



NVIDIA CUDA INSTALLATION GUIDE FOR MICROSOFT WINDOWS

DU-05349-001_v11.0 | June 2020

Installation and Verification on Windows



TABLE OF CONTENTS

Chapter 1. Introduction.....	1
1.1. System Requirements.....	1
1.1.1. x86 32-bit Support.....	2
1.2. About This Document.....	3
Chapter 2. Installing CUDA Development Tools.....	4
2.1. Verify You Have a CUDA-Capable GPU.....	4
2.2. Download the NVIDIA CUDA Toolkit.....	4
2.3. Install the CUDA Software.....	5
2.3.1. Uninstalling the CUDA Software.....	8
2.4. Use a Suitable Driver Model.....	8
2.5. Verify the Installation.....	9
2.5.1. Running the Compiled Examples.....	9
Chapter 3. Compiling CUDA Programs.....	11
3.1. Compiling Sample Projects.....	11
3.2. Sample Projects.....	11
3.3. Build Customizations for New Projects.....	12
3.4. Build Customizations for Existing Projects.....	12
Chapter 4. Additional Considerations.....	14

Chapter 1.

INTRODUCTION

CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA was developed with several design goals in mind:

- ▶ Provide a small set of extensions to standard programming languages, like C, that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- ▶ Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources.

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

This guide will show you how to install and check the correct operation of the CUDA development tools.

1.1. System Requirements

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

- ▶ A CUDA-capable GPU
- ▶ A supported version of Microsoft Windows
- ▶ A supported version of Microsoft Visual Studio
- ▶ the NVIDIA CUDA Toolkit (available at <http://developer.nvidia.com/cuda-downloads>)

The next two tables list the currently supported Windows operating systems and compilers.

Table 1 Windows Operating System Support in CUDA 11.0

Operating System	Native x86_64	Cross (x86_32 on x86_64)
Windows 10	YES	YES
Windows Server 2019	YES	NO
Windows Server 2016	YES	NO

Table 2 Windows Compiler Support in CUDA 11.0

Compiler*	IDE	Native x86_64	Cross (x86_32 on x86_64)
MSVC Version 192x	Visual Studio 2019 16.x (RTW and all updates)	YES	YES
MSVC Version 191x	Visual Studio 2017 15.x (RTW and all updates)	YES	YES
MSVC Version 1900	Visual Studio 2015 14.0 (RTW and updates 1, 2, and 3)	YES	YES
	Visual Studio Community 2015	YES	YES
MSVC Version 1800	Visual Studio 2013 12.0	YES	YES
MSVC Version 1700	Visual Studio 2012 11.0	YES	YES

x86_32 support is limited. See the [x86 32-bit Support](#) section for details.

For more information on MSVC versions, Visual Studio product versions, visit <https://dev.to/yumetodo/list-of-mscver-and-mscfullver-8nd>.

1.1.1. x86 32-bit Support

Native development using the CUDA Toolkit on x86_32 is unsupported. Deployment and execution of CUDA applications on x86_32 is still supported, but is limited to use with GeForce GPUs. To create 32-bit CUDA applications, use the cross-development capabilities of the CUDA Toolkit on x86_64.

Support for developing and running x86 32-bit applications on x86_64 Windows is limited to use with:

- ▶ GeForce GPUs
- ▶ CUDA Driver
- ▶ CUDA Runtime (cudart e.g. cudart32_xx.dll in lib\Win32)
- ▶ CUDA Math Library (math.h)
- ▶ CUDA C++ Compiler (nvcc)

1.2. About This Document

This document is intended for readers familiar with Microsoft Windows operating systems and the Microsoft Visual Studio environment. You do not need previous experience with CUDA or experience with parallel computation.

Chapter 2.

INSTALLING CUDA DEVELOPMENT TOOLS

Basic instructions can be found in the [Quick Start Guide](#). Read on for more detailed instructions.

The setup of CUDA development tools on a system running the appropriate version of Windows consists of a few simple steps:

- ▶ Verify the system has a CUDA-capable GPU.
- ▶ Download the NVIDIA CUDA Toolkit.
- ▶ Install the NVIDIA CUDA Toolkit.
- ▶ Test that the installed software runs correctly and communicates with the hardware.

2.1. Verify You Have a CUDA-Capable GPU

You can verify that you have a CUDA-capable GPU through the **Display Adapters** section in the **Windows Device Manager**. Here you will find the vendor name and model of your graphics card(s). If you have an NVIDIA card that is listed in <http://developer.nvidia.com/cuda-gpus>, that GPU is CUDA-capable. The Release Notes for the CUDA Toolkit also contain a list of supported products.

The **Windows Device Manager** can be opened via the following steps:

1. Open a run window from the Start Menu
2. Run:

```
control /name Microsoft.DeviceManager
```

2.2. Download the NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit is available at <http://developer.nvidia.com/cuda-downloads>. Choose the platform you are using and one of the following installer formats:

1. Network Installer: A minimal installer which later downloads packages required for installation. Only the packages selected during the selection phase of the installer

are downloaded. This installer is useful for users who want to minimize download time.

2. Full Installer: An installer which contains all the components of the CUDA Toolkit and does not require any further download. This installer is useful for systems which lack network access and for enterprise deployment.

The CUDA Toolkit installs the CUDA driver and tools needed to create, build and run a CUDA application as well as libraries, header files, CUDA samples source code, and other resources.

Download Verification

The download can be verified by comparing the MD5 checksum posted at <http://developer.nvidia.com/cuda-downloads/checksums> with that of the downloaded file. If either of the checksums differ, the downloaded file is corrupt and needs to be downloaded again.

To calculate the MD5 checksum of the downloaded file, follow the instructions at <http://support.microsoft.com/kb/889768>.

2.3. Install the CUDA Software

Before installing the toolkit, you should read the *Release Notes*, as they provide details on installation and software functionality.



The driver and toolkit must be installed for CUDA to function. If you have not installed a stand-alone driver, install the driver from the NVIDIA CUDA Toolkit.



The installation may fail if Windows Update starts after the installation has begun. Wait until Windows Update is complete and then try the installation again.

Graphical Installation

Install the CUDA Software by executing the CUDA installer and following the on-screen prompts.

Silent Installation

The installer can be executed in silent mode by executing the package with the `-s` flag. Additional parameters can be passed which will install specific subpackages instead of all packages. See the table below for a list of all the subpackage names.

Table 3 Possible Subpackage Names

Subpackage Name	Subpackage Description
Toolkit Subpackages (defaults to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.0)	
nvcc_11.0	CUDA compiler.
cuobjdump_11.0	Extracts information from cubin files.
nvprune_11.0	Prunes host object files and libraries to only contain device code for the specified targets.
cupti_11.0	The CUDA Profiler Tools Interface for creating profiling and tracing tools that target CUDA applications.
gpu_library_advisor_11.0	Identifies opportunities to improve application performance by replacing existing CPU libraries with GPU-accelerated versions of those libraries.
memcheck_11.0	Functional correctness checking suite.
nvdasm_11.0	Extracts information from standalone cubin files.
nvprof_11.0	Tool for collecting and viewing CUDA application profiling data from the command-line.
visual_profiler_11.0	GUI profiler for CUDA applications.
visual_studio_integration_11.0	Installs CUDA project wizard and builds customization files in VS.
nsight_vse_11.0	Installs the Nsight Visual Studio Edition plugin in all VS.
nsight_nvtx_11.0	Installs NVTX on Windows.
demo_suite_11.0	Prebuilt demo applications using CUDA.
documentation_11.0	CUDA HTML and PDF documentation files including the CUDA C++ Programming Guide, CUDA C++ Best Practices Guide, CUDA library documentation, etc.
cublas_11.0	CUBLAS runtime libraries.
cublas_dev_11.0	CUBLAS development libraries and headers.
cuda_11.0	CUDART runtime libraries.
cufft_11.0	CUFFT runtime libraries.
cufft_dev_11.0	CUFFT development libraries and headers.
curand_11.0	CURAND runtime libraries.
curand_dev_11.0	CURAND development libraries and headers.

Subpackage Name	Subpackage Description
cusolver_11.0	CUSOLVER runtime libraries.
cusolver_dev_11.0	CUSOLVER development libraries and headers.
cusparse_11.0	CUSPARSE runtime libraries.
cusparse_dev_11.0	CUSPARSE development libraries and headers.
npp_11.0	NPP runtime libraries.
npp_dev_11.0	NPP development libraries and headers.
nVRTC_11.0	NVRTC runtime libraries.
nVRTC_dev_11.0	NVRTC development libraries and headers.
nvml_dev_11.0	NVML development libraries and headers.
occupancy_calculator_11.0	Excel spreadsheet used to test occupancy scenarios.
fortran_examples_11.0	CUDA Fortran examples.
Samples Subpackages (defaults to C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.0)	
samples_11.0	Source code for many example CUDA applications using supported versions of Visual Studio. Note: C:\ProgramData\ is a hidden folder. It can be made visible within the Windows Explorer options at (Tools Options).
Driver Subpackages	
Display.Driver	The NVIDIA Display Driver. Required to run CUDA applications.

For example, to install only the compiler and driver components:

```
<PackageName>.exe -s nvcc_11.0 Display.Driver
```

Extracting and Inspecting the Files Manually

Sometimes it may be desirable to extract or inspect the installable files directly, such as in enterprise deployment, or to browse the files before installation. The full installation package can be extracted using a decompression tool which supports the LZMA compression method, such as [7-zip](#) or [WinZip](#).

Once extracted, the CUDA Toolkit files will be in the **CUDAToolkit** folder, and similarly for the CUDA Samples and CUDA Visual Studio Integration. Within each directory is a .dll and .nvi file that can be ignored as they are not part of the installable files.



Accessing the files in this manner does not set up any environment settings, such as variables or Visual Studio integration. This is intended for enterprise-level deployment.

2.3.1. Uninstalling the CUDA Software

All subpackages can be uninstalled through the Windows Control Panel by using the Programs and Features widget.

2.4. Use a Suitable Driver Model

On Windows 7 and later, the operating system provides two *driver models* under which the NVIDIA Driver may operate:

- ▶ The WDDM driver model is used for display devices.
- ▶ The *Tesla Compute Cluster (TCC)* mode of the NVIDIA Driver is available for non-display devices such as NVIDIA Tesla GPUs, and the GeForce GTX Titan GPUs; it uses the Windows WDM driver model.

The TCC driver mode provides a number of advantages for CUDA applications on GPUs that support this mode. For example:

- ▶ TCC eliminates the timeouts that can occur when running under WDDM due to the Windows *Timeout Detection and Recovery* mechanism for display devices.
- ▶ TCC allows the use of CUDA with Windows Remote Desktop, which is not possible for WDDM devices.
- ▶ TCC allows the use of CUDA from within processes running as Windows services, which is not possible for WDDM devices.
- ▶ TCC reduces the latency of CUDA kernel launches.

TCC is enabled by default on most recent NVIDIA Tesla GPUs. To check which driver mode is in use and/or to switch driver modes, use the **nvidia-smi** tool that is included with the NVIDIA Driver installation (see **nvidia-smi -h** for details).



Keep in mind that when TCC mode is enabled for a particular GPU, that GPU *cannot* be used as a display device.



NVIDIA GeForce GPUs (excluding GeForce GTX Titan GPUs) do not support TCC mode.

2.5. Verify the Installation

Before continuing, it is important to verify that the CUDA toolkit can find and communicate correctly with the CUDA-capable hardware. To do this, you need to compile and run some of the included sample programs.

2.5.1. Running the Compiled Examples

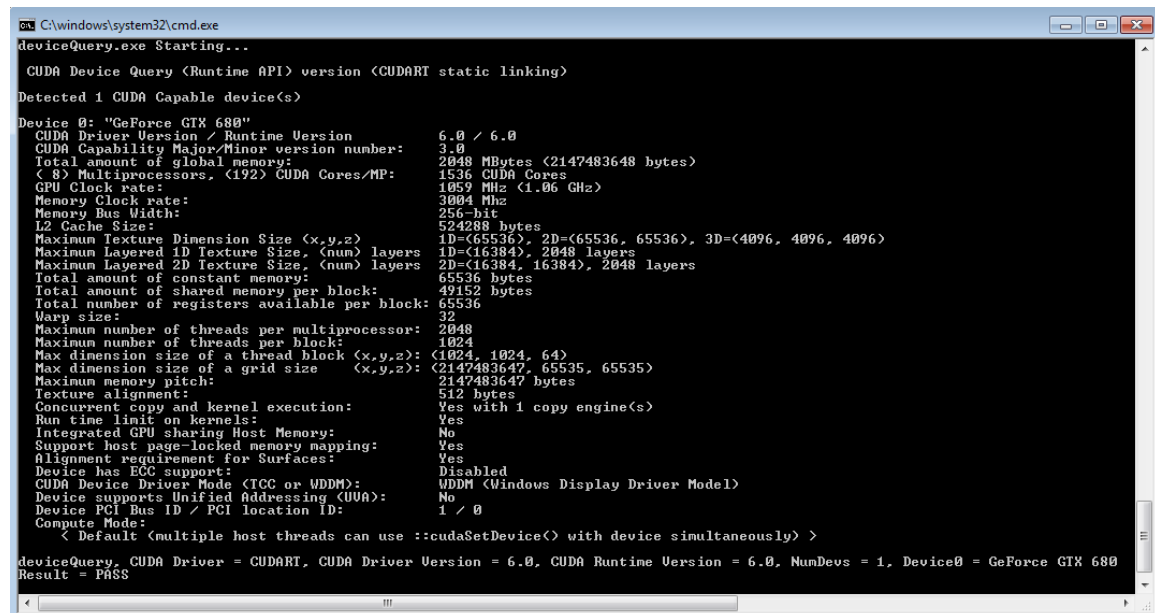
The version of the CUDA Toolkit can be checked by running **nvcc -v** in a **Command Prompt** window. You can display a **Command Prompt** window by going to:

Start > All Programs > Accessories > Command Prompt

CUDA Samples include sample programs in *source* form. To verify a correct configuration of the hardware and software, it is highly recommended that you build and run the **deviceQuery** sample program. The sample can be built using the provided VS solution files, and the compiled executable can be located at:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.0\bin\win64\Release

This assumes that you used the default installation directory structure. If CUDA is installed and configured correctly, the output should look similar to [Figure 1](#).



```

C:\windows\system32\cmd.exe
deviceQuery.exe Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 680"
  CUDA Driver Version / Runtime Version      6.0 / 6.0
  CUDA Capability Major/Minor version number: 3.0
  Total amount of global memory:              2048 MBytes (2147483648 bytes)
  < 8> Multiprocessors, (192) CUDA Cores/MP:  1536 CUDA Cores
  GPU Clock rate:                             1059 MHz (1.06 GHz)
  Memory Clock rate:                          3004 MHz
  Memory Bus Width:                           256-bit
  L2 Cache Size:                              524288 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:             65536 bytes
  Total amount of shared memory per block:     49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:         1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z):   (2147483647, 65535, 65535)
  Maximum memory pitch:                       2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:       Yes with 1 copy engine(s)
  Run time limit on kernels:                  Yes
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     Yes
  Alignment requirement for Surfaces:          Yes
  Device has ECC support:                     Disabled
  CUDA Device Driver Mode (TCG or UDDM):       UDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):    No
  Device PCI Bus ID / PCI location ID:        1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.0, CUDA Runtime Version = 6.0, NumDevs = 1, Device0 = GeForce GTX 680
Result = PASS
  
```

Figure 1 Valid Results from deviceQuery CUDA Sample

The exact appearance and the output lines might be different on your system. The important outcomes are that a device was found, that the device(s) match what is installed in your system, and that the test passed.

If a CUDA-capable device and the CUDA Driver are installed but **deviceQuery** reports that no CUDA-capable devices are present, ensure the device and driver are properly installed.

Running the **bandwidthTest** program, located in the same directory as **deviceQuery** above, ensures that the system and the CUDA-capable device are able to communicate correctly. The output should resemble [Figure 2](#).

```

C:\Windows\system32\cmd.exe

[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: Quadro K5000
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                  5751.2

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                  6342.7

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
Transfer Size (Bytes)      Bandwidth(MB/s)
33554432                  131928.7

Result = PASS

```

Figure 2 Valid Results from bandwidthTest CUDA Sample

The device name (second line) and the bandwidth numbers vary from system to system. The important items are the second line, which confirms a CUDA device was found, and the second-to-last line, which confirms that all necessary tests passed.

If the tests do not pass, make sure you do have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

To see a graphical representation of what CUDA can do, run the sample Particles executable at

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.0\bin\win64\Release
```

Chapter 3.

COMPILING CUDA PROGRAMS

The project files in the CUDA Samples have been designed to provide simple, one-click builds of the programs that include all source code. To build the Windows projects (for release or debug mode), use the provided ***.sln** solution files for Microsoft Visual Studio 2010, 2012, or 2013. You can use either the solution files located in each of the examples directories in

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.0\<category>\<sample_name>
```

or the global solution files **Samples*.sln** located in

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.0
```

CUDA Samples are organized according to **<category>**. Each sample is organized into one of the following folders: (**0_Simple, 1_Uilities, 2_Graphics, 3_Imaging, 4_Finance, 5_Simulations, 6_Advanced, 7_CUDALibraries**).

3.1. Compiling Sample Projects

The **bandwidthTest** project is a good sample project to build and run. It is located in the **NVIDIA Corporation\CUDA Samples\v11.0\1_Uilities\bandwidthTest** directory.

If you elected to use the default installation location, the output is placed in **CUDA Samples\v11.0\bin\win64\Release**. Build the program using the appropriate solution file and run the executable. If all works correctly, the output should be similar to [Figure 2](#).

3.2. Sample Projects

The sample projects come in two configurations: debug and release (where release contains no debugging information) and different Visual Studio projects.

A few of the example projects require some additional setup. The **simpleD3D9** example requires the system to have a Direct3D SDK (June 2010 or later) installed and the Visual C++ directory paths (located in **Tools > Options...**) properly configured. Consult the Direct3D documentation for additional details.

These sample projects also make use of the `$CUDA_PATH` environment variable to locate where the CUDA Toolkit and the associated `.props` files are.

The environment variable is set automatically using the Build Customization **CUDA 11.0.props** file, and is installed automatically as part of the CUDA Toolkit installation process.

Table 4 CUDA Visual Studio .props locations

Visual Studio	CUDA 11.0 .props file Install Directory
Visual Studio 2010	C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\BuildCustomizations
Visual Studio 2012	C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V110\BuildCustomizations
Visual Studio 2013	C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V120\BuildCustomizations
Visual Studio 2015	C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\BuildCustomizations
Visual Studio 2017	<Visual Studio Install Dir>\Common7\IDE\VC\VCTargets\BuildCustomizations
Visual Studio 2019	C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\MSBuild\Microsoft\VC\v160\BuildCustomizations

You can reference this **CUDA 11.0.props** file when building your own CUDA applications.

3.3. Build Customizations for New Projects

When creating a new CUDA application, the Visual Studio project file must be configured to include CUDA build customizations. To accomplish this, click File-> New | Project... NVIDIA-> CUDA->, then select a template for your CUDA Toolkit version. For example, selecting the "CUDA 11.0 Runtime" template will configure your project for use with the CUDA 11.0 Toolkit. The new project is technically a C++ project (.vcxproj) that is preconfigured to use NVIDIA's Build Customizations. All standard capabilities of Visual Studio C++ projects will be available.

To specify a custom CUDA Toolkit location, under **CUDA C/C++**, select **Common**, and set the **CUDA Toolkit Custom Dir** field as desired. Note that the selected toolkit must match the version of the Build Customizations.

3.4. Build Customizations for Existing Projects

When adding CUDA acceleration to existing applications, the relevant Visual Studio project files must be updated to include CUDA build customizations. This can be done using one of the following two methods:

1. Open the Visual Studio project, right click on the project name, and select **Build Customizations...**, then select the CUDA Toolkit version you would like to target.
2. Alternatively, you can configure your project always to build with the most recently installed version of the CUDA Toolkit. First add a CUDA build customization to your project as above. Then, right click on the project name and select **Properties**.

Under **CUDA C/C++**, select **Common**, and set the **CUDA Toolkit Custom Dir** field to **\$ (CUDA_PATH)** . Note that the **\$ (CUDA_PATH)** environment variable is set by the installer.

While Option 2 will allow your project to automatically use any new CUDA Toolkit version you may install in the future, selecting the toolkit version explicitly as in Option 1 is often better in practice, because if there are new CUDA configuration options added to the build customization rules accompanying the newer toolkit, you would not see those new options using Option 2.

If you use the **\$ (CUDA_PATH)** environment variable to target a version of the CUDA Toolkit for building, and you perform an installation or uninstallation of any version of the CUDA Toolkit, you should validate that the **\$ (CUDA_PATH)** environment variable points to the correct installation directory of the CUDA Toolkit for your purposes. You can access the value of the **\$ (CUDA_PATH)** environment variable via the following steps:

1. Open a run window from the Start Menu
2. Run:

```
control sysdm.cpl
```
3. Select the "Advanced" tab at the top of the window
4. Click "Environment Variables" at the bottom of the window

Files which contain CUDA code must be marked as a **CUDA C/C++** file. This can be done when adding the file by right clicking the project you wish to add the file to, selecting **Add\New Item**, selecting **NVIDIA CUDA 11.0\Code\CUDA C/C++ File**, and then selecting the file you wish to add.

Note for advanced users: If you wish to try building your project against a newer CUDA Toolkit without making changes to any of your project files, go to the Visual Studio command prompt, change the current directory to the location of your project, and execute a command such as the following:

```
msbuild <projectname.extension> /t:Rebuild /p:CudaToolkitDir="drive:/path/to/new/toolkit/"
```

Chapter 4.

ADDITIONAL CONSIDERATIONS

Now that you have CUDA-capable hardware and the NVIDIA CUDA Toolkit installed, you can examine and enjoy the numerous included programs. To begin using CUDA to accelerate the performance of your own applications, consult the *CUDA C Programming Guide*, located in the CUDA Toolkit documentation directory.

A number of helpful development tools are included in the CUDA Toolkit or are available for download from the NVIDIA Developer Zone to assist you as you develop your CUDA programs, such as NVIDIA® Nsight™ Visual Studio Edition, NVIDIA Visual Profiler, and cuda-memcheck.

For technical support on programming questions, consult and participate in the developer forums at <http://developer.nvidia.com/cuda/>.

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 of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2009-2020 NVIDIA Corporation. All rights reserved.