that oclUtils.lib can statically link against generic functions in shrUtils.

## Creating a new OpenCL Program using the SDK infrastructure

- Creating a new OpenCL Program using the SDK infrastructure is easy. Just follow these steps:
  - Copy one of the installed OpenCL SDK project folders, in its entirety, into "\NVIDIA GPU Computing SDK\OpenCL\src" and then rename the folder. Now you have something like "\NVIDIA GPU Computing SDK\OpenCL\src\<myproject>".
  - Edit the filenames of the project to suit your needs.
  - Edit the *.sln, *.vcproj and source files. Just search and replace all occurrences of the old filenames to the new ones you chose.
  - Open <myproject>.sln or <myproject>_vc9.sln in VS 8/2005 or VS9/2008 respectively, select the Win32 or x64 target platform (match the OS type for the current installation) and then rebuild the release and debug configurations of the project.
  - Run <myproject>.exe from the release or debug, directories located in "NVIDIA GPU Computing SDK\OpenCL\bin\win[32|64]\[release|debug]".
  - Modify the code to perform the computation you require.

## Creating a new OpenCL Program outside of the SDK infrastructure

- To create a new OpenCL Program without using the NVIDIA SDK infrastructure, a few key files are important to find and utilize.
- The only OpenCL-specific files needed to build an application to run with NVIDIA compute-capable GPU's with CUDA architecture on a system with a supported OS using the specified NVIDIA Display driver that supports OpenCL, are:

**Headers:**

cl.h
cl_platform.h
cl_ext.h
cl_gl.h
cl_gl_ext.h
cl_d3d10.h
cl_d3d10_ext.h
cl_d3d9.h
cl_d3d9_ext.h
cl_d3d9.h
cl_d3d9_ext.h
opencl.h

These header files are standard located in:

"NVIDIA GPU Computing SDK\OpenCL\common\inc\CL"

**Libs**

OpenCL.lib (different file for Win32 and x64 platforms)

These .lib files are for build-time implicit linking to the OpenCL driver/compiler, OpenCL.dll, and they are located in:

"NVIDIA GPU Computing SDK\OpenCL\common\lib\[Win32|x64]

**Note:** These lib files are not needed for applications implementing explicit DLL linkage at run-time.

**Dynamic Link Library (DLL)**

OpenCL.dll

This file is the binary OpenCL driver/compiler. The appropriate version of this file for a target platform is installed by the correct NVIDIA GPU driver installed on the system. It is needed during OpenCL program execution and is normally installed in the **\Windows\system32** directory (in the default path for the system).

## What Next?

- Now that you have CUDA-capable hardware and the OpenCL software installed, you can examine and enjoy the numerous included sample programs. For additional assistance accelerating your applications with OpenCL, consult the *OpenCL Programming Guide,  OpenCL Best Practices 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 and mailing list at http://forums.nvidia.com/index.php?showforum=134