NVIDIA CUDA TOOLKIT V7.0

RN-06722-001 _v7.0 | August 2014

Release Notes for Windows, Linux, and Mac OS

TABLE OF CONTENTS

Chapter 1. CUDA TOOLKIT MAJOR COMPONENTS

This section provides an overview of the major components of the CUDA Toolkit and points to their locations after installation.

Compiler

The CUDA-C and CUDA-C++ compiler, **`nvcc`**, is found in the **`bin/`** directory. It is built on top of the NVVM optimizer, which is itself built on top of the LLVM compiler infrastructure. Developers who want to target NVVM directly can do so using the Compiler SDK, which is available in the **`nvvm/`** directory.

Tools

The following development tools are available in the **`bin/`** directory (except for Nsight Visual Studio Edition (VSE) which is installed as a plug-in to Microsoft Visual Studio).

- IDEs: **`nsight`** (Linux, Mac), Nsight VSE (Windows)
- Debuggers: **`cuda-memcheck`**, **`cuda-gdb`** (Linux, Mac), Nsight VSE (Windows)
- Profilers: **`nvprof`**, **`nvvp`**, Nsight VSE (Windows)
- Utilities: **`cuobjdump`**, **`nvdisasm`**

Libraries

The scientific and utility libraries listed below are available in the **`lib/`** directory (DLLs on Windows are in **`bin/`**), and their interfaces are available in the **`include/`** directory.

- **`cublas`** (BLAS)
- **`cublas_device`** (BLAS Kernel Interface)
- **`cuda_occupancy`** (Kernel Occupancy Calculation [header file implementation])
- **`cudadevrt`** (CUDA Device Runtime)
- **`cudart`** (CUDA Runtime)
- **`cufft`** (Fast Fourier Transform [FFT])
- **`cupti`** (Profiling Tools Interface)
- **`curand`** (Random Number Generation)
- **`cusolver`** (Dense and Sparse Direct Linear Solvers and Eigen Solvers)
- **`cusparse`** (Sparse Matrix)
- **`npp`** (NVIDIA Performance Primitives [image and signal processing])
- **`nvblas`** ("Drop-in" BLAS)

- **nvcuvid** (CUDA Video Decoder [Windows, Linux])
- **thrust** (Parallel Algorithm Library [header file implementation])

CUDA Samples

Code samples that illustrate how to use various CUDA and library APIs are available in the **samples/** directory on Linux and Mac, and are installed to **C:\ProgramData\NVIDIA Corporation\CUDA Samples** on Windows. On Linux and Mac, the **samples/** directory is read-only and the samples must be copied to another location if they are to be modified. Further instructions can be found in the *Getting Started Guides* for Linux and Mac.

Documentation

The most current version of these release notes can be found online at http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html.

Documentation can be found in PDF form in the **doc/pdf/** directory, or in HTML form at **doc/html/index.html** and online at http://docs.nvidia.com/cuda/index.html.

Other

The Open64 source files are controlled under terms of the GPL license. Current and previously released versions are located at ftp://download.nvidia.com/CUDAOpen64/.

The CUDA-GDB source files are controlled under terms of the GPL license.

- The source code for CUDA-GDB that shipped with CUDA 5.5 and subsequent versions is located at https://github.com/NVIDIA/cuda-gdb.
- The source code for CUDA-GDB that shipped with CUDA 5.0 and previous versions is located at ftp://download.nvidia.com/CUDAOpen64/.

Chapter 2. NEW FEATURES

2.1. General CUDA

- Added a method to the CUDA Driver API, **`cuDevicePrimaryCtxRetain()`**, that allows a program to create (or to access if it already exists) the same CUDA context for a GPU device as the one used by the CUDART (CUDA Runtime API) library. This context is referred to as the primary context, and this new method allows for sharing the primary context between CUDART and other threads, which can reduce the performance overhead of creating and maintaining multiple contexts per device.
- Unified the device enumeration for CUDA, NVML, and related tools. Variable **`CUDA_DEVICE_ORDER`** can have a value of **`FASTEST_FIRST`** (default) or **`PCI_BUS_ID`**.
- Instrumented NVML (NVIDA Management Library) and the CUDA driver to ignore GPUs that have been made inaccessible via cgroups (control groups). This enables schedulers that rely on cgroups to enforce device access restrictions for their jobs. Job schedulers wanting to use cgroups for device restriction need CUDA and NVML to handle those restrictions in a graceful way.
- Implimented Multi-process Server (MPS), which enables concurrent execution of GPU tasks from multiple CPUs within a single node. It allows setting multiple MPI ranks in a single node.
- The Windows and Mac OS X installers are now also available as network installers. A network installer is much smaller than the traditional local installer and downloads only the components selected for installation.

2.2. CUDA Tools

2.2.1. CUDA Compiler

- Added support for GCC 4.9.
- Added support for the C++11 language dialect.

- On Mac OS X, **`libc++`** is supported with XCode 5.x. Command-line option **`-Xcompiler -stdlib=libstdc++`** is no longer needed when invoking NVCC. Instead, NVCC uses the default library that Clang chooses on Mac OS X. Users are still able to choose between **`libc++`** and **`libstdc++`** by passing **`-Xcompiler -stdlib=libc++`** or **`-Xcompiler -stdlib=libstdc++`** to NVCC.
- The Runtime Compilation library (**`nvrtc`**) provides an API to compile CUDA-C++ device source code at runtime. The resulting compiled PTX can be launched on a GPU using the CUDA Driver API. More details can be found in the *libNVRTC User Guide*.

2.2.2. CUDA-GDB

- Starting with CUDA 7.0, GPU core dumps can read by CUDA-GDB with the **`target cudacore ${gpucoredump}`** and **`target core ${cpucoredump} ${gpucoredump}`** commands.
- Enabled CUDA applications to generate a GPU core dump when an exception is hit on the GPU. The feature is supported on Windows, Mac OS X, and Linux desktops. (Android, L4T, and Vibrante support may come in the future.) On Windows, this feature is only supported in TCC mode. On Unix-like OSs (Linux, OS X, etc.), a CPU core dump is generated along with a GPU core dump.

2.2.3. CUDA-MEMCHECK

- Enabled the tracking and reporting of uninitialized global memory.

2.2.4. CUDA Profiler

- On supported chips (sm_30 and beyond), all hardware counters exposed by CUDA profiling tools (**`nvprof`**, **`nvvp`**, and Nsight Eclipse Edition) can now be profiled from multiple applications at the same time.

2.2.5. Nsight Eclipse Edition

- Cross compiling to the Power8 target architecture using the GNU tool-chain is now supported within the Nsight IDE.

2.2.6. NVIDIA Visual Profiler

- With GPU PC Sampling, which is supported for devices with compute capability 5.2, the Visual Profiler shows stall causes for each source and assembly line. This helps in pinpointing latency bottlenecks in a GPU kernel at the source level.

2.3. CUDA Libraries

2.3.1. cuBLAS Library

- The batched LU solver **`cublas{T}getrsBatched`** routine has been added to cuBLAS. It takes the output of the batched factorization routines **`cublas{T}getrfBatched`** to compute the solution given the provided batch of right-hand-side matrices.
- A license is no longer required in order to use cuBLAS-XT with more than two GPUs.

2.3.2. cuFFT Library

- For CUDA 7.0, support for callback routines, invoked when cuFFT loads and/or stores data, no longer requires an evaluation license file.
- For CUDA 7.0, cuFFT multiple-GPU execution is supported on up to four GPUs, except for single 1D complex-to-complex transforms, which are supported on two or four GPUs.
- In CUDA 7.0, transform execution may be distributed to four GPUs with the same CUDA architecture. In addition, multiple GPU support for two or four GPUs is no longer constrained to GPUs on a single board. Use of this functionality requires a free evaluation license file, which is available to registered developers via the cuFFT developer page.
- For CUDA 7.0, single complex-to-complex 2D and 3D transforms with dimensions that can be factored into primes less than or equal to 127 are supported on multiple GPUs. Single complex-to-complex 1D transforms on multiple GPUs continue to be limited to sizes that are powers of 2.

2.3.3. cuSOLVER Library

- CUDA 7.0 introduces cuSOLVER, a new library that is a collection of routines to solve linear systems and Eigen problems. It includes dense and sparse linear solvers and sparse refactorization.
- Enabled offloading dense linear algebra calls to the GPUs in a sparse direct solver.

2.3.4. cuSPARSE Library

- Added a new **`cusparse<t>csrgemm2()`** routine optimized for small matrices and operations **`C = a*A*B + b*D`**, where **`A`**, **`B`**, and **`D`** are CSR matrices.
- Added graph coloring.

2.3.5. CUDA Math Library

- Support for 3D and 4D Euclidean norm and 3D Euclidean reciprocal norm has been added to the math library.

2.3.6. Thrust Library

- Thrust version 1.8.0 introduces support for algorithm invocation from CUDA **`__device__ code`**, support for CUDA streams, and algorithm performance improvements. Users may now invoke Thrust algorithms from CUDA **`__device__`** code, providing a parallel algorithms library to CUDA programmers authoring custom kernels as well as allowing Thrust programmers to nest their algorithm calls within functors. The **`thrust::seq`** execution policy allows users to require sequential algorithm execution in the calling thread and makes a sequential algorithms library available to individual CUDA threads. The **`.on(stream)`** syntax allows users to request a CUDA stream for kernels launched during algorithm execution. Finally, new CUDA algorithm implementations provide substantial performance improvements.

2.4. CUDA Samples

- The CUDA Samples makefile **`x86_64=1`** and **`ARMv7=1`** options have been deprecated. Please use **`TARGET_ARCH`** to set the targeted build architecture instead. The CUDA Samples makefile **`GCC`** option has been deprecated. Please use **`HOST_COMPILER`** to set the host compiler instead.

Chapter 3. UNSUPPORTED FEATURES

The following features are officially unsupported in the current release. Developers must employ alternative solutions to these features in their software.

Support for 32-bit x86 Linux Systems

The CUDA Toolkit and CUDA Driver no longer support developing and running CUDA and OpenCL Applications on 32-bit x86 Linux operating systems.

Note 1: Developing and running 32-bit applications on 64-bit (x86_64) Linux operating systems is still supported, but that functionality is marked as deprecated and may be dropped in a future release. 64-bit applications are not impacted.

Note 2: This notice applies to x86 architectures only; 32-bit application support on the ARM architecture remains officially supported.

Red Hat Enterprise Linux 5 and CentOS 5

CUDA no longer supports the RHEL 5 and CentOS 5 Linux distributions. Please note that RHEL 6, RHEL 7, CentOS 6, and CentOS 7 are all supported.

CUDA Toolkit and CUDA Driver Support for Tesla Architecture

The CUDA Toolkit and CUDA Driver no longer supports the sm_10, sm_11, sm_12, and sm_13 architectures. As a consequence, **`CU_TARGET_COMPUTE_1x`** enum values have been removed from the CUDA headers.

Certain CUDA Features on 32-bit and 32-on-64-bit Windows Systems

The CUDA Toolkit no longer supports 32-bit Windows operating systems. Furthermore, the Windows CUDA Driver no longer supports Tesla and Quadro products on 32-bit Windows operating systems. Additionally, on 64-bit Windows operating systems, the following features are no longer supported by the CUDA driver or CUDA toolkit:

- Running 32-bit applications on Tesla and Quadro products
- Using the Thrust library from 32-bit applications
- 32-bit versions of the CUDA Toolkit scientific libraries, including cuBLAS, cuSPARSE, cuFFT, cuRAND, and NPP
- 32-bit versions of the CUDA samples

Note the above list doesn't impact any 64-bit components on Windows.

Using gcc as a Host Compiler on Mac OS X

On Mac OS X platforms, **`nvcc`** no longer supports the **`gcc`** toolchain for compiling host code, including the GNU stdlibc++ standard C++ library. Developers should use the **`clang/llvm`** toolchain for compiling host code instead; **`nvcc`** does this by default in CUDA 7.0 on supported Mac OS X platforms.

Chapter 4. DEPRECATED FEATURES

The following features are deprecated in the current release of the CUDA software. The features still work in the current release, but their documentation may have been removed, and they will become officially unsupported in a future release. We recommend that developers employ alternative solutions to these features in their software.

Developing and Running 32-bit CUDA and OpenCL Applications on x86 Linux Platforms

Support for developing and running 32-bit CUDA and OpenCL applications on 64-bit x86 Linux platforms is deprecated.

CUDA Samples Makefile `x86_64=1` and `ARMv7=1` Options

The CUDA Samples Makefile **`x86_64=1`** and **`ARMv7=1`** options have been deprecated. Please use **`TARGET_ARCH`** to set the targeted build architecture instead. The CUDA Samples Makefile **`GCC`** option has been deprecated. Please use **`HOST_COMPILER`** to set the host compiler instead.

Header File `sobol_direction_vectors.h`.

The **`sobol_direction_vectors.h`** header file is deprecated. This file allowed developers to employ the cuRAND device API with sobol distributions. However, since it is large and takes significant amounts of time and RAM to compile, this file is deprecated in favor of the **`curandGetDirectionVectors{32,64}()`** and **`curandGetScrambleConstants{32,64}()`** functions. These functions return a memory pointer to the direction vectors that are precompiled into the cuRAND library, and developers should use this pointer to copy the vectors to the GPU device memory.

Chapter 5. PERFORMANCE IMPROVEMENTS

5.1. CUDA Libraries

5.1.1. cuFFT Library

- For CUDA 7.0, a new mode has been added for copying single 1D transform input from the host to the GPU. This mode redistributes the inputs as required by the first computation phase of the algorithm and eliminates the overhead of this data redistribution from the execution phase.
- In CUDA 7.0, new composite-sized FFT kernels have been added for many sizes which can be factored into small primes. Composite sizes up to 256 are nearly completely represented, and there is some coverage for sizes up to 1920. This was done to reduce the number of kernel invocations done on the host and will result in significant speed improvements for many composite sizes. Note that these composite-size kernels are not used in combination with the cuFFT callback feature.

5.1.2. CUDA Math Library

- The performance of the double-precision reciprocal instruction **`rcp(x)`** in round-to-nearest mode was significantly improved.

Chapter 6. RESOLVED ISSUES

6.1. General CUDA

- On openSUSE and SLES, X no longer fails to load if the CUDA Toolkit RPM packages are installed using relocation immediately following an installation of the cuda-drivers package.
- The Windows toolkit installation no longer fails if Visual Studio, **`Nvda.Launcher.exe`**, **`Nsight.Monitor.exe`**, or **`Nvda.CrashReporter.exe`** is running.
- The **`cuda`** and **`gpu-deployment-kit`** packages must be installed by separate executions of **`yum`**. See the *Linux Getting Started Guide* for more details.
- The CUDA reference manual now correctly describes the CUDA device pointer **`CUdeviceptr`** as an unsigned integer type whose size matches the size of a pointer on the target platform.

6.2. CUDA Tools

6.2.1. CUDA Compiler

- C++11 support added. The new **`nvcc`** flag **`-std=c++11`** turns on C++11 features in the CUDA compiler as well as the host compiler and linker. The flag is supported by host compilers **`gcc`** >= 4.7 and clang. In addition, any C++11 features that are enabled by default by a supported host compiler are also allowed to be used in device code. Please see the *CUDA Programming Guide* for further details.

6.2.2. Nsight Eclipse Edition

- Starting with CUDA 7.0, to cross-compile a CUDA project, Nsight uses the default cross-compiler available on the host (or remote) operating system.

6.2.3. NVIDIA Visual Profiler

- In the Visual Profiler timeline, the colors of intervals on the **Compute** and **Stream** timelines are no longer incorrect. Also the timeline modes **color by stream** and **color by process** do work.

6.3. CUDA Libraries

6.3.1. cuSPARSE Library

- CUDA 7.0 fixed bugs in the **`csr2csc()`** and **`bsr2bsc()`** routines that were in the CUDA 6.0 and 6.5 releases. As a consequence, **`csrsv()`**, **`csrsv2()`**, **`csrsm()`**, **`bsrsv2()`**, **`bsrsm2()`**, and **`csrgemm()`** now produce correct results when working with transpose (**`CUSPARSE_OPERATION_TRANSPOSE`**) or conjugate-transpose (**`CUSPARSE_OPERATION_CONJUGATE_TRANSPOSE`**) operations.

6.4. CUDA Samples

- To properly build the `simpleCUFFT_callback` sample, the **`-dc`** compiler flag no longer must be added to the compilation commands.

Chapter 7. KNOWN ISSUES

7.1. General CUDA

- If the Windows toolkit installation fails, it may be because Visual Studio, **`Nvda.Launcher.exe`**, **`Nsight.Monitor.exe`**, or **`Nvda.CrashReporter.exe`** is running. Make sure these programs are closed and try to install again.
- Peer access is disabled between two devices if either of them is in SLI mode.

7.2. CUDA Tools

7.2.1. CUDA Compiler

- On Mac OS X, when Clang is used as the host compiler, 32-bit target compilation is not supported. This is because the Clang compiler doesn't support the **`-malign-double`** switch that the NVCC compiler needs to properly align double-precision structure fields when compiling for a 32-bit target (GCC does support this switch). Note that GCC is the default host compiler used by NVCC on Mac OS X 10.8 and Clang is the default on Mac OS X 10.9.