GETTING STARTED WITH CUDA SDK SAMPLES

DA-05723-001_v01 | May 2011

Application Note

TABLE OF CONTENTS

GETTING STARTED WITH CUDA SDK SAMPLES

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.

This document is intended to introduce to you a set of SDK samples that can be run as an introduction to CUDA. Most of these SDK samples use the CUDA runtime API except for ones explicitly noted that are CUDA Driver API.

To run these SDK samples, you should have experience with C and/or C++. It is not required that you have any parallel programming experience to start out.

The CUDA C SDK samples listed in this document are found in both the **`C`** and **`CUDALibraries`** default directories in the following folders:

- **Windows:**
  `ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0`
- **Linux:**
  `~/NVIDIA GPU Computing SDK`
- **Mac OSX:**
  `/Developer/GPU Computing`

BEFORE YOU BEGIN

This document assumes you have installed CUDA on your system. CUDA runs on Windows, Mac, and Linux environments. To install CUDA, refer to the *CUDA Getting Started Guide* available with the SDK and on the CUDA web site at:

http://www.nvidia.com/cuda)

GETTING STARTED WITH SDK SAMPLES

The list of SDK samples is divided up into three categories:

- Getting started samples
  If you are new to CUDA, these are the best SDK samples to begin with.
- Simple CUDA samples
- Samples that demonstrate CUDA + Graphics interoperability

Note: There are some overlaps between the three categories.

Getting Started Samples

matrixMul

This sample implements matrix multiplication. It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performance generic kernel for matrix multiplication. This SDK sample also shows how to use the CUBLAS to provide an example of high-performance matrix multiplication.

simpleTemplates

This sample is a templatized version of the template project. It also shows how to correctly templatize dynamically-allocated shared memory arrays.

template

This sample is a basic template project that can be used as a starting point for creating new CUDA projects.

template_runtime

This is a simple template project that can be used as a starting point to create a new CUDA project that does not use the **`cutil`** library.

Simpler CUDA Samples

BandwidthTest

This is a test program to measure the memory copy bandwidth of the GPU. It currently is capable of measuring device-to-device copy bandwidth, host-to-device copy bandwidth for pageable and page-locked memory, and device-to-host copy bandwidth for pageable and page-locked memory.

Clock

This example shows how to use the clock function in CUDA kernels to measure the performance within a kernel accurately.

cudaOpenMP

This is a sample application demonstrating how to use the OpenMP API for launching workloads across multiple GPUs. The binaries for this sample are not pre-built with the Windows SDK installer.

deviceQuery, deviceQueryDrv

These two SDK samples show how to enumerate properties of the CUDA devices present in the system (using the CUDA runtime API). The **`*Drv`** version is the same sample, but it uses the CUDA Driver API.

Ptxjit

This sample demonstrates JIT compilation of PTX code. This sample uses a PTX program embedded in a string array. The CUDA Driver API calls are used to compile and run a PTX program.

simpleAtomicIntrinsics

This is a simple demonstration of global memory atomic instructions. This sample requires Compute Capability 1.1 or higher.

simpleCUBLAS

This is a basic example demonstrating how to use the CUBLAS (CUDA Basic Linear Algebra) library. This sample can be found within the **`CUDALibraries`** folder.

For more details on how to use the CUBLAS Library, refer to the *CUBLAS_Library.pdf Programming Guide* included with the CUDA Toolkit.

simpleCUFFT

This is a basic example demonstrating how to use the CUFFT (CUDA Fast Fourier Transform) Library. In this sample, CUFFT is used to compute the 1D-convolution of a signal. The signal is transformed to the frequency domain, multiplied together with a filter kernel, and then the signal is transformed back to time domain. This sample can be found within the **`CUDALibraries`** folder.

For more details on how to use the CUFFT Library, refer to the *CUBLAS_Library.pdf Programming Guide* included with the CUDA Toolkit.

simpleMPI

This demonstrates how to use MPI in combination with CUDA to demonstrate how to launch workloads across multiple systems that have a GPU. This sample generates some random numbers on one node, dispatches to all nodes, then computes the square root on each node's GPU. Then the average results of the results are computed. The binary is not pre-built with the SDK installer.

simpleMultiCopy

Since Compute Capability 1.1, it is possible to overlap compute with one **`memcopy`** to/from the host. Compute Capability 2.0 with a Tesla or Quadro GPU improves on this by enabling a second parallel copy operation in the opposite direction at full speed (PCIe is symmetric). This sample illustrates the usage of CUDA streams to achieve overlapping of kernel execution with copying data to and from the device.

simpleMultiGPU

This application demonstrates how to use the CUDA API launch workloads across multiple GPUs. With CUDA 4.0, there is a new API for CUDA context management and multi-threaded access. This greatly simplifies the way that CUDA kernels can be launched across multiple GPUs.

For more details, refer to sections 3.2.4.1, 3.2.4.3, and 3.2.6 in the *CUDA C Programming Guide* and to the *CUDA_4.0_Readiness_Tech_Brief.pdf* about the new multi-device programming model.

simplePitchLinearTexture

This sample demonstrates how to use 1D Pitch Linear Textures in a CUDA program.

simplePrintf

This CUDA Runtime API sample is a very basic sample that implements how to use the **`printf`** function in the device code. Specifically, for devices with compute capability less than 2.0, the function **`cuPrintf`** is called; otherwise, **`printf`** can be used directly.

For more details, refer to Appendix B.14 in the *CUDA C Programming Guide* included with the CUDA Toolkit.

simpleStreams

This sample uses CUDA streams to overlap kernel executions with **`memcopies`** between the device and the host. With CUDA 4.0, this sample supports pinning of generic host memory. This SDK sample requires Compute Capability 1.1 or higher.

For more details, refer to sections 3.2.5.5 in the *CUDA C Programming Guide* and to the *CUDA_4.0_Readiness_Tech_Brief.pdf* included with the CUDA Toolkit.

simpleSurfaceWrite

This sample demonstrates the use of surface references, thus enabling write-to-texture. (This sample requires a Fermi-based GPU (Compute Capability 2.0)).

simpleTemplates

This sample is a templatized version of the template project. It also shows how to correctly templatize dynamically-allocated shared memory arrays.

simpleTexture, simpleTextureDrv

This sample demonstrates how to use textures with CUDA (Runtime API and the Driver API versions).

simpleVoteIntrinsics

This is a simple program that demonstrates how to use the Vote (any, all) intrinsic instruction in a CUDA kernel. This sample requires Compute Capability 1.2 or higher.

simpleZeroCopy

This sample illustrates how to use **`Zero MemCopy`**. Kernels can read directly from and write directly to pinned system memory. This sample requires GPUs that support this feature (MCP79, GT200, Fermi based GPUs).

Refer to Section 3.1.3 in the *CUDA_C_Best_Practices_Guide.pdf* for more details.

template

This sample is a basic template project that can be used as a starting point for creating new CUDA projects.

template_runtime

This is a simple template project that can be used as a starting point to create a new CUDA project that does not use the **`cutil`** library.

CUDA + Graphics Interoperability

These SDK samples demonstrate interoperability between CUDA and graphics.

simpleD3D9

This program demonstrates the interoperability between CUDA and Direct3D9. The program modifies vertex positions with CUDA and uses Direct3D9 to render the geometry. A Direct3D capable device is required.

simpleD3D9texture

This program demonstrates Direct3D9 texture interoperability with CUDA. The program creates a number of D3D9 textures (2D, 3D, and CubeMap) which are written to/from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D capable device is required.

simpleD3D10

This program demonstrates the interoperability between CUDA and Direct3D10. The program modifies vertex positions with CUDA and uses Direct3D10 to render the geometry. A Direct3D Capable device is required.

simpleD3D10Texture

This program demonstrates Direct3D10 texture interoperability with CUDA. The program creates a number of D3D10 textures (2D, 3D, and CubeMap) which are written to from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D Capable device is required.

simpleD3D11Texture

This program demonstrates Direct3D11 texture interoperability with CUDA. The program creates a number of D3D11 textures (2D, 3D, and CubeMap) which are written to from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D Capable device is required.

simpleGL

This program demonstrates interoperability between CUDA and OpenGL. The program modifies vertex positions with CUDA and uses OpenGL to render the geometry.

simpleTexture3D

This program demonstrates the use of 3D textures in CUDA.

Multi-GPU Programming

simpleP2P

This sample demonstrates the use of the new CUDA 4.0 API for multi-device programming with UVA (Unified Virtual Addressing) and GPU Direct 2.0 peer to peer communications (copying of data and memory addressing). This sample requires two GPUs with peer to peer capability.

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.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ROVI Compliance Statement

NVIDIA Products that support Rovi Corporation's Revision 7.1.L1 Anti-Copy Process (ACP) encoding technology can only be sold or distributed to buyers with a valid and existing authorization from ROVI to purchase and incorporate the device into buyer's products.

This device is protected by U.S. patent numbers 6,516,132; 5,583,936; 6,836,549; 7,050,698; and 7,492,896 and other intellectual property rights. The use of ROVI Corporation's copy protection technology in the device must be authorized by ROVI Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by ROVI Corporation. Reverse engineering or disassembly is prohibited.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, and CUDA are trademarks and/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