NVIDIA CUDA TOOLKIT V5.5 FOR POWER8

RN-06722-001 _v5.5 for POWER8 | February 2015

Release Notes for Linux

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. NVIDIA CUDA TOOLKIT V5.5 FOR POWER8 RELEASE NOTES

1.1. Release Overview

This is the first release of the CUDA development environment for the POWER8 architecture. It is based on the CUDA 5.5 release with the following differences

- **`cuda-gdb`**, **`cuda-gdbserver`**, and **`libnpp*.so`** are based on the CUDA 6.0 release.
- **`nsight`** and **`nvvp`** are not included in this release.
- OpenGL interop and VDPAU interop are not supported.
- The minimum driver version for the toolkit is 340.50. (NVIDIA launched CUDA 5.5 with 319.37 originally)
- The driver for POWER8 does not include any of the graphics drivers, video drivers, or GUI based utilities.
- **`nvidia-persistenced`** is not included in the driver for POWER8.
- UVM is not supported for POWER8 in this release.
- RDMA/GPUDirect is not supported for POWER8 in this release.
- Cross developement to POWER8 with an existing CUDA 5.5 toolkit is not supported.
- Code generation for **`sm_1x`** class GPUs is not supported.

Please refer to Linux sections in the CUDA 5.5 release notes in the next chapter for detailed information about the original CUDA 5.5 release.

1.2. Errata

1.2.1. CUDA Tools

- The libNVVM library no longer sets up signal handlers, removing any potential interference with the use of signals in application code that calls libNVVM.
- Profiler will fail with an internal error if the application suffers a GPU assert during profiling.

1.2.2. CUDA Samples

- When using the IBM XL C/C++ compiler as the host compiler with nvcc, the cudaOpenMP sample fails to build, as the compiler is unable to locate the **`omp.h`** header file.

1.3. Supported NVIDIA Hardware

The Tesla K40m is the only supported adapter for the POWER8 architecture.

1.4. Supported Operating Systems

1.4.1. Linux

The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for this CUDA Toolkit release.

Table 1 Linux Distributions Supported in CUDA 5.5 beta for POWER8

Distribution	32	64	Kernel	GCC	GLIBC
Ubuntu 14.10		x	3.16.0-10-generic	4.9.1	2.19

1.5. New Features

1.5.1. CUDA Tools

1.5.1.1. CUDA Compiler

- **`nvcc`** is compatible with **`gcc 4.9.x`** as the underlying host compiler.
- **`nvcc`** is compatible with IBM XL C/C++ for Linux, V13.1.1 (**`xlc`** and **`xlC`**) as the underlying host compiler.

Chapter 2. NVIDIA CUDA TOOLKIT V5.5 RELEASE NOTES

2.1. Errata

2.1.1. General CUDA

- The initial driver supporting CUDA 5.5 does not support Tesla-class (sm_1x architecture) GPUs on Mac OSX 10.9. This will be fixed in a future release of the CUDA driver on Mac OSX 10.9.
- Section 1.8.1, General CUDA, in the original version of these release notes had an item that described a CUDA 5.5 Toolkit restriction involving -Xlinker. That item has been replaced by the following:
  - With the CUDA 5.5 Toolkit, any objects that contain device code must be passed to both the device and host linkers. This was recommended practice before, but is now required.
  - For projects that use -Xlinker with nvcc, you must ensure the arguments after -Xlinker are quoted. In CUDA 5.0, **`-Xlinker -rpath /usr/local/cuda/lib`** would succeed; in CUDA 5.5 **`-Xlinker "-rpath /usr/local/cuda/lib"`** is now necessary.
- Visual Studio 2012 projects initially depended on the Visual Studio 2010 compiler being installed. When a user who only has VS2012 installed on the system opens **`samples_vs2012.sln`** for the first time, and the VS2012 projects are upgraded from VS2010 (with the original VS2010 compiler settings) to the VS2012 compiler, the CUDA CDP samples will fail to build with fatal error "LNK1319".

  Two fixes address not only the CDP samples, but all other samples that are built with VS2012.

  - The first fix addresses this bug explicitly: remove **`_MSC_VER=1600;`** from **`cdpSimplePrint_vs2012.vcxproj`**.

- The second fix addresses VS2012-only systems that are not properly building the CUDA samples: in **`*_vs2012.vcxproj`** files, search and replace **`<PlatformToolset>v100</PlatformToolset>`** with **`<PlatformToolset>v110</PlatformToolset>`**.

2.1.2. CUDA Libraries

2.1.2.1. CUBLAS

- The routine **`cublas<T>syrkx()`** has been added to the CUBLAS Library. This routine, which is a variation of **`cublas<T>syrk()`**, can be used advantageously to replace multiple calls to **`cublas<T>syr()`** where a different scalar **`alpha`** is applied to each vector. Those vectors would form the matrix **`A`**, and a second matrix, **`B`**, would be constructed by the same vectors, each scaled by the different **`alpha`**. Matrix **`B`** can be seen as the product of **`A`** with a diagonal matrix formed by the different **`alpha`** scalars, and it can be easily computed using **`cublas<T>dgmm()`**.
- 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`**.

2.1.2.2. CUFFT

There were a number of CUFFT documentation errors in CUDA 5.5.

- Some values of the enumerated type **`cufftResult`** are in error or are missing. Values 0 through 10 are correct, values 11 through 13 are as follows:

```
CUFFT_INCOMPLETE_PARAMETER_LIST = 10, //Internal plan configuration error
CUFFT_INVALID_DEVICE = 11,            //Execution of a plan was on a
                                      // different GPU than plan creation.
CUFFT_PARSE_ERROR = 12,               //Internal plan database error
CUFFT_NO_WORKSPACE = 13               //No workspace has been provided
                                      // prior to plan execution.
```

- The arguments for **`cufftMakePlan1d()`** are incorrect. The plan is a **`cufftHandle`** returned from a prior call to **`cufftCreate()`**. It is an input parameter only. There is an additional output parameter, which receives the size of the workspace required by the plan. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftMakePlan1d(cufftHandle plan, //Handle returned
                                                       // by cufftCreate
    int nx,            //Transform size
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    int batch,         //Number of transforms of size nx;
                       // deprecated—use cufftPlanMany.
    size_t *workSize); //Size of work area for the transform
```

- The arguments for c**`ufftMakePlan2d()`** are incorrect. The plan is a **`cufftHandle`** returned from a prior call to **`cufftCreate()`**. It is an input parameter only. There is an additional output parameter, which receives the size of the workspace required by the plan. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftMakePlan2d(cufftHandle plan, //Handle returned
```

```
                                                        // by cufftCreate
    int nx, int ny,    //Transform x and y dimensions
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftMakePlan3d()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. There is an additional output parameter, which receives the size of the workspace required by the plan. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftMakePlan3d(cufftHandle plan, //Handle returned
                                                       // by cufftCreate
    int nx, int ny, int nz, //Transform x, y, and z dimensions
    cufftType type,         //Transform type, e.g., CUFFT_C2C
    size_t *workSize);      //Size of work area for the transform
```

- The arguments for **cufftMakePlanMany()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. There is an additional output parameter, which receives the size of the workspace required by the plan. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftMakePlanMany(cufftHandle plan, //Handle returned
                                                         // by cufftCreate
    int rank,          //Dimensionality of the transform (1,2,or 3)
    int *n,            //Array of size rank, describing the size of each
                       // dimension
    int *inembed,      //Array of size rank, describing the storage
                       // dimensions of input data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int istride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int idist,         //Distance between the first element of two
                       // consecutive signals in a batch of input data
    int *onembed,      //Array of size rank, describing the storage
                       // dimensions of output data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int ostride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int odist,         //Distance between the first element of two
                       // consecutive signals in a batch of output data
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    int batch,         //Batch size for this transform
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftGetSize1d()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftGetSize1d(cufftHandle plan,
    int nx,            //Transform size
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    int batch,         //Number of transforms of size nx;
                       // deprecated—use cufftPlanMany.
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftGetSize2d()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftGetSize2d(cufftHandle plan, //Handle returned
                                                      // by cufftCreate
    int nx, int ny,    //Transform x and y dimensions
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftGetSize3d()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftGetSize3d(cufftHandle plan, //Handle returned
                                                      // by cufftCreate
    int nx, int ny, int nz, //Transform x, y, and z dimensions
    cufftType type,         //Transform type, e.g., CUFFT_C2C
    size_t *workSize);      //Size of work area for the transform
```

- The arguments for **cufftGetSizeMany()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftGetSizeMany(cufftHandle plan, //Handle returned
                                                        // by cufftCreate
    int rank,          //Dimensionality of the transform (1,2,or 3)
    int *n,            //Array of size rank, describing the size of each
                       // dimension
    int *inembed,      //Array of size rank, describing the storage
                       // dimensions of input data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int istride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int idist,         //Distance between the first element of two
                       // consecutive signals in a batch of input data
    int *onembed,      //Array of size rank, describing the storage
                       // dimensions of output data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int ostride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int odist,         //Distance between the first element of two
                       // consecutive signals in a batch of output data
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    int batch,         //Batch size for this transform
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftEstimateMany()** are incorrect. There is no plan argument for this call. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftEstimateMany(int rank, //Dimensionality of the
                                                 // transform (1,2,or 3)
    int *n,            //Array of size rank, describing the size of each
                       // dimension
    int *inembed,      //Array of size rank, describing the storage
                       // dimensions of input data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int istride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int idist,         //Distance between the first element of two
                       // consecutive signals in a batch of input data
    int *onembed,      //Array of size rank, describing the storage
                       // dimensions of output data. If set to NULL, all
                       // other advanced data layout parameters are
                       // ignored.
    int ostride,       //Distance between two successive input elements in
                       // the least significant (innermost) dimension
    int odist,         //Distance between the first element of two
                       // consecutive signals in a batch of output data
    cufftType type,    //Transform type, e.g., CUFFT_C2C
    int batch,         //Batch size for this transform
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftGetSize()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftGetSize(cufftHandle plan, //Handle returned
                                                    // by cufftCreate
    size_t *workSize); //Size of work area for the transform
```

- The arguments for **cufftSetAutollocation()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftSetAutoAllocation(cufftHandle plan, //Handle
                                            // returned by cufftCreate
    int autoAllocate); //Non-zero indicates CUFFT should allocate
                       // workspace automatically
```

- The arguments for **cufftSetWorkArea()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftSetWorkArea(cufftHandle plan, //Handle returned
                                                        // by cufftCreate
    void *workArea); //Pointer to device memory for CUFFT to use as
                     // its work area
```

- The arguments for **cufftExedC2C()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftExecC2C(cufftHandle plan, //Handle returned
                                                    // by cufftCreate
    cufftComplex *idata, //Pointer to the complex input data
                         // (in GPU memory) to transform
    cufftComplex *odata, //Pointer to the complex output data
                         // (in GPU memory)
    int direction);      //The transform direction: CUFFT_FORWARD or
                         // CUFFT_INVERSE
```

- The arguments for **cufftExecR2C()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftExecR2C(cufftHandle plan, //Handle returned
                                                    // by cufftCreate
    cufftReal *idata,     //Pointer to the real input data
                          // (in GPU memory) to transform
    cufftComplex *odata); //Pointer to the complex output data
                          // (in GPU memory)
```

- The arguments for **cufftExedZ2Z()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftExecZ2Z(cufftHandle plan, //Handle returned
                                                    // by cufftCreate
    cufftDoubleComplex *idata, //Pointer to the complex input data
                               // (in GPU memory) to transform
    cufftDoubleComplex *odata, //Pointer to the complex output data
                               // (in GPU memory)
    int direction);            //The transform direction: CUFFT_FORWARD or
                               // CUFFT_INVERSE
```

- The arguments for **cufftExecD2Z()** are incorrect. The plan is a **cufftHandle** returned from a prior call to **cufftCreate()**. It is an input parameter only. The actual calling sequence is as follows:

```
cufftResult CUFFTAPI cufftExecD2Z(cufftHandle plan, //Handle returned
                                                    // by cufftCreate
```

```
    cufftDoubleReal *idata,     //Pointer to the real input data
                                // (in GPU memory) to transform
    cufftDoubleComplex *odata); //Pointer to the complex output data
                                // (in GPU memory)
```

2.1.3. CUDA Samples

- When graphics samples targeting the i386 architecture are built on an x86_64 machine, the resulting binary is copied into the native, x86_64 **bin** directory instead of the i386 **bin** directory.
- When the Linux **.run** installer is used to install the CUDA Samples without the CUDA Toolkit, it will report an installation failure in the summary, even though the installation may have succeeded.
- During the cross-building of 32-bit samples on a 64-bit Linux machine, some libraries may not be found and the build will fail. Use the **EXTRA_LDFLAGS** Makefile variables to point to the needed libraries to fix the issue.

2.1.4. CUDA Tools

- When clang is used as the host compiler, 32-bit target compilation on Mac 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.
- Clang is now supported as a host compiler on Mac OS X 10.8 as a BETA feature in CUDA 5.5. To use Clang as the host compiler, invoke nvcc with **-ccbin=path-to-clang-executable**. There are some features that are not yet supported: building from within NSight Eclipse Edition, Clang language extensions (see **http://clang.llvm.org/docs/LanguageExtensions.html**), LLVM libc++ (only GNU libstdc++ is currently supported), language features introduced in C++11, the **__global__ function** template explicit instantiation definition, and 32-bit architecture cross-compilation. (*This replaces the previously released statement about Clang support.*)
- On Mac OS X 10.8, if you install XCode 5, including the command-line tools, the gcc compiler will get replaced with clang. You can continue to successfully compile with nvcc from the command-line by using the **--ccbin /usr/bin/clang** option, which instructs nvcc to use the clang compiler instead of gcc to compile any host code passed to it. However, this solution will not work when building with NSight Eclipse Edition. An alternative solution that will work from the command-line and with NSight Eclipse Edition is to download and older version of the command-line tools package from the Apple Developer website after installing XCode 5, which will re-install gcc to /usr/bin. In order to do this, go to **http://developer.apple.com/downloads/index.action**, sign in with your Apple ID, and search for **command-line tools** using the search pane on the left side of the screen.
- On the GK110, the kernel occasionally may produce incorrect results. This happens 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. "Outstanding" in this case means none of the results of these instructions have

been used. The underlying cause is that the texture barrier can track at most 63 outstanding texture/LDG instructions. If there are more than 63 such instructions, the texture barrier can no longer be relied on to ensure that any instruction's result is correct.

This issue can be worked around by adding **`-maxrregcount 63`** to ptxas. This guarantees there are at most 63 outstanding texture instructions because each texture/LDG will write at least one register. However, this may downgrade performance because it limits the maximum number of registers. (*This issue has been fixed for CUDA 6.0.*)

- The CUPTI (CUDA Profiling Tools Interface) release notes are now part of this document.

Changes Incompatible with CUPTI 4.0

A number of non-backward compatible API changes were made in CUPTI 4.1. These changes require minor source modifications to existing code compiled against CUPTI 4.0. In addition, some previously incorrect and undefined behavior is now prevented by improved error checking. Your code may need to be modified to handle these new error cases.

- Multiple CUPTI subscribers are not allowed. In CUPTI 4.0, **`cuptiSubscribe()`** could be used to enable multiple subscriber callback functions to be active at the same time. When multiple callback functions were subscribed, invocation of those callbacks did not respect the domain registration for those callback functions. In CUPTI 4.1 and later, **`cuptiSubscribe()`** returns **`CUPTI_ERROR_MAX_LIMIT_REACHED`** if there is already an active subscriber.
- The **`CUpti_EventID`** values for Tesla devices have changed in CUPTI 4.1 to make all **`CUpti_EventID`** values unique across all devices. Going forward, **`CUpti_EventID`** values will be added for new devices and events, but existing values will not be changed. If your application has stored **`CUpti_EventID`** values (for example, as part of the data collected for a profiling session), those **`CUpti_EventID`**s must be translated to the new ID values before being used in CUPTI 4.1 and later APIs.
- In enumeration **`CUpti_EventDomainAttribute`**, **`CUPTI_EVENT_DOMAIN_MAX_EVENTS`** has been removed. The number of events in an event domain can be retrieved with **`cuptiEventDomainGetNumEvents()`**.
- Routines **`cuptiDeviceGetAttribute()`**, **`cuptiEventGroupGetAttribute()`**, and **`cuptiEventGroupSetAttribute()`** now take a **`size`** parameter, and the **`value`** parameter now has type **`void *`**.
- Routine **`cuptiEventDomainGetAttribute()`** no longer takes a **`CUdevice`** parameter. This function is now used to get event domain attributes that are device independent. A new function **`cuptiDeviceGetEventDomainAttribute()`** has been added to get event domain attributes that are device dependent.
- Routines **`cuptiEventDomainGetNumEvents()`**, cuptiEventDomainEnumEvents(), and **`cuptiEventGetAttribute()`** no longer take a **`CUdevice`** parameter.

- The **contextUid** field of the **CUpti_CallbackData** structure has been changed from type uint64_t to type **uint32_t**.

Known Issues

The activity API functions **cuptiActivityEnqueueBuffer()** and **cuptiActivityDequeueBuffer()** are deprecated and will be removed in a future release. The new asynchronous API implemented by **cuptiActivityRegisterCallbacks()**, **cuptiActivityFlush()**, and **cuptiActivityFlushAll()** should be adopted. See the CUPTI documentation for details.

2.2. Documentation

For a list of documents supplied with this release, please refer to the **doc** directory of your CUDA Toolkit installation. PDF documents are available in the **doc/pdf** folder. Several documents are now also available in HTML format and are found in the **doc/html** folder.

- The HTML documentation is now fully available from a single entry page available both locally in the CUDA Toolkit installation folder under **doc/html/index.html** and online at http://docs.nvidia.com/cuda/index.html.
- The license information for the toolkit portion of this release can be found at **doc/EULA.txt**.
- The CUDA Occupancy Calculator spreadsheet can be found at **tools/CUDA_Occupancy_Calculator.xls**.
- The CHM documentation has been removed.

2.3. List of Important Files

If the CUDA 5.5 Toolkit was installed using the RPM/DEB installers, the installation directory has changed. There is a **targets** directory in the root of the installation directory with a sub-directory for each possible target. Currently, the follwoing targets are supported: **i386-linux**, **x86_64-linux**, **armv7-linux-gnueabihf**, and **ppc64le-linux**. In each of these target directories there is a **lib** directory for that target's libraries and an **include** directory for that target's header files. The installer creates the proper symbolic links in the installation's root directory for backward compatibility.

2.3.1. Core Files

```
bin/
    nvcc                        CUDA C/C++ compiler
    cuda-gdb                    CUDA Debugger
    cuda-memcheck               CUDA Memory Checker
    nsight                      Nsight Eclipse Edition (Linux and Mac OS)
    nvprof                      NVIDIA Command-Line Profiler
    nvvp                        NVIDIA Visual Profiler (Located in libnvvp/
                                                        on Windows)

include/
    cuda.h                      CUDA driver API header
    cudaGL.h                    CUDA OpenGL interop header for driver API
```

```
    cudaVDPAU.h                     CUDA VDPAU interop header for driver API
                                                                    (Linux)
    cuda_gl_interop.h               CUDA OpenGL interop header for toolkit API
                                                                    (Linux)
    cuda_vdpau_interop.h            CUDA VDPAU interop header for toolkit API
                                                                    (Linux)
    cudaD3D9.h                      CUDA DirectX 9 interop header  (Windows)
    cudaD3D10.h                     CUDA DirectX 10 interop header (Windows)
    cudaD3D11.h                     CUDA DirectX 11 interop header (Windows)
    cufft.h                         CUFFT API header
    cublas_v2.h                     CUBLAS API header
    cublas.h                        CUBLAS Legacy API header
    cusparse_v2.h                   CUSPARSE API header
    cusparse.h                      CUSPARSE Legacy API header
    curand.h                        CURAND API header
    curand_kernel.h                 CURAND device API header
    thrust/*                        Thrust headers
    npp.h                           NPP API header
    nvToolsExt*.h                   NVIDIA Tools Extension headers (Linux and Mac)
    nvcuvid.h                       CUDA Video Decoder header (Windows and Linux)
    cuviddec.h                      CUDA Video Decoder header (Windows and Linux)
    NVEncodeDataTypes.h             CUDA Video Encoder header (Windows; C-library
                                                                  or DirectShow)
    NVEncoderAPI.h                  CUDA Video Encoder header (Windows; C-library)
    INvTranscodeFilterGUIDs.h       CUDA Video Encoder header (Windows; DirectShow)
    INVVESetting.h                  CUDA Video Encoder header (Windows; DirectShow)

extras/
    CUPTI                           CUDA Profiling Tools Interface API
    Debugger                        CUDA Debugger API

nvvm/include/
    nvvm.h                          Optimizing Compiler Library API header

nvvm/libdevice/
    libdevice.compute*.bc           NVIDIA Common Device Math Functions Library

src/
    *fortran*.{c,h}                 FORTRAN interface files for CUBLAS and CUSPARSE
```

2.3.2. Windows lib Files

(Corresponding 32-bit or 64-bit DLLs are in **bin/**.)

```
lib/{Win32,x64}/
    cuda.lib                        CUDA driver library
    cudart.lib                      CUDA runtime library
    cudadevrt.lib                   CUDA runtime device library
    cublas.lib                      CUDA BLAS library
    cublas_device.lib               CUDA BLAS device library
    cufft.lib                       CUDA FFT library
    cuinj.lib                       CUDA internal library for profiling
    cusparse.lib                    CUDA Sparse Matrix library
    curand.lib                      CUDA Random Number Generation library
    npp.lib                         NVIDIA Performance Primitives library
    nvcuvenc.lib                    CUDA Video Encoder library
    nvcuvid.lib                     CUDA High-level Video Decoder library
    OpenCL.lib                      OpenCL library

nvvm/lib/{Win32,x64}/
    nvvm.lib                        Optimizing Compiler Library
```

2.3.3. Linux lib Files

```
lib{64}/
    libcudart.so                    CUDA runtime library
```

```
    libcuinj.so                 CUDA internal library for profiling
    libcublas.so                CUDA BLAS library
    libcublas_device.a          CUDA BLAS device library
    libcufft.so                 CUDA FFT library
    libcusparse.so              CUDA Sparse Matrix library
    libcurand.so                CUDA Random Number Generation library
    libnpp.so                   NVIDIA Performance Primitives library

nvvm/lib{64}/
    libnvvm.so                  Optimizing Compiler Library
```

2.3.4. Mac OS X lib Files

```
lib/
    libcudart.dylib             CUDA runtime library
    libcuinj.dylib              CUDA internal library for profiling
    libcublas.dylib             CUDA BLAS library
    libcublas_device.a          CUDA BLAS device library
    libcufft.dylib              CUDA FFT library
    libcusparse.dylib           CUDA Sparse Matrix library
    libcurand.dylib             CUDA Random Number Generation library
    libnpp.dylib                NVIDIA Performance Primitives library
    libtlshook.dylib            NVIDIA internal library

nvvm/lib/
    libnvvm.dylib               Optimizing Compiler Library
```

2.4. Supported NVIDIA Hardware

See http://www.nvidia.com/object/cuda_gpus.html.

2.5. Supported Operating Systems

2.5.1. Windows

The next two tables list the currently supported Windows operating systems and compilers.

Table 2 Windows Operating Systems Supported in CUDA 5.5

Windows 8	Windows XP
Windows 7	Windows Server 2012 (64-bit)
Windows Vista	Windows Server 2008 R2

Table 3 Windows Compilers Supported in CUDA 5.5

Compiler	IDE
Visual C++ 11.0	Visual Studio 2012
Visual C++ 11.0	Visual Studio 2012 Express (32-bit)
Visual C++ 10.0	Visual Studio 2010

Compiler	IDE
Visual C++ 9.0	Visual Studio 2008

2.5.2. Linux

The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for this CUDA Toolkit release.

Table 4 Linux Distributions Supported in CUDA 5.5

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 18		x	3.6.10-4.fc18.x86_64	4.7.2	2.16
ICC Compiler 12.1		x			
OpenSUSE 12.2		x	3.4.6-2.10-desktop	4.7.1	2.15
Red Hat Enterprise Linux (RHEL) 6.x		x	2.6.32-358.el6.i686 (RH 6.4)	4.4.7	2.12
RHEL 5.5+		x	2.6.18-238.el5	4.1.2	2.5
SUSE SLES 11 SP2		x	3.0.13-0.27-pae	4.3.4	2.11.3
SUSE SLES 11 SP1		x	2.6.32.12-0.7-pae	4.3.4	2.11.1
Ubuntu 12.10	x	x	3.5.0-17-generic	4.7.2	2.15
Ubuntu 12.04	x	x	2.6.35-23-generic	4.6	2.15
Ubuntu 10.04	x	x	2.6.35-23-generic	4.4.5	2.12.1

Table 5 Linux Distributions No Longer Supported as of CUDA 5.5

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 16	x	x	3.1.0-7.fc16.i686.pae	4.6.2	2.14.90
Ubuntu 11.10	x	x	3.0.0-19-generic-pae	4.6.1	2.13

2.5.3. Mac OS X

These Mac operating systems are supported in CUDA 5.5: Mac OS X 10.8.x (64-bit) and Mac OS X 10.7.5+.

2.6. Installation Notes

2.6.1. Windows

For silent installation:

- To install, use **msiexec.exe** from the shell, passing these arguments:

```
msiexec.exe /i <cuda_toolkit_filename>.msi /qn
```

- To uninstall, use **/x** instead of **/i**.

2.6.2. Linux

- In order to run CUDA applications, the CUDA module must be loaded and the entries in **/dev** created. This may be achieved by initializing X Windows, or by creating a script to load the kernel module and create the entries. An example script (to be run at boot time) follows.

```
#!/bin/bash

/sbin/modprobe nvidia

if [ "$?" -eq 0 ]; then

    # Count the number of NVIDIA controllers found.
    N3D=`/sbin/lspci | grep -i NVIDIA | grep "3D controller" | wc -l`
    NVGA=`/sbin/lspci | grep -i NVIDIA | grep "VGA compatible controller" \
                      | wc -l`

    N=`expr $N3D + $NVGA - 1`
    for i in `seq 0 $N`; do
        mknod -m 666 /dev/nvidia$i c 195 $i;
    done

    mknod -m 666 /dev/nvidiactl c 195 255

else
    exit 1
fi
```

- On some Linux releases, due to a GRUB bug in the handling of upper memory and a default **vmalloc** too small on 32-bit systems, it may be necessary to pass this information to the bootloader:

```
vmalloc=256MB, uppermem=524288
```

Here is an example of GRUB conf:

```
title Red Hat Desktop (2.6.9-42.ELsmp)
root (hd0,0)
uppermem 524288
kernel /vmlinuz-2.6.9-42.ELsmp ro root=LABEL=/1 rhgb quiet vmalloc=256MB
pci=nommconf
initrd /initrd-2.6.9-42.ELsmp.img
```

2.7. Deprecated Features

The following features are deprecated. 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 of the CUDA software. We recommend that developers employ alternate solutions to these features in their software.

There are no features to document at this time.

2.8. New Features

2.8.1. General CUDA

- MPS (Multi-Process Service) is a runtime service designed to let multiple MPI (Message Passing Interface) processes using CUDA run concurrently on a single GPU in a way that's transparent to the MPI program. A CUDA program runs in MPS mode if the MPS control daemon is running on the system. When a CUDA program starts, it connects to the MPS control daemon (if possible), which then creates an MPS server for the connecting client if one does not already exist for the user (UID) that launched the client. See the nvidia-cuda-mps-control man page for more information on how to configure an MPS environment.
- The CUDA 5.5 Toolkit adds support for Linux on the ARMv7 Architecture. The toolkit comes with a comprehensive set of tools to develop applications for Linux on ARMv7, either natively or cross-platform. Note that only the ARM hard-float floating point ABI is supported.
- The CUDA 5.5 Toolkit adds support for Linux on the ppc64le Architecture. The toolkit comes with a comprehensive set of tools to develop applications for Linux on ppc64le, either natively or cross-platform.
- With the CUDA 5.5 Toolkit, any objects that contain device code must be passed to both the device and host linkers. This was recommended practice before, but is now required.
- For projects that use -Xlinker with nvcc, you must ensure the arguments after -Xlinker are quoted. In CUDA 5.0, **`-Xlinker -rpath /usr/local/cuda/lib`** would succeed; in CUDA 5.5 **`-Xlinker "-rpath /usr/local/cuda/lib"`** is now necessary.
- The Toolkit is using a new installer on Windows. The installer is able to install any selection of components and to customize the installation locations per user request.
- The CUDA Sample projects have makefiles that are now more self-contained and robust. If some dependent libraries are not present on Linux, the top-level makefile does not build them.
- The CUDA Toolkit and the CUDA Driver are now available for installation as **`.rpm`** and **`.deb`** installation packages for all the supported Linux distributions, except Ubuntu 10.04 and RHEL 5.5. Those files are accessible on the CUDA Toolkit package repositories. The RPM and Debian package installations support installation of multiple versions. Installations can be updated when a new version of the CUDA Toolkit is available.
- The following documents are now available in the CUDA toolkit documentation portal:
  - Programming guides: *CUDA Video Encoder*, *CUDA Video Decoder*, *Developer Guide to Optimus*, *Parallel Thread Execution (PTX) ISA*, *Using Inline PTX Assembly in CUDA*, *NPP Library Programming Guide*.
  - Tools manuals: *CUDA Binary Utilities*.

- White papers: *Floating-Point and IEEE 754 Compliance, Incomplete-LU and Cholesky Preconditioned Iterative Methods*.
- Compiler SDK: *libNVVM API, libdevice Users's Guide, NVVM IR Specification*.
- General: *CUDA Toolkit Release Notes, End-User License Agreements*.

2.8.2. CUDA Libraries

2.8.2.1. CUBLAS

- The routines **`cublas{S,D,C,Z}getriBatched()`** and **`cublas{S,D,C,Z}matinvBatched()`** have been added to the CUBLAS Library. Routine **`cublas{S,D,C,Z}getriBatched()`** must be called after the LU batched factorization routine, **`cublas{S,D,C,Z}getrfBatched()`**, to obtain the inverse matrices. The routine **`cublas{S,D,C,Z}matinvBatched()`** does a direct inversion with pivoting based on the Gauss-Jordan algorithm but is limited to matrices of dimension <= 32×32.
- The limitation on the dimension **`n`** of the routine **`cublas<T>getrfbatched()`** has been removed. However, for performance reasons it is still recommended to use this routine for small values of n, typically n < 256.

2.8.2.2. CUFFT

- CUFFT 5.5 extends the existing API. The new calls allow creation of a CUFFT plan handle separate from the actual creation of the plan, allow insertion of new calls to set plan attributes before the work of plan creation is done, and allow advanced users more control over memory space allocation. Details can be found in the *CUFFT Library User's Guide*.
- CUFFT 5.5 provides FFTW3 interfaces that enables applications using FFTW to gain performance with NVIDIA CUFFT with minimal changes to program source code. The *CUFFT Library User's Guide* documents which FFTW3 API features are supported.

2.8.2.3. CURAND

CURAND 5.5 introduces support for the random number generator Philox4x32-10.

2.8.2.4. CUSPARSE

- The routine **`cusparse{S,D,C,Z)crsmm2()`** is an API extension of **`cusparse{S,D,C,Z}csrmm()`** which allows the matrix **`B`** to be passed in a transposed form. This can bring up to a 2× speedup in performance due to the better memory access efficiency of transposed matrix **`B`**.
- The **`cublas<T>gtsv()`** routines have been replaced with a version that supports pivoting. The previous version has been renamed **`cublas<T>gtsv_nopivot()`** to better reflect that it does not support pivoting. The new algorithm has been developed by Liwen Wang from the Impact Group of the University of Illinois.
- The routine **`cusparse<T>brsrxmv()`** is an extension of the routine **`cusparse<T>bsrmv()`** that allows the matrix vector product to be performed on a submatrix. This routine also works for block of dimension 1 (CSR format).

2.8.2.5. Thrust

The version of Thrust included with the current CUDA toolkit was upgraded from version 1.5.3 to version 1.7.0. A summary of included updates can be found here: https://github.com/thrust/thrust/blob/1.7.0/CHANGELOG.

2.8.3. CUDA Tools

2.8.3.1. CUDA Compiler

- The following changes have been made to the CUDA Compiler SDK:
  - An optimizing compiler library (**`libnvvm.so`**, **`nvvm.dll/nvvm.lib`**, **`libnvvm.dylib`**) and its header file **`nvvm.h`** are provided for compiler developers who want to generate PTX from a program written in NVVM IR, which is a compiler internal representation based on LLVM.
  - A set of libraries, **`libdevice.*.bc`**, that implement the common math functions for devices in the LLVM bitcode format are provided.
  - A set of samples that illustrate the use of the compiler SDK are provided.
  - Documents for the CUDA Compiler SDK (including the specification for LLVM IR, an API document for **`libnvvm`**, and an API document for **`libdevice`**) are provided.
- The default **`nvcc.profile`** no longer includes **`-lcudart`** (on Linux and Mac OS X) and **`cudart.lib`** (on Windows), and the use of the CUDA runtime is now controlled by the option **`--cudart (-cudart)`**. Consequently, the option **`--dont-use-profile (-noprof)`** no longer prevents **`nvcc`** from linking the object files against the CUDA runtime when the default **`nvcc.profile`** is used, and the option **`--cudart=none (-cudart=none)`** needs to be used instead. If the option **`--cudart=none (-cudart=none)`** is not specified, **`--cudart=static (-cudart=static)`** is assumed, and nvcc links the object files against the static CUDA runtime.
- CUDA 5.5 adds support for JIT linking. This can be done explicitly by using the driver API (see the **`cuLink*`** routines in the *CUDA Driver API* documentation); alternatively, runtime apps that use separate compilation will automatically JIT to a newer architecture if needed (see the *Separate Compilation* chapter in the *CUDA Compiler Driver NVCC* document). JIT linking requires rebuilding all objects with the 5.5 toolkit.

2.8.3.2. 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**.

2.8.3.3. CUDA-MEMCHECK

- Return code **cudaErrorNotReady** can be returned by **cudaStreamQuery()** and **cudaEventQuery()** in the case where the stream/event being waited on is still busy. This return code is not an error condition and is used by user programs to poll until the stream/event is ready. CUDA-MEMCHECK will no longer report the following conditions as errors when CUDA API call checking is enabled:
  - **cudaErrorNotReady** returned by CUDA Run Time API calls
  - **CUDA_ERROR_NOT_READY** returned by CUDA Driver API calls
- The racecheck tool in CUDA-MEMCHECK now has support for SM 3.5 devices.
- The **racecheck-report mode** option of the racecheck tool can be used to enable the generation of analysis records.
- CUDA-MEMCHECK now supports displaying error information as errors occur during program execution instead of waiting for program termination to display output.

2.8.3.4. CUDA Profiler

- The NVIDIA Visual Profiler now supports applications that use CUDA Dynamic Parallelism. The application timeline includes both host-launched and device-launched kernels, and shows the parent-child relationship between kernels.
- The application analysis performed by the NVIDIA Visual Profiler has been enhanced. A guided analysis mode has been added that provides step-by-step analysis and optimization guidance. Also, the analysis results now included graphical visualizations to more clearly indicate the optimization opportunities.
- The NVIDIA Visual Profiler and the command-line profiler, **nvprof**, now support power, thermal, and clock profiling.
- The NVIDIA Visual Profiler and the command-line profiler, **nvprof**, now support metrics that report the floating-point operations performed by a kernel. These metrics include both single-precision and double-precision counts for adds, multiplies, multiply-accumulates, and special floating-point operations.
- The NVIDIA command-line profiler, **nvprof**, now supports collection of any number of events and metrics during a single run of a CUDA application. It uses kernel replay to execute each kernel as many times as necessary to collect all the requested profile data.

- The NVIDIA command-line profiler, **nvprof**, now supports profiling of all CUDA processes executed on a system. In this "profile all processes" mode, a user starts **nvprof** on a system and all CUDA applications subsequently launched by that user are profiled.

2.8.3.5. Debugger API

- Two new symbols are introduced to control the behavior of the application: **CUDBG_ENABLE_LAUNCH_BLOCKING** and **CUDBG_ENABLE_INTEGRATED_MEMCHECK**. The two symbols, when set to 1, have the same effect as setting the environment variables **CUDA_LAUNCH_BLOCKING** and **CUDA_MEMCHECK** to 1. Both symbols also have the same restriction: the change takes effect on the next run of the application.
- Software preemption is available as a BETA. The option is enabled by setting the symbol **CUDBG_ENABLE_PREEMPTION_DEBUGGING** to 1. The option is used to debug a CUDA application on the same GPU that is rendering the desktop GUI.
- Software preemption (BETA) enables debugging of long-running or indefinite CUDA kernels that would otherwise encounter a launch timeout.
- Software preemption (BETA) allows multiple debugger sessions can simultaneously debug CUDA applications on the same GPU. This feature is available on Linux with devices of compute capability of 3.5.
- The parent grid information for each kernel is now available as either a new field in the **kernelReady** event, or as a field in the newly created **CUDBGGridInfo struct**, which is retrievable via the new **getGridInfo()** call. Both models, push and pull, complement each other and should be used hand-in-hand to get the most accurate and recent information about the status of a kernel in the application.
- To reduce the number of times the debugger stops and resumes the application, the debugger API can be made to defer non-essential host kernel launch notifications instead of producing events in the the synchronous event queue. This behavior is controlled with the new **setKernelLaunchNotificationMode()** function call. When set to **CUDBG_KNL_LAUNCH_NOTIFY_DEFER**, the debugger will not receive **kernelReady** events for every kernel launch. Instead, the debugger must reconstruct this information by calling **getGridInfo** for every previously unseen grid present on the device the next time it stops.
- The **gridId** is now available as a 64-bit value. New fields and new API functions were added to cover the new type. The old 32-bit values are still accessible but are now deprecated. Whenever possible the 64-bit **gridId** should be used.

2.8.3.6. Nsight Eclipse Edition

- Nsight Eclipse Edition now provides remote debugging of CUDA applications for Linux targets. The host system running Nsight may be Mac OS X or Linux, and the target system being debugged may be any supported version of Linux and may have a different CPU architecture. Nsight can upload a locally built application to the target system or can use an executable already available on the remote system. .
- The Nsight Eclipse Edition debugger now provides a memory viewer for both host and device memory. The memory viewer supports a number of different data types, including floating point.

- Nsight Eclipse Edition now provides CUDA Dynamic Parallelism support for both new and existing projects.
- For applications that use CUDA Dynamic Parallelism, the Nsight Eclipse Edition debugger now shows the parent/child launch trace for device-launched kernels.
- Nsight Eclipse Edition now includes the Remote System Explorer plug-in. This plug-in enables accessing of remote systems for file transfer, shell access, and listing running processes.
- Nsight Eclipse Edition is updated to use Eclipse Platform 3.8.2 and Eclipse CDT 8.1.2, introducing a number of new features and enhancements to existing features.

2.8.3.7. 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.

2.9. Performance Improvements

2.9.1. CUDA Libraries

2.9.1.1. CUBLAS

The **`cublas<T>trsv()`** routines have been significantly optimized with the work of Jonathan Hogg from The Science and Technology Facilities Council (STFC). Subsequently, **`cublas<T>trsm()`** was updated to use some of these optimizations in some cases.

2.9.1.2. Math

The performance of the double-precision functions **`fmod()`**, **`remainder()`**, and **`remquo()`** has been significantly improved for **`sm_30`**.

2.10. Resolved Issues

There are no issues to document at this time.

2.11. Known Issues

There are no issues to document at this time.

2.12. Source Code for Open64 and CUDA-GDB

- The Open64 and CUDA-GDB source files are controlled under terms of the GPL license. Current versions are located here: http://github.com/nvidia. Previously released versions can be found here: ftp://download.nvidia.com/CUDAOpen64/.
- Linux users can refer to the following:
  - The *Release Notes* and *Known Issues* sections in the *CUDA-GDB User Manual* (**`CUDA_GDB.pdf`**)
  - The **`CUDA_Memcheck.pdf`** for notes on supported error detection and known issues

2.13. More Information

- For more information, please visit http://www.nvidia.com/cuda and http://docs.nvidia.com/cuda.
- Please refer to the *LLVM Release License* text in **`EULA.txt`** for details on LLVM licensing.

Chapter 3. NVIDIA CUDA TOOLKIT V5.0 RELEASE NOTES

3.1. Errata

3.1.1. Known Issues

3.1.1.1. General CUDA

- Extracting the Linux installer via the **-extract=<path>** option currently requires root permissions.
- When the default CUDA 5.0 Windows installer option to silently install the NVIDIA display driver is used, an error message like "display driver has failed to install" may be displayed for certain hardware configurations. If this error message occurs, the installation can be completed by installing the display driver separately using the setup.exe saved under **C:\NVIDIA\DisplayDriver\...**.
- In certain hardware configurations, the CUDA 5.0 installer on Windows may fail to install the display driver. This failure occurs when the user disables silent installation of the display driver and instead chooses to interactively select the components of the display driver from the installer UI that appears after the CUDA toolkit and samples are installed. If the UI for interactive selection of the display driver components fails to appear, please reinstall just the display driver by running setup.exe saved under **C:\NVIDIA\DisplayDriver\...**.
- On GPUs that are not in Tesla Compute Cluster (TCC) mode under Windows, CUDA streams may not achieve as much concurrency as they did in prior releases.
- When running the Linux installer in silent mode without root permissions, the **-toolkitpath=<PATH>** and **-samplespath=<PATH>** flags must be passed.
- The CUDA 5.0 toolkit and samples require the associated CUDA driver version to be at least 304.54 on Linux and at least 306.94 on Windows. Make sure that such a CUDA driver is installed on your system before attempting to run the CUDA 5.0 samples or any CUDA applications.

- On certain Windows configuration installing Visual Studio integration files may not get updated, this could result in a build error when building CUDA application. To fix this problem follow these steps: The windows CUDA Toolkit installers installs a duplicate copy of the Visual Studio integration files into **<ProgramFiles>\NVIDIA GPU Computing Toolkit\CUDA \v5.0\extras\visual_studio_integration\MSBuildExtensions** for the CUDA Toolkit feature. Copy **Nvda.Build.CudaTasks.v5.0.dll** from this folder into the MSBuild Build Customization folder at **C:\Program Files\MSBuild\Microsoft.Cpp\v4.0\BuildCustomizations** on 32 bit operating systems or **C:\Program Files (x86)\MSBuild\Microsoft.Cpp \v4.0\BuildCustomizations** on 64 bit operating systems.
- On Linux and Mac OS X, the CUDA Toolkit 5.0 Samples do not generate PTX code required for forward compatibility with future GPU architectures. It is highly recommended to always compile CUDA applications with the PTX code associated with the latest available PTX generation supported by the compiler. To do so, the **-gencode arch=compute_35,code=sm_35** line in the CUDA Samples Makefiles must be replaced with **-gencode arch=compute_35,code= \"sm_35,compute_35\"**. For additional information, please consult the compiler documentation at http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#extended-notation

3.1.1.2. CUDA Libraries

- The **cublas<T>geam()** routine provides undefined results if the pointer mode is set to **CUBLAS_POINTER_MODE_DEVICE** and the value pointed to by alpha is zero. There are two possible workarounds for this issue. The first is to use **CUBLAS_PONTER_MODE_HOST** instead of **CUBLAS_POINTER_MODE_DEVICE**, but this may require an extra device-to-host memory copy, depending on the situation. The second is to swap the (transa, alpha, A, lda) parameters with the (transb, beta, B, ldb) parameters, which would make the value pointed to by beta equal to 0.
- The routine **cublasCsyrk()** may produce incorrect results on GPUs that implement the sm_30 architecture when the size of matrix parameter A exceeds (128M - 512) total elements.
- The CUSPARSE library routines **csrsv_analysis()**, **csrsv_solve()**, **csrsm_analysis()**, and **csrsm_solve()** support the CUSPARSE_MATRIX_TYPE_GENERAL matrix type in addition to the supported matrix types already listed in the documentation.

3.1.1.3. CUDA Tools

- The hardware counter (event) values may be incorrect in some cases on GPUs with compute capability (SM type) 3.5. Incorrect event values also result in incorrect metric values. These errors are more likely to occur when the same GPU is used for display and compute, or when other graphics applications are running simultaneously on the GPU.
- Beginning with CUDA 5.0, the ptxas portion of the compiler generates a warning when the command line option "-abi=no" is used that indicates the option may be deprecated in a future release.

- The current 5.0 linker will not support JIT to future architectures; objects will have to re-linked for each architecture.
- Source-level analysis in NVIDIA Nsight Eclipse Edition and NVIDIA Visual Profiler is not available for kernels accessed through static function pointers.

3.2. Documentation

- For a list of documents supplied with this release, please refer to the **doc** directory of your CUDA Toolkit installation. PDF documents are available in the **doc/pdf** folder. Several documents are now also available in HTML format and are found in the **doc/html** folder.
- The HTML documentation is now fully available from a single entry page available both locally in the CUDA Toolkit installation folder under **doc/html/index.html** and online at http://docs.nvidia.com/cuda/index.html.
- The license information for the toolkit portion of this release can be found at **doc/EULA.txt**.
- The CUDA Occupancy Calculator spreadsheet can be found at **tools/CUDA_Occupancy_Calculator.xls**.
- The CHM documentation has been removed.

3.3. List of Important Files

3.3.1. Core Files

```
bin/
    nvcc                        CUDA C/C++ compiler
    cuda-gdb                    CUDA Debugger
    cuda-memcheck               CUDA Memory Checker
    nsight                      Nsight Eclipse Edition (Linux and Mac OS)
    nvprof                      NVIDIA Command-Line Profiler
    nvvp                        NVIDIA Visual Profiler (Located in libnvvp/
                                                        on Windows)

include/
    cuda.h                      CUDA driver API header
    cudaGL.h                    CUDA OpenGL interop header for driver API
    cudaVDPAU.h                 CUDA VDPAU interop header for driver API
                                                                  (Linux)
    cuda_gl_interop.h           CUDA OpenGL interop header for toolkit API
                                                                  (Linux)
    cuda_vdpau_interop.h        CUDA VDPAU interop header for toolkit API
                                                                  (Linux)
    cudaD3D9.h                  CUDA DirectX 9 interop header  (Windows)
    cudaD3D10.h                 CUDA DirectX 10 interop header (Windows)
    cudaD3D11.h                 CUDA DirectX 11 interop header (Windows)
    cufft.h                     CUFFT API header
    cublas_v2.h                 CUBLAS API header
    cublas.h                    CUBLAS Legacy API header
    cusparse_v2.h               CUSPARSE API header
    cusparse.h                  CUSPARSE Legacy API header
    curand.h                    CURAND API header
    curand_kernel.h             CURAND device API header
    thrust/*                    Thrust headers
    npp.h                       NPP API header
```

```
    nvToolsExt*.h                NVIDIA Tools Extension headers (Linux and Mac)
    nvcuvid.h                    CUDA Video Decoder header (Windows and Linux)
    cuviddec.h                   CUDA Video Decoder header (Windows and Linux)
    NVEncodeDataTypes.h          CUDA Video Encoder header (Windows; C-library
                                                                or DirectShow)
    NVEncoderAPI.h               CUDA Video Encoder header (Windows; C-library)
    INvTranscodeFilterGUIDs.h    CUDA Video Encoder header (Windows; DirectShow)
    INVVESetting.h               CUDA Video Encoder header (Windows; DirectShow)

extras/
    CUPTI                        CUDA Performance Tool Interface API
    Debugger                     CUDA Debugger API

src/
    *fortran*.{c,h}              FORTRAN interface files for CUBLAS and CUSPARSE
```

3.3.2. Windows lib Files

(Corresponding 32-bit or 64-bit DLLs are in **bin/**.)

```
lib/{Win32,x64}/
    cuda.lib                     CUDA driver library
    cudart.lib                   CUDA runtime library
    cudadevrt.lib                CUDA runtime device library
    cublas.lib                   CUDA BLAS library
    cublas_device.lib            CUDA BLAS device library
    cufft.lib                    CUDA FFT library
    cusparse.lib                 CUDA Sparse Matrix library
    curand.lib                   CUDA Random Number Generation library
    npp.lib                      NVIDIA Performance Primitives library
    nvcuvenc.lib                 CUDA Video Encoder library
    nvcuvid.lib                  CUDA High-level Video Decoder library

    OpenCL.lib                   OpenCL library
```

3.3.3. Linux lib Files

```
lib{64}/
    libcudart.so                 CUDA runtime library
    libcuinj.so                  CUDA internal library for profiling
    libcublas.so                 CUDA BLAS library
    libcublas_device.a           CUDA BLAS device library
    libcufft.so                  CUDA FFT library
    libcusparse.so               CUDA Sparse Matrix library
    libcurand.so                 CUDA Random Number Generation library
    libnpp.so                    NVIDIA Performance Primitives library
```

3.3.4. Mac OS X lib Files

```
lib/
    libcudart.dylib              CUDA runtime library
    libcuinj.dylib               CUDA internal library for profiling
    libcublas.dylib              CUDA BLAS library
    libcublas_device.a           CUDA BLAS device library
    libcufft.dylib               CUDA FFT library
    libcusparse.dylib            CUDA Sparse Matrix library
    libcurand.dylib              CUDA Random Number Generation library
    libnpp.dylib                 NVIDIA Performance Primitives library
    libtlshook.dylib             NVIDIA internal library
```

3.4. Supported NVIDIA Hardware

See http://www.nvidia.com/object/cuda_gpus.html.

3.5. Supported Operating Systems

3.5.1. Windows

- Supported Windows Operating Systems

  Windows 8
  Windows 7
  Windows Vista
  Windows XP
  Windows Server 2012
  Windows Server 2008 R2

Table 6 Windows Compilers Supported in 5.0

Compiler	IDE
Visual C++ 10.0	Visual Studio 2010
Visual C++ 9.0	Visual Studio 2008

3.5.2. Linux

- The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for this CUDA Toolkit release.

Table 7 Linux Distributions Supported in 5.0

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 16	x	x	3.1.0-7.fc16	4.6.2	2.14.90
ICC Compiler 12.x		x			
OpenSUSE 12.1		x	3.1.0-1.2-desktop	4.6.2	2.14.1
Red Hat RHEL 6.x		x	2.6.32-131.0.15.el6	4.4.5	2.12
Red Hat RHEL 5.5+		x	2.6.18-238.el5	4.1.2	2.5
SUSE SLES 11 SP2		x	3.0.13-0.27-pae	4.3.4	2.11.3
SUSE SLES 11.1	x	x	2.6.32.12-0.7-pae	4.3.4	2.11.1
Ubuntu 11.10	x	x	3.0.0-19-generic-pae	4.6.1	2.13

Distribution	32	64	Kernel	GCC	GLIBC
Ubuntu 10.04	x	x	2.6.35-23-generic	4.4.5	2.12.1

Table 8 Linux Distributions Not Supported in 5.0

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 14	x	x	2.6.35.6-45	4.5.1	2.12.90
ICC Compiler 11.1		x			
OpenSUSE 11.2	x	x	2.6.31.5-0.1	4.4.1	2.10.1
Red Hat RHEL 6.x	x		2.6.32-131.0.15.el6	4.4.5	2.12
Red Hat RHEL 5.5+	x		2.6.18-238.el5	4.1.2	2.5
Ubuntu 11.04	x	x	2.6.38-8-generic	4.5.2	2.13

3.5.3. Mac OS X

- Supported Mac Operating Systems

  Mac OS X 10.8.x
  Mac OS X 10.7.x

3.6. Installation Notes

3.6.1. Windows

For silent installation:

- To install, use **`msiexec.exe`** from the shell, passing these arguments:

```
msiexec.exe /i <cuda_toolkit_filename>.msi /qn
```

- To uninstall, use **`/x`** instead of **`/i`**.

3.6.2. Linux

- In order to run CUDA applications, the CUDA module must be loaded and the entries in **`/dev`** created. This may be achieved by initializing X Windows, or by creating a script to load the kernel module and create the entries. An example script (to be run at boot time) follows.

```
#!/bin/bash

/sbin/modprobe nvidia

if [ "$?" -eq 0 ]; then

    # Count the number of NVIDIA controllers found.
    N3D=`/sbin/lspci | grep -i NVIDIA | grep "3D controller" | wc -l`
    NVGA=`/sbin/lspci | grep -i NVIDIA | grep "VGA compatible controller" \
                      | wc -l`
```

```
    N=`expr $N3D + $NVGA - 1`
    for i in `seq 0 $N`; do
        mknod -m 666 /dev/nvidia$i c 195 $i;
    done

    mknod -m 666 /dev/nvidiactl c 195 255

else
    exit 1
fi
```

- On some Linux releases, due to a GRUB bug in the handling of upper memory and a default **vmalloc** too small on 32-bit systems, it may be necessary to pass this information to the bootloader:

```
vmalloc=256MB, uppermem=524288
```

Here is an example of GRUB conf:

```
title Red Hat Desktop (2.6.9-42.ELsmp)
root (hd0,0)
uppermem 524288
kernel /vmlinuz-2.6.9-42.ELsmp ro root=LABEL=/1 rhgb quiet vmalloc=256MB
pci=nommconf
initrd /initrd-2.6.9-42.ELsmp.img
```

3.7. New Features

3.7.1. General CUDA

- Support compatibility between CUDA driver and CUDA toolkit is as follows:
  - Any **nvcc** generated PTX code is forward compatible to newer GPU architectures. This means any CUDA binaries that include PTX code will continue to run on newer GPUs and newer CUDA drivers released from NVIDIA; as the PTX code gets JIT compiled at runtime to the newer GPU architecture.
  - CUDA drivers are backward compatible with CUDA toolkit. This means systems can be upgraded to newer drivers independent of upgrading to newer toolkit. Apps built using old toolkit will load and run with the newer drivers however if they require PTX JIT compilation to run on a newer GPU architecture (SM version) then such apps cannot be used with CUDA tools from old toolkit. Any JIT compiled code implies using the newer compiler and thus a new ABI which requires upgrading to the matching newer toolkit and associated tools.
  - Any separately compiled NVCC binaries (enabled in 5.0) require that all device objects follow the same ABI, and must target the same GPU architecture (SM version). Any CUDA tools usage on these binaries must match the associated toolkit version of the compiler.
- The CUDA 4.2 toolkit for sm_30 implicitly increased a -maxrregcount that was less than 32 to 32. The CUDA 5.0 toolkit does not implicitly increase the -maxrregcount unless it is less than 16 (because the ABI requires at least 16 registers). Note that 32 is the "best minimum" for sm_3x, and the libcublas_device library is compiled for 32 registers.

- Any PTX code generated by NVCC is forward compatible with newer GPU architectures. CUDA binaries that include PTX code will continue to run on newer GPUs with newer NVIDIA CUDA drivers because the PTX code is JIT compiled at runtime to the newer GPU architectures.
- CUDA drivers are backward compatible with the CUDA toolkit. This means systems can be upgraded to newer drivers independently of upgrading to a newer toolkit. Applications built using an older toolkit will load and run with the newer drivers; however, if the applications require PTX JIT compilation to run on a newer GPU architecture (SM version) then they cannot be used with tools from an older CUDA toolkit. Any JIT-compiled code requires using the newer compiler and thus a new ABI, which, in turn, requires upgrading to the matching newer toolkit and associated tools.
- Any separately compiled NVCC binaries (enabled in 5.0) require that all device objects must follow the same ABI and must target the same GPU architecture (SM version). Any CUDA tool used with these binaries must match the associated toolkit version of the compiler.
- Using flag **`cudaStreamNonBlocking`** with **`cudaStreamCreateWithFlags()`** specifies that the created stream will run currently with stream 0 (the NULL stream) and will perform no synchronization with the NULL stream. This flag is functional in the CUDA 5.0 release.
- The **`cudaStreamAddCallback()`** routine introduces a mechanism to perform work on the CPU after work is finished on the GPU, without polling.
- The **`cudaStreamCallbackNonblocking`** option for **`cudaStreamAddCallback()`** and **`cuStreamAddCallback()`** has been removed from the CUDA 5.0 release. Option **`cudaStreamCallbackBlocking`** is supported and is the default behavior when no flags are specified.
- CUDA 5.0 introduces support for Dynamic Parallelism, which is a significant enhancement to the CUDA programming model. Dynamic Parallelism allows a kernel to launch and synchronize with new grids directly from the GPU using CUDA's standard **`<<< >>>`** syntax. A broad subset of the CUDA runtime API is now available on the device, allowing launch, synchronization, streams, events, and more. For complete information, please see the *CUDA Dynamic Parallelism* appendix in the *CUDA C Programming Guide*. CUDA Dynamic Parallelism is available only on SM 3.5 architecture GPUs.
- The use of a character string to indicate a device symbol, which was possible with certain API functions, is no longer supported. Instead, the symbol should be used directly.

3.7.1.1. Linux

- Added the **`cuIpc`** functions, which are designed to allow efficient shared memory communication and synchronization between CUDA processes. Functions **`cuIpcGetEventHandle()`** and **`cuIpcGetMemHandle()`** get an opaque handle that can be freely copied and passed between processes on the same machine. The accompanying **`cuIpcOpenEventHandle()`** and **`cuIpcOpenMemHandle()`** functions allow processes to map handles to resources created in other processes.

3.7.2. CUDA Libraries

3.7.2.1. CUBLAS

- In addition to the usual CUBLAS Library host interface that supports all architectures, the CUDA toolkit now delivers a static CUBLAS library (**`cublas_device.a`**) that provides the same interface but is callable from the device from within kernels. The device interface is only available on Kepler II because it uses the Dynamic Parallelism feature to launch kernels internally. More details can be found in the CUBLAS Documentation.
- The CUBLAS library now supports routines **`cublas{S,D,C,Z}getrfBatched()`**, for batched LU factorization with partial pivoting, and **`cublas{S,D,C,Z}trsmBatched()`** a batched triangular solver. Those two routines are restricted to matrices of dimension <= 32x32.
- The **`cublasCsyr()`**, **`cublasZsyr()`**, **`cublasCsyr2()`**, and **`cublasZsyr2()`** routines were added to the CUBLAS library to compute complex and double-complex symmetric rank 1 updates and complex and double-complex symmetric rank 2 updates respectively. Note, **`cublasCher()`**, **`cublasZher()`**, **`cublasCher2()`**, and **`cublasZher2()`** were already supported in the library and are used for Hermitian matrices.
- The **`cublasCsymv()`** and **`cublasZsymv()`** routines were added to the CUBLAS library to compute symmetric complex and double-complex matrix-vector multiplication. Note, **`cublasChemv()`** and **`cublasZhemv()`** were already supported in the library and are used for Hermitian matrices.
- A pair of utilities were added to the CUBLAS API for all data types. The **`cublas{S,C,D,Z}geam()`** routines compute the weighted sum of two optionally transposed matrices. The **`cublas{S,C,D,Z}dgmm()`** routines compute the multiplication of a matrix by a purely diagonal matrix (represented as a full matrix or with a packed vector).

3.7.2.2. CURAND

- The Poisson distribution has been added to CURAND, for all of the base generators. Poisson distributed results may be generated via a host function, **`curandGeneratePoisson()`**, or directly within a kernel via a device function, **`curand_poisson()`**. The internal algorithm used, and therefore the number of samples drawn per result and overall performance, varies depending on the generator, the value of the frequency parameter (lambda), and the API that is used.

3.7.2.3. CUSPARSE

- Routines to achieve addition and multiplication of two sparse matrices in CSR format have been added to the CUSPARSE Library.

  The combination of the routines **`cusparse{S,D,C,Z}csrgemmNnz()`** and **`cusparse{S,C,D,Z}csrgemm()`** computes the multiplication of two sparse matrices in CSR format. Although the transpose operations on the matrices are supported, only the multiplication of two non-transpose matrices has been

optimized. For the other operations, an actual transpose of the corresponding matrices is done internally.

The combination of the routines **cusparse{S,D,C,Z}csrgeamNnz()** and **cusparse{S,C,D,Z}csrgeam()** computes the weighted sum of two sparse matrices in CSR format.

- The location of the **csrVal** parameter in the **cusparse<t>csrilu0()** and **cusparse<t>csric0()** routines has changed. It now corresponds to the parameter ordering used in other CUSPARSE routines, which represent the matrix in CSR-storage format (**csrVal**, **csrRowPtr**, **csrColInd**).
- The **cusparseXhyb2csr()** conversion routine was added to the CUSPARSE library. It allows the user to verify that the conversion to HYB format was done correctly.
- The CUSPARSE library has added support for two preconditioners that perform incomplete factorizations: incomplete LU factorization with no fill in (ILU0), and incomplete Cholesky factorization with no fill in (IC0). These are supported by the new functions **cusparse{S,C,D,Z}csrilu0()** and **cusparse{S,C,D,Z}csric0()**, respectively.
- The CUSPARSE library now supports a new sparse matrix storage format called Block Compressed Sparse Row (Block-CSR). In contrast to plain CSR which encodes all non-zero primitive elements, the Block-CSR format divides a matrix into a regular grid of small 2-dimensional sub-matrices, and fully encodes all sub-matrices that have any non-zero elements in them. The library supports conversion between the Block-CSR format and CSR via **cusparse{S,C,D,Z}csr2bsr()** and **cusparse{S,C,D,Z}bsr2csr()**, and matrix-vector multiplication of Block-CSR matrices via **cusparse{S,C,D,Z}bsrmv()**.

3.7.2.4. Math

- Single-precision **normcdff()** and double-precision **normcdf()** functions were added. They calculate the standard normal cumulative distribution function.

  Single-precision **normcdfinvf()** and double-precision **normcdfinv()** functions were also added. They calculate the inverse of the standard normal cumulative distribution function.
- The **sincospi(x)** and **sincospif(x)** functions have been added to the math library to calculate the double- and single-precision results, respectively, for both **sin(x * PI)** and **cos(x * PI)** simultaneously. Please see the *CUDA Toolkit Reference Manual* for the exact function prototypes and usage, and the *CUDA C Programmer's Guide* for accuracy information. The performance of **sincospi{f}(x)** should generally be faster than calling **sincos{f}(x * PI)** and should generally be faster than calling **sinpi{f}(x)** and **cospi{f}(x)** separately.
- Intrinsic **__frsqrt_rn(x)** has been added to compute the reciprocal square root of single-precision argument **x**, with the single-precision result rounded according to the IEEE-754 rounding mode **nearest** or **even**.

3.7.2.5. NPP

- The NPP library in the CUDA 5.0 release contains more than 1000 new basic image processing primitives, which include broad coverage for converting colors, copying and moving images, and calculating image statistics.
- Added support for a new filtering-mode for Rotate primitives:

  NPPI_INTER_CUBIC2P_CATMULLROM

  This filtering mode uses cubic Catumul-Rom splines to compute the weights for reconstruction. This and the other two CUBIC2P filtering modes are based on the 1988 SIGGRAPH paper: *Reconstruction Filters in Computer Graphics* by Don P. Mitchell and Arun N. Netravali. At this point NPP only supports the Catmul-Rom filtering for Rotate.

3.7.3. CUDA Tools

3.7.3.1. CUDA Compiler

- The separate compilation **culib** format is not supported in the CUDA 5.0 release.
- From this release, the compiler checks the execution space compatibility among multiple declarations of the same function and generates warnings or errors based on the three rules described below.
  - Generates a warning if a function that was previously declared as **__host__** (either implicitly or explicitly) is redeclared with **__device__** or with **__host__ __device__**. After the redeclaration the function is treated as **__host__ __device__**.
  - Generates a warning if a function that was previously declared as **__device__** is redeclared with **__host__** (either implicitly or explicitly) or with **__host__ __device__**. After the redeclaration the function is treated as **__host__ __device__**.
  - Generates an error if a function that was previously declared as **__global__** is redeclared without **__global__**, or vice versa.
- With this release, **nvcc** allows more than one command-line switch that specifies a compilation phase, unless there is a conflict. Known conflicts are as follows:
  - lib cannot be used with **--link** or **--run**.
  - **--device-link** and **--generate-dependencies** cannot be used with other options that specify final compilation phases.

  When multiple compilation phases are specified, **nvcc** stops processing upon the completion of the compilation phase that is reached first. For example, **nvcc --compile --ptx** is equivalent to **nvcc --ptx**, and **nvcc --preprocess --fatbin** equivalent to **nvcc --preprocess**.
- Separate compilation and linking of device code is now supported. See the *Using Separate Compilation in CUDA* section of the **nvcc** documentation for details.

3.7.3.2. CUDA-GDB

- (Linux and Mac OS) CUDA-GDB fully supports Dynamic Parallelism, a new feature introduced with the 5.0 Toolkit. The debugger is able to track kernels launched from another kernel and to inspect and modify their variables like any CPU-launched kernel.
- When the environment variable **`CUDA_DEVICE_WAITS_ON_EXCEPTION`** is used, the application runs normally until a device exception occurs. The application then waits for the debugger to attach itself to it for further debugging.
- Inlined subroutines are now accessible from the debugger on SM 2.0 and above. The user can inspect the local variables of those subroutines and visit the call frame stack as if the routines were not inlined.
- Checking the error codes of all CUDA driver API and CUDA runtime API function calls is vital to ensure the correctness of a CUDA application. Now the debugger is able to report, and even stop, when any API call returns an error. See the CUDA-GDB documentation on **`set cuda api_failures`** for more information.
- It is now possible to attach the debugger to a CUDA application that is already running. It is also possible to detach it from the application before letting it run to completion. When attached, all the usual features of the debugger are available to the user, just as if the application had been launched from the debugger.

3.7.3.3. CUDA-MEMCHECK

- CUDA-MEMCHECK, when used from within the debugger, now displays the address space and the address of the faulty memory access.
- CUDA-MEMCHECK now displays the backtrace on the host and device when an error is discovered.
- CUDA-MEMCHECK now detects **`double free()`** and **`invalid free()`** on the device.
- The precision of the reported errors for **`local`**, **`shared`**, and **`global`** memory accesses has been improved.
- CUDA-MEMCHECK now reports leaks originating from the device heap.
- CUDA-MEMCHECK now reports error codes returned by the runtime API and the driver API in the user application.
- CUDA-MEMCHECK now supports reporting data access hazards in shared memory. Use the **`--tool racecheck`** command-line option to activate.

3.7.3.4. NVIDIA Nsight Eclipse Edition

- (Linux and Mac OS) Nsight Eclipse Edition is an all-in-one development environment that allows developing, debugging, and optimizing CUDA code in an integrated UI environment.

3.7.3.5. NVIDIA Visual Profiler, Command Line Profiler

- As mentioned in the Release Highlights, the tool, **`nvprof`**, is now available in release 5.0 for collecting profiling information from the command-line.

3.8. Performance Improvements

3.8.1. CUDA Libraries

3.8.1.1. CUBLAS

- On Kepler architectures, shared-memory access width can be configured for 4-byte banks (default) or 8-byte banks using the routine **`cudaDeviceSetSharedMemConfig()`**. The CUBLAS and CUSPARSE libraries do not affect the shared-memory configuration, although some routines might benefit from it. It is up to users to choose the best shared-memory configuration for their applications prior to calling the CUBLAS or CUSPARSE routines.
- In CUDA Toolkit 5.0, **`cublas<S,D,C,Z>symv()`** and **`cublas<C/Z>chemv()`** have an alternate, faster implementation that uses atomics. The regular implementation, which gives predictable results from one run to another, is run by default. The routine **`cublasSetAtomicsMode()`** can be used to choose the alternate, faster version.

3.8.1.2. CURAND

- In CUDA CURAND for 5.0, the Box-Muller formula, used to generate double-precision normally distributed results, has been optimized to use **`sincospi()`** instead of individual calls to **`sin()`** and **`cos()`** with multipliers to scale the parameters. This results in a 30% performance improvement on a Tesla C2050, for example, when generating double-precision normal results.

3.8.1.3. Math

- The performance of the double-precision **`fmod()`**, **`remainder()`**, and **`remquo()`** functions has been significantly improved for **`sm_13`**.
- The **`sin()`** and **`cos()`** family of functions [**`sin()`**, **`sinpi()`**, **`cos()`**, and **`cospi()`**] have new implementations in this release that are more accurate and faster. Specifically, all of these functions have a worst-case error bound of 1 ulp, compared to 2 ulps in previous releases. Furthermore, the performance of these functions has improved by 25% or more, although the exact improvement observed can vary from kernel to kernel. Note that the **`sincos()`** and **`sincospi()`** functions also inherit any accuracy improvements from the component functions.
- Function **`erfcinvf()`** has been significantly optimized for both the Tesla and Fermi architectures, and the worst case error bound has improved from 7 ulps to 4 ulps.

3.9. Resolved Issues

3.9.1. General CUDA

- When PTX JIT is used to execute **sm_1x**- or **sm_2x**-native code on Kepler, and when the maximum grid dimension is selected based on the grid-size limits reported by **cudaGetDeviceProperties()**, a conflict can occur between the grid size used and the size limit presumed by the JIT'd device code.

  The grid size limit on devices of compute capability 1.x and 2.x is 65535 blocks per grid dimension. If an application attempts to launch a grid with >= 65536 blocks in the x dimension on such devices, the launch fails outright, as expected. However, because Kepler increased the limit (for the x dimension) to 2^{31}-1 blocks per grid, previous CUDA Driver releases allowed such a grid to launch successfully; but this grid exceeds the number of blocks that can fit into the 16-bit grid size and 16-bit block index assumed by the compiled device code. Beginning in CUDA release 5.0, launches of kernels compiled native to earlier GPUs and JIT'd onto Kepler now return an error as they would have with the earlier GPUs, avoiding the silent errors that could otherwise result.

  This can still pose a problem for applications that select their grid launch dimensions based on the limits reported by **cudaGetDeviceProperties()**, since this function reports 2^{31}-1 for the grid size limit in the x dimension for Kepler GPUs. Applications that correctly limited their launches to 65535 blocks per grid in the x dimension on earlier GPUs may attempt bigger launches on Kepler--yet these launches will fail. To work around this issue for existing applications that were not built with Kepler-native code, a new environment variable has been added for backward compatibility with earlier GPUs: setting **CUDA_GRID_SIZE_COMPAT = 1** causes **cudaGetDeviceProperties()** to conservatively underreport 65535 as the maximum grid dimension on Kepler, allowing such applications to work as expected.
- Functions **cudaGetDeviceProperties()**, **cuDeviceGetProperties()**, and **cuDeviceGetAttribute()** may return the incorrect clock frequency for the SM clock on Kepler GPUs.

3.9.2. CUDA Libraries

3.9.2.1. CURAND

- In releases prior to CUDA 5.0, the CURAND pseudorandom generator MRG32k3a returned integer results in the range 1 through 4294967087 (the larger of two primes used in the generator). CUDA 5.0 results have been scaled to extend the range to 4294967295 (2^{32} - 1). This causes the generation of integer sequences that are somewhat different from previous releases. All other distributions (that is, uniform, normal, log-normal, and Poisson) were already correctly scaled and are not affected by this change.

3.9.2.2. CUSPARSE

- An extra parameter (**int * nnzTotalDevHostPtr**) was added to the parameters accepted by the functions **cusparseXcsrgeamNnz()** and

cusparseXcsrgemmNnz(). The memory pointed to by **nnzTotalDevHostPtr** can be either on the device or host, depending on the selected **CUBLAS_POINTER_MODE**. On exit, ***nnzTotalDevHostPtr** holds the total number of non-zero elements in the resulting sparse matrix C.

3.9.2.3. NPP

- The **nppiLUT_Linear_8u_C1R** and all other LUT primitives that existed in NPP release 4.2 have undergone an API change. The pointers provided for the parameters **pValues** and **pLevels** have to be device pointers from version 5.0 onwards. In the past, those two values were expected to be host pointers, which was in violation of the general NPP API guideline that all pointers to NPP functions are device pointers (unless explicitly noted otherwise).
- The implementation of the **nppiWarpAffine*()** routines in the NPP library have been completely replaced in this release. This fixes several outstanding bugs related to these routines.
- Added these two primitives, which were temporarily removed from release 4.2:

  nppiAbsDiff_8u_C3R
  nppiAbsDiff_8u_C4R

3.9.2.4. Thrust

- The version of Thrust included with the current CUDA toolkit was upgraded to version 1.5.3 in order to address several minor issues.

3.9.3. CUDA Tools

- (Windows) The file **fatbinary.h** has been released with the CUDA 5.0 Toolkit. The file, which replaces **__cudaFatFormat.h**, describes the format used for all fat binaries since CUDA 4.0.

3.9.3.1. CUDA Compiler

- The CUDA compiler driver, **nvcc**, predefines the macro **__NVCC__**. This macro can be used in C/C++/CUDA source files to test whether they are currently being compiled by **nvcc**. In addition, **nvcc** predefines the macro **__CUDACC__**, which can be used in source files to test whether they are being treated as CUDA source files. The **__CUDACC__** macro can be particularly useful when writing header files.
- It is to be noted that the previous releases of **nvcc** also predefined the **__CUDACC__** macro; however, the description in the document *The CUDA Compiler Driver NVCC* was incorrect. The document has been corrected in the CUDA 5.0 release.

3.9.3.2. CUDA Occupancy Calculator

- There was an issue in the CUDA Occupancy Calculator that caused it to be overly conservative in reporting the theoretical occupancy on Fermi and Kepler when the number of warps per block was not a multiple of 2 or 4.

3.10. Known Issues

3.10.1. General CUDA

- 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`**.
- Individual GPU program launches are limited to a run time of less than 5 seconds on a GPU with a display attached. Exceeding this time limit usually causes a launch failure reported through the CUDA driver or the CUDA runtime. GPUs without a display attached are not subject to the 5 second runtime restriction. For this reason it is recommended that CUDA be run on a GPU that is NOT attached to a display and does not have the Windows desktop extended onto it. In this case, the system must contain at least one NVIDIA GPU that serves as the primary graphics adapter.

3.10.1.1. Linux, Mac OS

- Device code linking does not support object files that are in Mac OS fat-file format. As a result, the device libraries included in the toolkit (libcudadevrt.a and libcublas_device.a) do not use the fat file format and only contain code for a 64-bit architecture. In contrast, the other libraries in the toolkit on the Mac OS platform do use the fat file format and support both 32-bit and 64-bit architectures.
- At the time of this release, there are no Mac OS configurations available that support GPUs that implement the sm_35 architecture. Code that targets this architecture can be built, but cannot be run or tested on a Mac OS platform with the CUDA 5.0 toolkit.
- The Linux kernel provides a mode where it allows user processes to overcommit system memory. (Refer to kernel documentation for **`/proc/sys/vm/`** for details). If this mode is enabled (the default on many distros) the kernel may have to kill processes in order to free up pages for allocation requests. The CUDA driver process, especially for CUDA applications that allocate lots of zero-copy memory with **`cuMemHostAlloc()`** or **`cudaMallocHost()`**, is particularly vulnerable to being killed in this way. Since there is no way for the CUDA SW stack to report an OOM error to the user before the process disappears, users, especially on 32-bit Linux, are encouraged to disable memory overcommit in their kernel to avoid this problem. Please refer to documentation on **`vm.overcommit_memory`** and **`vm.overcommit_ratio`** for more information.
- When compiling with GCC, special care must be taken for structs that contain 64-bit integers. This is because GCC aligns long longs to a 4-byte boundary by default, while **`nvcc`** aligns long longs to an 8-byte boundary by default. Thus, when using GCC to compile a file that has a struct/union, users must give the **`-malign-double`** option to GCC. When using **`nvcc`**, this option is automatically passed to GCC.
- (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**.

- (Mac OS) To save power, some Apple products automatically power down the CUDA- capable GPU in the system. If the operating system has powered down the CUDA-capable GPU, CUDA fails to run and the system returns an error that no device was found. In order to ensure that your CUDA-capable GPU is not powered down by the operating system do the following:

  1. Go to **System Preferences**.
  2. Open the **Energy Saver** section.
  3. Uncheck the **Automatic graphics switching** box in the upper left.

3.10.1.2. Windows

- Individual kernels are limited to a 2-second runtime by Windows Vista. Kernels that run for longer than 2 seconds will trigger the Timeout Detection and Recovery (TDR) mechanism. For more information, see http://www.microsoft.com/whdc/device/display/wddm_timeout.mspx.
- The maximum size of a single memory allocation created by **`cudaMalloc()`** or **`cuMemAlloc()`** on WDDM devices is limited to

  `MIN( (`System Memory Size in MB - 512 MB**`) / 2,`**
  `PAGING_BUFFER_SEGMENT_SIZE ).`

  For Vista, **`PAGING_BUFFER_SEGMENT_SIZE`** is approximately 2 GB.

3.10.2. CUDA Libraries

3.10.2.1. NPP

- The NPP **`ColorTwist_32f_8u_P3R`** primitive does not work properly for line strides that are not 64-byte aligned. This issue can be worked around by using the image memory allocators provided by the NPP library.

3.10.3. CUDA Tools

With separate compiled binaries the values of the local variables may be incorrect in the debugger, please use fully compiled binaries while debugging.

3.10.3.1. CUDA Compiler

- (Windows) Because Microsoft changed the declaration of the **`hypot()`** function between MSVC v9 and MSVC v10, users of Microsoft Visual Studio 2010 who link with the new **`cublas_device.lib`** and **`cudadevrt.lib`** device-code libraries may encounter an error. Specifically, performing device- and host-linking in a single pass using **`NVCC`** on a system with Visual Studio 2010 gives the error **`unresolved external symbol hypot`**. Users who encounter this error can avoid it by linking in two stages: first device-link with **`nvcc -dlink`** and then host-link using **`cl`**. This error should not arise from the VS2010 IDE when using the CUDA plug-in, as that plug-in already links in two stages.
- 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.

- (Linux) The **__float128** data type is not supported for the **gcc** host compiler.
- (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 work around for this issue is to manually add padding so that the structs between the host compiler and CUDA are consistent.
- (Windows) When the **PATH** environment variable contains double quotes (**"**), **nvcc** may fail to set up the environment for Microsoft Visual Studio 2010, generating an error. This is because **nvcc** runs **vcvars32.bat** or **vcvars64.bat** to set up the environment for Microsoft Visual Studio 2010 and these batch files are not always able to process **PATH** if it contains double quotes.

  One workaround for this issue is as follows:

  1. Make sure that **PATH** does not contain any double quotes.
  2. Run **vcvars32.bat** or **vcvars64.bat**, depending on the system.
  3. Add the directories that need to be added to **PATH** with double quotes.
  4. Run **NVCC** with the **--use-local-env** switch.

3.10.3.2. NVIDIA Visual Profiler, Command Line Profiler

- On Mac OS X systems with NVIDIA drivers earlier than version 295.10.05, the Visual Profiler may fail to import session files containing profile information collected from GPUs with compute capability 3.0 or later.
- If required, a Java installation is triggered the first time the Visual Profiler is launched. If this occurs, the Visual Profiler must be exited and restarted.
- Visual Profiler fails to generate events or counter information. Here are a couple of reasons why Visual Profiler may fail to gather counter information.

  More than one tool is trying to access the GPU. To fix this issue please make sure only one tool is using the GPU at any given point. Tools include the CUDA command line profiler, Parallel NSight Analysis Tools and Graphics Tools, and applications that use either CUPTI or PerfKit API (NVPM) to read counter values.

  More than one application is using the GPU at the same time Visual Profiler is profiling a CUDA application. To fix this issue please close all applications and just run the one with Visual Profiler. Interacting with the active desktop should be avoided while the application is generating counter information. Please note that for some types of counters Visual Profiler gathers counters for only one context if the application is using multiple contexts within the same application.
- Enabling certain counters can cause GPU kernels to run longer than the driver's watchdog time-out limit. In these cases the driver will terminate the GPU kernel resulting in an application error and profiling data will not be available. Please

disable the driver watchdog time out before profiling such long running CUDA kernels.

- On Linux, setting the X Config option **`Interactive`** to **`false`** is recommended.
- For Windows, detailed information on disabling the Windows TDR is available at

  http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx#E2

- Enabling counters on GPUs with compute capability (SM type) 1.x can result in occasional hangs. Please disable counters on such runs.
- The **`warp serialize`** counter for GPUs with compute capability 1.x is known to give incorrect and high values for some cases.
- To ensure that all profile data is collected and flushed to a file, **`cudaDeviceSynchronize()`** followed by either **`cudaDeviceReset()`** or **`cudaProfilerStop()`** should be called before the application exits.
- Counters **`gld_incoherent`** and **`gst_incoherent`** always return zero on GPUs with compute capability (SM type) 1.3. A value of zero doesn't mean that all load/stores are 100% coalesced.
- Use Visual Profiler version 4.1 onwards with NVIDIA driver version 285 (or later). Due to compatibility issues with profile counters, Visual Profiler 4.0 (or earlier) must not be used with NVIDIA driver version 285 (or later).

3.11. Source Code for Open64 and CUDA-GDB

- The Open64 and CUDA-GDB source files are controlled under terms of the GPL license. Current and previously released versions are located here:

  ftp://download.nvidia.com/CUDAOpen64/.
- Linux users:
  - Please refer to the *Release Notes* and *Known Issues* sections in the *CUDA-GDB User Manual* (**`CUDA_GDB.pdf`**).
  - Please refer to **`CUDA_Memcheck.pdf`** for notes on supported error detection and known issues.

3.12. More Information

- For more information and help with CUDA, please visit http://www.nvidia.com/cuda.
- Please refer to the *LLVM Release License* text in **`EULA.txt`** for details on LLVM licensing.

Chapter 4. NVIDIA CUDA TOOLKIT V4.2 RELEASE NOTES

4.1. Errata

4.1.1. Known Issues

- Functions **cudaGetDeviceProperties**, **cuDeviceGetProperties**, and **cuDeviceGetAttribute** may return the incorrect clock frequency for the SM clock on Kepler GPUs. [Windows and Linux]
- In CUDA Toolkit 4.2, the functions **cudaDeviceGetSharedMemConfig()** and **cudaDeviceSetSharedMemConfig()** were added for Kepler. However, the CUDA Reference Manual included with CUDA Toolkit 4.2 was not regenerated to include documentation for these functions. The functions are documented in the Doxygen comments in the file **include/cuda_runtime_api.h** in the toolkit installation directory.
- If required, a Java installation is triggered the first time the Visual Profiler is launched. If this occurs, the Visual Profiler must be exited and restarted.
- *GraphCut* is not supported on GPUs with less than compute capability 1.1.
- In the *CUDA C Programming Guide* for CUDA Toolkit 4.2, some of the instruction throughputs listed for compute capability 3.0 in Table 5.1 are incorrect. The table has been corrected in the externally linked document on *DevZone* and will be corrected in the next version of the *CUDA C Programming Guide*.

4.2. Release Highlights

- Added support for GK10x Kepler GPUs.
- This release contains the following:
  - NVIDIA CUDA Toolkit documentation
  - NVIDIA OpenCL documentation

- NVIDIA CUDA compiler (nvcc) and supporting tools
- NVIDIA CUDA runtime libraries
- NVIDIA CUDA-GDB debugger
- NVIDIA CUDA-MEMCHECK
- NVIDIA Visual Profiler
- NVIDIA CUBLAS, CUFFT, CUSPARSE, CURAND, Thrust, and NPP libraries

4.3. Documentation

For a list of documents supplied with this release, please refer to the /doc directory of your CUDA Toolkit installation.

The NVML development package is not shipped with CUDA 4.2. For changes related to *nvidia-smi* and NVML, please refer to the nvidia-smi man page and the *Tesla Deployment Kit* package located on the developer site http://developer.nvidia.com/tesla-deployment-kit; NVML documentation and the SDK are included.

4.4. List of Important Files

```
bin/
    nvcc                        CUDA C/C++ compiler
    cuda-gdb                    CUDA Debugger
    cuda-memcheck               CUDA Memory Checker
    nvvp                        NVIDIA Visual Profiler
                                (On Windows, nvvp is located in libnvvp/)

include/
    cuda.h                      CUDA driver API header
    cudaGL.h                    CUDA OpenGL interop header for driver API
    cudaVDPAU.h                 CUDA VDPAU interop header for driver API
                                (Linux only)
    cuda_gl_interop.h           CUDA OpenGL interop header for toolkit API
                                (Linux only)
    cuda_vdpau_interop.h        CUDA VDPAU interop header for toolkit API
                                (Linux only)
    cudaD3D9.h                  CUDA DirectX 9 interop header (Windows only)
    cudaD3D10.h                 CUDA DirectX 10 interop header (Windows only)
    cudaD3D11.h                 CUDA DirectX 11 interop header (Windows only)
    cufft.h                     CUFFT API header
    cublas_v2.h                 CUBLAS API header
    cublas.h                    CUBLAS Legacy API header
    cusparse_v2.h               CUSPARSE API header
    cusparse.h                  CUSPARSE Legacy API header
    curand.h                    CURAND API header
    curand_kernel.h             CURAND device API header
    thrust/*                    Thrust Headers
    npp.h                       NPP API Header
    nvcuvid.h                   CUDA Video Decoder header (Windows and Linux)
    cuviddec.h                  CUDA Video Decoder header (Windows and Linux)
    NVEncodeDataTypes.h         CUDA Video Encoder (C-library or DirectShow)
                                (Windows only)
    NVEncodeAPI.h               CUDA Video Encoder (C-library) (Windows only)
    INvTranscodeFilterGUIDs.h   CUDA Video Encoder (DirectShow) (Windows only)
    INVVESetting.h              CUDA Video Encoder (DirectShow) (Windows only)

extras/
```

```
    CUPTI                       CUDA Profiling APIs
    Debugger                    CUDA Debugger APIs
```

4.4.1. Windows lib Files

```
lib/
    cuda.lib                    CUDA driver library
    cudart.lib                  CUDA runtime library
    cublas.lib                  CUDA BLAS library
    cufft.lib                   CUDA FFT library
    cusparse.lib                CUDA Sparse Matrix library
    curand.lib                  CUDA Random Number Generation library
    npp.lib                     NVIDIA Performance Primitives library
    nvcuvenc.lib                CUDA Video Encoder library
    nvcuvid.lib                 CUDA Video Decoder library
```

4.4.2. Linux lib Files

```
lib/
    libcuda.so                  CUDA driver library
    libcudart.so                CUDA runtime library
    libcublas.so                CUDA BLAS library
    libcufft.so                 CUDA FFT library
    libcusparse.so              CUDA Sparse Matrix library
    libcurand.so                CUDA Random Number Generation library
    libnpp.so                   NVIDIA Performance Primitives library
```

4.4.3. Mac OS X lib Files

```
lib/
    libcudart.dylib               CUDA runtime library
    libcuinj.dylib                CUDA internal library for profiling
    libcublas.dylib               CUDA BLAS library
    libcublas_device.a            CUDA BLAS device library
    libcufft.dylib                CUDA FFT library
    libcusparse.dylib             CUDA Sparse Matrix library
    libcurand.dylib               CUDA Random Number Generation library
    libnpp.dylib                  NVIDIA Performance Primitives library
    libtlshook.dylib              NVIDIA internal library
```

4.5. Supported NVIDIA Hardware

See http://www.nvidia.com/object/cuda_gpus.html.

4.6. Supported Operating Systems

4.6.1. Windows

- Supported Operating Systems

  Windows 7
  Windows Vista
  Windows XP
  Windows Server 2008

Table 9 Windows Compilers Supported in 4.2

Compiler	IDE
MSVC8 (14.00)	VS 2005
MSVC9 (15.00)	VS 2008
MSVC2010 (16.00)	VS 2010

4.6.2. Linux

The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on *distro* versions that have been qualified for this CUDA Toolkit release.

Table 10 Linux Distributions Supported in 4.2

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 14	x	x	2.6.35.6-45	4.5.1	2.12.90
ICC Compiler 11.1	x	x			
OpenSUSE-11.2	x	x	2.6.31.5-0.1	4.4.1	2.10.1
RHEL-5.>=5 (5.5, 5.6, 5.7)	x	x	2.6.18-238.el5	4.1.2	2.5
RHEL-6.X (6.0, 6.1)		x	2.6.32-131.0.15.el6	4.4.5	2.12
SLES 11.1	x	x	2.6.32.12-0.7-pae	4.3-62.198	2.11.1-0.17.4
Ubuntu-10.04	x	x	2.6.35-23-generic	4.4.5	2.12.1
Ubuntu-11.04	x	x	2.6.38-8-generic	4.5.2	2.13

Table 11 Linux Distributions Not Supported in 4.2

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 13	x	x	2.6.33.3-85	4.4.4	2.12
RHEL-4.8	x		2.6.9-89.ELsmpl	3.4.6	2.3.4
Ubuntu-10.10	x	x	2.6.35-23-generic	4.4.5	2.12.1

32-bit versions of RHEL 4.8 and RHEL 6.0 have not been tested with this release and are therefore not supported in this CUDA Toolkit release.

4.6.3. Mac OS X

Table 12 Mac OS X Platforms Supported in 4.2

Platform	32	64	GCC
Mac OS X 10.7	x	x	4.2.1 (build 5646)
Mac OS X 10.6	x	x	4.2.1 (build 5646)

4.7. Installation Notes

4.7.1. Windows

For silent installation:

- To install, use **msiexec.exe** from the shell, passing these arguments:

```
msiexec.exe /i cudatoolkit.msi /qn
```

- To uninstall, use **/x** instead of **/i**.

4.7.2. Linux

- In order to run CUDA applications, the CUDA module must be loaded and the entries in **/dev** created. This may be achieved by initializing X Windows, or by creating a script to load the kernel module and create the entries. An example script (to be run at boot time):

```
#!/bin/bash

/sbin/modprobe nvidia

if [ "$?" -eq 0 ]; then

    # Count the number of NVIDIA controllers found.
    N3D=`/sbin/lspci | grep -i NVIDIA | grep "3D controller" | wc -l`
    NVGA=`/sbin/lspci | grep -i NVIDIA | grep "VGA compatible controller" \
                      | wc -l`

    N=`expr $N3D + $NVGA - 1`
    for i in `seq 0 $N`; do
        mknod -m 666 /dev/nvidia$i c 195 $i;
    done

    mknod -m 666 /dev/nvidiactl c 195 255

else
    exit 1
fi
```

- On some Linux releases, due to a GRUB bug in the handling of upper memory and a default **vmalloc** too small on 32-bit systems, it may be necessary to pass this information to the bootloader:

```
vmalloc=256MB, uppermem=524288
```

Example of GRUB conf:

```
title Red Hat Desktop (2.6.9-42.ELsmp)
root (hd0,0)
uppermem 524288
kernel /vmlinuz-2.6.9-42.ELsmp ro root=LABEL=/1 rhgb quiet vmalloc=256MB
pci=nommconf
initrd /initrd-2.6.9-42.ELsmp.im
```

- Pinned memory in CUDA is only supported on Linux kernel versions >= 2.6.18. Host side memory allocations pinned for CUDA using **cudaHostRegister()** API can be passed to 3rd party drivers. Pinned memory allocations returned from **cudaHostAlloc()** and **cudaMallocHost()** can also be passed to 3rd party drivers and starting with 4.1, **CUDA_NIC_INTEROP** is no longer needed on these APIs; thus this flag is now deprecated.

4.8. New Features

Support for GK10x Kepler GPUs.

4.9. Resolved Issues

- In the routines **cusparse<T>csr2hyb** and **cusparse<T>dense2hyb**, upon the occurrence of an error (typically a device memory allocation problem), the handle to the hybrid format descriptor (**cusparseHybMat_t**) was wrongly destroyed using **cusparseDestroyHybMat**. A subsequent call to **cusparseDestroyHybMat** by the user would then result in an error. This issue has been fixed in the 4.2 toolkit and now the user can and should call **cusparseDestroyHybMat** to clean up, either after an error or when the matrix is no longer needed.
- **CUDA-MEMCHECK** now explicitly reports calls to **assert()** inside a CUDA kernel.
- The version of Thrust included with the CUDA toolkit has been upgraded from 1.5.1 to 1.5.2.
- Rotate primitives falsely used to enforce that the source image's pitch (**nSrcStep**) was large enough to accommodate the destination ROI's size. This bug was fixed and the restriction no longer exists.
- Starting with CUDA Toolkit 4.0, **cublasDestroy** did not properly free all of the GPU resources, leading to a GPU memory leak of about 256 KB per CUBLAS handle. This could also lead to GPU memory fragmentation when the unreleased resources were scattered over the GPU memory. This issue has been resolved in the 4.2 Toolkit.

4.10. Known Issues

4.10.1. Windows

- In the NPP library, the **nppiGraphcut_32s8u()** and **nppiGraphcut8_32s8u()** primitives may fail with an error while running on a GPU that supports the sm1.0 architecture, especially on systems with a 64-bit operating system.
- Individual kernels are limited to a 2-second runtime by Windows Vista. Kernels that run for longer than 2 seconds will trigger the *Timeout Detection and Recovery* (TDR) mechanism. For more information, see http://www.microsoft.com/whdc/device/display/wddm_timeout.mspx
- The maximum size of a single memory allocation created by **cudaMalloc** or **cuMemAlloc** on WDDM devices is limited to: MIN((System Memory Size in MB - 512 MB) / 2, **PAGING_BUFFER_SEGMENT_SIZE**) For Vista, **PAGING_BUFFER_SEGMENT_SIZE** is approximately 2GB.
- (Windows and Linux): Individual GPU program launches are limited to a run time of less than 5 seconds on a GPU with a display attached. Exceeding this time limit usually causes a launch failure reported through the CUDA driver or the CUDA runtime. GPUs without a display attached are not subject to the 5 second runtime restriction. For this reason it is recommended that CUDA be run on a GPU that is NOT attached to a display and does not have the Windows desktop extended onto it. In this case, the system must contain at least one NVIDIA GPU that serves as the primary graphics adapter.

4.10.2. Linux & Mac

- In the NPP library, the **nppiGraphcut_32s8u()** and **nppiGraphcut8_32s8u()** primitives may fail with an error while running on a GPU that supports the **sm1.0** architecture, especially on systems with a 64-bit operating system.
- The Linux kernel provides a mode where it allows user processes to overcommit system memory. (Refer to kernel documentation for **/proc/sys/vm/** for details). If this mode is enabled (the default on many distros) the kernel may have to kill processes in order to free up pages for allocation requests. The CUDA driver process, especially for CUDA applications that allocate lots of zero-copy memory with **cuMemHostAlloc** or **cudaMallocHost**, is particularly vulnerable to being killed in this way. Since there is no way for the CUDA SW stack to report an OOM error to the user before the process disappears, users, especially on 32-bit Linux, are encouraged to disable memory overcommit in their kernel to avoid this problem. Please refer to documentation on *vm.overcommit_memory* and *vm.overcommit_ratio* for more information.
- When compiling with GCC, special care must be taken for structs that contain 64-bit integers. This is because GCC aligns long longs to a 4 byte boundary by default, while NVCC aligns long longs to an 8 byte boundary by default. Thus, when using GCC to compile a file that has a struct/union, users must give the -malign-double option to GCC. When using NVCC, this option is automatically passed to GCC.

4.10.3. Mac

To save power, some Apple products automatically power-down the CUDA-capable GPU in the system. If the operating system has powered down the CUDA-capable GPU, CUDA fails to run and the system returns an error that no device was found. In order to ensure that your CUDA-capable GPU is not powered down by the operating system do the following:

- Go to **System Preferences**
- Open the **Energy Saver** section
- Un-check the **Automatic graphics switching** check box in the upper left.

4.10.4. Visual Profiler and Command Line Profiler

- Visual Profiler fails to generate events or counter information. There are several reasons why Visual Profiler may fail to gather counter information:
  - If more than one tool is trying to access the GPU. To fix this issue please make sure only one tool is using the GPU at any given point. Tools include the CUDA command line profiler, Parallel NSight Analysis Tools and Graphics Tools, and applications that use either CUPTI or PerfKit API (NVPM) to read counter values.
  - If more than one application is using the GPU at the same time when Visual Profiler is profiling a CUDA application. To fix this issue please close all applications and just run the one with Visual Profiler. Interacting with the active desktop should be avoided while the application is generating counter information. Please note that Visual Profiler gathers counters for only one context if the application is using multiple contexts within the same application.
- Enabling `{gld|gst} instructions {8|16|32|64|128}bit` counters can cause GPU kernels to run longer than the driver's watchdog timeout limit. In these cases the driver will terminate the GPU kernel resulting in an application error and profiling data will not be available. Please disable driver watchdog timeout before profiling such long running CUDA kernels.
  - On Linux, setting the X Config option 'Interactive' to false is recommended.
  - For Windows, detailed information on disabling the Windows TDR is available at: http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx#E2
- On Windows Vista/Win7 profiling an application which makes more than 32K CUDA kernel launch, memory copy, or memory set API calls without a synchronization call can result in an application hang. To work around this issue add synchronization calls like `cudaDeviceSynchronize()` or `cudaStreamSynchronize()`.
- Enabling counters on GPUs with compute capability (SM type) 1.x can result in occasional hangs. Please disable counters on such runs.
- The *warp serialize* counter for GPUs with compute capability 1.x is known to give incorrect and high values for some cases.
- Prof triggers are not supported on GPUs with compute capability (SM type) 1.0.

- Profiler data gets flushed to a file only at synchronization calls like **`cudaDeviceSynchronize()`** and **`cudaStreamSynchronize()`** or when the profiler buffer gets full. If an app terminates without these sync calls then profiler data may be lost.
- Counters **`gld_incoherent`** and **`gst_incoherent`** always return zero on GPUs with compute capability (SM type) 1.3. A value of zero doesn't mean that all load/stores are 100% coalesced.
- Use Visual Profiler version 4.1 onwards with driver version 285 (or later).Due to compatibility issues with profile counters, Visual Profiler 4.0 (or earlier) must not be used with driver version 285 (or later).

4.11. Source Code for Open64 and CUDA-GDB

- The Open64 and CUDA-GDB source files are controlled under terms of the GPL license. Current and previously released versions are located at: ftp://download.nvidia.com/CUDAOpen64
- Linux users:
  - Please refer to the *Release Notes* and *Known Issues* sections in the *CUDA-GDB User Manual* (**`CUDA_GDB.pdf`**).
  - Please refer to **`CUDA_Memcheck.pdf`** for notes on supported error detection and known issues.

4.12. More Information

- For more information and help with CUDA, please visit http://www.nvidia.com/cuda.
- Please refer to the *LLVM Release License* text in **`EULA.txt`** for details on LLVM licensing.

Chapter 5. NVIDIA CUDA TOOLKIT V4.1 RELEASE NOTES

5.1. Release Highlights

This release contains:

- NVIDIA CUDA Toolkit documentation
- NVIDIA OpenCL documentation
- NVIDIA CUDA compiler (nvcc) and supporting tools
- NVIDIA CUDA runtime libraries
- NVIDIA CUBLAS, CUFFT, CUSPARSE, CURAND, Thrust, and NPP libraries

Visual Profiler release notes and ChangeLog information are now consolidated into this release notes documents.

NVIDA CUDA Toolkit version 4.1 has the following new features:

- Advanced application development features
  - New LLVM-based compiler delivers up to 10% faster performance for many applications
  - Access to 3D surfaces and cube maps from device code
  - Peer-to-peer communication between processes
  - Support for resetting a GPU in **`nvidia-smi`**, without rebooting the system
- New and improved *drop-in* acceleration with GPU-Accelerated Libraries
  - Over 1000 new image processing functions in the NPP library
  - New **`cuSPARSE`** tri-diagonal solver up to 10x faster than MKL on a 6 core CPU
  - Up to 2x faster sparse matrix vector multiply using ELL hybrid format
  - New support in cuRAND for MRG32k3a and *Mersenne Twister* (MTGP11213) RNG algorithms

- Bessel functions now supported in the CUDA standard Math library (**j0**, **j1**, **jn**, **y0**, **y1**, **yn**)
- Learn more about GPU-Accelerated Libraries at: http://developer.nvidia.com/gpu-accelerated-libraries

- Enhanced and redesigned developer tools
  - Redesigned Visual Profiler with automated performance analysis and expert guidance
  - CUDA-GDB support for multi-context debugging and **assert()** in device code
  - CUDA-MEMCHECK now detects out of bounds access for memory allocated in device code
  - Learn more about debugging and performance analysis tools for GPU developers at: http://developer.nvidia.com/cuda-tools-ecosystem

5.2. Documentation

For a list of documents supplied with this release, please refer to the **/doc** directory of your CUDA Toolkit installation.

The NVML development package is no longer shipped with CUDA 4.1. For changes related to nvidia-smi and NVML, please refer to nvidia-smi man page and the *Tesla Deployment Kit* package located on the developer site; NVML documentation and the SDK are included.

5.3. List of Important Files

```
bin/
    nvcc                        CUDA C/C++ compiler
    cuda-gdb                    CUDA Debugger
    cuda-memcheck               CUDA Memory Checker
    nvvp                        NVIDIA Visual Profiler

include/
    cuda.h                      CUDA driver API header
    cudaGL.h                    CUDA OpenGL interop header for driver API
    cudaVDPAU.h                 CUDA VDPAU interop header for driver API
                                (Linux only)
    cuda_gl_interop.h           CUDA OpenGL interop header for toolkit API
                                (Linux only)
    cuda_vdpau_interop.h        CUDA VDPAU interop header for toolkit API
                                (Linux only)
    cudaD3D9.h                  CUDA DirectX 9 interop header (Windows only)
    cudaD3D10.h                 CUDA DirectX 10 interop header (Windows only)
    cudaD3D11.h                 CUDA Directx 11 interop header (Windows only)
    cufft.h                     CUFFT API header
    cublas_v2.h                 CUBLAS API header
    cublas.h                    CUBLAS Legacy API header
    cusparse_v2.h               CUSPARSE API header
    cusparse.h                  CUSPARSE Legacy API header
    curand.h                    CURAND API header
    curand_kernel.h             CURAND device API header
    thrust/*                    Thrust Headers
    npp.h                       NPP API Header
```

```
    nvcuvid.h                   CUDA Video Decoder header (Windows and Linux)
    cuviddec.h                  CUDA Video Decoder header (Windows and Linux)
    NVEncodeDataTypes.h         CUDA Video Encoder (C-library or DirectShow)
                                required for projects (Windows only)
    NVEncodeAPI.h               CUDA Video Encoder (C-library) required for
                                projects (Windows only)
    INvTranscodeFilterGUIDs.h   CUDA Video Encoder (DirectShow) required for
                                projects (Windows only)
    INVVESetting.h              CUDA Video Encoder (DirectShow) required for
                                projects (Windows only)

extras/
    CUPTI                       CUDA Profiling APIs
    Debugger                    CUDA Debugger APIs
```

5.3.1. Windows lib Files

```
lib/
    cuda.lib                    CUDA driver library
    cudart.lib                  CUDA runtime library
    cublas.lib                  CUDA BLAS library
    cufft.lib                   CUDA FFT library
    cusparse.lib                CUDA Sparse Matrix library
    curand.lib                  CUDA Random Number Generation library
    npp.lib                     NVIDIA Performance Primitives library
    nvcuvenc.lib                CUDA Video Encoder library
    nvcuvid.lib                 CUDA Video Decoder library
```

5.3.2. Linux lib Files

```
lib/
    libcuda.so                  CUDA driver library
    libcudart.so                CUDA runtime library
    libcublas.so                CUDA BLAS library
    libcufft.so                 CUDA FFT library
    libcusparse.so              CUDA Sparse Matrix library
    libcurand.so                CUDA Random Number Generation library
    libnpp.so                   NVIDIA Performance Primitives library
```

5.3.3. Mac OS X lib Files

```
lib/
    libcuda.dylib               CUDA driver library
    libcudart.dylib             CUDA runtime library
    libcublas.dylib             CUDA BLAS library
    libcufft.dylib              CUDA FFT library
    libcusparse.dylib           CUDA Sparse Matrix library
    libcurand.dylib             CUDA Random Number Generation library
    libnpp.dylib                NVIDIA Performance Primitives library
```

5.4. Supported NVIDIA Hardware

See http://www.nvidia.com/object/cuda_gpus.html.

5.5. Supported Operating Systems

5.5.1. Windows

- Supported Operating Systems (32-bit and 64-bit)

  WinServer 2008
  WinXP
  Vista/Win7

Table 13 Windows Compilers Supported in 4.1

Compiler	IDE
MSVC8 (14.00)	VS 2005
MSVC9 (15.00)	VS 2008
MSVC2010 (16.00)	VS 2010

5.5.2. Linux

The CUDA development environment relies on tight integration with the host development environment--including the host compiler and C runtime libraries, and is therefore only supported on *distro* versions that have been qualified for this CUDA Toolkit release.

Table 14 Linux Distributions Supported in 4.1

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 14	x	x	2.6.35.6-45	4.5.1	2.12.90
ICC Compiler 11.1	x	x			
OpenSUSE-11.2	x	x	2.6.31.5-0.1	4.4.1	2.10.1
RHEL-5.>=5 (5.5, 5.6, 5.7)	x	x	2.6.18-238.el5	4.1.2	2.5
RHEL-6.X (6.0, 6.1)		x	2.6.32-131.0.15.el6	4.4.5	2.12
SLES 11.1	x	x	2.6.32.12-0.7-pae	4.3-62.198	2.11.1-0.17.4
Ubuntu-10.04	x	x	2.6.35-23-generic	4.4.5	2.12.1
Ubuntu-11.04	x	x	2.6.38-8-generic	4.5.2	2.13

Table 15 Linux Distributions Not Supported in 4.1

Distribution	32	64	Kernel	GCC	GLIBC
Fedora 13	x	x	2.6.33.3-85	4.4.4	2.12
RHEL-4.8	x		2.6.9-89.ELsmpl	3.4.6	2.3.4

Distribution	32	64	Kernel	GCC	GLIBC
Ubuntu-10.10	x	x	2.6.35-23-generic	4.4.5	2.12.1

32-bit versions of RHEL 4.8 and RHEL 6.0 have not been tested with this release and are therefore not supported in this CUDA Toolkit release.

5.5.3. Mac OS X

Table 16 Mac OS X Platforms Supported in 4.1

Platform	32	64	Kernel	GCC
Mac OS X 10.7	x	x	10.0.0	4.2.1 (build 5646) XCode 4.1
Mac OS X 10.6	x	x	10.0.0	4.2.1 (build 5646)

5.6. Installation Notes

5.6.1. Windows

For silent installation:

- To install, use **msiexec.exe** from the shell, passing these arguments:

```
msiexec.exe /i cudatoolkit.msi /qn
```

- To uninstall, use **/x** instead of **/i**.

5.6.2. Linux

- In order to run CUDA applications, the CUDA module must be loaded and the entries in **/dev** created. This may be achieved by initializing X Windows, or by creating a script to load the kernel module and create the entries. An example script (to be run at boot time):

```
#!/bin/bash

/sbin/modprobe nvidia

if [ "$?" -eq 0 ]; then

    # Count the number of NVIDIA controllers found.
    N3D=`/sbin/lspci | grep -i NVIDIA | grep "3D controller" | wc -l`
    NVGA=`/sbin/lspci | grep -i NVIDIA | grep "VGA compatible controller" \
                      | wc -l`

    N=`expr $N3D + $NVGA - 1`
    for i in `seq 0 $N`; do
        mknod -m 666 /dev/nvidia$i c 195 $i;
    done

    mknod -m 666 /dev/nvidiactl c 195 255

else
```

```
    exit 1
fi
```

- On some Linux releases, due to a GRUB bug in the handling of upper memory and a default **vmalloc** too small on 32-bit systems, it may be necessary to pass this information to the bootloader:

```
vmalloc=256MB, uppermem=524288
```

Example of GRUB conf:

```
title Red Hat Desktop (2.6.9-42.ELsmp)
root (hd0,0)
uppermem 524288
kernel /vmlinuz-2.6.9-42.ELsmp ro root=LABEL=/1 rhgb quiet vmalloc=256MB
pci=nommconf
initrd /initrd-2.6.9-42.ELsmp.img
```

- CUDA Requirements for using Pinned Memory on Linux: Pinned memory in CUDA is only supported on Linux kernel version >= 2.6.18. Host side memory allocations pinned for CUDA using **cudaHostRegister()** API can be passed to 3rd party drivers. Pinned memory allocations returned from **cudaHostAlloc()** and **cudaMallocHost()** can also be passed to 3rd party drivers and starting with **4.1 CUDA_NIC_INTEROP** is no longer needed on these APIs, thus this flag is now deprecated.

5.7. Upgrading from Previous CUDA Toolkit 4.0

Please refer to the *CUDA_4.1_Readiness_Tech_Brief.pdf* document.

5.7.1. Vista, Server 2008 and Windows 7 Related

- Individual kernels are limited to a 2-second runtime by Windows Vista. Kernels that run for longer than 2 seconds will trigger the *Timeout Detection and Recovery* (TDR) mechanism. For more information, see http://www.microsoft.com/whdc/device/display/wddm_timeout.mspx.
- The maximum size of a single memory allocation created by **cudaMalloc** or **cuMemAlloc** on WDDM devices is limited to:

  MIN ((System Memory Size in MB - 512 MB) / 2, **PAGING_BUFFER_SEGMENT_SIZE**).

  For Vista, PAGING_BUFFER_SEGMENT_SIZE is approximately 2GB.
- (Windows and Linux): Individual GPU program launches are limited to a run time of less than 5 seconds on a GPU with a display attached. Exceeding this time limit usually causes a launch failure reported through the CUDA driver or the CUDA runtime. GPUs without a display attached are not subject to the 5 second runtime restriction. For this reason it is recommended that CUDA be run on a GPU that is NOT attached to a display and does not have the Windows desktop extended onto it. In this case, the system must contain at least one NVIDIA GPU that serves as the primary graphics adapter.
- The Linux kernel provides a mode where it allows user processes to overcommit system memory. (Refer to kernel documentation for **/proc/sys/vm/** for details). If this mode is enabled- the default on many distros- the kernel may have to kill processes in order to free up pages for allocation requests. The CUDA driver

process, especially for CUDA applications that allocate lots of zero-copy memory with **cuMemHostAlloc** or **cudaMallocHost**, is particularly vulnerable to being killed in this way. Since there is no way for the CUDA SW stack to report an OOM error to the user before the process disappears, users, especially on 32bit Linux, are encouraged to disable memory overcommit in their kernel to avoid this problem. Please refer to documentation on **vm.overcommit_memory** and **vm.overcommit_ratio** for more information.

5.7.2. Linux and Mac

When compiling with GCC, special care must be taken for structs that contain 64-bit integers. This is because GCC aligns long longs to a 4 byte boundary by default, while NVCC aligns long longs to an 8 byte boundary by default. Thus, when using GCC to compile a file that has a **struct/union**, users must give the **-malign-double** option to GCC. When using NVCC, this option is automatically passed to GCC.

5.7.3. Mac Related

To save power, some Apple products automatically power-down the CUDA-capable GPU in the system. If the operating system has powered down the CUDA-capable GPU, CUDA fails to run and the system returns an error that no device was found. In order to ensure that your CUDA-capable GPU is not powered down by the operating system do the following:

1. Go to **System Preferences**.
2. Open the **Energy Saver** section.
3. Un-check the **Automatic graphics switching** check box in the upper left.

5.8. CUDA Toolkit Known Issues

5.8.1. SDK Related

- The SDK sample- **boxFilter**, provided with the CUDA 4.1 SDK package for Linux and Mac may crash upon exit. The SDK sample incorrectly tries to device Memory using **free()**. The correct code should use **cudaFree()** instead for the device memory. This is a known issue and can be fixed. To fix the sample so that it does not crash upon exit, update **boxFilter.cpp**, lines 568-569 as follows: Replace:

  ```
  free(d_img);
  free(d_temp);
  ```

  With:

  ```
  cudaFree(d_img);
  cudaFree(d_temp);
  ```

- Please note that although the Linux and Mac SDK packages include **DirectCompute** documentation, the **DirectCompute** API is only supported

on Windows Vista and Windows 7, and will not work with Linux and Mac OS environments.

- String-based API functions (referencing static variables) are being deprecated in this release.
- **cudaHostUnregister** returns previous errors after kernel synchronization and **cudaGetLastError**
- The CUDA driver creates worker threads on all platforms, and this can cause issues at process cleanup in some multithreaded applications on all supported operating systems. On Linux, for example, if an application spawns multiple host pthreads, calls into **CUDART**, and then exits all user-spawned threads with **pthread_exit()**, the process may never terminate. Driver threads will not automatically exit once the user's threads have gone down.

  The proper solution is to either:

  1. call **cudaDeviceReset()** on all used devices before termination of host threads, or,
  2. trigger process termination directly (i.e, with **exit()**) rather than relying on the process to die after only user-spawned threads have been individually exited.
- Assertions in device code are not supported on OS X. If kernel code can call into assert on these platforms, all calls into runtime functions will fail with **cudaErrorOperatingSystem**, indicating that the device code cannot be loaded. Kernel code which references assert, but disables it at compile time with the **NDEBUG** define can still be loaded.
- Windows7-x64: Building project yields path not found errors for missing include and library files.

  Problem: Environment variables written by the installer may have mistakenly included an extra slash in the path specification.

  Solution: Remove the extra backslash at the end of the environment variable **CUDA_PATH**. Original value: **...\NVIDIA GPU Computing Toolkit\CUDA\v4.1**.

  New value: **...\NVIDIA GPU Computing Toolkit\CUDA\v4.1**.
- MAC 10.7: **cuda-gdb** is not supported on compute capability (SM type) 1.x on MAC OS 10.7
- The host linker on Mac OS 10.7 generates position-independent executables by default. As CUDA does not support position-independent executable currently, the linker must generate position-dependent executable by passing in the **-no_pie** option. If **nvcc** is being used to link the application, this option will be passed to the linker by default. To override the default behavior, the **-Xlinker -pie** option can be passed to **nvcc**.

5.8.2. Visual Profiler and Command Line Profiler

- Visual Profiler fails to generate events or counter information. There are several reasons due to which Visual Profiler may fail to gather counter information:

  1. If more than one tool is trying to access the GPU. To fix this issue please make sure only one tool is using the GPU at any given point. Tools include the CUDA command line profiler, *Parallel NSight* Analysis Tools and Graphics Tools, and

applications that use either **CUPTI** or PerfKit API (NVPM) to read counter values.

2. If more than one application is using the GPU at the same time when Visual Profiler is profiling a CUDA application. To fix this issue please close all applications and just run the one with Visual Profiler. Interacting with the active desktop should be avoided while the application is generating counter information. Please note that Visual Profiler gathers counters for only one context if the application is using multiple contexts within the same application.
3. On Windows platform if anytime the attach feature in Parallel NSight was enabled even on an older installation of Parallel NSight. To fix this issue:
    a. Please disable attach feature in Parallel NSight by right clicking on your Monitor tray icon then hit Properties, and go to the CUDA section, and disable **Use this Monitor for CUDA attach**.
    b. If disabling **Attach** in the Nsight Monitor does not fix the problem then you can go to the Windows **Advanced System Settings**, **Environment variables**, **System Variables** and delete **CUDA_INJECTION32_PATH** and/or **CUDA_INJECTION64_PATH** if these exist. The simplest way to get to the Windows **Advanced System Settings** is press **<windows+break>** buttons on your keyboard which takes you to the Windows **Control Panel** from where you can select **Advanced System Settings** in the left pane.

- Enabling **{gld|gst} instructions {8|16|32|64|128}bit** counters can cause GPU kernels to run longer than the driver's watchdog timeout limit. In these cases the driver will terminate the GPU kernel resulting in an application error and profiling data will not be available. Please disable driver watchdog timeout before profiling such long running CUDA kernels. On Linux, setting the **X Config** option **Interactive** to false is recommended. For Windows, detailed information on disabling the Windows TDR is available at: http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx#E2
- On Windows Vista/Win7, profiling an application which makes more than 32K CUDA kernel launch, memory copy, or memory set API calls without a synchronization call can result in an application hang. To work around this issue add synchronization calls like **cudaDeviceSynchronize()** or **cudaStreamSynchronize()**.
- Enabling counters on GPUs with compute capability (SM type) 1.x can result in occasional hangs. Please disable counters on such runs.
- On Windows Vista/Win7 systems occasional *Timeout Detection and Recovery* (TDR) can be hit when profiling with counters enabled. Please disable TDR before profiling such long running CUDA kernels. Detail information on disabling Windows TDR can be found at http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx#E2
- The *warp serialize* counter for GPUs with compute capability 1.x is known to give incorrect and high values for some cases.
- Prof triggers are not supported on GPUs with compute capability (SM type) 1.0.
- Profiler data gets flushed to a file only at synchronization calls like **cudaDeviceSynchronize()** and **cudaStreamSynchronize()** or when the profiler buffer gets full. If an app terminates without these sync calls then profiler

data may be lost. Similarly for OpenCL apps the OpenCL resources like the contexts, events should be freed before the app terminates.

- Counters **gld_incoherent** and **gst_incoherent** always return zero on GPUs with compute capability (SM type) 1.3. A value of zero doesn't mean that all load/stores are 100% coalesced.
- Use Visual Profiler version 4.1 onwards with driver version 285 (or later). Due to compatibility issues with profile counters, Visual Profiler 4.0 (or earlier) must not be used with driver version 285 (or later).

5.8.3. CUDA-MEMCHECK

- The **--device** option for cuda-memcheck in CUDA Toolkit v4.1 does not have any effect. This option is always silently ignored.
- **CUDA-MEMCHECK** may report an unknown error when running applications which call **assert()** in the CUDA kernel.

5.9. New Features in CUDA Release 4.1

- Cross process *P2P* is now supported.
- Added the ability to use **assert()** within kernels. This feature is supported only on the Fermi architecture.

5.9.1. CUDA Runtime

The **cuIpc** functions are designed to allow efficient shared memory communication and synchronization between CUDA processes. **cuIpcGetEventHandle** and **cuIpcGetMemHandle** get an opaque handle that can be freely copied and passed between processes on the same machine. The accompanying **cuIpcOpenEventHandle** and **cuIpcOpenMemHandle** functions allow processes to map handles to resources created in other processes. Equivalent runtime API functions are available.

5.9.2. Compiler Related

- The **nvcc** compiler switch, **--fmad** (short name: **-fmad**), to control the contraction of floating-point multiplies and add/subtracts into floating-point multiply-add operations (**FMAD**, **FFMA**, or **DFMA**) has been added: **--fmad**=true and **--fmad**=false enables and disables the contraction respectively. This switch is supported only when the **--gpu-architecture** option is set with **compute_20**, **sm_20**, or higher. For other architecture classes, the contraction is always enabled. The **--use_fast_math** option implies **--fmad**=true, and enables the contraction.
- For target architecture **sm_2x**, a new compiler component **cicc** is used instead of **nvopencc**.
- PTX version 3.0 is used for target architectures **sm_2x**. PTX version 1.4 is used for target architectures **sm_1x**.
- nvcc **--cuda** compiles the **.cu** input files to output files with the **.cu.cpp.ii** (instead of **.cu.cpp**) file extension in this release. This change has been made in order to avoid triggering an implicit rule in GNU Make which deletes the **.cu** files.

Note also that nvcc **--keep** produces the **.cu.cpp.ii** as one of the intermediate files, instead of the **.cu.cpp** output.

The **nvcc** option **-Xopencc** is deprecated.

5.9.3. CUDA Libraries

- In CUDA Toolkit version 4.1, the *Thrust* library supports the version of transform_if that does not require a *stencil* range. This was missing in previous releases.
- In previous releases of the CUDA toolkit, the CUFFT library included compiled kernel PTX and compiled kernel binaries for compute capability 1.0, 1.3 and 2.0. Starting with this release, the compiled kernel PTX will only be shipped for the highest supported compute capability (i.e., 2.0 for this release). This results in a significant reduction of file size for the dynamically linked libraries for all platforms.

There is no change to the compiled kernel binaries.

- The CUFFT Library now supports the advanced data layout parameters **inembed**, **istride**, **idist**, **onembed**, **ostride** and **odist**, as accepted by the **cufftPlanMany()** API, for real-to-complex (R2C) and complex-to-real (C2R) transforms. The previous release only supported these parameters for complex-to-complex (C2C) transform. Please refer to the CUFFT documentation for more details.
- The CURAND library supports the MTGP32 pseudo-random number generator, which is a member of the *Mersenne Twister* family of generators.
- The CUSPARSE library now provides a routine (**csrsm**) to perform a triangular solve with multiple right-hand-sides. This routine will generally perform better than calling a single triangular solve multiple times, once for each right-hand-side.
- The sparse triangular solve (**csrsv_analysis** and **csrsv_solve** routines) can now accept a general sparse matrix and work only on its triangular part. In the previous release, the **csrsv** routines would only accept matrices where the *MatrixType* was set to TRIANGULAR. Now, it can accept matrices of type GENERAL, but only operate on the triangular portion indicated by the **FillMode** setting (UPPER or LOWER). In addition, the sparse triangular solve can now ignore the diagonal elements by assuming that they are unity. The diagonal elements must be always present in the matrix, but will be assumed to be unity when the user sets the **DiagType** field in the matrix descriptor to be UNIT. This is particularly useful when processing sparse matrices where the lower and upper triangular parts have been stored together in a single general matrix.
- The **cusparseXgtsv()** and **cusparseXgtsvStridedBatch()** routines have been added to the CUSPARSE library in order to support solving linear systems represented by tri-diagonal sparse matrices.
- The CUSPARSE library now supports a *Hybrid* matrix storage format based on the *ELL* and *COO* formats. This format usually provides a significant speedup for the sparse matrix-vector multiplication operation compared to the CSR matrix storage format. Since the format is implemented using an opaque datatype

(**cusparseHybMat_t**), users cannot directly view nor operate on matrices in this format. The **dense2hyb** and **csr2hyb** conversion functions are provided to convert an existing matrix into the Hybrid format. Matrix-vector multiplication can be performed on Hybrid matrices using the **hybmv** routine and a triangular solve can be performed using the **hybsv** routine.

- The CUSPARSE Library now supports a new API for certain routines that allows an application to more easily take advantage of parallelism using streams. In particular, the new API accepts and returns certain scalar parameters by reference to device or host memory instead of by value on the host. This allows these APIs to execute asynchronously without blocking the caller host thread.

  The new APIs are exposed in the header file **cusparse_v2.h**. The older forms of the APIs are still supported and are exposed in the header file **cusparse.h**.

  Existing applications that use the CUSPARSE library can be recompiled and linked against the legacy version of CUSPARSE without any changes to the existing application source code. Furthermore, the binary interface for these older routines are still available as entry points into the CUSPARSE **.so** and **.dll**.

  NVIDIA recommends that new applications use the new API and that existing applications that need maximum stream parallelism be converted to the new API. Refer to CUSPARSE Library documentation (*doc/CUSPARSE_Library.pdf*) which has been rewritten to focus on the new APIs. Some treatment of the older APIs is still included.
- The CUBLAS library now supports a *batched* matrix multiply routine, **cublas{S,D,C,Z}gemmBatched**, that multiplies two arrays of matrices and produces another array of matrices. This API will multiply all of the matrices in a single launch and can improve performance compared to multiplying each pair of matrices with a separate call to the GEMM routine, especially for smaller matrices.
- Added new *Graphcut* that supports regular 8-neighborhood graphs to enable higher fidelity computations (**nppiGraphcut8_32s8u**). In addition, the existing primitive that supports 4-neighborhood graphs (**nppiGraphcut_32s8u**) has been significantly optimized. This release also changes the way scratch-memory (device buffer) is passed to the GraphCut primitives. This change is not backwards compatible.
- In previous releases of the CUDA Toolkit, the NPP library included compiled kernel PTX and compiled kernel binaries for compute capability 1.0, 1.3 and 2.0. Starting with this release, the compiled kernel PTX will only be shipped for the highest supported compute capability (i.e., 2.0 for this release). This results in a significant reduction of file size for the dynamically linked libraries for all platforms.

  

  There is no change to the compiled kernel binaries.
- Almost 1,000 new image processing primitives have been added to the NPP library (in **nppi.h**) for arithmetic and logical operations. As of this release, the NPP library has broad coverage for these types of image operations on formats that have 1 component, 2 components with alpha, 3 components, 4 components, and 4 components with alpha, where the component sizes are 8-, 16-, and 32-bit integer or 32-bit floating point.

- The CURAND library now supports *L'Ecuyer's MRG32k3* a pseudo-random number generator.
- The CURAND library in the previous releases would dynamically allocate memory for internal usage within the **curandCreateGenerator()** API when it would create an *XORWOW* generator, and it would deallocate the memory for that generator within the **curandDestroyGenerator()** API. Starting with this release, the memory is allocated and deallocated dynamically each time the **curandGenerateSeeds()** API is called on an XORWOW generator, so that the dynamically allocated memory is not tied up for the entire life of an XORWOW generator.
- The CUDA math library now supports Bessel functions of the first and second kinds of orders 0, 1, and n, both in single and double precision. These can be accessed via the j0f, j1f, jnf, y0f, y1f, and ynf functions in single precision and j0, j1, jn, y0, y1, and yn functions in double precision. Please refer to *Appendix C* in the *CUDA C Programming Guide* and the relevant entries in the *CUDA Toolkit Reference Manual* (Cuda_Toolkit_Reference_Manual.pdf) for more information.
- The scaled complementary error function has been added to **math.h**. This is equivalent to **exp(x*x)*erfc(x)**. The double-precision routine is exposed as **erfcx()** and the single-precision routine as **erfcxf()**.
- New functions for halving addition and rounded halving addition for 32-bit signed and unsigned integers have been added to the math header files. These new functions perform the addition and halving without overflow in the intermediate sum. They are available as **__{u}{r}hadd()**. Please refer to the *CUDA C Programming Guide* for more details.

5.9.4. CUDA Driver

- For 2D texture references bound to pitched memory, the pitch has to be aligned to the HW specific texture pitch alignment attribute. This value can be queried using the device attribute:
  - **CU_DEVICE_ATTRIBUTE_TEXTURE_PITCH_ALIGNMENT** in the driver API
  - **cudaDeviceProp::texturePitchAlignment** in the runtime API.

  If a misaligned pitch is specified the following error will be returned:
  - **CUDA_ERROR_INVALID_VALUE** in the driver API
  - **cudaErrorInvalidValue** in the runtime API.
- In the CUDA Driver, **cuMemHostRegister** and **cudaHostRegister** now accept memory ranges with arbitrary size and alignment; **cuMemHostRegister** and **cudaHostRegister** are still restricted to non-overlapping memory ranges.
- Cubemaps can be created by specifying the flag **cudaArrayCubemap** during CUDA array creation. Cubemap Layered CUDA arrays can be created by specifying two flags - **cudaArrayCubemap** and **cudaArrayLayered**. New intrinsics have been added to perform texture fetches. e.g. calling **texCubemap(texRef, x, y, z)** fetches from a cubemap texture.
- For changes related to *NVSMI* and *NVML*, please refer to nvidia-smi man page and the *Tesla Deployment Kit* package (found on the developer site) which includes NVML documentation and the SDK.

- CUDA-OpenGL interop API now allows querying the device on which OpenGL is running. If SLI is enabled, the application can query the current rendering device on a per-frame basis. For more information, refer to the *CUDA API Reference Manual* and the *CUDA C Programming Guide*.
- 1D Layered, 2D Layered and 3D surfaces can now be bound to surface references. New intrinsics have been added to perform loads/stores to such surfaces . For example, **`surf3Dread(data, surfref, x, y, z)`** reads from a location (x, y, z) of a 3D surface.
- Texture gather operations can now be performed on 2D CUDA arrays by specifying a flag **`cudaArrayTextureGather`** during CUDA array creation. Texture gather allows obtaining the *bilerp* footprint of a regular texture fetch. New intrinsics of the form **`tex2Dgather(texref, x, y, comp)`** have been added, where *comp* can be one of {0,1,2,3} to indicate the component to be fetched.

5.10. Performance Improvements in CUDA Release 4.1

- Various performance improvements have been made to the device reduction and host sorting algorithms in the Thrust library. A new CUDA **`reduce_by_key`** implementation provides up to 3x faster performance. A faster host sort provides up to 10x faster performance for sorting arithmetic types on (single-threaded) CPUs. A new *OpenMP* sort provides up to 3x speedup over the single-threaded host sort using a quad-core CPU. When sorting arithmetic types with the OpenMP backend the combined performance improvement is ~6x for 32-bit integers and ranges from 3x (64-bit types) to more than 10x (8-bit types).
- The performance of double precision floating-point square-root has been significantly optimized for the Tesla and Fermi architectures for the default rounding mode (IEEE round-to-nearest), accessible via the **`sqrt()`** math function or the **`__dsqrt_rn()`** intrinsic.
- The double-precision **`cosh()`** math library routine has been optimized for both the Tesla and Fermi architectures.
- Single-precision floating-point reciprocal has been optimized significantly for the Fermi architecture for all four IEEE rounding modes. This improvement applies to the *1/x* operator in C, when compiled with the compiler defaults, or when **`-prec-div=true`** is explicitly specified on the **`nvcc`** command-line. In addition, this improvement applies to the **`__frcp_{rn,rz,ru,rd}() intrinsics`**.
- Single-precision square-root has been optimized significantly for the Fermi architecture for all four IEEE rounding modes. This improvement applies to the **`sqrtf()`** math function when compiled with the compiler defaults, or when **`-prec-sqrt=true`** is explicitly specified on the nvcc command-line. In addition, this improvement applies to the **`__fsqrt_{rn,rz,ru,rd}()`** intrinsics.
- IEEE-754 compliant single-precision floating-point division for the default rounding mode (round-to-nearest-or-even) has been accelerated significantly for the Fermi architecture. This operation is generated for the single-precision division operator / when building with the compiler defaults, or when **`-prec-div=true`** is explicitly

specified on the nvcc command line. In addition it is accessible via the **__fdiv_rn()** intrinsic."

- The **erfcf()** function has been optimized for the Fermi architecture. With the compiler defaults for Fermi (**-prec-div=true** and **-no-ftz=true**), the function executes at twice the speed of the previous implementation, although exact observed performance improvement will depend on the specific application code that calls **erfcf()**.
- The accuracy of the double-precision **erfinv()** math library routine has been improved from a worst-case error bounds of 8 ULPs (units in the last place) over the full range of inputs to only 5 ULPs.
- The **cublasXgemv()** routines in the CUBLAS library have been optimized, specifically for non-square matrices when the number of columns is much greater than the number of rows.

5.11. Resolved Issues

- In the NPP library, the two quantization-table initialization functions used for JPEG compression, **nppiQuantFwdTableInit_JPEG_8u16u()** and **nppiQuantInvTableInit_JPEG_8u16u()**, expect an input quantization table in a zigzaged format as described in the JPEG standard. However, now the resulting tables are de-zigzaged; this was not true in previous versions. The de-zigzaged result tables are in the proper format for use with the **nppiDCTQuantFwd8x8LS_JPEG_8u16s_C1R()** or **nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R()** routines. User programs should not see any functional difference if they never inspect the output of **nppiQuantFwdTableInit_JPEG_8u16u()** or **nppiQuantInvTableInit_JPEG_8u16u()**, and simply pass the output to the DCT functions listed earlier.
- In previous versions of the NPP Library, the Rotate primitives set pixel values inside the destination ROI to 0 (black) if there is no pixel value from the source image that corresponds to a particular destination pixel. This incorrect behavior has been fixed. Now, these destination pixels are left untouched so that they stay at the original background color.
- In previous releases of the NPP Library, the Signal primitives in the Arithmetic, Logical and Shift, and Vector Initialization families would fail for signals beyond a certain size. In this release, these primitives should be function correctly for signals of any size - assuming of course that the input and output signals have been successfully allocated within the available GPU memory.
- In the previous release, the NPP Color Conversion primitives did not work properly for line strides that were not 64 byte aligned. In particular the P3R, P3P2R, P3C3 variants of those primitives were affected. This issue is now fixed.
- In the previous release of the NPP library, the **nppiMinMax_8u_C4R** function would erroneously provide copies of the result from the first channel in the 2nd, 3rd, and 4th channels. So the result would be **{min(channel1), min(channel1), min(channel1), min(channel1)}** and not **{min(channel1), min(channel2), min(channel3), min(channel4)}**, and similar for the maximums. This bug has been fixed in this release of the NPP library.

- This production release of the CUDA 4.1 Toolkit has been upgraded to include v1.5.1 of Thrust, which includes several bugfixes identified during earlier CUDA Toolkit v4.1 release candidates. Please see the *Thrust CHANGELOG* for a complete list.
- The Thrust library is now thread-safe, and hence the various Thrust APIs can all be called safely from multiple concurrent host threads.
- The **device_ptr<void>** datatype in Thrust now requires an explicit case to convert to **device_ptr<T>**, where **T != void**. Use the expression **device_pointer_cast(static_cast<int*>(void_ptr.get()))** to convert, for example, **device_ptr<void>** to **device_ptr<int>**. Existing code that used to unsafely convert without an explicit case will no longer compile.
- The previous version of the **cublasXnrm2()** routines in the CUBLAS library could produce *NaNs* in the output incorrectly in some cases when the input contained at least one denormal value. This has been fixed in the current release.
- For certain cases related to the CUSPARSE library, in the previous version of the CUDA Toolkit (v4.0), **cusparse{S,D,C,Z}csrmv** could return an erroneous result due to a race condition if at least one of following conditions is verified:
  - *Trans* parameter is NOT set to **CUSPARSE_OPERATION_NON_TRANSPOSE** and the sparse matrix A had an average number of non-zeros per row above 32,
  - matrix A type is set to **CUSPARSE_MATRIX_TYPE_SYMMETRIC** or **CUSPARSE_MATRIX_TYPE_HERMITIAN** This issue is now fixed in this version (v4.1) of CUSPARSE.
- Useful error codes added:
  - **CUDA_ERROR_HOST_MEMORY_ALREADY_REGISTERED** (**cudaErrorHostMemoryAlreadyRegistered**) will be returned when user calls **cuMemHostRegister** (**cudaHostRegister**) on memory registered by a previous call to **cuMemHostRegister** (**cudaHostRegister**).
  - **CUDA_ERROR_HOST_MEMORY_NOT_REGISTERED** (**cudaErrorHostMemoryNotRegistered**) will be returned when user calls **cuMemHostUnregister** (**cudaHostUnregister**) on memory not registered by any previous call to **cuMemHostRegister** (**cudaHostRegister**).
- In the earlier CUDA Toolkit version 4.1 *release candidates* (RC), the function **curandSetGeneratorOffset()** had no impact on the generated results for the **CURAND_RNG_PSEUDO_MRG32K3A** generator. This issue is fixed in this production release of CUDA Toolkit version 4.1.
- In previous releases, the **curand_precalc.h** header file described a large array in a single line with no newlines, which can cause problems with some source control systems. In this release, newlines have been added periodically throughout the file.
- In previous releases **cuMemsetD2D16/32** failed in some corner cases. This has been fixed in this release.
- In the previous version (v4.0) of the CUBLAS library, the routine **cublas_Xgemv()** with the *trans* parameter NOT set to **CUBLAS_OP_N**, returned incorrect numeric results for the output vector *y*, if the number of columns of the input matrix A exceeded 2097120 for **cublas_Sgemv()** or 1048560 for the other datatypes. The issue is now resolved in this version (v4.1) of CUBLAS.

- The CUBLAS library in v4.0 of the CUDA Toolkit had added support for a new API. The older API was still supported via a header file, but the entry points were removed from the CUBLAS **.so** and **.dll**. While existing source code written in C/C++ was still backwards compatible after a simple recompile, compatibility was broken for projects that were directly using the entry points (i.e., the binary interface) of the **.so** and **.dll**. In this release, the old entry points have been added back into the **.so** and **.dll** to provide better compatibility for such projects. Now the **.so** and **.dll** contain entry points for both the new and old APIs.
- In certain cases, the **thrust::adjacent_difference()** operation in the previous release would produce incorrect results when operating in-place. This has been fixed in the Thrust library in the current release.
- Previous releases of the CUFFT library were not thread-safe, and hence could not be accessed concurrently from multiple threads in the same process. This has been fixed in the current release. Once created, any plan can be accessed safely from any thread in the same process until the plan is destroyed.
- In previous releases of the CUFFT Library, certain configurations would produce slightly different results for the same input when ECC is on versus when ECC is off (though both were within the expected tolerance compared to the infinite precision mathematically correct reference). In this release, the results are now identical for the same configuration whether ECC is on or off.
- A possible bug associated with **cuFFT** occurred if GTX480 and GT240 are both present in system. This is no longer the case.
- The host linker on Mac OS X generates position-independent executables by default, unless the target platform is Mac OS X 10.6 or earlier. Since *cuda-gdb* does not support position-independent executables, **nvcc** passes **-no_pie** to the host linker and generates position-dependent executables. With this release, users can force nvcc to produce position-independent executables by specifying **-Xlinker -pie** as an **nvcc** option.

5.12. Source Code for Open64 and CUDA-GDB

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

Linux users:

- Please refer to the *Release Notes* and *Known Issues* sections in the *CUDA-GDB User Manual* (**CUDA_GDB.pdf**).
- Please refer to **CUDA_Memcheck.pdf** for notes on supported error detection and known issues.

5.13. More Information

For more information and help with CUDA, please visit http://www.nvidia.com/cuda.

Please refer to the *LLVM Release License* text in **EULA.txt** for details on LLVM licensing.

5.14. Acknowledgements

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 **`erfinv()`** function found in this release of the CUDA toolkit.

Chapter 6. NVIDIA CUDA TOOLKIT V4.0 RELEASE NOTES

6.1. Release Highlights

This release contains:

- NVIDIA CUDA Toolkit documentation
- NVIDIA OpenCL documentation
- NVIDIA CUDA compiler (nvcc) and supporting tools
- NVIDIA CUDA runtime libraries
- NVIDIA CUBLAS, CUFFT, CUSPARSE, CURAND, Thrust, and NPP libraries

NVIDA CUDA Toolkit version 4.0 has the following new features:

- NVIDIA cuda-gdb debugger
- NVIDIA Visual Profiler for CUDA C/C++ and OpenCL applications
- Easier application porting
  - Share GPUs across multiple threads
  - Single thread access to GPUs
  - No-copy pinning of system memory
  - New CUDA C/C++ language features
  - Thrust templated primitives library
  - NPP image/video processing library
  - Layered Textures
- Faster multi-GPU programming
  - Unified virtual addressing
  - GPUDirect v2.0 with peer-to-peer communication
- New and improved developer tools
  - Automated performance analysis
  - C++ debugging

- Debugger cuda-gdb for Mac OS
- GPU binary disassembler

6.2. Documentation

For a list of documents supplied with this release, please refer to the */doc* directory of your CUDA Toolkit installation.

For issues related to the Visual Profiler, please refer to the *Visual Profiler release notes* for the specific platform. Refer the *Visual Profiler change log* - "Changelog.txt" - for changes in Visual Profiler with respect to the previous version.

6.3. Errata for Windows, Linux, and Mac OS X

6.3.1. Linux

CUDA Requirements for Using Pinned Memory on Linux:

Pinned memory in CUDA is only supported on Linux kernel version >= 2.6.18. Host side memory allocations pinned for CUDA using **`cudaHostRegister()`** API can be passed to 3rd party drivers. Pinned memory allocations returned from **`cudaHostAlloc()`** and **`cudaMallocHost()`** can also be passed to 3rd party drivers and starting with 4.1 **`CUDA_NIC_INTEROP`** is no longer needed on these APIs, thus this flag is now deprecated.

6.3.2. Resolved Issues

Previous version of the *Errata* reported that for applications using multiple streams CUDA Visual Profiler can drop profiler data rows and that the following error is reported: **In this profiling session some profiler output rows are dropped due to incorrect gpu time stamp values and the profiler output is incomplete.**

This issue has been fixed with a patch for Linux toolkits. You can download the patches from the main download page: http://developer.nvidia.com/cuda-toolkit-40 Each patch is associated with its appropriate Linux package; the description section in the Downloads column specifies **Visual Profiler Patch** in parentheses

6.3.3. Known Issues

- Visual Profiler incorrectly treats kernels with names that start with **`memcpy`** as being memory copies. As a result, profiling data reported for these kernels is incorrect. To workaround this issue the kernel name should be changed so that it does not start with **`memcpy`**.
- A 64-bit application, with the OS configured as 32-bit kernel running on driver versions prior to the CUDA 4.0.31, may crash.

  Follow these steps to determine your default OS kernel configuration:

  1. Choose **About This Mac** from the Apple menu.

2. Click on **More Info**.
3. Select **Software in the Contents** pane.
4. Look for **64-bit Kernel and Extensions: Yes (or No)** under the **System Software Overview** heading.

With the CUDA driver 4.0.31 driver for Mac, a CUDA context cannot be created in this mode- 32-bit kernel, 64-bit CUDA application. If a 64-bit CUDA application tries to create a CUDA context in this mode, **`cuInit()`** will return a CUDA error.

The CUDA driver 4.0.31 on Mac OSX 10.7 supports the following configurations:

32-bit Kernel running with 32-bit CUDA application.
64-bit Kernel running with 64-bit CUDA application.

Support for 32-bit OS kernel with 64-bit CUDA applications will require a future CUDA driver update in conjunction with a *Lion Software Update*.

If your system is running as a 32-bit kernel, and you want to run a 64-bit CUDA application, one option is to set your OS to run in 64-bit kernel mode. This requires the Apple system hardware to support the OS running in 64-bit kernel; please refer to the *Apple website* for a detailed list of supported hardware.

You can enable your OS to run in 64-bit kernel mode using one of the following ways:

- At startup time, if 32-bit kernel is your default configuration, holding *6* and *4* keys during startup will boot into 64-bit kernel mode.
- To change the default configuration for the current startup disk (persistent),

  - to 64-bit kernel, open a **Terminal Window** with the command: `sudo systemsetup -setkernelbootarchitecture x86_64`

  - to 32-bit kernel, open a **Terminal Window** with the command: `sudo systemsetup -setkernelbootarchitecture i386`

Any OSX using XCODE4.0 or higher will be supported starting with CUDA 4.1 due for release late this year. Any pre-built CUDA applications will work with the released CUDA driver for 10.7 but there is no tool chain support to create new CUDA applications on 10.7 or XCODE version 4.0 or higher until CUDA 4.1.

The CUDA 4.0 SDK code samples for Windows platforms have been updated from version 4.0.17 to 4.0.19 to address the following issues:

1. Problems with building *DEBUG* targets using Visual Studio 2010.

   Specifically, the Visual Studio 2010 **`cutil`** project solution file did not build correctly when a DEBUG configuration was chosen. The **`.sln/.vcxproj`** solution and project files have been updated to resolve this.
2. The CUDA 4.0 SDK projects build using the last installed CUDA Toolkit instead of the latest one.

   In some cases, where a developer had both CUDA 3.2 or 4.0 Toolkit installed, Visual Studio 2010 SDK projects would choose the last installed toolkit, instead

of the newest one. CUDA project files previously specified the include paths to be **$(CUDA_PATH)\include**. To address this, SDK sample projects now specify either **$(CudaToolkitIncludeDir)** or **$(CudaToolkitDir)\include**.

3. Individual SDK solutions from VS2005, VS2008, VS2010 do not build properly.

   Each SDK sample solution may depend on **cutil**, **shrUtils**, or **oclUtils** libraries which are also part of the SDK. In order to build with the proper dependencies, developers needed to open the **release_vs200?.sln** solution file for all dependencies to work. The individual SDK sample solutions for CUDA, CUDALibraries, and OpenCL now include dependencies from individual solution files.

- In some cases, Visual Profiler global memory derived statistics and hints may be incorrect. If the kernel has local memory accesses, the derived statistics- **global memory excess load %** and **global memory excess store %** can yield incorrect results. This is because the L2 throughput that is used to calculate these values include local memory accesses too. As a result, the hints which use these statistics are incorrect as well since the excess loads given by this formula are caused due to the local memory accesses (in addition to possibly uncoalesced memory access pattern)
- In a multi-gpu setup, when compute mode is set to **compute prohibited** for some GPUs, the Visual Profiler cannot profile a CUDA runtime application; Visual Profiler reports an error and profiling data is not shown.
- **CudaHostRegister()** is not supported in RHEL4. Please refer to the *NVIDIA CUDA C Programming Guide* for details on **CudaHostRegister()**

6.3.4. More Information

For more information and help with CUDA, please visit http://www.nvidia.com/cuda.

6.4. List of Important Files

```
bin/nvcc                      Command line compiler

include/
    cuda.h                      CUDA driver API header
    cudaGL.h                    CUDA OpenGL interop header for driver API
    cudaVDPAU.h                 CUDA VDPAU interop header for driver API
                                (Linux only)
    cuda_gl_interop.h           CUDA OpenGL interop header for toolkit API
                                (Linux only)
    cuda_vdpau_interop.h        CUDA VDPAU interop header for toolkit API
                                (Linux only)
    cudaD3D9.h                  CUDA DirectX 9 interop header (Windows only)
    cudaD3D10.h                 CUDA DirectX 10 interop header (Windows only)
    cudaD3D11.h                 CUDA Directx 11 interop header (Windows only)
    cufft.h                     CUFFT API header
    cublas.h                    CUBLAS API header
    cusparse.h                  CUSPARSE API header
    curand.h                    CURAND API header
    curand_kernel.h             CURAND device API header
    thrust/*                    Thrust Headers
    npp.h                       NPP API Header
    nvcuvid.h                   CUDA Video Decoder header (Windows and Linux)
    cuviddec.h                  CUDA Video Decoder header (Windows and Linux)
```

```
    NVEncodeDataTypes.h         CUDA Video Encoder (C-library or DirectShow)
                                required for
                                projects (Windows only)
    NVEncodeAPI.h               CUDA Video Encoder (C-library) required for
                                projects
                                (Windows only)
    INvTranscodeFilterGUIDs.h   CUDA Video Encoder (DirectShow) required for
                                projects
                                (Windows only)
    INVVESetting.h              CUDA Video Encoder (DirectShow) required for
                                projects (Windows only)
```

6.4.1. Windows lib Files

```
lib/
    cuda.lib                    CUDA driver library
    cudart.lib                  CUDA runtime library
    cublas.lib                  CUDA BLAS library
    cufft.lib                   CUDA FFT library
    cusparse.lib                CUDA Sparse Matrix library
    curand.lib                  CUDA Random Number Generation library
    npp.lib                     NVIDIA Performance Primitives library
    nvcuvenc.lib                CUDA Video Encoder library
    nvcuvid.lib                 CUDA Video Decoder library
```

6.4.2. Linux lib Files

```
lib/
    libcuda.so                  CUDA driver library
    libcudart.so                CUDA runtime library
    libcublas.so                CUDA BLAS library
    libcufft.so                 CUDA FFT library
    libcusparse.so              CUDA Sparse Matrix library
    libcurand.so                CUDA Random Number Generation library
    libnpp.so                   NVIDIA Performance Primitives library
```

6.4.3. Mac OS X lib Files

```
lib/
    libcuda.dylib               CUDA driver library
    libcudart.dylib             CUDA runtime library
    libcublas.dylib             CUDA BLAS library
    libcufft.dylib              CUDA FFT library
    libcusparse.dylib           CUDA Sparse Matrix library
    libcurand.dylib             CUDA Random Number Generation library
    libnpp.dylib                NVIDIA Performance Primitives library
```

6.5. Supported NVIDIA Hardware

See http://www.nvidia.com/object/cuda_gpus.html.

6.6. Supported Operating Systems for Windows, Linux, and Mac OS X

6.6.1. Windows

- Supported Operating Systems (32-bit and 64-bit)
  - Windows 7
  - Windows Vista
  - Windows XP
  - Windows Server 2008 R2
  - Windows Server 2008
  - Windows Server 2003

Table 17 Windows Compilers Supported in 4.0

Compiler	IDE
MSVC8 (14.00)	VS 2005
MSVC9 (15.00)	VS 2008
MSVC2010 (16.00)	VS 2010

6.6.2. Linux

The CUDA development environment relies on tight integration with the host development environment--including the host compiler and C runtime libraries, and is therefore only supported on distro versions that have been qualified for this CUDA Toolkit release. For example, since the CUDA Toolkit 4.0 was not tested with any Linux distros that use the GNU C Compiler (GCC) version 4.5, it is not supported on those distros.

Table 18 Linux Distributions Supported in 4.0

Distribution	32	64	Kernel	GCC	GLIBC
SLES11-SP1	X	X	2.6.32.12-0.7-pae	4.3-62.198	2.11.1-0.17.4
RHEL-6.0		X	2.6.32-71.el6	4.4.4	2.12
Ubuntu-10.10	X	X	2.6.35-23-generic	4.4.5	2.12.1
OpenSUSE-11.2	X	X	2.6.31.5-0.1	4.4.1	2.10.1
Fedora13	X	X	2.6.33.3-85	4.4.4	2.12
RHEL-4.8		X	2.6.9-89.ELsmpl	3.4.6	2.3.4
RHEL-5.5	X	X	2.6.18-194.el5	4.1.2	2.5

Table 19 Linux Distributions Not Supported in 4.0

Distribution	32	64	Kernel	GCC	GLIBC
RHEL-4.8	X		2.6.9-89.ELsmpl	3.4.6	2.3.4
Ubuntu-10.04	X	X	2.6.32-21-generic	4.4.3	2.11.1
SLED11-SP1	X	X	2.6.32.12.0.7	4.3.4	2.11.1

32-bit versions of RHEL 4.8 and RHEL 6.0 have not been tested with this release and are therefore not supported in the CUDA Toolkit 4.0 release.

6.6.3. Mac OS X

Table 20 Mac OS X Platforms Supported in 4.0

Platform	32	64	Kernel	GCC	Status
Mac OS X 10.6	X	X	10.0.0	4.2.1 (build 5646)	Continued
Mac OS X 10.5.2+	X	X			Removed
Mac OS X 10.5.7	X	X	9.7.0	4.0.1 (build 5490)	Removed

6.7. Installation Notes

6.7.1. Windows

Silent Installation:
Install using **`msiexec.exe`** from the shell and pass the following arguments:

```
msiexec.exe /i cudatoolkit.msi /qn
```

To uninstall:
Use **`/x`** instead of **`/i`**

6.7.2. Linux

On some Linux releases, due to a GRUB bug in the handling of upper memory and a default **`vmalloc`** too small on 32-bit systems, it may be necessary to pass this information to the bootloader:

```
vmalloc=256MB, uppermem=524288
```

Example of grub conf:

```
title Red Hat Desktop (2.6.9-42.ELsmp)
root (hd0,0)
uppermem 524288
kernel /vmlinuz-2.6.9-42.ELsmp ro root=LABEL=/1 rhgb quiet vmalloc=256MB
```

```
pci=nommconf
initrd /initrd-2.6.9-42.ELsmp.img
```

6.8. Upgrading from Previous CUDA Toolkit 3.2

Please refer to the *CUDA_4.0_Readiness_Tech_Brief.pdf* document.

Mac-related Note

CUDA 4.0 does not have support for XCODE4.0.

6.9. Notes on New Features and Performance Improvements

6.9.1. CUDA Driver Features

- **cudaMemcpyAsync** works with non pinned heap memory. The asynchronous copy APIs (**cudaMemcpyAsync** et al in the runtime API and **cuMemcpyHtoDAsync** et al in the driver API) may take ordinary pageable host memory as its source or destination argument.

  This is in contrast to CUDA 3.2 where host memory could only be used if it was allocated through CUDA (using **cudaMallocHost** et al through the runtime API or **cuMemAllocHost** through the driver API).

  

  While using pageable host memory is now permitted for use with the asynchronous copy APIs, using pageable host memory will result in the copies being performed synchronously.

- **cudaMemcpy** is supported across contexts. The ability to copy memory between devices in the runtime API (and between context in the driver API) has been added.

  When using unified addressing, the function **cudaMemcpy** (and its variants) with the copy direction **cudaMemcpyDefault** may be used to copy between devices in the runtime API (the function **cuMemcpy** may be used in the driver API). When not using unified addressing, the function **cudaMemcpyPeer** in the runtime API (and **cuMemcpyPeer** in the driver API) and its variants may be used to copy between devices.

  This functionality is supported on all platforms and all devices. This functionality will take advantage of direct peer access where it is enabled.

  

  This functionality may not be optimal on compute level 1.0 devices and across non-SLI-linked devices using the WDDM driver model on Vista and Win7.

- **cudaStreamWaitEvent** supported across contexts. The function **cudaStreamWaitEvent** (or **cuStreamWaitEvent** in the driver API) may be used

to effect cross-device (or cross-context, in the driver API) synchronization. An event recorded on one device may be waited on by a stream created by another device.

The dependency added will be resolved asynchronously, and this will be very efficient.

This may not be optimally efficient yet for compute 1.0 devices or for devices that are not in SLI on Windows Vista/7, using the WDDM driver model.

- Added flag for property *Concurrent Data Transfer* to indicate two simultaneous DMA transfers.

  The ability of the device to concurrently pull data (from host or a peer device) and push data (to host or a peer) may be queried.

  In the runtime API, this may be done by examining the device property **`asyncEngineCount`** will be set to *1* if only one direction of a transfer may be active at a time and *2* if both directions may be active at a time.

  The driver API device property query is **`CU_DEVICE_ATTRIBUTE_ASYNC_ENGINE_COUNT`**.

- (Windows and Linux) Added support for unified virtual address space.

  Devices supporting 64-bit and compute 2.0 ahd higher capability now share a single unified address space between the host and all devices. This means that the pointer used to access memory on the host is the same as the pointer to used to access memory on the device. Therefore, the location of memory may be queried directly from its pointer value; the direction of a memory copy need not be specified.

  The function **`cudaPointerGetAttribute`** in the runtime API (and **`cuPointerGetAttribute`** in the driver API) may be used to query attributes about a pointer. The copy direction **`cudaMemcpyDefault`** in the runtime API (and the functions **`cuMemcpy`**, its variants, and the memory type **`CU_MEMORYTYPE_UNIFIED`** in the driver API) may be used to copy data without specifying the direction.

This functionality is available only on Linux-64, Windows XP-64, and Windows Vista/7 using the TCC driver model.

- The ability of directly accessing memory on peer devices has been added.

  If direct access of memory on the peer device is possible (which can be queried by runtime API **`cudaDeviceCanAccessPeer`** or driver API **`cuDeviceCanAccessPeer`**), this functionality can be enabled by **`cudaDeviceEnablePeerAccess`** (or **`cuCtxEnablePeerAccess`**).

  This functionality is supported on all NVIDIA CUDA devices with compute level 2.0 and up running 64-bit Linux, XP, and TCC drivers.

Peer access is not supported on WDDM.

- (Linux) DX and OGL textures (shared through interop), mapped as CUDA arrays, can now be bound to surface references in CUDA. In order to be able to do so, the DX/OGL resource should be registered with the appropriate register flag as follows:

  For the driver api, it's **CU_GRAPHICS_REGISTER_FLAGS_SURFACE_LDST**.

  For the runtime api, it's **cudaGraphicsRegisterFlagsSurfaceLoadStore**.

  

  Surface has smaller width/height restrictions than textures. If the texture is registered with the surface load/store flag, and the size is too big, then that will generate an error.

- Removed alignment requirments from **cuMemcpy*** functions.

  All restrictions on the alignment of the source and destination pointer and pitch for all 2D and 3D copies (using **cudaMemcpy3D** et al in the runtime API and **cuMemcpy3D** et al in the driver API) have been removed.

  

  Using unaligned operands for a copy may result in poorer performance than using aligned operands.

- Added 64-bit support to WinXP-64
- (Windows and Linux) *CUDA-OpenGL interop* currently supports the following set of texture formats:

  {**GL_R**, **GL_RG**, **GL_RGBA**, **GL_LUMINANCE**, **GL_LUMINANCE_ALPHA**, **GL_ALPHA**, **GL_INTENSITY**} X {,8,16,16F,32F,8UI,16UI,32UI,8I,16I,32I}

  These formats are also supported for OpenCL-OpenGL interop.

  For further details on these texture formats, please refer to the OpenGL specification.

- Event and stream creation/destruction improved in this version.

  The functions **cudaStreamDestroy** and **cudaEventDestroy** (**cuStreamDestroy** and **cuEventDestroy**) are now asynchronous and light-weight. Destroying a stream or event will return immediately, even if there is still pending work in the stream or pending work behind the event. The stream or event's resources will be released asynchronous once the stream or event has completed its work.

- Added device attributes for memory clock and number of threads per *SM*.

  The following new device attributes are supported in the CUDA driver API:

  CU_DEVICE_ATTRIBUTE_MEMORY_CLOCK_RATE: gives the peak memory clock frequency in kilohertz.

  CU_DEVICE_ATTRIBUTE_GLOBAL_MEMORY_BUS_WIDTH: gives the global memory bus width in bits.

  CU_DEVICE_ATTRIBUTE_L2_CACHE_SIZE: gives the size of the L2 cache in bytes.

  CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_MULTIPROCESSOR: gives the number of maximum threads that can be resident at one time on a multiprocessor.

- (Windows) This version allows a single CUcontext to be current to multiple threads simultaneously.
- A kernel that is compiled with a **`__launch_bounds__`** directive will have the max threads/block taken into account when querying the max thread count via **`cuFuncGetAttribute(&i, CU_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK, f)`**. Also **`cuFuncSetBlockShape(f, x, y, z)`** will reject block shapes that exceed the max threads/block set via a **`__launch_bounds__`**. These changes in behavior will be likewise be visible in the *CUDART* counterparts to these CUDA APIs.
- Querying the maximum grid Z dimension on Fermi and later architectures will now return values greater than 1 (on Fermi it is 65535). Methods for querying the max grid Z dimension are as follows:
  - CUDART:
    1. call **`cudaGetDeviceProperties(&prop, dev)`** and check **`prop.maxGridSize[2]`**
  - CUDA driver:
    1. call **`cuDeviceGetProperties(&devProps, hDev)`** and check **`devProps.maxGridSize[2]`**
    2. call **`cuDeviceGetAttribute(&i, CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z, hDev)`**

  Launching 3D grids is accomplished in CUDART by passing in a 3rd grid dimension in <<< >>> or in **`cudaConfigureCall()`**. Launching 3D grids with the CUDA driver requires the use of the new **`cuLaunchKernel`** API, which has **`gridDimX`**, **`gridDimY`** and **`gridDimZ`** parameters.

  It is important to note that only on Fermi and later architectures will an app be able to actually use 3D grid launches.
- (Windows) Layered Textures (2D)implemented.

Note: Layered textures are currently not supported on the Tesla architecture (sm_1x).

Layered textures are better known as *array textures* in graphics APIs. A layered texture is a collection of either 1D or 2D textures of identical size and format, arranged in layers. Such textures can be created as follows:

- by specifying the flag CUDA_ARRAY3D_LAYERED when creating the CUDA array using the driver API.
- by specifying the flag cudaArrayLayered when creating the CUDA array using the runtime API.

Kernels can access any texel from any particular layer using a new set of intrinsics that have the following format:

- **`tex1DLayered(texref, float x, int layer)`**
- **`tex2DLayered(texref, floay x, float y, int layer)`**

In a 2D layered texture, no filtering is performed between layers i.e. there is no trilinear filtering done like it is done for 3D textures. Similarly, for 1D layered texture, there is no bilinear filtering done like the way it is for 2D textures.

The second argument in the template for texture references now means *texture type* instead of *dim*. i.e. instead of *texture<returnType, dim, readMode>*, it is *texture<returnType, textureType, readMode>* The *textureType* arguments can be one of the following #defines:

```
#define cudaTextureType1D 0x01
#define cudaTextureType2D 0x02
#define cudaTextureType3D 0x03
#define cudaTextureType1DLayered 0xF1
#define cudaTextureType2DLayered 0xF2
```

Backward compatibility for the existing 1D, 2D and 3D textures is maintained by aliasing the corresponding *#defines* to their *dim* value. As a reult, sample texture references would look like:

```
texture<float4, cudaTextureType3D> texRef3D;
texture<float4, cudaTextureType1DLayered> texRef1DLayered;
```

- This version has a new launching API called **`cuLaunchKernel`**. This API offers many improvements over previous launching APIs:
  1. All function state associated with a launch is specified via one API call. This makes multithreaded launching of kernels feasible.
  2. Support for 3D dimensional grid launches on h/w that supports it (see associated *NVbug 599870 - 3D grid launches*)
  3. Kernel parameter passing can either be done via an easy to use method where addresses of parameters are passed in and the driver worries about packing the parameters together, or an expert mode (much like **`cuParamSetv`**) where all parameters are pre-packed by the application in one chunk.
- Added mechanism for registering system memory for DMA.

6.9.2. CUDA Compiler Features

Among the new features added in the CUDA 4.0 compiler are:

- Support for inline PTX: much like an **`__asm__`** directive, PTX can now be inlined with CUDA C/C++.
- Support for driver-loadable fatbins: fatbin files can contain multiple PTX and cubin files targeted at different architectures. In previous releases, only applications that used the runtime API were able to use fatbin files. Now with CUDA 4.0, driver API applications can use them too.

For more details on these features, please consult the nvcc documentation (*nvcc.pdf*) that comes with the release.

Starting with CUDA 4.0 release, the compiler implements enhanced error checks for function calls. The compiler checks that the calling function and the called function have compatible **`__host__`**, **`__device__ and __global__`** attributes. The compatibility rules for calls between functions with such attributes are documented in the *CUDA Programming Guide*.

If the compiler detects an incompatible call, it will generate error or warning messages. Warnings may be turned into errors in a future release. Additional error checks may be implemented in a future release. It is recommended that the user modify the calling function or the called function to ensure compatibility with the function call restrictions documented in the *CUDA Programming Guide*.

6.9.3. CUDA Libraries Features

- The CUBLAS Library now supports a new API that is thread-safe and allows the application to more easily take advantage of parallelism using streams (especially for functions with scalar return parameters). Because this new API is thread-safe, the *CUBLAS* library will work cleanly with applications that use the new multi-threading features of the CUDA Runtime Library (*CUDART*) in the CUDA Toolkit v4.0.

  The legacy CUBLAS API is still supported, but it is not thread-safe and does not offer as many opportunities for parallelism with streams as the new API. Existing applications that use CUBLAS should work without any changes to the existing code, they only need to explicitly link to the CUDART dynamic library during compilation. Note that this link requirement was not necessary with the previous versions of CUBLAS if the application only used CUBLAS entry points (and hence did not use any explicit CUDART entry points).

  We recommend that new applications use the new API. In addition, we recommend that you convert to the new API for exisiting applications that need maximum stream parallelism or correct operation in a multi-threaded scenario.

  The documentation in *doc/CUBLAS_Library.pdf* has been rewritten to focus on the new API; some treatment of the legacy API is still included.
- The *TRMM* routines in the CUBLAS Library can selectively operate either out-of-place or in-place (the traditional BLAS interface only operates in-place). The out-of-place option, which is new in this release, offers a significant speedup, up to 3x, on the Fermi architecture compared to the previous release, and a modest speedup on the Tesla architecture compared to the previous release. In general, as the input matrix sizes get larger, the performance of the TRMM routine can now approach the performance of the corresponding raw *GEMM* routines when operating out-of-place.
- The performance of the *ZGEMM* routine in the CUBLAS library, specifically for input matrices larger than about 100x100, has been optimized for the Fermi architecture.
- Added the **`cublasGetVersion()`** function to the CUBLAS Library.
- Performance has significantly improved (>1.5x) for double-precision power-of-2 transforms up to size 2048, especially on the Fermi architecture. Certain API features such as non-standard element strides, etc. will not trigger these new kernels, therfore performance is improved only in some cases.
- In the previous release candidate, the CUFFT Library had a performance regression for some 2D FFT sizes as compared to the 3.2 release. These regressions have been fixed.
- Added the **`cufftGetVersion()`** function to the CUFFT Library.

- In the previous version of the CUFFT Library, the *Bluestein* or *chirp* FFT algorithm was used to accelerate transforms for sizes that cannot be factored into a combination of powers-of-2, -3, -5, or -7 for 1D transforms only. This release employs the Bluestein algorithm to accelerate 2-D and 3-D transforms as well.
- The *CUFFT* Library APIs now support multiple batches for all 1D, 2D and 3D transforms. The previous release had limited support for multiple batches for 2D and 3D transforms.
- In this version of the CUDA Toolkit (v4.0), the CUFFT Library now supports more complex input and output data layouts via the advanced data layout parameters **`inembed`**, **`istride`**, **`idist`**, **`onembed`**, **`ostride`** and **`odist`**, as accepted by the **`cufftPlanMany()`** API. In this release, these parameters are supported only for complex-to-complex (C2C) transforms. This feature allows transforming a subset of an input array, or outputting to only a portion of a larger data structure. If the user sets **`inembed`** or **`onembed`** to NULL, then the CUFFT Library will function as it did in the previous releases and assume a basic data layout and ignore the other advanced parameters. If the user intends to use the advanced parameters, then all of the advanced interface parameters should be specified correctly. Advanced parameters are defined in units of the relevant data type (**`cufftReal`**, **`cufftDoubleReal`**, **`cuComplex`**, **`cuDoubleComplex`**).
- The *CUSPARSE* library now provides a solver for triangular sparse linear systems, via the **`cusparse*csrsv_analysis()`** and **`cusparse*csrsv_solve()`** APIs. Refer to the document, *CUSPARSE_Library.pdf* for detailed usage information.
- The **`cusparse*csrmv()`** and **`cusparse*csrmm()`** routines in the CUSPARSE library now support symmetric (**`CUSPARSE_MATRIX_TYPE_SYMMETRIC`**) and Hermitian (**`CUSPARSE_MATRIX_TYPE_HERMITIAN`**) matrix types.
- Renamed **`cudaDeviceBlockingSync`** to **`cudaDeviceScheduleBlockingSync`**
- The **`cospi()`** routine has been added for single-precision and double-precision floating-point datatypes. The function cospi(x) implements cos(x * PI). No special include file is required to access this routine. Note: the **`sinpi()`** routine has already been available in previous releases.
- In previous releases of the CUDA toolkit, the CUBLAS and CUSPARSE libraries included compiled kernel PTX and compiled kernel binaries for compute capability 1.0, 1.3 and 2.0. Starting with this release, the compiled kernel PTX will only be shipped for the highest supported compute capability (i.e., 2.0 for this release). This results in a significant reduction of file size for the CUBLAS and CUSPARSE dynamically linked libraries for all platforms. Note: there is no change to the compiled kernel binaries.
- The CURAND library now supports generation of double-precision floating point *Sobol' quasi-random sequences* with 53 bits of randomness, as well as 64 bit integer Sobol' quasi-random sequences. These are accessed via the **`CURAND_RNG_QUASI_SOBOL64`** and **`CURAND_RNG_QUASI_SCRAMBLED_SOBOL64`** generator types in the host API and the **`curandStateSobol64_t`** and **`curandStateScrambledSobol64_t`** generator structures in the device API.
- The CURAND library now supports generation of log-normally distributed random numbers, via the **`curandGenerateLogNormal()`** and **`curandGenerateLogNormalDouble()`** host API functions and the **`curand_log_normal()`**, **`curand_log_normal2()`**,

curand_log_normal_double() and **curand_log_normal2_double()** device API functions.

- The CURAND library now supports generation of scrambled Sobol' quasi-random numbers, via the **CURAND_RNG_QUASI_SCRAMBLED_SOBOL32** and **CURAND_RNG_QUASI_SCRAMBLED_SOBOL64** generator types in the host API and the **curandStateScrambledSobol32_t** and **curandStateScrambledSobol64_t** generator structures in the device API.
- The CURAND library documentation (*doc/CURAND_Library.pdf*) now contains a summary and selected detailed results of the statistical quality tests run against the generators provided by CURAND.
- Beginning with this release, the NVIDIA Performance Primitives (*NPP*) library is included directly within the CUDA Toolkit. Currently, the NPP library supports a variety of basic signal and image processing primitives that are optimized across the range of CUDA capable GPUs. Documentation is found at *doc/NPP_Library.pdf* and the public header file is at *include/npp.h*.
- Added a complete set of Arithmetic and Logical Signal Processing Primitives.
- NPP has added Beta support for asynchronous operation using CUDA streams via the **nppSetStream()** and **nppGetStream()** functions. This feature is provided in an early form in this release and will be provided in a non-Beta fully tested form in a future release.
- The Thrust CUDA library is now included with the CUDA Toolkit in the */include/thrust* directory. A Quick Start document is available at *doc/Thrust_Quick_Start_Guide.pdf*. Additionally, several code samples in the *NVIDIA GPU Computing SDK* now employ *Thrust*. The Thrust library source code, additional detailed documentation, example programs and a discussion group will continue to be available at the project's original home at *http://code.google.com/p/thrust/*.
- This version of Thrust introduces **discard_iterator**, an output iterator which ignores values assigned to it. **discard_iterator** is useful for discarding unnecessary output from algorithms with multiple output ranges (such as **reduce_by_key**), and measuring in advance the total size of the result of algorithms which produce variably-sized output (such as set_intersection).
- The Thrust library now provides set operations for sorted ranges, including union, difference and symmetric difference. These new operations are exposed via *thrust/set_operations.h*.
- Added CUDA runtime API functions to control profiling:

  cudaProfilerInitialize() - Initialize profiling
  cudaProfilerStart() - Start profiling
  cudaProfilerStop() - Stop profiling

  A new header file "cuda_profiler_api.h" has been added for these runtime API functions. The corresponding driver APIs are cuProfilerInitialize(), cuProfilerStart(), cuProfilerStop() and the header file is "cudaProfiler.h".

6.9.4. CUDA Libraries Performance

- The performance of transforms in the *CUFFT* library that are pure powers of 3, 5, and 7 have been optimized significantly in this release, especially for double precision.
- In version 3.2 of *CUSPARSE*, the **`csrmv()`** and **`csrmm()`** functions ran slower when the *beta* parameter was =0 than when it was =1. In this version, the performance variation has been removed, and **`csrmv()`** and **`csrmm()`** should run slightly faster when *beta* =0.
- The *GEMV* routines, for all datatypes, in the *CUBLAS* library have been significantly optimized for the case in which the input matrix, A, is transposed. Performance has improved up to 2x, especially when the input matrix, A, is large. The performance improvements apply to both the Tesla (GT200) and Fermi (GF100) architectures.
- The performance of the *TRSM* routines in the CUBLAS library for large matrices has been significantly improved on Fermi and Tesla architecture platforms.
- The performance of the double-precision hyperbolic sine function, **`sinh()`**, has been improved significantly on GF100 (Fermi architecture) and GT200 (Tesla architecture). The exact improvement achieved for end applications using **`sinh`** will vary based on the specific characteristics of each application.
- Improved performance of CUFFT on *R2C* and *C2R* transforms whose input data size along the X (or, least significant) dimension is a multiple of 2 but not a multiple of 4. In the previous release, the performance was much better when this size was a full multiple of 4; now, both cases should run at the same higher performance.
- The performance of double-precision floating point division on the Fermi architecture has been significantly optimized for the round-to-nearest-even case, which is the default rounding mode employed when using the **`/`** operator in CUDA-C device code. The round-to-nearest-even mode can be explicitly employed in CUDA using the **`__ddiv_rn()`** intrinsic. The exact improvement achieved for end applications that perform double precision divides will vary based on the specific characteristics of each application.
- CURAND supports a new ordering technique for pseudo-random generators (**`CURAND_ORDERING_PSEUDO_SEEDED`**) that significantly reduces the state setup time. However, since this ordering technique uses a different starting seed for each thread on the device, it may result in statistical weaknesses of the pseudorandom output for some user seed values.
- The performance of the **`SYR2K`** and **`HER2K`** routines in the CUBLAS library has been optimized for the Fermi architecture.
- The **`SYMM`** and **`HEMM`** routines in CUBLAS have been significantly optimized for the Fermi architecture. For instance, in some cases there is a 3x performance improvement over the previous version of these routines, both for single and for double precision.
- The performance of the double-precision reciprocal square-root function, **`rsqrt()`**, has been improved significantly for GT200 (the Tesla architecture) and GF100 (the Fermi architecture). The exact improvement achieved for end applications that use rsqrt will vary based on the specific characteristics of each application.

- The performance and accuracy of the double-precision erfc() function have been improved. This function is now accurate to 4 ulps, and the performance has significantly improved on both the Tesla and Fermi architectures. The exact improvement achieved for end applications that use erfc will vary based on the specific characteristics of each application.

6.10. Known Issues

- In the current release, the *TCC* driver cannot be run under a guest account; admin privileges are needed to run TCC. This requirement will be removed in a future release.
- GPUs without a display attached are not subject to the 2 second runtime restriction. For this reason it is recommended that CUDA be run on a GPU that is *NOT* attached to a display and does not have the Windows desktop extended onto it. In this case, the system must contain at least one NVIDIA GPU that serves as the primary graphics adapter. Thus, for devices like S1070 that do not have an attached display, users may disable the Windows *TDR* timeout. Disabling the TDR timeout will allow kernels to run for extended periods of time without triggering an error.

  The following is an example .reg script:

  ```
  Windows Registry Editor Version 5.00
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\GraphicsDrivers]
  "TdrLevel"=dword:00000000
  ```

- The header file search locations and the order that they are visited have been revised.

  Until CUDA 3.2, nvcc searched the following locations, in order:

  1. The toolkit include paths,
  2. The current working directory,
  3. The paths specified with **`-I`**,
  4. The paths specified with **`-isystem`**, and
  5. The system include paths.

  The header files in the toolkit include path could not be overridden as the toolkit include paths were always visited first.

  From CUDA 4.0, nvcc searches through the include paths in the following order:

  1. The paths specified with **`-I`**,
  2. The toolkit include paths,
  3. The paths specified with -isystem, and
  4. The system include paths.

  The current working directory is not added to the include paths by default anymore, adhering to the C/C++ compiler convention. That is, to add the current working directory to the include search paths, **`-I`**. or -isystem. must be given to **`nvcc`**, depending on the desired search order. Alternatively, the **`#include`** directives can be used in the quoted form, instead of the angle-bracket form, to include header files in the current working directory.

- 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.
- (Windows) Structure and union types with bit fields may not work correctly in device code on the Windows platform.

  In addition:
  - Transferring variables that contain such types, from host to device or from device to host, may not work correctly.
  - Use of variables with such types in device code may not work correctly.

  This issue will be addressed in a future release.
- When compiling **thrust::reduce** **cudafe** generates use of private **typedefs**
- (Windows) The CUDA C compiler may produce a different memory layout, compared to the host Microsoft compiler, for a C++ object of class type T that satisfies any of the following conditions:

  1. T has virtual functions or derives from a direct or indirect base class that has virtual functions.
  2. T has a direct or indirect virtual base.
  3. T has multiple inheritance with more than one direct or indirect empty base class.

  The size for such an object may also be different in host and device code. As long as type **T** is used exclusively in host or device code, the program should work correctly. Do not pass objects of type **T** between host and device code (e.g. as arguments to **__global__** functions or through **cudaMemcpy** calls).”
- For certain configurations, the CUFFT Library will produce slightly different results for the same input when *ECC* is on versus when ECC is off, even on the same architecture.

  

  In both cases the results are mathematically within the expected tolerance. The difference arises from optimizations specific to the ECC on and ECC off cases that result in slightly different factorizations of the overall transform into smaller radixes.
- The CUFFT library is not thread-safe, and hence cannot be accessed concurrently from multiple threads in the same process. This will be fixed in a future release.
- CUDALibraries has 4 SDK samples that do not build on certain Linux 32-bit Operating Systems. The *Makefile* links incorrectly to **-lUtilNPP_i686**; it should be **-lUtilNPP_i386**.

  To build NPP samples properly on 32-bit Linux replace all instances of **-lUtilNPP_$(OS_ARCH)** with **-lUtilNPP_$(LIB_ARCH)** in the following Makefiles:
  - **CUDALibraries/src/boxFilterNPP/Makefile**
  - **CUDALibraries/src/freeImageInteropNPP/Makefile**

- **CUDALibraries/src/imageSegmentationNPP/Makefile**
- **CUDALibraries/src/histEqualizationNPP/Makefile**

- When a program is terminated while waiting on a breakpoint, the system needs to be rebooted. This affects the TCC driver for Windows Vista and Windows 7.
- There is a known driver bug when debugging CUDA applications which use TCC. If the application terminates while paused at a GPU breakpoint, internal driver state can be corrupted. Until the system is rebooted, further attempts to create CUDA contexts will enter an infinite loop during **cuCtxCreate()**.
- GPU enumeration order on multi-GPU systems is non-deterministic and may change with this or future releases. Users should make sure to enumerate all CUDA-capable GPUs in the system and select the most appropriate one(s) to use.

6.10.1. Vista, Server 2008 and Windows 7 Related

- In order to run CUDA on a non-TESLA GPU, either the Windows desktop must be extended onto the GPU, or the GPU must be selected as the PhysX GPU.
- Individual kernels are limited to a 2-second runtime by Windows Vista. Kernels that run for longer than 2 seconds will trigger the *Timeout Detection and Recovery* (TDR) mechanism. For more information, see http://www.microsoft.com/whdc/device/display/wddm_timeout.mspx..
- The CUDA Profiler does not support performance counter events on Windows Vista. All profiler configuration regarding performance counter events is ignored.
- The maximum size of a single memory allocation created by **cudaMalloc** or **cuMemAlloc** on WDDM devices is limited to:

  ```
  MIN ( ( System Memory Size in MB - 512 MB ) / 2,
  PAGING_BUFFER_SEGMENT_SIZE )
  ```

  For Vista, **PAGING_BUFFER_SEGMENT_SIZE** is approximately 2GB.
- The OS may impose artificial limits on the amount of memory you can allocate using the Cuda APIs for both system and video memory. In many cases, these limits are significantly less than the size of physical system and video memory, but there are exceptions that make it difficult to quantify the expected behavior for a particular application.

6.10.2. XP, Vista, Server 2008 and Windows 7 Related

- Applications that try to use too much memory may cause a CUDA memcopy or kernel to fail with the error **CUDA_ERROR_OUT_OF_MEMORY**. If this happens, the CUDA context is placed into an error state and must be destroyed and recreated if the application wants to continue using CUDA.
- **malloc** may fail due to running out of virtual memory space. The address space limitation is fixed by a Microsoft issued hotfix. Please install the patch located at: http://support.microsoft.com/kb/940105 if this is an issue. Windows Vista SP1 includes this hotfix.
- When compiling a source file that includes **vector_types.h** with the Microsoft compiler on a 32-bit Windows system, the 16-byte aligned vector types are not properly aligned at 16 bytes.

6.10.3. XP Related

- OpenGL interoperability
  - OpenGL can not access a buffer that is currently *mapped*. If the buffer is registered but not mapped, OpenGL can do any requested operations on the buffer.
  - Deleting a buffer while it is mapped for CUDA results in undefined behavior.
  - Attempting to map or unmap while a different context is bound than was current during the buffer register operation will generally result in a program error and should thus be avoided.
  - Interoperability will use a software path on SLI
  - Interoperability will use a software path if monitors are attached to multiple GPUs and a single desktop spans more than one GPU (i.e. WinXP dualview).
- OpenCL program binary formats may change in this or future releases. Users should create programs from source and should not rely on compatibility of generated binaries between different versions of the driver.
- (Windows and Linux) Individual GPU program launches are limited to a run time of less than 5 seconds on a GPU with a display attached. Exceeding this time limit usually causes a launch failure reported through the CUDA driver or the CUDA runtime. GPUs without a display attached are not subject to the 5 second runtime restriction. For this reason it is recommended that CUDA be run on a GPU that is NOT attached to a display and does not have the Windows desktop extended onto it. In this case, the system must contain at least one NVIDIA GPU that serves as the primary graphics adapter.
- (Windows and Linux) It is a known issue that **`cudaThreadExit()`** may not be called implicitly on host thread exit. Due to this, developers are recommended to explicitly call **`cudaThreadExit()`** while the issue is being resolved.per email thread started by Cliff Woolley
- For maximum performance when using multiple byte sizes to access the same data, coalesce adjacent loads and stores when possible rather than using a union or individual byte accesses. Accessing the data via a union may result in the compiler reserving extra memory for the object, and accessing the data as individual bytes may result in non-coalesced accesses. This will be improved in a future compiler release.

6.10.4. Linux Only

- There is a known bug in ICC with respect to passing 16-byte aligned types by value to GCC-built code such as the CUDA Toolkit libraries e.g. CUBLAS. At this time, passing a **`double2`** or **`cuDoubleComplex`** or any other 16-byte aligned type by value to GCC-built code from ICC-built code will pass incorrect data. Intel has been informed of this bug. As a workaround, a GCC-built wrapper function that accepts the data by reference from the ICC-built code can be linked with the ICC-built code; the GCC-built wrapper can then, in turn, pass the data by value to the CUDA Toolkit libraries.

- In order to run CUDA applications, the CUDA module must be loaded and the entries in **/dev** created. This may be achieved by initializing X Windows, or by creating a script to load the kernel module and create the entries. An example script (to be run at boot time):

```
#!/bin/bash
/sbin/modprobe nvidia
if [ "$?" -eq 0 ]; then
    # Count the number of NVIDIA controllers found.
    N3D=`/sbin/lspci | grep -i NVIDIA | grep "3D controller" | wc -l`
    NVGA=`/sbin/lspci | grep -i NVIDIA | grep "VGA compatible controller" |
 wc -l`
    N=`expr $N3D + $NVGA - 1`
    for i in `seq 0 $N`; do
        mknod -m 666 /dev/nvidia$i c 195 $i;
    done
    mknod -m 666 /dev/nvidiactl c 195 255
else
    exit 1
fi
```

- The Linux kernel provides a mode where it allows user processes to overcommit system memory. (Refer to kernel documentation **/proc/sys/vm/** for details). If this mode is enabled-the default on many distros-the kernel may have to kill processes in order to free up pages for allocation requests. The CUDA driver process, especially for CUDA applications that allocate lots of zero-copy memory with **cuMemHostAlloc** or **cudaMallocHost**, is particularly vulnerable to being killed in this way. Since there is no way for the CUDA SW stack to report an OOM error to the user before the process disappears, users, especially on 32bit Linux, are encouraged to disable memory overcommit in their kernel to avoid this problem.

  Please refer to documentation on **vm.overcommit_memory** and **vm.overcommit_ratio** for more information.

6.10.5. Linux and Mac

- When compiling with GCC, special care must be taken for structs that contain 64-bit integers. This is because GCC aligns long longs to a 4 byte boundary by default, while NVCC aligns long longs to an 8 byte boundary by default. Thus, when using GCC to compile a file that has a struct/union, users must give the **-malign-double** option to GCC. When using NVCC, this option is automatically passed to GCC.
- It is a known issue that **cudaThreadExit()** may not be called implicitly on host thread exit. Due to this, developers are recommended to explicitly call **cudaThreadExit()** while the issue is being resolved.

6.10.6. Mac Only

- OpenGL interop will always use a software path leading to reduced performance when compared to interop on other platforms.
- CUDA kernels which do not terminate or run without interruption for several tens of seconds may trigger the GPU to reset causing a disruption of any attached displays. This may cause display image to become corrupted, which will disappear upon a reboot.

- The kernel driver may leak wired (i.e. unpageable memory) if CUDA applications terminate in unexpected ways. Continued leaks will lead to severely degraded system performance and requires a reboot to fix.
- On systems with multiple GPUs installed or systems with multiple monitors connected to a single GPU, OpenGL interoperability always copies shared buffers through host memory.
- Current hardware limits the number of asynchronous memcopies that can be overlapped with kernel execution. Overlap is also limited to kernels executing for less than 1 second. These limitations are expected to improve on future hardware.
- The following APIs exhibit high CPU utilization if they wait for the hardware for a significant amount of time.. As a workaround, apps may use **`cu(da)StreamQuery`** and/or **`cu(da)EventQuery`** to check whether the GPU is busy and yield the thread as desired.

  - **`cuCtxSynchronize`**
  - **`cuEventSynchronize`**.
  - **`cuStreamSynchronize`**.
  - **`cudaThreadSynchronize`**
  - **`cudaEventSynchronize`**.
  - **`cudaStreamSynchronize`**
- The MacBook Pro currently presents both GPUs as available for use in Performance mode. This is incorrect behavior, as only one GPU is available at a time. CUDA applications that try to run on the second GPU (device ID 1) will potentially hang. This hang may be terminated by pressing **ctrl-C** or closing the offending application.
- There is a potential for a system hang if any running CUDA application terminates abnormally while executing divergent code on the MAC OS. This issue has been fixed in the newer Mac driver version 256.01.00f03 available on: http://www.nvidia.com/.

6.11. Resolved Issues

The following known issues that were published in *CUDA Toolkit 3.2* (and 4.0 RC, RC2) release notes and errata documents have been fixed:

- For devices with compute capability 1.x, only the *Occupancy analysis* part of *Kernel analysis* was supported by the Visual Profiler. The information displayed under *Limiting Factor Identification* in the kernel analysis window was not accurate and was not to be used. This issue has been fixed.
- When profiling OpenCL applications on devices with compute capability 1.x. an *Invalid cta_launched column* error was previously reported. This issue has been fixed.
- Visual Profiler was reported to crash when trying to profile a application on Ubuntu 10.10. This issue has been fixed.
- Earlier version reported a known issue that when profiling an application in Visual Profiler on a device with compute capability 1.x with the *Normalized counters* option enabled, incorrect signals are selected resulting in warnings. This issue has been fixed.

- Earlier version reported a known issue that for some SDK applications (e.g. **simpleMultiGPU**) which run on multiple GPU devices, the Visual Profiler output is generated only for one device. This issue has been fixed.
- **NV50_P2P** allocations are limited to only allow P2P objects to be allocated between GPUs in the same peer group.

  The details are as follows:

  pciDomainID is added to the **cudaDeviceProp** structure
  description: **pciDomainID** is the PCI domain identifier of the device
  CU_DEVICE_ATTRIBUTE_PCI_DOMAIN_ID added as a constant for **cuDeviceGetAttribute**

6.11.1. Mac Related

- To save power, some Apple products automatically power-down the CUDA-capable GPU in the system. If the operating system has powered down the CUDA-capable GPU, CUDA fails to run and the system returns an error that no device was found.

  In order to ensure that your CUDA-capable GPU is not powered down by the operating system do the following:

  1. Go to **System Preferences**.
  2. Open the **Energy Saver** section.
  3. Un-check the **Automatic graphics switching** check box in the upper left.
- This issue described in the previous version has been fixed in CUDA Toolkit 4.0. On Mac OS only, the NVIDIA C Compiler (nvcc) handles **size_t** incorrectly during 64-bit compilation. The version of nvcc included with CUDA Toolkit 3.2 fails to handle variables of type **size_t** as an 8-byte entity in PTX when compiling 64-bit device code. To address this issue, NVIDIA has released a patch that updates components of nvcc.

  The patch is available as *CUDA Toolkit: GFEC Patch for MacOS* from the following location: http://developer.nvidia.com/object/cuda_3_2_downloads.html

  Please refer to additional information and installation instructions in the *README file* distributed with the patch.
- The following issue reported in the previous version has been fixed in CUDA Toolkit 4.0.

  In CUBLAS 3.2, the *GEMM*, *SYRK*, and *HERK* routines for Fermi GPUs can enter an infinite recursion leading to an application crash for certain input sizes meeting the criteria below. To work around this problem, the input to CUBLAS must be recursively subdivided until the individual calls to these CUBLAS routines do not match these criteria.

  Given threshold size *T*, where *T* is equal to 2^27 - 512 (i.e., 134217216), the crash might be seen in any of the following circumstances:

  1. A is not transposed, lda * k >= T, and T is divisible by lda.
  2. B is not transposed, ldb * n >= T, T is divisible by n, and n is divisible by 32.
  3. A is transposed, lda * m >= T, T is divisible by m, and m is divisible by 32.

4. B is transposed, ldb * k >= T, and T is divisible by ldb.

- The performance of the *TRMM* routine in this 4.0 release has regressed compared to the performance in the 3.2 release. This will be fixed in the final 4.0 production release. As a work-around, the new out-of-place option provided in the new CUBLAS API for TRMM can be used. The performance of this out-of-place implementation is much higher than the 3.2 performance.
- In the previous release of the CUBLAS Library, the **`cublasDgemm()`** routine produced incorrect results in some cases when k < 32 and matrix A is transposed. This has been fixed in this release.
- (Windows and Linux) In the previous version, **`divergent_branch`** counter in Visual Profiler reported an incorrect value (of zero) for Fermi. This issue has been fixed in CUDA Toolkit 4.0.
- (Windows) **`cudaMempy3D`** no longer ignores src and dst position parameters for host memory.
- The **`cublasCgemm()`** routine in the CUBLAS library would crash in a few specific cases in the previous release; fixed in this release.
- The **`cufftPlanMany()`** API in the 4.0 RC release had a bug that caused previously working application code to fail. In particular, when **`inembed`** was set to NULL, and **`istride`** or **`idist`** were set to invalid values, the API would return the **`CUFFT_INVALID_VALUE`** error code. This has been fixed, and now the error checks are only executed if **`inembed`** is not NULL. This applies to the **`onembed`**, **`ostride`** and **`odist`** parameters as well.
- In the previous version of the CUFFT Library, there was a memory leak in some cases when creating and subsequently destroying a plan for a FFT transform whose size had a prime factor larger than 47. This has been fixed in the current release.
- The **`cublasFree()`** interface in the Legacy CUBLAS API has been corrected to remove the *const* type qualifier from the **`void *devicePtr`** argument in order to match the **`cudaFree()`** and the standard C **`free()`** APIs. Note that this may cause user code that depends on that parameter being const not to compile with the latest version of CUBLAS, though this should be an uncommon scenario.
- In the previous release, in certain situations, the CUFFT library would print the following error message to stderr: **cufft: Failed to find applicable transform**.

  In the current release, all errors are reported via API return codes and the library does not print anything directly to **`stdout`** or **`stderr`**.
- Fixed in this release: When profiling an application in Visual Profiler on a device with compute capability 1.x with the *Normalized counters* option enabled, incorrect signals are selected resulting in warnings. To avoid the warnings, do not enable the Normalized counters option.
- Fixed in this release: Issue reported in earlier release notes: For some SDK applications (e.g. **`simpleMultiGPU`**) which run on multiple GPU devices, the Visual Profiler output is generated only for one device.
- Fixed in this release: In the earlier release, Visual Profiler sample project **`Nbody.cvp`** could not be opened on Linux unless the file was remaned from **`Nbody_nbody_Context_0.csv`** to **`Nbody_Nbody_Context_0.csv`**.
- Fixed in this release: Issue reported in earlier release notes: GPU enumeration order on multi-GPU systems is non-deterministic and may change with this or future

releases. Users should make sure to enumerate all CUDA-capable GPUs in the system and select the most appropriate one(s) to use.

- Fixed in this release (Vista, Server 2008 and Windows 7 related): Issue reported in earlier release notes: The CUDA Profiler does not support performance counter events on Windows Vista. All profiler configuration regarding performance counter events is ignored.
- In previous releases, the **nppiNormDiff_8u_C1R** function in the NPP library returned both output values into host pointers. In this release, the semantics of this API function have been changed and now the pointers provided for the two outputs are assumed to be pointing to device memory. There will be no compilation error as the prototype of the function has not changed and the program may fail silently; hence if this function is being used we recommend that the code be updated proactively by users.
- In previous versions of the NPP Library, the Rotate primitives set pixel values inside the destination ROI to 0 (black) if there is no pixel value from the source image that corresponds to a particular destination pixel. This incorrect behavior has been fixed. Now, these destination pixels are left untouched so that they stay at the original background color.
- In the previous CUDA Toolkit 4.0 release candidates, the NPP Library header file, nppi.h, made use of const references for passing structs to functions. This causes compilation errors when included from within a C file (as opposed to from within a C++ file). Since the NPP API is intended to be a pure C API, the offending C++ constructs have been removed from the header file.
- In the previous release of the NPP Library, the **nppiGraphcut_32s8u** API function would return a **NPP_TEXTURE_BIND_ERROR** in some cases when the API should have executed to completion without error. This has been fixed in the current release.
- Improved the accuracy of the generation of normally distributed single-precision pseudo-random numbers in the CURAND library. The main observed impacts of this improvement are:

  1. the maximum difference between the results generated by a GPU generator and a HOST generator are much smaller for single-precision normally distributed random numbers; and
  2. the performance of GPU random number generation is now slower than the previous version for single-precision normally distributed random numbers.
- The *Sobol'* direction vectors used by the CURAND library have been updated using the latest Joe-Kuo file **new-joe-kuo-6.21201**. The file was obtained from this website: http://web.maths.unsw.edu.au/%7Efkuo/sobol/. The smallest dimension with updated values in the new file is the 212th dimension. Therefore, the exact Sobol' sequences generated by CURAND may differ from the previous release even for the same exact input parameters, if more than 211 dimensions are requested. The authors of the direction vectors indicate that the previous set of vectors were corrupted and that their use be discontinued.
- The previous version of the NPP library had a bug in the **nppsDiv_32s_C1R** primitive when dividing by *0*. This bug has been fixed, and now the primitive will correctly return **NPP_MAX_32S** or **NPP_MIN_32S** when dividing by 0.

- (Operating Systems: Windows2008 Server64, WinXP-x64) In the previous version a setup consisting of GF100 M2070-Q + R260.27 driver resulted in SDK sample **`DeviceQuery`** not running when switched from OS to regular user account. This has been fixed in this version.
- In the previous release of the NPP Library, the **`nppiMinMax_8u_C1R()`** function would not work in certain situations; this has been fixed in this release.
- For an OpenCL C program, the maximum alignment of a function scope local variable and a function parameter variable is limited to 16-byte.
- In previous releases, the **`nppiMean_StdDev_8u_C1R`** function in the NPP library returned both output values into host pointers. In this release, the semantics of this API function have been changed and now the pointers provided for the two outputs are assumed to be pointing to device memory. There will be no compilation error as the prototype of the function has not changed and the program may fail silently; hence if this function is being used we recommend that the code be updated proactively by users.
- In the previous release, the **`*Filter_8u_C1R`** functions in the NPP library produced incorrect results when the **`nSrcStep`** input parameter was not a multiple of 4. This has been corrected, and now the functions work for all values of **`nSrcStep`**. The exact list of impacted functions is **`nppiFilterRow_8u_C1R`**, **`nppiFilterBox_8u_C1R`**, **`nppiFilter_8u_C1R`**, **`nppiFilterMax_8u_C1R`**, and **`nppiFilterMin_8u_C1R`**.
- In previous releases, the **`nppiMinMax_8u_C1R`** function in the NPP library returned both output values into host pointers. In this release, the semantics of this API function have been changed and now the pointers provided for the two outputs are assumed to be pointing to device memory. There will be no compilation error as the prototype of the function has not changed and the program may fail silently; hence if this function is being used we recommend that the code be updated proactively by users.
- The accuracy of single-precision transforms in the CUFFT Library has been signifcantly improved, especially for larger transforms and multi-dimensional transforms.

The accuracy improvements in general did not impact performance compared to the previous version of CUFFT, however some single precision power-of-2 kernels on the Fermi architecture will show a minor performance regression compared to the previous version of the library.

- In previous versions of the CUFFT Library, for some 1D transform sizes larger than 32M elements, the first call to **`cufftExec*()`** would fail due to insufficient memory or due to grid size limitations. These resource limitations are now properly checked for and reported by **`cufftPlan*()`** such that if sufficient resources are not available to execute an FFT of the requested size, the error will be reported at plan time rather than at execution time.
- Thrust no longer supports scatter and gather directly between host and device memory; instead the output needs to be staged through a temporary object and copied explicitly with **`thrust::copy()`**.

- Thrust no longer supports operations on **device_vector** when the backend is CUDA in the absence of **nvcc**. Hence, operations which modify device_vector's size or elements are unavailable in a **.cpp** file.

6.12. Source Code for Open64 and CUDA-GDB

- The Open64 and CUDA-GDB source files are controlled under terms of the GPL license. Current and previously released versions are located at: ftp://download.nvidia.com/CUDAOpen64
- Linux users:
  - Please refer to the *Release Notes* and *Known Issues* sections in the *CUDA-GDB User Manual* (**CUDA_GDB.pdf**).
  - Please refer to **CUDA_Memcheck.pdf** for notes on supported error detection and known issues.

6.13. More Information

For more information and help with CUDA, please visit http://www.nvidia.com/cuda.

6.14. Acknowledgements

NVIDIA extends thanks to EM Photonics (http://www.emphotonics.com) for their contributions to the matrix-vector multiplication functions in the CUBLAS library incorporated into the v4.0 release.