SDK CODE SAMPLE GUIDE TO NEW FEATURES IN CUDA TOOLKIT v4.0

DA-05689-001_v01 | May 2011

Application Note

OVERVIEW OF NEW FEATURES

NVIDIA® CUDA™ Toolkit version 4.0 introduces some exciting new features and capabilities. To illustrate the capabilities and advantages of the new features, the CUDA SDK includes many new and improved code samples. In addition, existing code samples have been upgraded to take advantage of the new features. This document serves as a guide to the new SDK code samples as they relate to the new CUDA Toolkit Version 4.0 feature list.

CUDA TOOLKIT 4.0 HIGHLIGHTS

The CUDA 4.0 Toolkit has the following new features:

- Easier application porting
  - Share GPUs across multiple threads
  - Single thread access to GPUs
  - No-copy pinning of system memory
  - New CUDA C/C++ language features
  - Thrust templated primitives library
  - NPP image/video processing library
  - Layered Textures
- Faster multi-GPU programming
  - Unified virtual addressing
  - GPUDirect v2.0 with peer-to-peer communication

- New and improved developer tools
    - Automated performance analysis
    - C++ debugging
    - Debugger cuda-gdb for MacOS
    - GPU binary disassemble

Code Samples for New Library Features

CUSPARSE

For additional information about CUSPARSE, please refer to the document CUSPARSE_Library.pdf included with the CUDA toolkit.

conjugateGradientPrecond (New!)

This example this shows a preconditioned conjugate gradient solver using CUBLAS and CUSPARSE libraries.

CUBLAS

For information about CUBLAS, please refer to the document CUBLAS_Library.pdf included with the CUDA toolkit.

matrixMul (Updated)

The sample shows how to perform matrix multiplication using CUDA and also CUBLAS. This sample illustrates how to implement a basic kernel and a CUDA Library version for matrix multiplication.

matrixMulDrv (Updated)

This sample shows how to perform matrix multiplication using the CUDA driver API. It has been updated to call the CUBLAS library function which provides high-performance matrix multiplication.. It demonstrates how to link to CUDA driver at runtime and how to use JIT (just-in-time) compilation from PTX code. It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performant generic kernel for matrix multiplication.

BatchCUBLAS (New!)

A sample to demonstrate batching CUBLAS GEMM calls to improve performance

simpleCUBLAS (Updated)

A simple sample showing how to use CUBLAS v2.0 API interface.

CURAND

For additional information about CURAND, please refer to the document CURAND_Library.pdf included with the CUDA toolkit.

randomFog (Updated)

This sample illustrates pseudo- and quasi- random numbers produced by CURAND.

EsimatePiQ (Updated)

This sample uses Monte Carlo simulation for Estimation of Pi (using batch QRNG). This sample also uses the NVIDIA CURAND library.

EstimatePiInlineQ (Updated)

This sample uses Monte Carlo simulation for Estimation of Pi (using batch inline QRNG). This sample also uses the NVIDIA CURAND library.

Thrust

For additional information about THRUST, please refer to the document Thrust_Quick_Start_Guide.pdf included with the CUDA toolkit.

With the release of CUDA 4.0, the Thrust library is now included as part of the CUDA Toolkit (http://code.google.com/p/thrust/). The following code samples illustrate how to use the Thrust library's templated primitives library:

radixSortThrust (Updated)

This sample demonstrates a very fast and efficient parallel radix sort uses Thrust library. The included RadixSort class can sort either by using key-value pairs (with float or unsigned integer keys) or keys only.

Particles (Updated)

This sample uses CUDA to simulate and visualize a large set of particles and their physical interaction. It implements a uniform grid data structure using either atomic operations and also uses the fast radix sort from the Thrust library

smokeParticles (Updated)

Smoke simulation with volumetric shadows using half-angle slicing technique. The procedural simulation is done with CUDA, Thrust Library for sorting algorithms, and OpenGL is used to render the graphics.

lineofSight (Updated)

This sample is an implementation of a simple line-of-sight algorithm: Given a height map and a ray originating at some observation point, it computes all the points along the ray that are visible from the observation point. The implementation is based on the Thrust library.

marchingCubes (Updated)

This sample extracts a geometric isosurface from a volume dataset using the marching cubes algorithm. It uses the scan (prefix sum) function from the Thrust library to perform stream compaction.

NVIDIA Performance Primitives (NPP)

Srarting with CUDA Toolkit 4.0, the NPP SDK is now integrated into the CUDA Toolkit. For additional information please refer to the NPP_Library.pdf API document included with the CUDA toolkit.

The following SDK code samples illustrate the usage of NPP.

imageSegmentationNPP

This NPP SDK sample demonstrates how to perform image segmentation using the NPP GraphCut function.

histEqualizationNPP

This SDK sample demonstrates how to use NPP for histogram equalization for image data.

FreeImageInteropNPP

A simple SDK sample demonstrate how to use FreeImage library with the NPP library.

BoxFilterNPP

This NPP SDK sample demonstrates how to use NPP FilterBox function to perform a Box Filter. A fast image box filter is implemented using CUDA with OpenGL rendering.

C++ and Virtual Functions

NewDelete (New!)

This sample demonstrates dynamic global memory allocation through device C++ new and delete operators and virtual function declarations available with CUDA 4.0.

CUDA Runtime (CUDART)

template_runtime (Updated)

This template has been updated to be self-contained in that it does not depend on or contain references to any SDK-specific libraries. This template project can be used as a starting point to create new CUDA Runtime API projects.

No-Copy Pinning of System Memory

simpleStreams (Updated)

This sample uses CUDA streams to overlap kernel executions with memory copies between the host and the device. It uses a new CUDA 4.0 feature that supports pinning of generic host memory. Requires Compute Capability 1.1 or higher.

For more details about using page-locked host memory, refer to section 3.2.4 and 3.3.6 in the CUDA C Programming Guide.

Multi-GPU Programming

MonteCarloMultiGPU (Updated)

This sample evaluates fair call price for a given set of European options using the Monte Carlo approach, taking advantage of all CUDA-capable GPUs installed in the system. This sample use double precision hardware if a GTX 200 class GPU is present. The sample also takes advantage of CUDA 4.0 capability to supporting using a single CPU thread to control multiple GPUs.

For more details, please refer to sections 3.2.4.1, 3.2.4.3, and 3.2.6 in the CUDA C Programming Guide about the new multi-device programming.

simpleMultiGPU (Updated)

CUDA 4.0 has two new features:

- - Improved multi-threaded access to CUDA contexts
- - New context management primitives

This application demonstrates how to use the new CUDA 4.0 API for CUDA context management and multi-threaded access to run CUDA kernels on multiple-GPUs.

For more details, please refer to sections 3.2.4.1, 3.2.4.3, and 3.2.6 in the CUDA C Programming Guide about the new multi-device programming.

simpleP2P (New!)

CUDA 4.0 has two new features:

- Unified virtual addressing
- GPUDirect v2.0 with peer-to-peer communication

This application demonstrates how to use 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).

For more details please refer to sections 3.2.6.4, 3.2.6.5, and 3.2.7 in the CUDA C Programming Guide.

n-Body (Updated)

This sample highlights the following two new CUDA 4.0 features:

- Single thread access to GPUs
- CUDA streams can be synchronized to events in other CUDA contexts

The nBody sample has been updated to use these new features to easily scale the n-body simulation across multiple GPUs in a single PC. Adding the "-numdevices=<N>" command line option will cause the sample to use N devices (if available) for simulation. In this mode, the position and velocity data for all bodies are read from system memory using "zero copy" rather than from device memory. For a small number of devices (4 or fewer) and a large enough number of bodies, bandwidth is not a bottleneck so we can achieve strong scaling across these devices.

Inline PTX

The NVIDIA® CUDA™ programming environment provides a parallel thread execution (PTX) instruction set architecture (ISA) for using the GPU as a data-parallel computing device. For more information on how to inline PTX assembly language statements into CUDA code refer to the application note ***Using PTX Assembly in CUDA.pdf*** and for more information on the PTX ISA refer to the PTX ISA reference document ***ptx_isa_[version].pdf***. Both documents are in the CUDA Toolkit **doc** folder.

inlinePTX (New!)

This SDK code sample is a simple test application that demonstrates the new CUDA 4.0 ability to embed PTX in a CUDA kernel.

Graphics Features

CUDA 4.0 has a new feature that enables support for layered textures or texture arrays. Texture arrays of size (16k x 16k) x 2k (layers) are supported with CUDA 4.0 and requires a GPU based on the NVIDIA® Fermi Architecture.

For more details please refer to section 3.2.10.1.5 in the CUDA C Programming Guide.