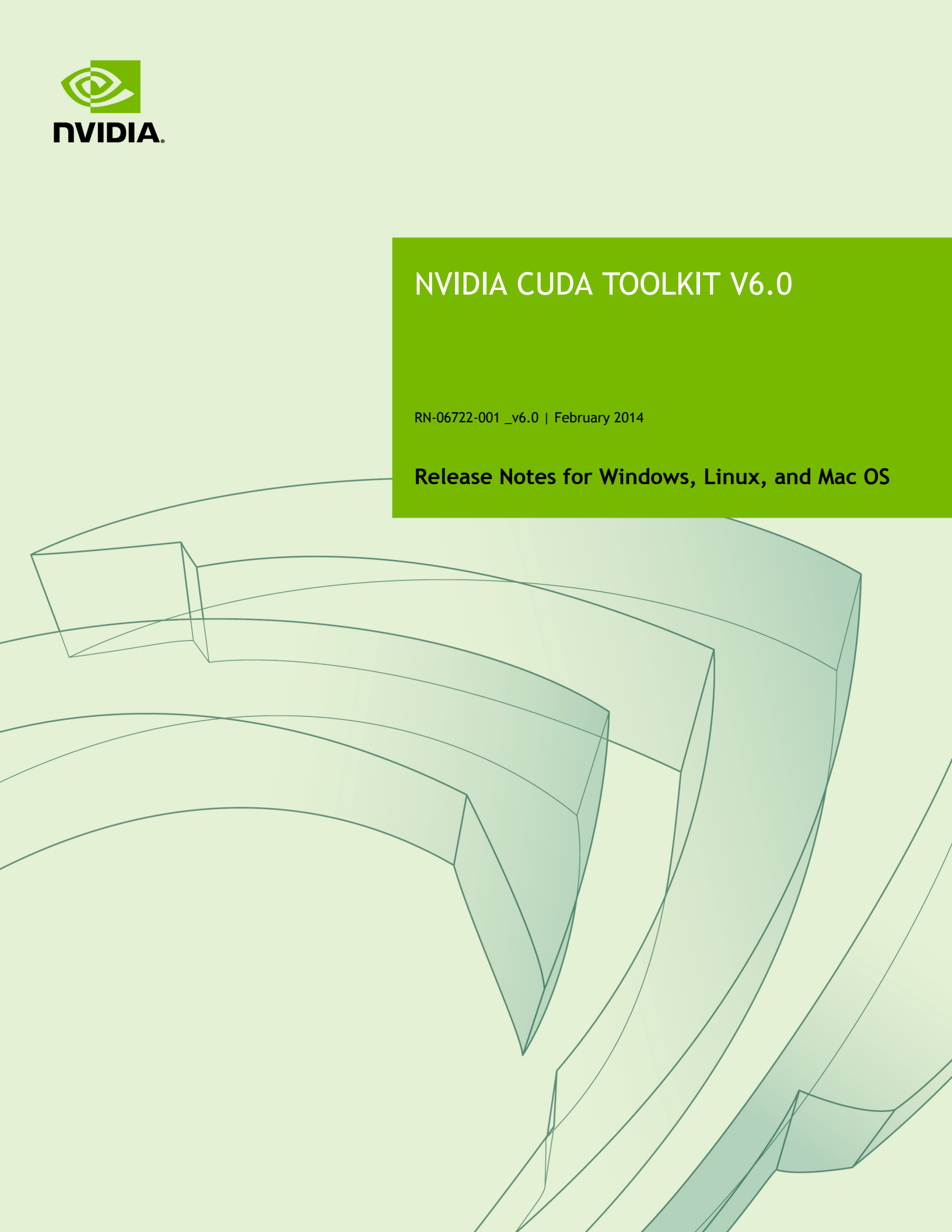

# NVIDIA CUDA TOOLKIT V6.0

RN-06722-001 _v6.0 | February 2014

## Release Notes for Windows, Linux, and Mac OS

# TABLE OF CONTENTS


Errata ..... iii
CUDA 6.0 Release Candidate ..... iii
Chapter 1. CUDA Toolkit Major Components ..... 1
Chapter 2. New Features ..... 3
2.1. General CUDA ..... 3
2.2. CUDA Tools ..... 4
2.2.1. CUDA Compiler ..... 4
2.2.2. CUDA-GDB ..... 4
2.2.3. CUDA-MEMCHECK ..... 4
2.2.4. CUDA Profiling Tools Interface (CUPTI) ..... 5
2.2.5. Nsight Eclipse Edition ..... 5
2.2.6. NVIDIA Visual Profiler ..... 5
2.3. CUDA Libraries ..... 5
2.3.1. cuBLAS Library ..... 5
2.3.2. cuFFT Library ..... 6
2.3.3. cuRAND Library ..... 6
2.3.4. cuSPARSE Library ..... 6
2.3.5. CUDA Math Library ..... 6
2.3.6. NVIDIA Performance Primitives (NPP) ..... 6
2.4. CUDA Samples ..... 7
Chapter 3. Unsupported Features ..... 8
Chapter 4. Deprecated Features ..... 9
Chapter 5. Performance Improvements ..... 11
5.1. General CUDA ..... 11
Chapter 6. Resolved Issues ..... 12
6.1. General CUDA ..... 12
6.2. CUDA Tools ..... 12
6.2.1. CUDA Compiler ..... 12
6.2.2. Nsight Eclipse Edition ..... 12
6.3. CUDA Libraries ..... 13
6.3.1. cuBLAS Library ..... 13
6.3.2. NVIDIA Performance Primitives (NPP) ..... 13
Chapter 7. Known Issues ..... 14
7.1. Linux on ARMv7 Specific Issues ..... 14
7.2. General CUDA ..... 14
7.3. CUDA Tools ..... 15
7.3.1. CUDA Compiler ..... 15
7.3.2. NVIDIA Visual Profiler ..... 16

# ERRATA

## CUDA 6.0 Release Candidate

The following are known issues with the CUDA 6.0 Release Candidate that will be resolved in the production release:

- The **`minBlocksPerMultiprocessor`** parameter for the **`launch_bounds()`** qualifier only accepts values up to 16 when used in compiling for sm_50, even though values up to 32 are possible on that architecture.
- There is a performance issue with the new SIMD video intrinsics **`__v*2()`** and **`__v*4()`** when used in compiling for the sm_50 architecture.
- The sm_50 architecture supports 48 KB of shared memory per block; however, the check for this limit is not functioning properly in the compiler. This can allow programs that use more than 48 KB of shared memory per block to compile successfully, although they will fail to run because the driver component does check the limit properly.
- The MT19937 random number generator in the cuRAND library generates non-deterministic results for **`curandGenerateUniformDouble()`**.
- The NPP library function **`nppiAlphaComp_8u_AC4R()`** generates incorrect results when used with the **`NPPI_OP_ALPHA_ATOP_PREMUL`** option.
- The NPP library functions **`FilterSobelHorizSecondBorder()`** and **`FilterSobelVertSecondBorder()`** may generate incorrect results.

# 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)
- **`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

- Unified Memory is a new feature enabling a type of memory that can be accessed by both the CPU and GPU without explicit copying between the two. This is called "managed memory" in the software APIs. Unified Memory is automatically migrated to the physical memory attached to the processor that is accessing it. This migration provides high performance access from either processor, unlike "zero-copy" memory where all accesses are out of CPU system memory.
- Added a standalone header library for calculating occupancy (the library is not dependent on the CUDA Runtime or CUDA Driver APIs). The header library provides a programmatic interface for the occupancy calculations previously contained in the CUDA Occupancy Calculator.
- The Dynamic Parallelism runtime should no longer generate a **`cudaErrorLaunchPendingCountExceeded`** error when the number of pending launches exceeds **`cudaLimitDevRuntimePendingLaunchCount`**. Instead, the runtime automatically extends the pending launch buffer beyond **`cudaLimitDevRuntimePendingLaunchCount`**, albeit with a performance penalty.
- Support for the following Linux distributions has been added as of CUDA 6.0: Fedora 19, Ubuntu 13.04, CentOS 5.5+, CentOS 6.4, OpenSUSE 12.3, and SLES SP11.
- Support for the ICC Compiler has been upgraded to version 13.1.
- Support for the Windows Server 2012 R2 operating system has been added as of CUDA 6.0.
- RDMA (remote direct memory access) for GPUDirect is now supported for applications running under MPS (Multi-Process Service).
- CUDA Inter-Process Communication (IPC) is now supported for applications running under MPS. CUDA IPC event and memory handles can be exported and opened by the MPS clients of a single MPS server.
- Applications running under MPS can now use **`assert()`** in their kernels. When an **`assert`** is triggered, all work submitted by MPS clients will be stalled until the **`assert`** is handled. The MPS client that triggered the **`assert`** will exit, but will not interfere with other running MPS clients.

- Previously, a wide variety of errors were reported by an "Unspecified Launch Failure (ULF)" message or by the corresponding error codes **CUDA_ERROR_LAUNCH_FAILED** and **cudaErrorLaunchFailed**. The CUDA driver now supports enhanced error reporting by providing richer error messages when exceptions occur. This will help developers determine the causes of application faults without the need of additional tools.
- Man page documentation has been added for the following:
  - All the libraries, at a high level
  - All of the tools binaries, at a high level
  - The CUDA driver API, in detail
  - The CUDA runtime API, in detail

# 2.2. CUDA Tools

## 2.2.1. CUDA Compiler

- Parameters with reference type are now supported for **__global__** functions. If the object bound to the reference is read or written from the GPU during the execution of the **__global__** function, the object must reside in memory that is accessible from the GPU.

## 2.2.2. CUDA-GDB

- The code base for CUDA-GDB was upgraded from GDB 7.2 to GDB 7.6, which enables support for DWARF 4 and thus enables distributions using GCC 4.8. GDB 7.6 also provides better ARM debugging support: allowing single stepping between 32-bit and 16-bit instructions.
- To mitigate the issue of variables not being accessible at some code addresses, the debugger offers two new options. With **set cuda value_extrapolation**, the latest known value is displayed with (possibly) a prefix. With **set cuda ptx_cache**, the latest known value of the PTX register associated with a source variable is display with the (cached) prefix.
- It is now possible for the **set cuda break_on_launch** option to break on kernels launched from the GPU. Also, enabling this option does not disable deferred kernel launch notifications.
- Remote debugging has been made considerably faster: up to two orders of magnitude. Local debugging is also considerably faster.
- CUDA-GDB can now use optimized methods to single-step a program; these accelerate single-stepping most of the time. This feature can be disabled with **set cuda single_stepping_optimizations off**.

## 2.2.3. CUDA-MEMCHECK

- The CUDA-MEMCHECK tool now works with applications that use the multi-process server (MPS) to share a single GPU among multiple processes.

### 2.2.4. CUDA Profiling Tools Interface (CUPTI)

- The nvprof profiling tool now works with multiple processes that are sharing a single GPU with the multi-process server (MPS).

### 2.2.5. Nsight Eclipse Edition

- Nsight Eclipse Edition now supports developing (building, debugging, and profiling) CUDA applications on remote systems.

### 2.2.6. NVIDIA Visual Profiler

- The Visual Profiler guided analysis system has been updated with several enhancements.
  - It now includes a side-by-side source and disassembly view annotated with instruction execution counts, inactive thread counts, and predicated instruction counts. This view enables you to find hotspots and inefficient code sequences within your kernels.
  - It has been updated with several new analysis passes: (1) kernel instructions are categorized into classes so that you can see if the instruction mix matches your expectations, (2) inefficient shared memory access patterns are detected and reported, and (3) the per-SM activity level is presented to help you detect detect load-balancing issues across the blocks of your kernel.
  - It can now generate a kernel analysis report. The report is a PDF version of the per-kernel information presented by the guided analysis system.

## 2.3. CUDA Libraries

### 2.3.1. cuBLAS Library

- The cuBLAS library now supports execution of Level-3 BLAS routines out-of-core (that is, out of system memory, even if the total data is too large to fit completely in GPU memory) and across two GPUs in parallel. This functionality is exposed via the cublasXt interface. Note that the GPUs must both be on the same GPU board, such as a Tesla K10 or GeForce GTX 690; furthermore, with the appropriate licensing, more than two GPUs on the same board can be supported (seehttps://developer.nvidia.com/cublasxt).
- CUDA 6.0 includes a new library called NVBLAS, which intercepts certain calls to a host BLAS implementation and routes them to cuBLAS via the cublasXt interface. For an existing application that makes use of BLAS calls, NVBLAS allows for drop-in GPU acceleration with only relinking.
- Conversion LAPACK routines **`cublas{T}tpttr`** and **`cublas{T}trttp`**, which convert from Packed Triangular format to Triangular and to Packed Triangular format from Triangular, have been added to cuBLAS.

### 2.3.2. cuFFT Library

- The cuFFT library now supports execution of batched and single FFTs across two GPUs by using the cufftXt interface. Note that the GPUs must both be on the same GPU board, such as a Tesla K10 or GeForce GTX 690. As of CUDA 6.0, all types of batched transforms are supported, with the exception that strides are not allowed. For single transforms, C2C/Z2Z 1-D, 2-D, and 3-D transforms with power-of-2 sizes are supported.

### 2.3.3. cuRAND Library

- In CUDA 6.0, support for a new random number generator, Mersenne Twister 19937, has been added. This generator is widely recognized to be very efficient at producing random variates with excellent statistical properties. The generator is callable from the host and generates results in GPU memory. A kernel-callable interface is not available at this time.

### 2.3.4. cuSPARSE Library

- The routines **`cusparse(X)bsrsv2`**, **`cusparse(X)bsric02`**, and **`cusparse(X)bsrilu02`** have been added to the cuSPARSE library. They perform triangular solve, incomplete Cholesky factorization, and incomplete LU factorization on the matrices stored in block CSR format, respectively. The routines handle arbitrary block sizes; moreover, the user can query the size of working space used internally and pre-allocate the buffer, passing a pointer to it to the routines. If the matrix becomes numerically singular (during factorization), the user can query its properties and use the boost value to proceed with computation.

### 2.3.5. CUDA Math Library

- Support for over 80 new SIMD instructions (**`__v*2()`** and **`__v*4()`**, particularly useful in video processing) has been added to the math library.
- Functions **`rhypot()`** and **`rhypotf()`** have been added to the math library to support reciprocal hypotenuse, which is particularly useful for robust 2-D vector normalization.
- Support for **`cyl_bessel_i0{f}()`** and **`cyl_bessel_i1{f}()`**, modified Bessel functions of the first kind of order 0 and order 1 respectively, which are widely used in computations for physical phenomena, has been added to the math library.

### 2.3.6. NVIDIA Performance Primitives (NPP)

- The NPP library in the CUDA 6.0 release contains more than 500 new image and signal processing primitives, including BGR/YUV conversions, color transformations using a 3D LUT with trilinear interpolation, optimized Huffman coding for JPEG, filter median routines, error metric computations, and other miscellaneous color and pixel routines.

- In CUDA 6.0, NPP adds a new primitive, **`FilterMedian`**, to filter the image using a median filter. The new primitive supports variants with different data types and channel settings.

## 2.4. CUDA Samples

- (Windows) The HTML and PDF documentation for the CUDA Samples, available on all platforms, replaces the former CUDA Samples Browser, which was available only on Windows. The documentation provides a listing of the samples similar to that of the CUDA Samples Browser and also provides links to download individual samples.

# 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.

**CUDA Profiling Tools Interface**

CUPTI users should adopt the new asynchronous activity buffering API implemented by `cuptiActivityRegisterCallbacks()`, `cuptiActivityFlush()`, and `cuptiActivityFlushAll()`. Further information can be found in the CUPTI documentation.

**cuSPARSE "Legacy" API**

Please use the cuSPARSE API in cusparse.h. Further information on the new cuSPARSE API can be found in the cuSPARSE library documentation.

**Linux Distributions That Are No Longer Supported as of CUDA 6.0**

The following Linux distributions are no longer supported: Fedora 18, OpenSUSE 12.2, SLES 11 SP1, Ubuntu 12.10, and Ubuntu 10.04.

**Linux Compilers That Are No Longer Supported as of CUDA 6.0**

ICC version 12.1 is no longer supported, but a newer version of ICC is supported.

**Mac Operating Systems That Are No Longer Supported as of CUDA 6.0**

OS X 10.7.*x* is no longer supported.

**Mac Models with the MCP79 Chipset**

The CUDA 6.0 toolkit and r331 driver do not work on Macs with GPUs based on the MCP79 chipset. To continue developing or running CUDA applications, please use the CUDA 5.5 toolkit or r319 driver, as appropriate. The affected Mac models include the following:

- iMac (20-inch, Early 2009)
- iMac (24-inch, Early 2009)
- iMac (21.5-inch, Late 2009)
- MacBook Air (Late 2008, Mid 2009)
- MacBook Pro (17-inch, Early 2009)
- MacBook Pro (17-inch, Mid 2009)
- MacBook Pro (15-inch, Late 2008)
- MacBook Pro (15-inch, Mid 2009)
- MacBook Pro (15-inch 2.53 GHz, Mid 2009)
- MacBook Pro (13-inch, Mid 2009)
- Mac mini (Early 2009)
- Mac mini (Late 2009)

# 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.

**Windows XP 64-bit Edition Support**

Support for CUDA on the 64-bit version of the Windows XP operating system is deprecated in this CUDA release and will be dropped in the next major release. CUDA on the 32-bit version of Windows XP is still supported. We recommend that developers and users on 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**

Support for CUDA on the Windows Vista operating system is deprecated in this CUDA release and will be dropped in the next major release. 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**

Support for CUDA on the Windows Server 2012 operating system is deprecated in this CUDA release and will be dropped in the next major release. We recommend that users and developers migrate to Windows Server 2012 R2, which is supported in the current and future releases.

**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.

**(Mac) Developing and Running 32-bit CUDA and OpenCL Applications on OS X Platforms**

Support for developing and running 32-bit CUDA and OpenCL applications on OS X 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 OS X applications.

**Targeting G80 (sm_10) for CUDA and OpenCL Applications**

Support for targeting the G80 architecture (sm_10) for CUDA and OpenCL applications is deprecated. This implies the following:

- Support is currently still available in the toolkit.
- Support may be dropped from the toolkit in a future release.
- Support in the driver for running sm_10 binaries is unaffected in this release by the deprecation.
- New features may not have support for sm_10.

**CUDA Video Encoder**

Support for building applications with the CUDA Video Encoder interface, NVCUVENC, is deprecated and will be dropped in a subsequent release. This functionality is still available in the current release of the CUDA Toolkit, and the driver will continue to run applications built against this interface. Moving forward, we recommend using a newer video encoding interface, NVENC, which is available at https://developer.nvidia.com/nvidia-video-codec-sdk.

# Chapter 5. PERFORMANCE IMPROVEMENTS

## 5.1. General CUDA

- The latency of launching a child kernel (that is, grid) from the GPU using Dynamic Parallelism has been reduced.

# Chapter 6. RESOLVED ISSUES

## 6.1. General CUDA

- In the CUDA 6.0 release, a Dynamic Parallelism kernel will not be launched if an illegal stream handle is used. Although previous releases would set the proper error code in the case of an invalid stream handle, a kernel could still be launched and in some cases complete and generate its output results. The new version of the Dynamic Parallelism runtime behaves differently and does not launch the kernel.
- (Mac) The initial CUDA 5.5 driver did not support Tesla-class (sm_1x architecture) GPUs on Mac OSX 10.9. This was fixed by the CUDA 6.0 Production Release driver on Mac OSX 10.9.

## 6.2. CUDA Tools

### 6.2.1. CUDA Compiler

- *The following issue has been fixed in CUDA 6.0.* On the GK110, the kernel occasionally may produce incorrect results when, either by loop unrolling or straight lines of code, there are more than 63 outstanding texture/LDG instructions at one point during the program execution.

### 6.2.2. Nsight Eclipse Edition

- Before Nsight 6.0, the CPU architecture was set in the linker and compiler properties. Now this setting is configurable on a per-target system basis.

# 6.3. CUDA Libraries

## 6.3.1. cuBLAS Library

- *The following issue has been fixed in CUDA 6.0.* In CUDA 5.0 and CUDA 5.5, the cuBLAS routine **`SGEMM()`** for operations **`NN`** and **`NT`** can give wrong results on Kepler Architecture SM35 when the following conditions are met :

  **`4 * ldc * n >= 2^32`** and **`m >= 256`**

  where **`m`**, **`n`**, and **`ldc`** are respectively the number of rows, the number of columns, and the leading dimension of the resulting matrix **`C`**.

## 6.3.2. NVIDIA Performance Primitives (NPP)

- In CUDA 6.0, NPP routines are fully validated on the ARM platform. Please report any functional errors via the CUDA registered developer website.
- In CUDA 6.0, NPP has improved the stability of JPEG-codec-related functions in multi-threaded and multi-GPU applications. This includes the DCT and quantization primitives.
- Fixed a bug in the progressive refine encoding mode.
- Optimized the performance of the CPU Huffman decoder code.

# 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 device memory heap size, set using **`cudaDeviceSetLimit(cudaLimitMallocHeapSize, *)`** or **`cuCtxSetLimit(CU_LIMIT_MALLOC_HEAP_SIZE, *)`**, is currently limited to a size of 4,294,967,296 (4 GB).
- 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.
- The Unified Memory sample has a known incompatibility with the Exclusive Thread mode that can be set via **`nvidia-smi`**. Using the two together causes the sample not to run correctly.
- The **`libfreeimage-dev`** package must be manually installed on ARM hosts by using **`sudo apt-get install libfreeimage-dev`**. Without this step, certain samples that depend on freeimage will fail to build.
- Peer access is disabled between two devices if either of them is in SLI mode.

- Documentation on the CUDA Multi-Process Service (MPS) is only available online at http://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf. A few new features have been enabled in CUDA 6.0 for MPS applications:
  - MPS applications are now supported by **cuda-memcheck**, which will detect any MPS-specific execution errors.
  - Visual Profiler and **nvprof** also support profiling MPS applications and have the ability to import profile data from multiple CUDA applications.
  - Device assert, CUDA IPC, and GPUDirect RDMA can also be used in MPS applications.
- To revert an NVIDIA driver that is version 331.17 or newer to one that is earlier than 331.17, the current driver must be manually removed by running **nvidia-uninstall** before the earlier version is installed.
- The new **__managed__** variables in CUDA 6.0 create global-scope symbols accessible from both device and host code. In a program using the runtime API, because these variables are valid for CPU access, their lifetime is that of the whole program instead of the CUDA context that contains them. Unlike **__device__** variables, their value should persist across calls to **cudaDeviceReset()**. For the CUDA 6.0 Early Access release, however, calling **cudaDeviceReset()** destroys the storage for **__managed__** variables, causing CPU access to fail.

  For programs using the driver API, the lifetime of **__managed__** variables is always limited to the context in which they were created, so no persistence is expected.
- 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**.

# 7.3. CUDA Tools

## 7.3.1. CUDA Compiler

- (Mac) 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) 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 work around for this issue is to manually add padding so that the structs between the host compiler and CUDA are consistent.

### 7.3.2. NVIDIA Visual Profiler

- (Windows) Using the mouse wheel button to scroll does not work within the Visual Profiler on Windows.
- The Visual Profiler cannot correctly import profiler data generated by **`nvprof`** when the option **`--kernels <kernel filter>`** is used. Visual Profiler reports a warning, "Some collected events or source-level results could not be associated with the session timeline." One workaround is to use the **`nvprof`** option **`--kernels :::1`** to profile the first invocation for all kernels.
- (Mac) Visual Profiler events and metrics do not work correctly on OS X 10.8.5. Please upgrade to OS X 10.9.2 to use Visual Profiler events and metrics.