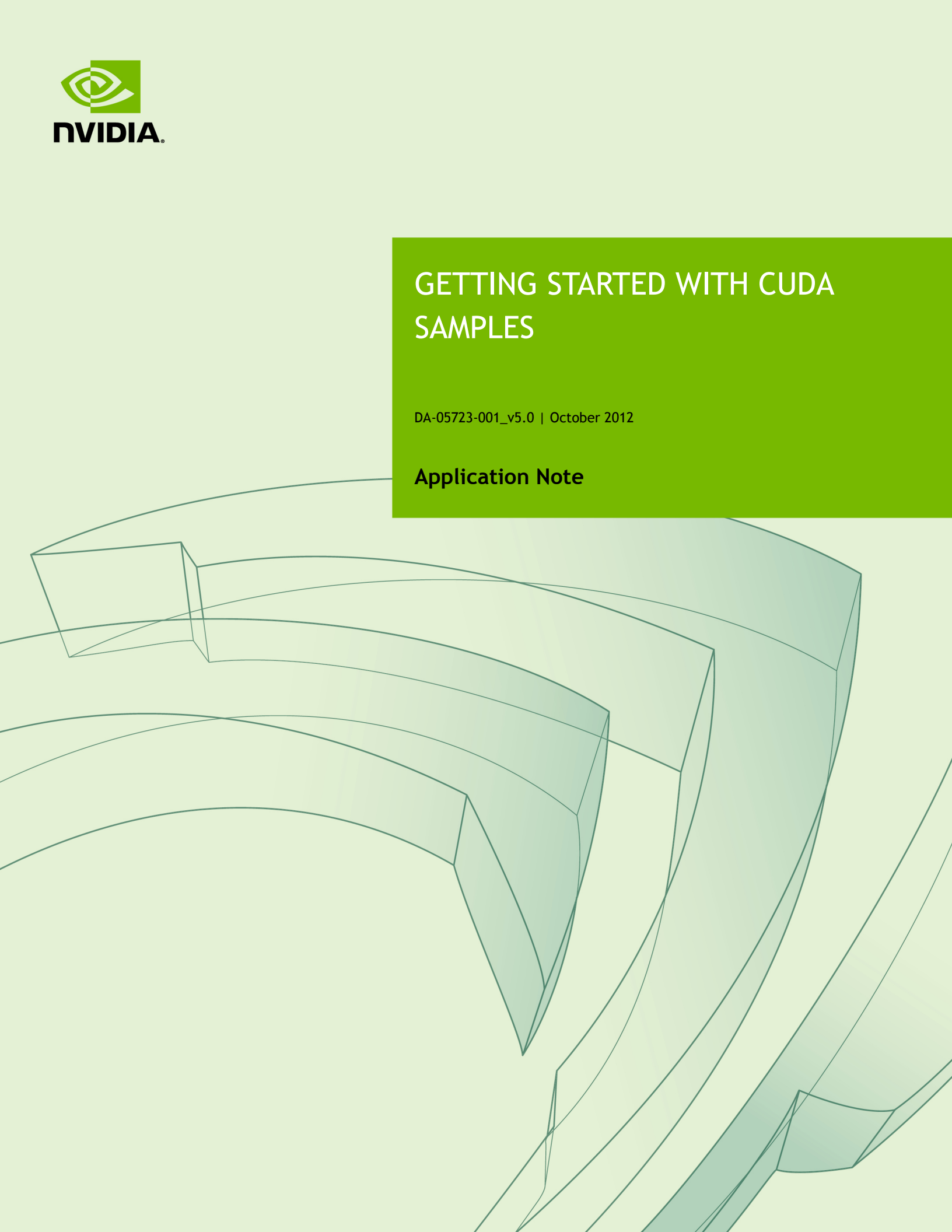

GETTING STARTED WITH CUDA SAMPLES

DA-05723-001_v5.0 | October 2012

Application Note

TABLE OF CONTENTS

Chapter 1. GETTING STARTED WITH CUDA SAMPLES

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).

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

To run these 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 samples listed in this document are found in both the C and `7_CUDALibraries` default directories in the following folders:

Windows

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

Linux

```
~/NVIDIA_CUDA-5.0_Samples
```

Mac OSX

```
/Developer/NVIDIA/CUDA-5.0/Samples
```

1.1 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

1.2 Getting Started

The list of samples is divided up into three categories:

- Getting started samples. If you are new to CUDA, these are the best samples to begin with.
- Simple CUDA samples

- Samples that demonstrate CUDA + Graphics interoperability

There are some overlaps between the three categories.

1.2.1 Getting Started Samples

matrixMul, matrixMulCUBLAS

This sample implements matrix multiplication as a CUDA kernel. It has been written for clarity of exposition to illustrate various CUDA programming principles, but not with the goal of providing the best performance kernel for matrix multiplication. For performance, this sample also uses the `CUBLAS` library to show 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.

1.2.2 Simple 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 installer.

deviceQuery, deviceQueryDrv

These two samples show how to enumerate properties of the CUDA devices present in the system (using the CUDA runtime API). The `*Drv` version has the same functions as the runtime sample, but 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 `7_CUDALibraries` folder.

For more details on how to use the CUBLAS Library, refer to the *CUBLAS Library 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 `7_CUDALibraries` folder.

For more details on how to use the `CUFFT` Library, refer to the *CUBLAS Library 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 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. Starting 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 *Portable Memory*, *Mapped Memory*, and *Multi-Device-System* 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 *Assertion* 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. Starting with CUDA 4.0, this sample adds support to pin of generic host memory. This sample requires Compute Capability 1.1 or higher.

For more details, refer to *Streams* 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 Memory Copy*. 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 *Parallel Libraries* in the *CUDA C Best Practices Guide* 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.

1.2.3 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.

1.2.4 Multi-GPU Programming

simpleP2P

This sample demonstrates how to use a new feature in 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 based on GF100 or GF110 or newer.