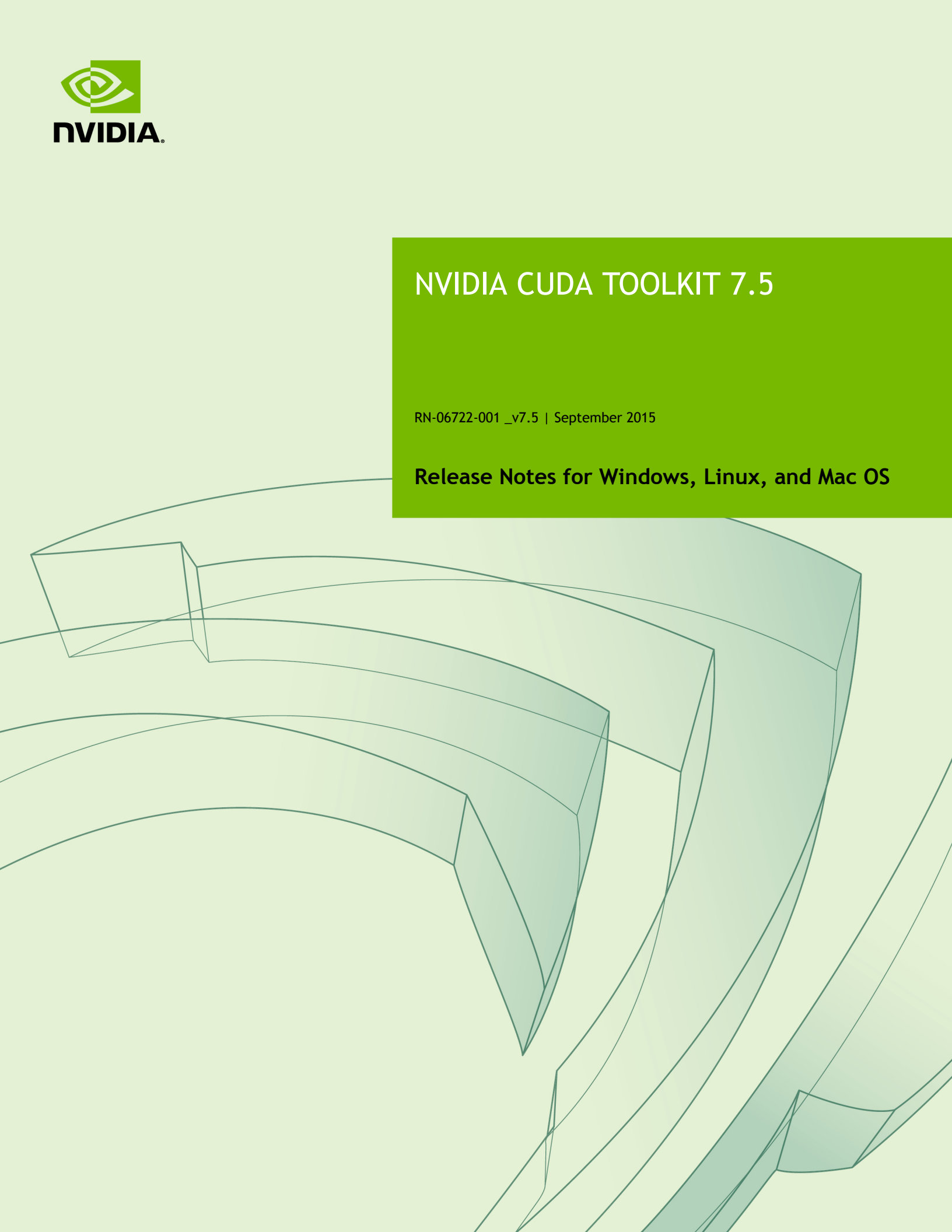

NVIDIA CUDA TOOLKIT 7.5

RN-06722-001 _v7.5 | September 2015

Release Notes for Windows, Linux, and Mac OS

TABLE OF CONTENTS

ERRATA

Unsupported and Deprecated Features

Unsupported Features

Cross-building from x86_64 to ppc64le is no longer supported.

Resolved Issues

General CUDA

- The **`--no-recommends`** option must be used when installing the driver packages without GL on OpenSUSE or SLES.

CUDA Tools

- To build the NPP samples in Nsight Eclipse Edition, please either install the libfreeimage package or use the identical samples from the CUDA 7.0 SDK.

Known Issues

General CUDA

- Enabling per-thread synchronization behavior [http://docs.nvidia.com/cuda/cuda-runtime-api/stream-sync-behavior.html#stream-sync-behavior__per-thread-default-stream] does not work correctly with the following CUDA runtime routines, which use the legacy synchronization behavior [http://docs.nvidia.com/cuda/cuda-runtime-api/stream-sync-behavior.html#stream-sync-behavior__legacy-default-stream]:
  - **`cudaMemcpyPeer()`**
  - **`cudaMemcpyPeerAsync()`** with a **`NULL`** stream argument
  - **`cudaGraphicsMapResources()`** with a **`NULL`** stream argument
  - **`cudaGraphicsUnmapResources()`** with a **`NULL`** stream argument

To work around this issue, use **cudaMemcpyPeerAsync()** with the **cudaPerThreadStream** stream argument instead of **cudaMemcpyPeer()**, and pass the **cudaPerThreadStream** argument instead of **NULL** to the other routines.

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`**, **`gwiz`**

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])
- **nvrtc** (CUDA Runtime Compilation)
- **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. Also, the **version.txt** file in the root directory of the toolkit will contain the version and build number of the installed toolkit.

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.

CUDA-GDB Sources

CUDA-GDB sources are available as follows:

- For CUDA Toolkit 7.0 and newer, in the installation directory **extras/**. The directory is created by default during the toolkit installation unless the **.rpm** or **.deb** package installer is used. In this case, the **cuda-gdb-src** package must be manually installed.
- For CUDA Toolkit 6.5, 6.0, and 5.5, at https://github.com/NVIDIA/cuda-gdb.
- For CUDA Toolkit 5.0 and earlier, at ftp://download.nvidia.com/CUDAOpen64/.
- Upon request by sending an e-mail to mailto:oss-requests@nvidia.com.

Chapter 2. NEW FEATURES

2.1. General CUDA

- CUDA applications can now run on Windows systems when launched from within a Remote Desktop session. They can also be run as Windows service applications. Note: the CUDA-GL interop APIs are only supported via Remote Desktop on GPUs that also support OpenGL via Remote Desktop, such as the Quadro line of GPUs.
- The **`cudaStreamCreateWithPriority()`** function is now supported on all GPUs, not just Tesla and Quadro GPUs.
- The Tesla Compute Cluster mode is now supported on all Tesla, Quadro, and GeForce GTX TITAN GPUs. Previously, it was only supported on Tesla and some Quadro GPUs.
- Added two topology functions to NVML and the NVIDIA-SMI topo command. The NVML functions are listed below:
  - Function **`nvmlSystemGetGpuSet()`**: discovers the set of GPUs that are on the same IO-HUB (root complex) for a given CPU socket.
  - Function **`nvmlDeviceGetNearestDevices()`**: discovers the set of GPUs that are closest to a given GPU. The closest connection is determined by the input device handle and its hierarchy in the topology, that is, a GPU that shares a Gemini board, is on the same PLX switch, has a direct PCI link, or is connected to the same host bridge.

  These functions help applications optimize data access and memory transfers between GPUs and/or supported I/O devices.

2.2. CUDA Tools

2.2.1. CUDA Compilers

- Compilers **`nvcc`** and **`nvrtc`** predefine the following macros: **`__CUDACC_VER_MAJOR__`**, **`__CUDACC_VER_MINOR__`**, **`__CUDACC_VER_BUILD__`**,

and **__CUDACC_VER__** to help users identify the compiler version in the source code. Macros **__CUDACC_VER_MAJOR__**, **__CUDACC_VER_MINOR__**, and **__CUDACC_VER_BUILD__** are defined with the compiler major, minor, and build version numbers respectively; **__CUDACC_VER__** is defined by **__CUDACC_VER_MAJOR__ * 10000 + __CUDACC_VER_MINOR__ * 100 + __CUDACC_VER_BUILD__**.

- The PTX ISA has been extended to support the **lop3** instruction, which can perform arbitrary logic operations involving three inputs.
- The **-warn-spills** and **-warn-lmem-usage** options have been added to the **ptxas** compilation stage to provide warnings when the compiler needs to spill values from registers, or if there is any local memory usage, respectively.
- Added a command line option to **ptxas** to give warnings if the input **ptx** is doing double-precision computations. The long form of the option is **--warn-on-double-precision-use** and the short form is **-warn-double-usage**.
- The **nvcc** compiler has added an experimental feature to define a **__device__** annotated C++11 lambda function on the host and pass it to a device kernel. This feature is enabled by the **--std=c++11 --expt-extended-lambda** command-line options to **nvcc**.
- The **clang** LLVM-based C and C++ compiler versions 3.5 and 3.6 are now supported as a host-compiler by **nvcc** on Linux operating systems. Please see the *Linux Installation Guide* for specific version support.

2.2.2. CUDA Profiler

- Functionality for CPU sampling and call stack tracing has been added to **nvprof**, and is enabled by the command line option **--cpu-profiling on**. This new option is a work in progress, and in the CUDA 7.5 release is not yet integrated with the GPU profiling functionality of **nvprof**. Similarly, CPU profiling information is not yet visible in **nvvp**.
- Enabled GPU PC sampling with instruction-level analysis of application bottlenecks.

2.2.3. GPU Wizard

- The GPU Wizard is available as a standalone tool and is also bundled with CUDA Toolkit releases. The Wizard instruments the target application and detects BLAS, FFT, and other common math routines. It then provides runtime analysis and reports potential speedups for those library routines if they used GPU acceleration via NVIDIA math libraries. In addition, the tool also supports hotspot detection and the corresponding call trace to help with performance tuning. The Wizard is supported on Windows in this release.

2.2.4. Nsight Eclipse Edition

- NSight Eclipse Edition is now supported on the IBM Power8 platform.
- Nsight Eclipse Edition supports multiple toolkit versions. This enables the latest version of Nsight, shipped with CUDA 7.5, to support cross-compilation and remote profiling on platforms, such as Jetson TK1, that are on older versions of CUDA.

2.3. CUDA Libraries

2.3.1. cuBLAS Library

- The routine **`cublasHgemm()`** has been added to support half-precision floating point (FP16). This routine is only supported for GPU architectures **`sm_53`** or greater.
- The cuBLAS Library has an extension of SGEMM called **`cublasSgemmEx()`** that supports different data formats (FP16 and INT8) in input and/or output while the computation is still done in single-precision floating point. Please refer to the documentation for the description of the supported formats.

2.3.2. cuFFT Library

- Two new cuFFT functions have been added to support the specification of transforms on arrays spanning more than 4G elements. Functions **`cufftMakePlanMany64()`** and **`cufftGetSizeMany64()`** are identical to the 32-bit versions, except that all of the dimensions and strides are specified as 64-bit integers.

2.3.3. cuRAND Library

- The **`sobol_direction_vectors.h`** header file, which allowed developers to employ the cuRAND device API with sobol distributions, has been removed. The file was removed in favor of the **`curandGetDirectionVectors{32,64}()`** and **`curandGetScrambleConstants{32,64}()`** functions, which return memory pointers to the direction vectors that are precompiled into the cuRAND library. These pointers can then be used to copy the vectors to the GPU device memory.

  Sample code demonstrating this approach is included in the cuRAND documentation. Please see the CUDA 7.5 *cuRAND Library Programming Guide* for more details.

2.3.4. cuSPARSE Library

- The cuSPARSE library now supports the **`cusparse{S,D,C,Z}gemvi()`** routine, which multiplies a dense matrix by a sparse vector.

2.3.5. CUDA Math Library

- Support for the *n*-dimensional Euclidean norm and the four-dimensional and *n*-dimensional Euclidean reciprocal norms has been added to the math library.
- A new header file, **`cuda_fp16.h`**, provides an interface for the 16-bit floating point format. This interface includes scalar and vector datatypes, **`half`** and **`half2`**, as well as conversion functions to and from 32-bit floating point (**`float`**) format. In addition, basic arithmetic, comparison, and data movement operations are provided on architectures with compute capability 5.3 (**`sm_53`**) and higher.

2.3.6. NVIDIA Performance Primitives (NPP) Library

- Bayer CFA-to-RGB color conversion routines, **`nppiCFAToRGB*()`**, have been added to the NPP library.

2.4. CUDA Samples

- CUDA samples were added to illustrate usage of the cuSOLVER library.

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.

General CUDA

- **The `sobol_direction_vectors.h` Header File**

  The **`sobol_direction_vectors.h`** header file, which allowed developers to employ the cuRAND device API with sobol distributions, has been removed.

CUDA Tools

- **CUDA-GDB on Mac OS X**

  Debugging GPGPU code using **`cuda-gdb`** is no longer supported on the Mac platform.
- **Developer Tools Support for 32-bit Applications on 64-bit Linux Systems**

  This has no impact on Windows or Mac 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.

General CUDA

- **Double Arithmetic Support**

  All CUDA supported devices now natively support double arithmetic. As a result, `cudaSetDoubleForDevice()` and `cudaSetDoubleForHost()` are no longer needed and are deprecated.
- **Ctrl+C Behavior with MPS Server**

  When a particular process is killed via `SIGINT` (Ctrl+C), the MPS server behavior has been to allow other processes to continue execution. This behavior is being deprecated. In a future CUDA release, when any process that is connected to the MPS server is killed before its outstanding GPU work has completed, other processes connected to the same MPS server may not be able to continue issuing CUDA commands.
- **Exclusive Thread Compute Mode**

  The `EXCLUSIVE_THREAD` option for the `nvidia-smi` compute-mode setting is deprecated and will be unsupported in a future version of the 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 Tools

- **Microsoft Visual Studio 2010**

  Support for the compiler in Microsoft Visual Studio 2010 (VC++ 10.0) is deprecated as of CUDA 7.5 and will be discontinued in a future release of the CUDA Toolkit.

- **Compatibility Modes in cuFFT 7.5**

  In cuFFT 7.5, compatibility modes different from **CUFFT_COMPATIBILITY_FFT_PADDING** are deprecated. In the next major cuFFT release, the function **cufftSetCompatibilityMode()** will no longer accept the following values for the mode parameter: **CUFFT_COMPATIBILITY_NATIVE**, **CUFFT_COMPATIBILITY_FFTW_ALL**, and **CUFFT_COMPATIBILITY_FFT_ASYMMETRIC**. The error code **CUFFT_NOT_SUPPORTED** will be returned in each case.

CUDA Libraries

- **Compatibility Modes in cuFFT 7.5**

  In cuFFT 7.5, compatibility modes different from **CUFFT_COMPATIBILITY_FFT_PADDING** are deprecated. In the next major cuFFT release, the function **cufftSetCompatibilityMode()** will no longer accept the following values for the mode parameter: **CUFFT_COMPATIBILITY_NATIVE**, **CUFFT_COMPATIBILITY_FFTW_ALL**, and **CUFFT_COMPATIBILITY_FFT_ASYMMETRIC**. The error code **CUFFT_NOT_SUPPORTED** will be returned in each case.

Chapter 5. PERFORMANCE IMPROVEMENTS

5.1. CUDA Tools

5.1.1. CUDA Compilers

- Improved the performance of the `sqrt` macro expansion. The speed has increased by 5–10%, and marginal values are handled at least 40% more efficiently.

Chapter 6. RESOLVED ISSUES

6.1. General CUDA

- It is no longer necessary to shut down the nvidia-persistenced daemon prior to the removal of the NVIDIA driver debian packages.
- The POWER8 driver installation no longer overrides the mesa GL alternative.
- Exceptions caused by MPS clients are now propagated to all the other clients. Once all clients have seen the exception and have detached from the MPS server, the server exits.
- The surface object read and write intrinsics for cubemap layered surfaces, **`surfCubemapLayeredread()`** and **`surfCubemapLayeredwrite()`**, erroneously took an extraneous argument, **`int z`**. This argument has been removed from the interface of these functions, and any application using these functions needs to be modified to compile successfully with this version of the CUDA Toolkit.

6.2. CUDA Tools

6.2.1. CUDA Compilers

- The CUDA disassembler, **`nvdisasm`**, now correctly prints live range analysis information for Maxwell GPUs (SM 5.x).
- When C++11 code (**`-std=c++11`**) is compiled on Linux with **`gcc`** as the host compiler, invoking **`pow()`** or **`std::pow()`** from device code with (**`float`**, **`int`**) or (**`double`**, **`int`**) arguments now compiles successfully.
- Device linking (in order to use separate compilation) was causing severe performance degradation when constant memory was used.

6.2.2. CUDA Profiler

- The profiler no longer fails to collect events or metrics when application replay mode is turned on for an application that uses CUDA driver APIs to launch the kernel.

6.2.3. CUDA Profiling Tools Interface (CUPTI)

- The **`cuptiActivityConfigurePCSampling()`** function is not supported, and therefore the PC sampling period cannot be changed. The PC sampling period used during sampling is given in the **`samplingPeriodInCycles`** field of the **`CUpti_ActivityPCSamplingRecordInfo`** record.
- The double-precision flops-per-cycle device attribute and the **`flop_dp_efficiency`** metric values reported by the profiler are now correct.

6.3. CUDA Libraries

6.3.1. cuFFT Library

- Fixed a known issue in the cuFFT library that could produce incorrect results for certain input sizes, if they were less than or equal to 1920 in any dimension, when **`cufftSetStream()`** was passed a non-blocking stream.
- The static library version of cuFFT had several known issues that were manifested only when execution was on a Maxwell GPU (**`sm_50`** or higher) and when a transform contained sizes that factor to primes in the range of 67–127. This has been fixed.

6.3.2. Thrust Library

- On the SLES 11 Linux distribution, an issue that causes the **`segmentationTreeThrust`** CUDA sample in the **`6_Advanced`** directory to fail has been fixed.
- The version of Thrust included with CUDA 7.5 has been upgraded to Thrust v1.8.2. Note that CUDA 7.0 shipped with Thrust v1.8.1. A changelog of the bugs fixed in v1.8.2 can be found at https://github.com/thrust/thrust/blob/1.8.2/CHANGELOG.

6.4. CUDA Samples

- With multiple devices, the N-Body CUDA sample rounds up the number of bodies per device to (number of SMs * 256). This can lead to issues where the last GPU has little or no work, leading to a performance drop and/or a kernel launch failure. To resolve this, line 103 in file **`bodysystemcuda_impl.h`** should be modified from **`unsigned int round = numSms[i] * 256`** to **`unsigned int round = 256`**.

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.
- Unified memory is not currently supported with IOMMU. The workaround is to disable IOMMU in the BIOS. Please refer to the vendor documentation for the steps to disable it in the BIOS.
- The latest version of Xcode (Xcode 6.3) is not compatible with this CUDA release. To avoid this issue, download an older version of Xcode, such as Xcode 6.2, from https://developer.apple.com/downloads, copy it to a version-specific location within **`/Applications`**, such as **`/Applications/Xcode_6.2.app`**, and select it by running **`sudo xcode-select -s /Applications/Xcode_6.2.app/Contents/Developer`**.

7.2. CUDA Tools

7.2.1. CUDA Profiling Tools Interface (CUPTI)

- The **`cuptiActivityConfigurePCSampling()`** function is not supported, and therefore the PC sampling period cannot be changed. The PC sampling period used during sampling is given in the **`samplingPeriodInCycles`** field of the **`CUpti_ActivityPCSamplingRecordInfo`** record.
- The double-precision flops-per-cycle device attribute and the **`flop_dp_efficiency`** metric values reported by the profiler are now correct.
- The CUDA 7.5 PC Sampling profiling tool fails on Windows Server 2012.

7.3. CUDA Libraries

7.3.1. Thrust Library

- On the SLES 11 Linux distribution, there is a known issue that causes the TestGetTemporaryBufferDispatchExplicit and TestGetTemporaryBufferDispatchImplicit unit tests provided with the Thrust library to fail.

Acknowledgment

NVIDIA extends thanks to Professor Mike Giles of Oxford University for providing the initial code for the optimized version of the device implementation of the double-precision `exp()` function found in this release of the CUDA toolkit.