CUDA Samples

Reference Manual

TRM-06704-001_v11.6 | March 2022

Table of Contents

Chapter 1. Release Notes

This section describes the release notes for the CUDA Samples only. For the release notes for the whole CUDA Toolkit, please see CUDA Toolkit Release Notes.

1.1. CUDA 11.6

- All CUDA samples are now only available on GitHub repository. They are no longer available via CUDA toolkit.
- Added new folder structure for samples.
- Added Visual Studio 2022 support to all the samples.

1.2. CUDA 11.5

- All CUDA samples are now available on GitHub repository.
- Added `4_CUDA_Libraries/cuDLAHybridMode`. Demonstrate usage of cuDLA in hybrid mode. (available only on GitHub repository)
- Added `4_CUDA_Libraries/cuDLAStandaloneMode`. Demonstrate usage of cuDLA in standalone mode. (available only on GitHub repository)
- Added `4_CUDA_Libraries/cuDLAErrorReporting`. Demonstrate DLA error detection via CUDA. (available only on GitHub repository)
- Added `3_CUDA_Features/graphMemoryNodes`. Demonstrates memory allocations and frees within CUDA graphs using Graph APIs and Stream Capture APIs. (available only on GitHub repository)
- Added `3_CUDA_Features/graphMemoryFootprint`. Demonstrates how graph memory nodes re-use virtual addresses and physical memory. (available only on GitHub repository)

1.3. CUDA 11.4 Update 1

- Added support for VS Code on linux platform.

1.4. CUDA 11.4

- Added `7_CUDALibraries/simpleCUBLAS_LU`. Demonstrates batched matrix LU decomposition using cuBLAS API `cublas<t>getrfBatched()`.
- Updated `2_Graphics/simpleVulkan`, `2_Graphics/simpleVulkanMMAP` and `3_Imaging/vulkanImageCUDA`. Demonstrates use of SPIR-V shaders.
- Removed `7_CUDALibraries/boundSegmentsNPP`.

1.5. CUDA 11.3

- Added `0_Simple/streamOrderedAllocationIPC`. Demonstrates IPC pools of stream ordered memory allocated using cudaMallocAsync and cudaMemPool family of APIs.
- Updated `2_Graphics/simpleVulkan`. Demonstrates use of timeline semaphore.
- Updated `0_Simple/globalToShmemAsyncCopy` with a partitioned cuda pipeline producer-consumer GEMM kernel.
- Updated multiple samples to use pinned memory using `cudaMallocHost()`.

1.6. CUDA 11.2

- FreeImage is no longer distributed with the CUDA Samples. On Windows, see the Dependencies section for more details on how to set up FreeImage. On Linux, it is recommended to install FreeImage with your distribution's package manager.

1.7. CUDA 11.1

- Added `2_Graphics/simpleVulkanMMAP`. Demonstrates Vulkan CUDA Interop via cuMemMap APIs where CUDA buffer is imported in vulkan.
- Added `7_CUDALibraries/watershedSegmentationNPP`. Demonstrates how to use the NPP watershed segmentation function.
- Added `7_CUDALibraries/batchedLabelMarkersAndLabelCompressionNPP`. Demonstrates how to use the NPP label markers generation and label compression functions based on a Union Find (UF) algorithm including both single image and batched image versions.
- Deprecated Visual Studio 2015 support for all Windows supported samples.
- Dropped Visual Studio 2012, 2013 support from all the Windows supported samples.

1.8. CUDA 11.0

- Added `0_Simple/globalToShmemAsyncCopy`. Demonstrates asynchronous copy of data from global to shared memory using cuda pipeline. Also demonstrates arrive-wait barrier for synchronization.
- Added `0_Simple/simpleAttributes`. Demonstrates the stream attributes that affect L2 locality.
- Added `0_Simple/dmmaTensorCoreGemm`. Demonstrates double precision GEMM computation using the WMMA API for double precision employing the Tensor Cores. Also makes use of asynchronous copy from global to shared memory using cuda pipeline which leads to further performance gain.
- Added `0_Simple/bf16TensorCoreGemm`. Demonstrates __nv_bfloat16 (e8m7) GEMM computation using the WMMA API for __nv_bfloat16 employing the Tensor Cores. Also makes use of asynchronous copy from global to shared memory using cuda pipeline which leads to further performance gain.
- Added `0_Simple/tf32TensorCoreGemm`. Demonstrates tf32 (e8m10) GEMM computation using the WMMA API for tf32 employing the Tensor Cores. Also makes use of asynchronous copy from global to shared memory using cuda pipeline which leads to further performance gain.
- Added `0_Simple/simpleAWBarrier`. Demonstrates the arrive wait barriers.
- Added warp aggregated atomic multi bucket increments kernel using labeled_partition cooperative groups in `6_Advanced/warpAggregatedAtomicsCG` which can be used on compute capability 7.0 and above GPU architectures.
- Added `0_Simple/binaryPartitionCG`. Demonstrates binary_partition cooperative groups creation and usage in divergent path.
- Added `6_Advanced/cudaCompressibleMemory`. Demonstrates compressible memory allocation using cuMemMap API.
- Removed `7_CUDALibraries/nvgraph_Pagerank`, `7_CUDALibraries/nvgraph_SemiRingSpMV`, `7_CUDALibraries/nvgraph_SpectralClustering`, `7_CUDALibraries/nvgraph_SSSP` as the NVGRAPH library is dropped from CUDA Toolkit 11.0.
- Added two new reduction kernels in `6_Advanced/reduction` one which demonstrates reduce_add_sync intrinstic supported on compute capability 8.0 and another which uses cooperative_groups::reduce function which does thread_block_tile level reduction introduced from CUDA 11.0
- Added windows support to `6_Advanced/c++11_cuda`.

1.9. CUDA 10.2

- Added `6_Advanced/jacobiCudaGraphs`. Demonstrates Instantiated CUDA Graph Update usage.
- Added `0_Simple/memMapIPCDrv`. Demonstrates Inter Process Communication using cuMemMap APIs with one process per GPU for computation.
- Added `0_Simple/vectorAddMMAP`. Demonstrates how cuMemMap API allows the user to specify the physical properties of their memory while retaining the contiguous nature of their access, thus not requiring a change in their program structure.
- Added `0_Simple/simpleDrvRuntime`. Demonstrates how CUDA Driver and Runtime APIs can work together to load cuda fatbinary of vector add kernel.
- Added `0_Simple/cudaNvSci`. Demonstrates CUDA-NvSciBuf/NvSciSync Interop.

1.10. CUDA 10.1 Update 2

- Added `3_Imaging/vulkanImageCUDA`. Demonstrates how to perform Vulkan Image-CUDA Interop.
- Added `7_CUDALibraries/nvJPEG_encoder`. Demonstrates encoding of jpeg images using NVJPEG Library.
- Added Windows support to `7_CUDALibraries/nvJPEG`.
- Removed DirectX SDK (June 2010 or newer) installation requirement, all the DirectX-CUDA samples now use DirectX from Windows SDK shipped with Microsoft Visual Studio 2012 or higher

1.11. CUDA 10.1 Update 1

- Added `3_Imaging/NV12toBGRandResize`. Demonstrates how to convert and resize NV12 frames to BGR planars frames using CUDA in batch.
- Added Visual Studio 2019 support to all the samples.

1.12. CUDA 10.1

- Added `0_Simple/immaTensorCoreGemm`. Demonstrates integer GEMM computation using the Warp Matrix Multiply and Accumulate (WMMA) API for integers employing the Tensor Cores.
- Added `2_Graphics/simpleD3D12`. Demonstrates Direct3D12 interoperability with CUDA.

- Added `7_CUDALibraries/nvJPEG`. Demonstrates single and batched decoding of jpeg images using NVJPEG Library.
- Added `7_CUDALibraries/conjugateGradientCudaGraphs`. Demonstrates conjugate gradient solver on GPU using CUBLAS/CUSPARSE library calls captured and called using CUDA Graph APIs.
- Updated `0_Simple/simpleIPC` to work on Windows OS as well with TCC enabled GPUs.

1.13. CUDA 10.0

- Added `1_Utilities/UnifiedMemoryPerf`. Demonstrates the performance comparision of Unified Memory and other types of memory like zero copy buffers, pageable, pagelocked memory on a single GPU.
- Added `2_Graphics/simpleVulkan`. Demonstrates the Vulkan-CUDA Interop. CUDA imports the Vulkan vertex buffer and operates on it to create sinewave, and synchronizes with Vulkan through vulkan semaphores imported by CUDA.
- Added `0_Simple/simpleCudaGraphs`. Demonstrates how to use CUDA Graphs through Graphs APIs and Stream Capture APIs.
- Removed `3_Imaging/cudaDecodeGL`, `3_Imaging/cudaDecodeD3D9` as the cuvid library is dropped from CUDA Toolkit 10.0.
- Removed `6_Advanced/cdpLUDecomposition`, `7_CUDALibraries/simpleDevLibCUBLAS` as the CUBLAS Device library is dropped from CUDA Toolkit 10.0.

1.14. CUDA 9.2

- Added `7_CUDALibraries/boundSegmentsNPP`. Demonstrates nppiLabelMarkers to generate connected region segment labels.
- Added `6_Advanced/conjugateGradientMultiDeviceCG`. Demonstrates a conjugate gradient solver on multiple GPUs using Multi Device Cooperative Groups, also uses Unified Memory optimized using prefetching and usage hints.
- Updated `0_Simple/fp16ScalarProduct` to use fp16 native operators for half2 and other fp16 features, it also compare results of using native vs intrinsics fp16 operations.

1.15. CUDA 9.0

- Added `7_CUDALibraries/nvgraph_SpectralClustering`. Demonstrates Spectral Clustering using NVGRAPH Library.
- Added `6_Advanced/warpAggregatedAtomicsCG`. Demonstrates warp aggregated atomics using Cooperative Groups.

- Added `6_Advanced/reductionMultiBlockCG`. Demonstrates single pass reduction using Multi Block Cooperative Groups.
- Added `6_Advanced/conjugateGradientMultiBlockCG`. Demonstrates a conjugate gradient solver on GPU using Multi Block Cooperative Groups.
- Added Cooperative Groups(CG) support to several samples notable ones to name are `6_Advanced/cdpQuadtree`, `6_Advanced/cdpAdvancedQuicksort`, `6_Advanced/threadFenceReduction`, `3_Imaging/dxtc`, `4_Finance/MonteCarloMultiGPU`, `0_Simple/matrixMul_nvrtc`.
- Added `0_Simple/simpleCooperativeGroups`. Illustrates basic usage of Cooperative Groups within the thread block.
- Added `0_Simple/cudaTensorCoreGemm`. Demonstrates a GEMM computation using the Warp Matrix Multiply and Accumulate (WMMA) API introduced in CUDA 9, as well as the new Tensor Cores introduced in the Volta chip family.
- Updated `0_Simple/simpleVoteIntrinsics` to use newly added *_sync equivalent of the vote intrinsics _any, _all.
- Updated `6_Advanced/shfl_scan` to use newly added *_sync equivalent of the shfl intrinsics.

1.16. CUDA 8.0

- Added `7_CUDALibraries/FilterBorderControlNPP`. Demonstrates how any border version of an NPP filtering function can be used in the most common mode (with border control enabled), can be used to duplicate the results of the equivalent non-border version of the NPP function, and can be used to enable and disable border control on various source image edges depending on what portion of the source image is being used as input.
- Added `7_CUDALibraries/cannyEdgeDetectorNPP`. Demonstrates the recommended parameters to use with the nppiFilterCannyBorder_8u_C1R Canny Edge Detection image filter function. This function expects a single channel 8-bit grayscale input image. You can generate a grayscale image from a color image by first calling nppiColorToGray() or nppiRGBToGray(). The Canny Edge Detection function combines and improves on the techniques required to produce an edge detection image using multiple steps.
- Added `7_CUDALibraries/cuSolverSp_LowlevelCholesky`. Demonstrates Cholesky factorization using cuSolverSP's low level APIs.
- Added `7_CUDALibraries/cuSolverSp_LowlevelQR`. Demonstrates QR factorization using cuSolverSP's low level APIs.
- Added `7_CUDALibraries/BiCGStab`. Demonstrates Bi-Conjugate Gradient Stabilized (BiCGStab) iterative method for nonsymmetric and symmetric positive definite linear systems using CUSPARSE and CUBLAS
- Added `7_CUDALibraries/nvgraph_Pagerank`. Demonstrates Page Rank computation using nvGRAPH Library.

- Added `7_CUDALibraries/nvgraph_SemiRingSpMV`. Demonstrates Semi-Ring SpMV using nvGRAPH Library.
- Added `7_CUDALibraries/nvgraph_SSSP`. Demonstrates Single Source Shortest Path(SSSP) computation using nvGRAPH Library.
- Added `7_CUDALibraries/simpleCUBLASXT`. Demonstrates simple example to use CUBLAS-XT library.
- Added `6_Advanced/c++11_cuda`. Demonstrates C++11 feature support in CUDA.
- Added `1_Utilities/topologyQuery`. Demonstrates how to query the topology of a system with multiple GPU.
- Added `0_Simple/fp16ScalarProduct`. Demonstrates scalar product calculation of two vectors of FP16 numbers.
- Added `0_Simple/systemWideAtomics`. Demonstrates system wide atomic instructions on migratable memory.
- Removed `0_Simple/template_runtime`. Its purpose is served by `0_Simple/template`.

1.17. CUDA 7.5

- Added `7_CUDALibraries/cuSolverDn_LinearSolver`. Demonstrates how to use the CUSOLVER library for performing dense matrix factorization using cuSolverDN's LU, QR and Cholesky factorization functions.
- Added `7_CUDALibraries/cuSolverRf`. Demonstrates how to use cuSolverRF, a sparse re-factorization package of the CUSOLVER library.
- Added `7_CUDALibraries/cuSolverSp_LinearSolver`. Demonstrates how to use cuSolverSP which provides sparse set of routines for sparse matrix factorization.
- The `2_Graphics/simpleD3D9`, `2_Graphics/simpleD3D9Texture`, `3_Imaging/cudaDecodeD3D9`, and `5_Simulations/fluidsD3D9` samples have been modified to use the Direct3D 9Ex API instead of the Direct3D 9 API.
- The `7_CUDALibraries/grabcutNPP` and `7_CUDALibraries/imageSegmentationNPP` samples have been removed. These samples used the NPP graphcut APIs, which have been deprecated in CUDA 7.5.

1.18. CUDA 7.0

- Removed support for Windows 32-bit builds.
- The Makefile x86_64=1 and ARMv7=1 options have been deprecated. Please use TARGET_ARCH to set the targeted build architecture instead.
- The Makefile GCC option has been deprecated. Please use HOST_COMPILER to set the host compiler instead.

- The CUDA Samples are no longer shipped as prebuilt binaries on Windows. Please use VS Solution files provided to build respective executable.
- Added `0_Simple/clock_nvrtc`. Demonstrates how to compile clock function kernel at runtime using libNVRTC to measure the performance of kernel accurately.
- Added `0_Simple/inlinePTX_nvrtc`. Demonstrates compilation of CUDA kernel having PTX embedded at runtime using libNVRTC.
- Added `0_Simple/matrixMul_nvrtc`. Demonstrates compilation of matrix multiplication CUDA kernel at runtime using libNVRTC.
- Added `0_Simple/simpleAssert_nvrtc`. Demonstrates compilation of CUDA kernel having assert() at runtime using libNVRTC.
- Added `0_Simple/simpleAtomicIntrinsics_nvrtc`. Demonstrates compilation of CUDA kernel performing atomic operations at runtime using libNVRTC.
- Added `0_Simple/simpleTemplates_nvrtc`. Demonstrates compilation of templatized dynamically allocated shared memory arrays CUDA kernel at runtime using libNVRTC.
- Added `0_Simple/simpleVoteIntrinsics_nvrtc`. Demonstrates compilation of CUDA kernel which uses vote intrinsics at runtime using libNVRTC.
- Added `0_Simple/vectorAdd_nvrtc`. Demonstrates compilation of CUDA kernel performing vector addition at runtime using libNVRTC.
- Added `4_Finance/binomialOptions_nvrtc`. Demonstrates runtime compilation using libNVRTC of CUDA kernel which evaluates fair call price for a given set of European options under binomial model.
- Added `4_Finance/BlackScholes_nvrtc`. Demonstrates runtime compilation using libNVRTC of CUDA kernel which evaluates fair call and put prices for a given set of European options by Black-Scholes formula.
- Added `4_Finance/quasirandomGenerator_nvrtc`. Demonstrates runtime compilation using libNVRTC of CUDA kernel which implements Niederreiter Quasirandom Sequence Generator and Inverse Cumulative Normal Distribution functions for the generation of Standard Normal Distributions.

1.19. CUDA 6.5

- Added `7_CUDALibraries/cuHook`. Demonstrates how to build and use an intercept library with CUDA.
- Added `7_CUDALibraries/simpleCUFFT_callback`. Demonstrates how to compute a 1D-convolution of a signal with a filter using a user-supplied CUFFT callback routine, rather than a separate kernel call.
- Added `7_CUDALibraries/simpleCUFFT_MGPU`. Demonstrates how to compute a 1D-convolution of a signal with a filter by transforming both into frequency domain, multiplying them together, and transforming the signal back to time domain on Multiple GPUs.

- Added `7_CUDALibraries/simpleCUFFT_2d_MGPU`. Demonstrates how to compute a 2D-convolution of a signal with a filter by transforming both into frequency domain, multiplying them together, and transforming the signal back to time domain on Multiple GPUs.
- Removed `3_Imaging/cudaEncode`. Support for the CUDA Video Encoder (NVCUVENC) has been removed.
- Removed `4_Finance/ExcelCUDA2007`. The topic will be covered in a blog post at Parallel Forall.
- Removed `4_Finance/ExcelCUDA2010`. The topic will be covered in a blog post at Parallel Forall.
- The `4_Finance/binomialOptions` sample is now restricted to running on GPUs with SM architecture 2.0 or greater.
- The `4_Finance/quasirandomGenerator` sample is now restricted to running on GPUs with SM architecture 2.0 or greater.
- The `7_CUDALibraries/boxFilterNPP` sample now demonstrates how to use the static NPP libraries on Linux and Mac.
- The `7_CUDALibraries/conjugateGradient` sample now demonstrates how to use the static CUBLAS and CUSPARSE libraries on Linux and Mac.
- The `7_CUDALibraries/MersenneTwisterGP11213` sample now demonstrates how to use the static CURAND library on Linux and Mac.

1.20. CUDA 6.0

- New featured samples that support a new CUDA 6.0 feature called UVM-Lite
- Added `0_Simple/UnifiedMemoryStreams` - new CUDA sample that demonstrates the use of OpenMP and CUDA streams with Unified Memory on a single GPU.
- Added `1_Utilities/p2pBandwidthTestLatency` - new CUDA sample that demonstrates how measure latency between pairs of GPUs with P2P enabled and P2P disabled.
- Added `6_Advanced/StreamPriorities` - This sample demonstrates basic use of the new CUDA 6.0 feature stream priorities.
- Added `7_CUDALibraries/ConjugateGradientUM` - This sample implements a conjugate gradient solver on GPU using cuBLAS and cuSPARSE library, using Unified Memory.

1.21. CUDA 5.5

- Linux makefiles have been updated to generate code for the AMRv7 architecture. Only the ARM hard-float floating point ABI is supported. Both native ARMv7 compilation and cross compilation from x86 is supported
- Performance improvements in CUDA toolkit for Kepler GPUs (SM 3.0 and SM 3.5)

- Makefiles projects have been updated to properly find search default paths for OpenGL, CUDA, MPI, and OpenMP libraries for all OS Platforms (Mac, Linux x86, Linux ARM).
- Linux and Mac project Makefiles now invoke NVCC for building and linking projects.
- Added `0_Simple/cppOverload` - new CUDA sample that demonstrates how to use C++ overloading with CUDA.
- Added `6_Advanced/cdpBezierTessellation` - new CUDA sample that demonstrates an advanced method of implementing Bezier Line Tessellation using CUDA Dynamic Parallelism. Requires compute capability 3.5 or higher.
- Added `7_CUDALibrariess/jpegNPP` - new CUDA sample that demonstrates how to use NPP for JPEG compression on the GPU.
- CUDA Samples now have better integration with Nsight Eclipse IDE.
- `6_Advanced/ptxjit` sample now includes a new API to demonstrate PTX linking at the driver level.

1.22. CUDA 5.0

- New directory structure for CUDA samples. Samples are classified accordingly to categories: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, and `7_CUDALibraries`
- Added `0_Simple/simpleIPC` - CUDA Runtime API sample is a very basic sample that demonstrates Inter Process Communication with one process per GPU for computation. Requires Compute Capability 2.0 or higher and a Linux Operating System.
- Added `0_Simple/simpleSeparateCompilation` - demonstrates a CUDA 5.0 feature, the ability to create a GPU device static library and use it within another CUDA kernel. This example demonstrates how to pass in a GPU device function (from the GPU device static library) as a function pointer to be called. Requires Compute Capability 2.0 or higher.
- Added `2_Graphics/bindlessTexture` - demonstrates use of `cudaSurfaceObject`, `cudaTextureObject`, and MipMap support in CUDA. Requires Compute Capability 3.0 or higher.
- Added `3_Imaging/stereoDisparity` - demonstrates how to compute a stereo disparity map using SIMD SAD (Sum of Absolute Difference) intrinsics. Requires Compute Capability 2.0 or higher.
- Added `0_Simple/cdpSimpleQuicksort` - demonstrates a simple quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.
- Added `0_Simple/cdpSimplePrint` - demonstrates simple printf implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- Added `6_Advanced/cdpLUDecomposition` - demonstrates LU Decomposition implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.
- Added `6_Advanced/cdpAdvancedQuicksort` - demonstrates an advanced quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.
- Added `6_Advanced/cdpQuadtree` - demonstrates Quad Trees implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.
- Added `7_CUDALibraries/simpleDevLibCUBLAS` - implements a simple cuBLAS function calls that call GPU device API library running cuBLAS functions. cuBLAS device code functions take advantage of CUDA Dynamic Parallelism and requires compute capability of 3.5 or higher.

1.23. CUDA 4.2

- Added `segmentationTreeThrust` - demonstrates a method to build image segmentation trees using Thrust. This algorithm is based on Boruvka's MST algorithm.

1.24. CUDA 4.1

- Added `MersenneTwisterGP11213` - implements Mersenne Twister GP11213, a pseudorandom number generator using the `cuRAND` library.
- Added `HSOpticalFlow` - When working with image sequences or video it's often useful to have information about objects movement. Optical flow describes apparent motion of objects in image sequence. This sample is a Horn-Schunck method for optical flow written using CUDA.
- Added `volumeFiltering` - demonstrates basic volume rendering and filtering using 3D textures.
- Added `simpleCubeMapTexture` - demonstrates how to use `texcubemap` fetch instruction in a CUDA C program.
- Added `simpleAssert` - demonstrates how to use GPU assert in a CUDA C program.
- Added `grabcutNPP` - CUDA implementation of Rother et al. GrabCut approach using the 8 neighborhood `NPP` Graphcut primitive introduced in CUDA 4.1. (C. Rother, V. Kolmogorov, A. Blake. *GrabCut: Interactive Foreground Extraction Using Iterated Graph Cuts. ACM Transactions on Graphics (SIGGRAPH'04)*, 2004).

Chapter 2. Getting Started

The CUDA Samples are an educational resource provided to teach CUDA programming concepts. The CUDA Samples are not meant to be used for performance measurements.

For system requirements and installation instructions, please refer to the Linux Installation Guide and the Windows Installation Guide.

2.1. Getting CUDA Samples

Windows

On Windows, the CUDA Samples are installed using the CUDA Toolkit Windows Installer. By default, the CUDA Samples are installed in:

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\
```

The installation location can be changed at installation time.

Linux

On Linux, to install the CUDA Samples, the CUDA toolkit must first be installed. See the Linux Installation Guide for more information on how to install the CUDA Toolkit.

Then the CUDA Samples can be installed by running the following command, where <target_path> is the location where to install the samples:

```
$ cuda-install-samples-11.6.sh <target_path>
```

2.2. Building Samples

Windows

The Windows samples are built using the Visual Studio IDE. Solution files (.sln) are provided for each supported version of Visual Studio, using the format:

```
*_vs<version>.sln - for Visual Studio <version>
```

Complete samples solution files exist at:

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\
```

Each individual sample has its own set of solution files at:

```
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\<sample_dir>\
```

To build/examine all the samples at once, the complete solution files should be used. To build/examine a single sample, the individual sample solution files should be used.

Linux

The Linux samples are built using makefiles. To use the makefiles, change the current directory to the sample directory you wish to build, and run `make`:

```
$ cd <sample_dir>
$ make
```

The samples makefiles can take advantage of certain options:

- **TARGET_ARCH=<arch>** - cross-compile targeting a specific architecture. Allowed architectures are x86_64, armv7l, aarch64, sbsa, and ppc64le.

  By default, TARGET_ARCH is set to HOST_ARCH. On a x86_64 machine, not setting TARGET_ARCH is the equvalent of setting TARGET_ARCH=x86_64.

  ```
  $ make TARGET_ARCH=x86_64
  $ make TARGET_ARCH=armv7l
  $ make TARGET_ARCH=aarch64
  $ make TARGET_ARCH=sbsa
  $ make TARGET_ARCH=ppc64le
  ```

  See here for more details.

- **dbg=1** - build with debug symbols

  ```
  $ make dbg=1
  ```

- **SMS="A B ..."** - override the SM architectures for which the sample will be built, where `"A B ..."` is a space-delimited list of SM architectures. For example, to generate SASS for SM 35 and SM 50, use `SMS="35 50"`.

  ```
  $ make SMS="35 50"
  ```

- **HOST_COMPILER=<host_compiler>** - override the default g++ host compiler. See the Linux Installation Guide for a list of supported host compilers.

  ```
  $ make HOST_COMPILER=g++
  ```

2.3. CUDA Cross-Platform Samples

CUDA Samples are now located in https://github.com/nvidia/cuda-samples, which includes instructions for obtaining, building, and running the samples.

2.4. Using CUDA Samples to Create Your Own CUDA Projects

2.4.1. Creating CUDA Projects for Windows

Creating a new CUDA Program using the CUDA Samples infrastructure is easy. We have provided a `template` project that you can copy and modify to suit your needs. Just follow these steps:

(`<category>` refers to one of the following folders: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, `7_CUDALibraries`.)

1. Copy the content of:

   ```
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\<category>\template
   ```

   to a directory of your own:

   ```
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\<category>\myproject
   ```
2. Edit the filenames of the project to suit your needs.
3. Edit the `*.sln`, `*.vcproj` and source files.
   Just search and replace all occurrences of `template` with `myproject`.
4. Build the 64-bit, release or debug configurations using:

   ```
   myproject_vs<version>.sln
   ```
5. Run `myproject.exe` from the `release` or `debug` directories located in:

   ```
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v11.6\bin\win64\[release|debug]
   ```
6. Now modify the code to perform the computation you require.
   See the *CUDA Programming Guide* for details of programming in CUDA.

2.4.2. Creating CUDA Projects for Linux

Note: The default installation folder `<SAMPLES_INSTALL_PATH>` is `NVIDIA_CUDA_11.6_Samples` and <category> is one of the following: `0_Simple`, `1_Utilities`, `2_Graphics`, `3_Imaging`, `4_Finance`, `5_Simulations`, `6_Advanced`, `7_CUDALibraries`.

Creating a new CUDA Program using the NVIDIA CUDA Samples infrastructure is easy. We have provided a `template` project that you can copy and modify to suit your needs. Just follow these steps:

1. Copy the `template` project:

   ```
   cd <SAMPLES_INSTALL_PATH>/<category>
   cp -r template <myproject>
   ```

```
cd <SAMPLES_INSTALL_PATH>/<category>
```

2. Edit the filenames of the project to suit your needs:

```
mv template.cu myproject.cu
mv template_cpu.cpp myproject_cpu.cpp
```

3. Edit the `Makefile` and source files.

   Just search and replace all occurrences of `template` with `myproject`.

4. Build the project as (release):

```
make
```

   To build the project as (debug), use "make dbg=1":

```
make dbg=1
```

5. Run the program:

```
../../bin/x86_64/linux/release/myproject
```

6. Now modify the code to perform the computation you require.

   See the *CUDA Programming Guide* for details of programming in CUDA.

Chapter 3. Samples Reference

This document contains a complete listing of the code samples that are included with the NVIDIA CUDA Toolkit. It describes each code sample, lists the minimum GPU specification, and provides links to the source code and white papers if available.

The code samples are divided into the following categories:

Introduction Reference
: Basic CUDA samples for beginners that illustrate key concepts with using CUDA and CUDA runtime APIs.

Utilities Reference
: Utility samples that demonstrate how to query device capabilities and measure GPU/CPU bandwidth.

Concepts and Techniques Reference
: Samples that demonstrate CUDA related concepts and common problem solving techniques.

CUDA Features Reference
: Samples that demonstrate CUDA Features.

CUDA Libraries Reference
: Samples that demonstrate how to use CUDA platform libraries (NPP, NVJPEG, NVGRAPH cuBLAS, cuFFT, cuSPARSE, cuSOLVER and cuRAND).

Domain Specific Reference
: Samples that are specific to domain (Graphics, Finance, Image Processing).

Performance Reference
: Samples that demonstrate performance optimization.

3.1. Introduction Reference

3.2. Utilities Reference

3.3. Concepts and Techniques Reference

3.4. CUDA Features Reference

3.5. CUDA Libraries Reference

3.6. Domain Specific Reference

3.7. Performance Reference

Chapter 4. Dependencies

Some CUDA Samples rely on third-party applications and/or libraries, or features provided by the CUDA Toolkit and Driver, to either build or execute. These dependencies are listed below.

If a sample has a dependency that is not available on the system, the sample will not be installed. If a sample has a third-party dependency that is available on the system, but is not installed, the sample will waive itself at build time.

Each sample's dependencies are listed in the Samples Reference section.

Third-Party Dependencies

These third-party dependencies are required by some CUDA samples. If available, these dependencies are either installed on your system automatically, or are installable via your system's package manager (Linux) or a third-party website.

FreeImage

FreeImage is an open source imaging library. FreeImage can usually be installed on Linux using your distribution's package manager system. FreeImage can also be downloaded from the FreeImage website.

To set up FreeImage on a Windows system, extract the FreeImage DLL distribution into the `7_CUDALibraries/common/` folder such that `7_CUDALibraries/common/FreeImage/Dist/x64/` contains the `.h`, `.dll`, and `.lib` files.

Message Passing Interface

MPI (Message Passing Interface) is an API for communicating data between distributed processes. A MPI compiler can be installed using your Linux distribution's package manager system. It is also available on some online resources, such as Open MPI. On Windows, to build and run MPI-CUDA applications one can install MS-MPI SDK.

Only 64-Bit

Some samples can only be run on a 64-bit operating system.

DirectX

DirectX is a collection of APIs designed to allow development of multimedia applications on Microsoft platforms. For Microsoft platforms, NVIDIA's CUDA Driver supports DirectX. Several CUDA Samples for Windows demonstrates CUDA-DirectX Interoperability, for building such samples one needs to install Microsoft Visual Studio 2012 or higher which provides Microsoft Windows SDK for Windows 8.

DirectX 12

DirectX 12 is a collection of advanced low-level programming APIs which can reduce driver overhead, designed to allow development of multimedia applications on Microsoft platforms starting with Windows 10 OS onwards. For Microsoft platforms, NVIDIA's CUDA Driver supports DirectX. Few CUDA Samples for Windows demonstrates CUDA-DirectX12 Interoperability, for building such samples one needs to install Windows 10 SDK or higher , with VS 2015 or VS 2017.

OpenGL

OpenGL is a graphics library used for 2D and 3D rendering. On systems which support OpenGL, NVIDIA's OpenGL implementation is provided with the CUDA Driver.

OpenGL ES

OpenGL ES is an embedded systems graphics library used for 2D and 3D rendering. On systems which support OpenGL ES, NVIDIA's OpenGL ES implementation is provided with the CUDA Driver.

Vulkan

Vulkan is a low-overhead, cross-platform 3D graphics and compute API. Vulkan targets high-performance realtime 3D graphics applications such as video games and interactive media across all platforms. On systems which support Vulkan, NVIDIA's Vulkan implementation is provided with the CUDA Driver. For building and running Vulkan applications one needs to install the Vulkan SDK.

OpenMP

OpenMP is an API for multiprocessing programming. OpenMP can be installed using your Linux distribution's package manager system. It usually comes preinstalled with GCC. It can also be found at the OpenMP website.

Screen

Screen is a windowing system found on the QNX operating system. Screen is usually found as part of the root filesystem.

X11

X11 is a windowing system commonly found on *-nix style operating systems. X11 can be installed using your Linux distribution's package manager, and comes preinstalled on Mac OS X systems.

EGL

EGL is an interface between Khronos rendering APIs (such as OpenGL, OpenGL ES or OpenVG) and the underlying native platform windowing system.

EGLOutput

EGLOutput is a set of EGL extensions which allow EGL to render directly to the display.

EGLSync

EGLSync is a set of EGL extensions which provides sync objects that are synchronization primitive, representing events whose completion can be tested or waited upon.

NVSCI

NvSci is a set of communication interface libraries out of which CUDA interops with NvSciBuf and NvSciSync. NvSciBuf allows applications to allocate and exchange buffers in memory. NvSciSync allows applications to manage synchronization objects which coordinate when sequences of operations begin and end.

NvMedia

NvMedia provides powerful processing of multimedia data for true hardware acceleration across NVIDIA Tegra devices. Applications leverage the NvMedia Application Programming Interface (API) to process the image and video data.

CUDA Features

These CUDA features are needed by some CUDA samples. They are provided by either the CUDA Toolkit or CUDA Driver. Some features may not be available on your system.

CUFFT Callback Routines

CUFFT Callback Routines are user-supplied kernel routines that CUFFT will call when loading or storing data. These callback routines are only available on Linux x86_64 and ppc64le systems.

CUDA Dynamic Paralellism

CDP (CUDA Dynamic Paralellism) allows kernels to be launched from threads running on the GPU. CDP is only available on GPUs with SM architecture of 3.5 or above.

Multi-block Cooperative Groups

Multi Block Cooperative Groups(MBCG) extends Cooperative Groups and the CUDA programming model to express inter-thread-block synchronization. MBCG is available on GPUs with Pascal and higher architecture.

Multi-Device Cooperative Groups

Multi Device Cooperative Groups extends Cooperative Groups and the CUDA programming model enabling thread blocks executing on multiple GPUs to cooperate and synchronize as they execute. This feature is available on GPUs with Pascal and higher architecture.

CUBLAS

CUBLAS (CUDA Basic Linear Algebra Subroutines) is a GPU-accelerated version of the BLAS library.

CUDA Interprocess Communication

IPC (Interprocess Communication) allows processes to share device pointers.

CUFFT

CUFFT (CUDA Fast Fourier Transform) is a GPU-accelerated FFT library.

CURAND

CURAND (CUDA Random Number Generation) is a GPU-accelerated RNG library.

CUSPARSE

CUSPARSE (CUDA Sparse Matrix) provides linear algebra subroutines used for sparse matrix calculations.

CUSOLVER

CUSOLVER library is a high-level package based on the CUBLAS and CUSPARSE libraries. It combines three separate libraries under a single umbrella, each of which can be used independently or in concert with other toolkit libraries. The intent ofCUSOLVER is to provide useful LAPACK-like features, such as common matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver. In

addition cuSolver provides a new refactorization library useful for solving sequences of matrices with a shared sparsity pattern.

NPP

NPP (NVIDIA Performance Primitives) provides GPU-accelerated image, video, and signal processing functions.

NVJPEG

NVJPEG library provides high-performance, GPU accelerated JPEG decoding functionality for image formats commonly used in deep learning and hyperscale multimedia applications.

NVRTC

NVRTC (CUDA RunTime Compilation) is a runtime compilation library for CUDA C++.

Stream Priorities

Stream Priorities allows the creation of streams with specified priorities. Stream Priorities is only available on GPUs with SM architecture of 3.5 or above.

Unified Virtual Memory

UVM (Unified Virtual Memory) enables memory that can be accessed by both the CPU and GPU without explicit copying between the two. UVM is only available on Linux and Windows systems.

16-bit Floating Point

FP16 is a 16-bit floating-point format. One bit is used for the sign, five bits for the exponent, and ten bits for the mantissa. FP16 is only available on specific mobile platforms.

C++11 CUDA

NVCC Support of C++11 features.

Chapter 7. Frequently Asked Questions

Answers to frequently asked questions about CUDA can be found at http://developer.nvidia.com/cuda-faq and in the CUDA Toolkit Release Notes.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

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

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

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com