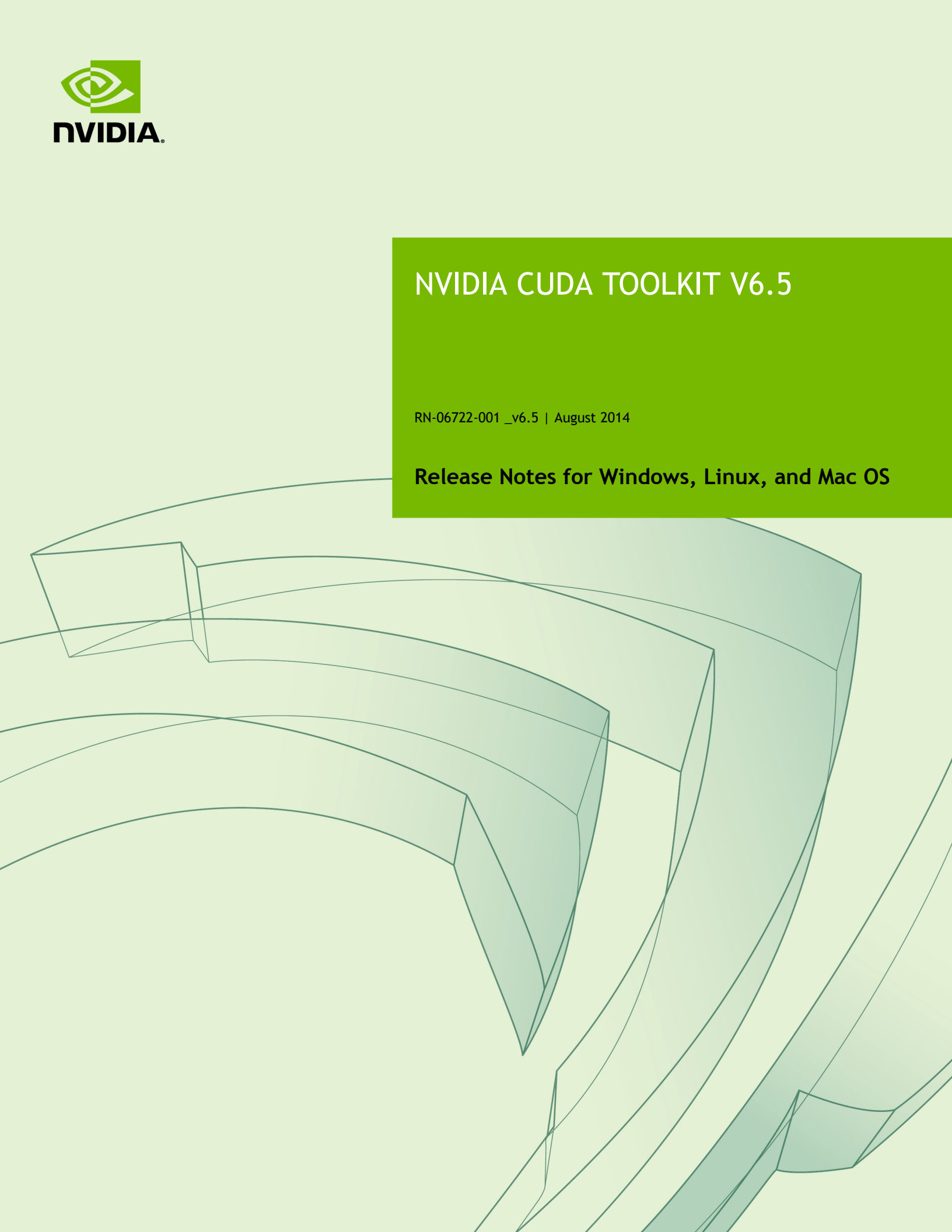

NVIDIA CUDA TOOLKIT V6.5

RN-06722-001 _v6.5 | August 2014

Release Notes for Windows, Linux, and Mac OS

TABLE OF CONTENTS

ERRATA

Known Issues

Any listed issue was not documented in the original version of these release notes.

Static Library Compilation Command-lines Incorrect in cuBLAS, cuSPARSE, and cuFFT Documentation

The incorrect command-lines use **`gcc`** instead of **`g++`** and omit the **`cuda`** and **`cudart`** libraries, as well as the **`include`** and **`lib`** directories. For example, the cuBLAS document shows this command-line:

```
gcc myCublasApp.c libcublas_static.a libculibos.a -o myCublasApp
```

The correct command-line in this case is as follows:

```
g++ myCublasApp.c -lcublas_static -lculibos -lcudart_static -lcuda -o
 myCublasApp -I /usr/local/cuda-6.5/include -L /usr/local/cuda-6.5/lib64
```

ARM Cross-Compilation with Nsight Eclipse Edition

To use the ARM cross-compilation feature of Nsight Eclipse Edition without an ARM cross-compiler based on GCC-4.6, please create a symlink that points to the cross-compiler executable present on your system. For example, it can be done by issuing the following command:

```
sudo ln -sf `which arm-linux-gnueabihf-g++` /usr/bin/arm-linux-gnueabihf-g+
+-4.6
```

NVIDIA Visual Profiler Timeline Color

In the Visual Profiler timeline, the colors of intervals on the **Compute** and **Stream** timelines are incorrect. Also the timeline modes "**color by stream**" and "**color by process**" do not work.

NVIDIA Visual Profiler on Mac OS X 10.9.3

Visual Profiler events and metrics do not work correctly on Mac OS X 10.9.3. Please use Mac OS X 10.9.4 or later. *This item updates information found in Section 7.3.4.*

Building CUDA Sample "simpleCUFFT_callback"

To properly build the `simpleCUFFT_callback` sample, the **`-dc compiler`** flag must be added to the compilation commands.

CUDA Samples on AArch64 Systems

On AArch64 systems, CUDA Samples using the CUDA Driver API must be built using **`CUDA_SEARCH_PATH=/usr/lib`**.

Chapter 1. CUDA TOOLKIT OVERVIEW

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

System Requirements

The CUDA Toolkit is supported for Linux, Mac OS X, and Microsoft Windows. Specific system requirements are referenced below.

- Linux: The latest information about support for the Linux platform can be found online at http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html.
- Mac OS: The latest information about support for Mac OS X can be found online at http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-mac-os-x/index.html.
- Windows: The latest information about support for Microsoft Windows can be found online at http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/index.html.

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 OS), NSight VSE (Windows)
- Debuggers: **`cuda-memcheck`**, **`cuda-gdb`** (Linux, Mac OS), NSight VSE (Windows)
- Profilers: **`nvprof`**, **`nvvp`**, NSight VSE (Windows)
- Utilities: **`cuobjdump`**, **`nvdisasm`**, **`nvprune`**

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)
- **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 OS, and are installed to **C:\ProgramData\NVIDIA Corporation\CUDA Samples** on Windows. On Linux and Mac OS, 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 OS.

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, including Getting Started Guides, Programming Guides, API References, and Tools Guides, 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

- Introduced support for the latest architecture in the Maxwell family (sm_52).
- Added support for using **`_shfl`** intrinsics with all first class types. User source code already implementing this feature should be guarded with **`(CUDA_VERSION <= 6000)`** in order to compile against CUDA 6.5.
- On Linux, Xid 13 dmesg error reporting has been improved to provide more detail and also to indicate which of the various potential causes of the Xid 13 error was to blame.
- The Linux **`.run`** installation now comes with an uninstallation script, **`uninstall_cuda_6.5.pl`**, to help with uninstalling the toolkit during conversions to Debian/RPM installations.
- On Linux, stubs that applications can link against at build time have been added for each library. This removes the need to have the full library installed when building an application. In addition to the CUDA Toolkit libraries, a stub has been provided for the CUDA Driver library (**`libcuda.so`**). See the *NVIDIA CUDA Getting Started Guide for LINUX* for details on how to use these stubs.

2.2. CUDA Tools

2.2.1. General CUDA Tools

- Improved support for CUDA FORTRAN in the command-line debugging and profiling tools in the CUDA Toolkit, including new debugging support for FORTRAN arrays (in Linux only), improved source-to-assembly code correlation, and improved documentation. This improved support is available with PGI compiler version 14.4 and higher. CUDA FORTRAN support is a beta feature in the CUDA 6.5 release.

2.2.2. CUDA Compiler

- (Windows) Support has been added for the C++ compiler (VC 12) in Microsoft Visual Studio 2013 for Windows.
- The default target GPU architecture (**-arch**) for **nvcc** has changed from sm_10 in previous releases to sm_20 in this release. Note that sm_20 is not the minimum target architecture supported by **nvcc**, since sm_11, sm_12, and sm_13 are still valid target GPU architectures if specified explicitly.
- A new tool in CUDA 6.5, **nvprune**, prunes an object to only contain the compiled code for the specified architectures (for example, selects only the sm_35 code for **libcublas_static.a**). See the *CUDA Binary Utilities* document for more information.
- (Linux) The **cuobjdump** utility for examining CUDA binaries is now available on Linux distributions running natively on the ARM architecture; this includes Android OS.

2.2.3. CUDA Occupancy Calculator

- Added CUDA occupancy calculator and occupancy-based launch configuration API interfaces. These functions help set up execution configurations with reasonable occupancy.

  The stand-alone programmatic occupancy calculator implementation, **cuda_occupancy.h**, is rewritten and out of beta. Note that the API has changed significantly from the beta version included with CUDA 6.0. This file includes stand-alone implementations of both the occupancy calculator and the occupancy-based launch configuration functions, so applications can use them without depending on the entire CUDA software stack.

2.2.4. CUDA Profiling Tools Interface (CUPTI)

- Instruction classification is done for the source-correlated instruction execution activity **CUpti_ActivityInstructionExecution**. See **CUpti_ActivityInstructionClass** for the instruction classes.
- Two new device attributes were added to the activity **CUpti_DeviceAttribute**:
  - **CUPTI_DEVICE_ATTR_FLOP_SP_PER_CYCLE**. Peak single-precision floating-point operations that can be performed in one cycle by the device.
  - **CUPTI_DEVICE_ATTR_FLOP_DP_PER_CYCLE**. Peak double-precision floating-point operations that can be performed in one cycle by the device.
- Two new metric device properties were added:
  - **CUPTI_METRIC_PROPERTY_FLOP_SP**. Peak single-precision floating point operations that can be performed in one cycle by the device.
  - **CUPTI_METRIC_PROPERTY_FLOP_DP**. Peak double-precision floating-point operations that can be performed in one cycle by the device.
- Activity record **CUpti_ActivityGlobalAccess2** for source-level global access information replaces **CUpti_ActivityGlobalAccess**, which has been deprecated.

The new record adds information needed to map SASS assembly instructions to CUDA C source code; it also provides ideal L2 transaction counts based on access patterns.

- Activity record **`CUpti_ActivityBranch2`** for source-level branch information replaces **`CUpti_ActivityBranch`**, which has been deprecated. The new record adds information needed to map SASS assembly instructions to CUDA C source code.
- Added a new sample to show how to map SASS assembly instructions to CUDA C source lines.

2.2.5. NVIDIA Visual Profiler

- Visual Profiler now displays peak single-precision flops and peak double-precision flops for a GPU under **Device** properties.
- The Visual Profiler **Kernel** profile analysis view has been updated with several enhancements.
  - Initially, the instruction with the maximum execution count is highlighted.
  - A bar is shown in the background of the counter value for the **Exec Count** column to make it easier to identify instructions with high execution counts.
  - The current assembly instruction block is highlighted using two horizontal lines around the block. Also, **next** and **previous** buttons have been added to move to the next or previous block of assembly instructions.
  - Syntax highlighting is done for the CUDA C source.
  - A tooltip describing each column has been added.
- The Visual Profiler **Kernel** memory analysis view has been updated with several enhancements.
  - Added ECC overhead, which provides a count of memory transactions required for ECC.
  - For L2 cache, added a split of transactions for L1 reads, L1 writes, texture reads, atomic reads, and noncoherent reads.
  - For L1 cache, added a count of atomic transactions.
- Visual Profiler and **`nvprof`** now support a new application replay mode for collecting multiple events and metrics. In this mode, the application is run multiple times instead of using kernel replay. This is useful for cases when the kernel uses a large amount of device memory and the use of kernel replay is slow due to the high overhead of saving and restoring device memory for each kernel replay run. In Visual Profiler, this new application replay mode is enabled in the **New Session** dialog.

2.3. CUDA Libraries

2.3.1. General CUDA Libraries

- Starting with the CUDA 6.5 release on Linux and Mac OS, the cuBLAS, cuSPARSE, cuFFT, cuRAND, and NPP libraries are provided as static libraries in addition to

being provided as shared libraries. These new static libraries depend on a common thread abstraction layer library cuLIBOS (**libculibos.a**) that is now distributed as part of the toolkit. Consequently, cuLIBOS must be provided to the linker when at least one of these static libraries is being linked against. For example, on Linux, to compile an application using cuBLAS and cuRAND against the static versions of these libraries, the following command should be used:

```
g++ myCublasApp.c -lcublas_static -lculibos -lcudart_static -lcuda -o
 myCublasApp -I /usr/local/cuda-6.5/include -L /usr/local/cuda-6.5/lib64
```

Note that **libculibos.a** is **not** needed when the shared version of these libraries is used.

2.3.2. cuBLAS Library

- The **cublas<T>trsmBatched()** routine no longer limits the **m** and **n** dimensions to 32. However, the routine is still intended to be used for matrices of relatively small size, for which the performance of calling **cublas<T>trsm()** multiple times would be limited by kernel launch overhead. Performance has also been significantly improved for **n** > 1.
- The cuBLAS Library now offers the batched routines **cublas<T>geqrfBatched()** and **cublas<T>gelsBatched()**, which are respectively a batched QR factorization and a batched least-squares solver for over-determined systems.

2.3.3. cuFFT Library

- User-specified routines can now operate directly on cuFFT input or output data. The new **cufftXt*Callback()** APIs are used to specify which user-defined routines will be called when each data point is loaded or stored by the cuFFT kernels, potentially reducing the overall number of accesses to device memory.
- Starting with cuFFT in CUDA 6.5, single 2D or 3D FFTs on multiple GPUs can be performed without the need for transposing data between successive FFTs. In prior releases it was necessary to transpose the data before performing a second FFT on multiple GPUs.

2.3.4. cuSPARSE Library

- The cuSPARSE Library added two new routines that support the BSR format, **cusparse<T>bsrmm()** and **cusparse<T>bsrsm()**, which are respectively the multiplication of a matrix in BSR format by a dense matrix, and the solve of a triangular matrix in BSR format against multiple right-hand sides.

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.

Windows XP 64-bit Edition Support

With this release, CUDA no longer supports the 64-bit version of the Windows XP operating system, although CUDA on the 32-bit version of Windows XP is still supported. We recommend that developers and users of the 64-bit version of Windows XP migrate to Windows 7 or Windows 8.1, which are supported in the current and future CUDA releases.

Windows Vista Support

This CUDA release no longer supports the Windows Vista operating system. We recommend that users and developers migrate to Windows 7 or Windows 8.1, which are supported in the current and future releases.

Windows Server 2012 Support

CUDA on the Windows Server 2012 operating system is not supported in this CUDA release. We recommend that users and developers migrate to Windows Server 2012 R2, which is supported in the current and future releases.

(Linux) Support for 32-bit Applications on x86-based Linux Distributions

Several portions of the CUDA Toolkit are no longer available for developing 32-bit applications on x86-based Linux distributions:

- .deb installer packages for 32-bit CUDA Toolkit components
- CUDA Toolkit scientific libraries, including cuBLAS, cuSPARSE, cuFFT, cuRAND, and NPP
- Thrust
- Quadro and Tesla products
- Tesla (sm_1x) and Fermi (sm_2x) architectures
- CUDA Samples

The above list also applies to 32-bit components and 32-bit rpm/deb packages on 64-bit x86-based Linux distributions. The 64-bit components are unaffected by these changes.

(Mac OS X) Support for 32-bit CUDA and OpenCL Applications on Mac OS X

Developing and running 32-bit CUDA and OpenCL applications on Mac OS X platforms is no longer supported in the CUDA Toolkit and in the CUDA Driver.

Legacy 32-bit CUDA and OpenCL applications will not run on this version of the CUDA Driver on Mac OS X platforms.

Targeting sm_10 (G80) for CUDA Applications

The CUDA Toolkit no longer supports the sm_10 target architecture (the G80 architecture) for CUDA and OpenCL applications.

CUDA Video Encoder (NVCUVENC)

Building applications with the CUDA Video Encoder interface is no longer supported; however, the driver will continue to run applications built against this interface. We recommend using the NVIDIA Encoder API (NVENC), a newer video encoding interface that is available at https://developer.nvidia.com/nvidia-video-codec-sdk.

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.

Tesla and Quadro Products and CUDA Toolkit on 32-bit Windows Platforms

Support for the CUDA Toolkit on 32-bit Windows platforms is deprecated, as is support for Tesla and Quadro products for the CUDA driver on 32-bit Windows platforms. Additionally, on 64-bit Windows platforms, support for the following features for 32-bit CUDA and OpenCL applications is deprecated from the CUDA driver and CUDA toolkit, as appropriate:

- Tesla and Quadro products
- CUDA Toolkit scientific libraries, including cuBLAS, cuSPARSE, cuFFT, cuRAND, and NPP
- Thrust
- CUDA samples

This deprecation notice doesn't impact any 64-bit components.

Interop with IDirect3D9 objects on Microsoft Windows 7 and Later

This release deprecates support for interop with **`IDirect3D9`** objects on Windows 7 and later Microsoft operating systems. This applies to the **`cuD3D9*()`** and **`cuGraphicsD3D9RegisterResource()`** routines in the Driver API, as well as the corresponding **`cudaD3D9*()`** and **`cudaGraphicsD3D9RegisterResource()`** routines in the Runtime API. We recommend using **`IDirect3D9ex`** objects, which will work with these same routines, instead.

Linux RHEL 5 and CentOS 5 Support

Support for CUDA on the RHEL 5 and CentOS 5 Linux distributions is deprecated in this CUDA release and will be dropped in a future release. We recommend that users and developers migrate to RHEL 6, which is supported in the current and future releases.

Support for sm_10, sm_11, sm_12, and sm_13 Architectures

The sm_10 architecture is deprecated within the CUDA Driver, and the sm_11, sm_12, and sm_13 architectures are deprecated within the CUDA Toolkit and the CUDA

Driver. Support for these architectures will be removed in the next major version of the CUDA Toolkit and Driver. Note that support for the sm_10 architecture has already been removed from the CUDA Toolkit.

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 x86 Linux platforms is deprecated. This implies the following:

- Support is currently still available in the toolkit and driver.
- Support may be dropped from the toolkit in a future release, and similarly from the driver.
- New features may not have support for 32-bit x86 Linux applications.
- This notice applies to running applications on a 32-bit Linux kernel, and also to running 32-bit applications on a 64-bit Linux kernel.
- This notice applies to x86 architectures only; 32-bit Linux applications are still officially supported and are not deprecated on the ARM architecture.

CUPTI Activity Records

Activity record **`CUpti_ActivityGlobalAccess`** for source-level global access information has been deprecated and replaced by the new activity record **`CUpti_ActivityGlobalAccess2`**. Activity record **`CUpti_ActivityBranch`** for source-level branch information has been deprecated and replaced by the new activity record **`CUpti_ActivityBranch2`**.

Chapter 5. PERFORMANCE IMPROVEMENTS

5.1. General CUDA

- MPS performance has been improved: launch performance has been improved from 7 to 5 microseconds; launch and synchronize performance has been improved from 35 to 15 microseconds.

5.2. CUDA Libraries

5.2.1. CUDA Math Library

- Performance has been increased for these single-precision functions in CUDA 6.5: **`acoshf()`**, **`asinhf()`**, **`atanf()`**, **`atan2f()`**, **`atanhf()`**, **`cyl_bessel_i0f()`**, **`cyl_bessel_i1f()`**, **`cbrtf()`**, **`coshf()`**, **`erfcf()`**, **`erfcinvf()`**, **`erfcxf()`**, **`erfinvf()`**, **`expf()`**, **`exp10f()`**, **`expm1f()`**, **`fdiv_rd()`**, **`fdiv_rn()`**, **`fdiv_ru()`**, **`fdiv_rz()`**, **`fmodf()`**, **`frcp_rd()`**, **`frcp_rn()`**, **`frcp_ru()`**, **`frcp_rz()`**, **`frsqrt_rn()`**, **`hypotf()`**, **`logf()`**, **`log10f()`**, **`log1pf()`**, **`log2f()`**, **`normcdff()`**, **`normcdfinvf()`**, **`powf()`**, **`remainderf()`**, **`remquof()`**, **`rhypotf()`**, **`sincospif()`**, **`sinhf()`**, **`sinpif()`**, and **`tanhf()`**.

  Of these single-precision functions, **`atanf()`**, **`expf()`**, **`exp10f()`**, **`expm1f()`**, **`hypotf()`**, and **`rhypotf()`** show especially marked improvement.
- Performance has been increased for these double-precision functions in CUDA 6.5: **`acosh()`**, **`asin()`**, **`asinh()`**, **`atan()`**, **`atanh()`**, cyl_bessel_i0, **`cyl_bessel_i1()`**, **`cbrt()`**, **`cospi()`**, **`div()`**, **`erfc()`**, **`erfcx()`**, **`erfinv()`**, **`exp2()`**, **`fmod()`**, **`hypot()`**, **`log()`**, **`log10()`**, **`log1p()`**, **`log2()`**, **`normcdf()`**, pow(), **`rcbrt()`**, **`remainder()`**, **`remquo()`**, **`rhypot()`**, **`sincospi()`**, **`sinpi()`**, and **`tan()`**.

  Of these double-precision functions, **`acosh()`**, **`atan()`**, **`cbrt()`**, **`hypot()`**, and **`rhypot()`** show especially marked improvement.
- Performance of the double-precision square root function, **`sqrt()`**, was significantly improved for GPUs with compute capability 2.0 and above.

- Performance of the double-precision reciprocal square root function, **`rsqrt()`**, was significantly improved for GPUs with compute capability 2.0 and above.

Chapter 6. RESOLVED ISSUES

6.1. General CUDA

- (Linux) A driver packaging issue that forced users on Redhat and Fedora to ensure that the **`xorg-x11-drv-nvidia-devel`** package was installed has been resovled.
- The device memory heap size, set using **`cudaDeviceSetLimit(cudaLimitMallocHeapSize, *)`** or **`cuCtxSetLimit(CU_LIMIT_MALLOC_HEAP_SIZE, *)`**, is no longer limited to a size of 4,294,967,296 (4 GB).

Chapter 7. KNOWN ISSUES

7.1. Linux on ARMv7 Specific Issues

- Mapping host memory allocated outside of CUDA to device memory is not allowed on ARM; because of this, **`cudaHostRegister()`** is not supported by the CUDA driver on ARM platforms. If required, **`cudaHostAlloc()`** with the flag **`cudaHostAllocMapped`** can be used to allocate device-mapped host-accessible memory.

7.2. General CUDA

- The **`cuda`** and **`gpu-deployment-kit`** packages must be installed by separate executions of **`yum`**. See the *Linux Getting Started Guide* for more details.
- On openSUSE and SLES, X will fail to load if the CUDA Toolkit RPM packages are installed using relocation immediately following an installation of the **`cuda-drivers`** package (and its dependencies). Users should reboot in between the driver and toolkit installations. Executing **`nvidia-xconfig`** may rescue a system where X has failed to load in this situation.
- The CUDA drivers may fail to install if the RPMFusion repository is enabled at the same time as the CUDA repository. When installing CUDA, the **`--disablerepo="rpmfusion-nonfree*"`** option should be used. For example, to install the **`cuda`** package: **`yum --disablerepo="rpmfusion-nonfree*" install cuda`**.
- (Mac OS) When CUDA applications are run on 2012 MacBook Pro models, allowing or forcing the system to go to sleep causes a system crash (kernel panic). To prevent the computer from automatically going to sleep, set the **Computer Sleep** option slider to **Never** in the **Energy Saver** pane of the **System Preferences**.
- The CUDA reference manual incorrectly describes the type of **`CUdeviceptr`** as an **`unsigned int`** on all platforms. On 64-bit platforms, a **`CUdeviceptr`** is an **`unsigned long long`**, not an **`unsigned int`**.
- Peer access is disabled between two devices if either of them is in SLI mode.

- On multi-GPU configurations without P2P support between any pair of devices that support Unified Memory, managed memory allocations are placed in zero-copy memory. When data is migrated, this results in lower performance than the default managed memory behavior. In certain cases, the environment variable **CUDA_MANAGED_FORCE_DEVICE_ALLOC** can be set to force managed allocations to be in device memory and to enable migration on these hardware configurations. Normally, using the environment variable **CUDA_VISIBLE_DEVICES** is recommended to restrict CUDA to only use those GPUs that have P2P support. Please refer to the environment variables section in the *CUDA C Programming Guide* for further details.

7.3. CUDA Tools

7.3.1. CUDA Compiler

- (Mac OS) When Clang is used as the host compiler, 32-bit target compilation on OS X 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 OS X 10.8 and Clang is the default on OS X 10.9.
- The NVCC compiler doesn't accept Unicode characters in any filename or path provided as a command-line parameter.
- A CUDA program may not compile correctly if a **type** or **typedef T** is private to a class or a structure, and at least one of the following is satisfied:
  - **T** is a parameter type for a **__global__** function.
  - **T** is an argument type for a template instantiation of a **__global__** function.

  This restriction will be fixed in a future release.
- (Mac OS) The documentation surrounding the use of the flag **-malign-double** suggests it be used to make the struct size the same between host and device code. We know now that this flag causes problems with other host libraries. The CUDA documentation will be updated to reflect this. The workaround for this issue is to manually add padding so that the structs between the host compiler and CUDA are consistent.

7.3.2. CUDA-GDB

- There can be a significant performance degradation for large routines when the debugger steps over inlined routines. This happens because inlined code blocks may have multiple exit points under the hood, and the debugger steps every single instruction until an exit point is reached.

7.3.3. Nsight Eclipse Edition

- On Linux, the NVIDIA Visual Profiler (**nvvp**) and the Nsight IDE (**nsight**) do not run properly when the oxygen-gtk theme is used. If you experience such crashes,

please uninstall the oxygen-gtk theme. The command to do this on OpenSUSE is **`sudo zypper rm gtk2-theme-oxygen`** and on Ubuntu is **`sudo apt-get remove gtk2-engines-oxygen`**.

7.3.4. NVIDIA Visual Profiler

- (Windows) Using the mouse wheel button to scroll does not work within the Visual Profiler on Windows.
- (Mac OS) Visual Profiler events and metrics do not work correctly on Mac OS X 10.9.3. Mac OS X 10.9.2 can be used as a workaround.

7.4. CUDA Libraries

7.4.1. cuFFT Library

- In the CUDA 6.5 Early Access release, there are some limitations in the cuFFT callback implementation.
  - The static version of the cuFFT library is not supported on 32-bit Windows systems; consequently, the callback feature is not supported there either.
  - If the size of any dimension cannot be factored into a combination of powers of 2, 3, 5, and 7 (that is, the size has a prime factor of 11 or greater), the callback routine cannot safely call **`__syncthreads()`**.
  - For 2D and 3D transforms, if the size of any dimension has a prime factor of 131 or greater, **`cufftUnsetCallback()`** does not function correctly.
  - For 2D and 3D C2C transforms, if any dimension has a prime factor of 131 or greater, the **`store()`** callback does not function correctly.
  - For multi-GPU C2R and R2C plans, callbacks are not supported if the batch size is greater than one and any dimension has a prime factor of 131 or greater.

7.4.2. Thrust Library

- (Linux) There is a known issue that causes the **`TestGetTemporaryBufferDispatchExplicit`** and **`TestGetTemporaryBufferDispatchImplicit`** unit tests provided with the Thrust library to fail on the SLES 11 Linux distribution.
- (Linux) There is a known issue that causes the **`segmentationTreeThrust`** CUDA sample in the **`6_Advanced`** directory to fail on the SLES 11 Linux distribution.

7.4.3. CUDA Samples

- On 32-bit Windows systems, certain samples may fail to compile due to the compiler exhausting available memory, especially if compiling is done in Debug mode or if the sample is using Dynamic Parallelism.