NVIDIA CUDA TOOLKIT 8.0.61

RN-06722-001 _v8.0 | June 2017

Release Notes for Windows, Linux, and Mac OS

TABLE OF CONTENTS

ERRATA

New Features

cuBLAS 8.0.61.2 New Features

cuBLAS Library

This update contains performance enhancements and bug-fixes to the cuBLAS library in CUDA Toolkit 8. Deep Learning applications based on Recurrent Neural Networks (RNNs) and Fully Connected Networks (FCNs) will benefit from new GEMM kernels and improved heuristics in this release.

This update supports the x86_64 architecture on Linux, Windows, and Mac OS operating systems, and the ppc64le architecture on Linux only.

The highlights of this update are as follows:

- Performance enhancements for GEMM matrices used in speech and natural language processing
- Integration of OpenAI GEMM kernels
- Improved GEMM heuristics to select optimized algorithms for given input sizes
- Heuristic fixes for batched GEMMs
- GEMM performance bug fixes for Pascal and Kepler platforms

CUDA 8.0.61 (CUDA 8.0 Production) New Features

CUDA Tools

- **CUDA Compilers.** The CUDA compiler now supports Xcode 8.2.1.
- **NVRTC.** NVRTC is no longer considered a preview feature.

CUDA Libraries

- **cuBLAS.** The cuBLAS library added a new function `cublasGemmEx()`, which is an extension of `cublas<t/>gemm()`. It allows the user to specify the algorithm, as well as the precision of the computation and of the input and output matrices.

The function can be used to perform matrix-matrix multiplication at lower precision.

Resolved Issues

General CUDA

- **CUDA Installer.** On some SLES or openSUSE system configurations, the NVIDIA GL library package may need to be locked before the steps for a GL-less installation are performed. The NVIDIA GL library package can be locked with this command:

```
sudo zypper addlock nvidia-glG04
```

- **Unified memory.** On GP10x systems, applications that use **`cudaMallocManaged()`** and attempt to use **`cuda-gdb`** will incur random spurious MMU faults that will take down the application.
- **Unified memory.** Functions **`cudaMallocHost()`** and **`cudaHostRegister()`** don't work correctly on multi-GPU systems with the IOMMU enabled on Linux. The only workaround is to disable unified memory support with the **`CUDA_DISABLE_UNIFIED_MEMORY=1`** environment variable.
- **Unified memory.** Fixed an issue where **`cuda-gdb`** or **`cuda-memcheck`** would crash when used on an application that calls **`cudaMemPrefetchAsync()`**.
- **Unified memory.** Fixed a potential issue that can cause an application to hang when using **`cudaMemPrefetchAsync()`**.

CUDA Tools

- **CUDA Compilers.** Fixed an issue with wrong code generation for computing the address of an array when using a 64-bit index.
- **CUDA Compilers.** When a program is compiled with whole program optimization, applying launch bounds to recursive functions or to indirect function calls may have unpredictable results.
- **CUDA Profiler.** The PC sampling warp state counts were incorrect in some cases.
- **CUDA Profiler.** Profiling applications using **`nvprof`** or Visual Profiler on systems without an NVIDIA driver resulted in an error. This is now reported as a warning.
- **cuSOLVER.** Fixed an issue with the cuSOLVER library where some of its functions were not exposed, resulting in link errors.
- **NVTX.** The NVIDIA Tools Extension SDK (NVTX) function **`nvtxGetExportTable()`** was missing from the export table list.

CUDA Libraries

- **cuBLAS.** Fixed GEMM performance issues on Kepler and Pascal for different matrix sizes, including small batches. Note that this fix is available **only** in the cuBLAS packages on the CUDA network repository.
- **cuBLAS.** Updated the cuBLAS headers to use comments that are in compliance with ANSI C standards.

- **cuBLAS.** Made optimizations for mixed-precision (**`FP16`**, **`INT8`**) matrix-matrix multiplication of matrices with a small number of columns **`(n)`**.
- **cuBLAS.** Fixed an issue with the **`trsm()`** function for large-sized matrices.

Known Issues

General CUDA

- **CUDA library.** Function **`cuDeviceGetP2PAttribute()`** was not published in the cuda library (**`libcuda.so`**). Until a new build of the toolkit is issued, users can either use the driver version, **`cudaDeviceGetP2PAttribute()`**, or perform the link to use **`libcuda`** directly instead of the stub (usually it can be done by adding **`-L/usr/lib64`**).

CUDA Tools

- **CUDA Profiler.** When a device is in the "exclusive" process compute mode, the profiler may fail to collect events or metrics in "application replay" mode. In this case, use "kernel replay" mode.
- **CUDA Profiler.** In the Visual Profiler, the **Run > Configure Metrics and Events...** dialog does not work for the device that has NVLink support. It's suggested to collect all metrics and events using **`nvprof`** and then import into **`nvvp`**.
- **CUDA Profiler, CUPTI.** Some devices with compute capability 6.1 don't support multi-context scope collection for metrics. This issue affects **`nvprof`**, Visual Profiler, and CUPTI.

Chapter 1. CUDA TOOLKIT MAJOR COMPONENTS

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

Compiler

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

Tools

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

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

Libraries

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

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

- **nvcuvid** (CUDA Video Decoder [Windows, Linux])
- **nvgraph** (CUDA nvGRAPH [accelerated graph analytics])
- **nvml** (NVIDIA Management Library)
- **nvrtc** (CUDA Runtime Compilation)
- **nvtx** (NVIDIA Tools Extension)
- **thrust** (Parallel Algorithm Library [header file implementation])

CUDA Samples

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

Documentation

The most current version of these release notes can be found online at http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html. Also, the **version.txt** file in the root directory of the toolkit will contain the version and build number of the installed toolkit.

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

CUDA-GDB Sources

CUDA-GDB sources are available as follows:

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

Chapter 2. NEW FEATURES

2.1. General CUDA

- CUDA 8.0 adds support for GPUDirect Async, which improves application throughput by eliminating the CPU as the critical path in GPU-initiated data transfers. The GPU now directly triggers data transfers without CPU coordination, unblocking the CPU to perform other tasks.
- The NVLink high-speed interconnect is now supported.
- Added new RPM packages that help automate the deployment of CUDA installations in large-scale cluster environments, using tools such as Puppet.
- Added absolute GPU numbering in NVML and NVIDIA-SMI that is based on the order of the GPUs on the PCI bus, so that the numbering matches the **/dev/nvidiaX** indices.
- Unified Memory is now supported with OS X 10.11 for Mac.

2.2. CUDA Tools

2.2.1. CUDA Compilers

- The CUDA compiler now supports Xcode 8.2.1.
- NVRTC is no longer considered a preview feature.
- Microsoft Visual Studio 2015 Update 3 (VC14) is now supported.
- Intel C++ Compilers 16.0 and 15.0.4 are now supported.
- POWER8 IBM XL compiler 13.1.3 is now supported.
- The Clang 3.7 and 3.8 LLVM-based C and C++ compilers are now supported as host compilers by **nvcc** on Linux operating systems.
- The **-std=c++11** option for **nvcc** is now supported when IBM xlC compiler version 13.1 (and above) is used as the host compiler.

- The **-std=c++11** option for **nvcc** is now supported when Intel ICC compiler 15 (and above) is used as the host compiler. Note that the CUDA extended lambda feature is not supported with the Intel ICC compiler.
- For debug compilations, the compiler now maintains the live ranges of variables that are in-scope according to C language rules. This feature allows users to inspect variables they expect to be live and reduces "value optimized out" errors during debugging.
- Improved 32-bit overflow checking when detecting common sub-expressions in array index operations.
- Improved the loop unroller with respect to nested loops. There are cases when the inner loop's trip count may depend on the outer loop's induction variable; after the outer loop is fully unrolled, the inner loop's trip count may become compile-time constants and thus become candidates for complete unrolling.
- Improved loop unrolling in the presence of **unroll pragma** information.
- The argument to the **unroll pragma** is now allowed to be any integral constant expression (as defined by the C++ standard). This includes integral template arguments and **constexpr** function calls (in modes where **constexpr** is enabled).
- The **nvcc** compiler defines the macros **__CUDACC_EXTENDED_LAMBDA__** and **__CUDACC_RELAXED_CONSTEXPR__** when the **--expt-extended-lambda** and **--expt-relaxed-constexpr** flags are specified, respectively.
- For the function **nvrtcCreateProgram()**, the type of the **headers** and **includeNames** parameters has been changed from **const char **** to **const char * const ***. For the function **nvrtcCompileProgram()**, the type of the **options** parameter has been changed from **const char **** to **const char * const ***. These changes are intended to facilitate easier use of the NVRTC API, such as using the pointer returned by the **std::initializer_list<const char *>::begin()** function.
- To reduce compile time, the compiler may remove unused **__device__** functions before generating PTX in whole-program compilation mode. The unused **__device__** functions are not removed when compiling in debug mode (**-G**) or in separate compilation mode. Note that a **__device__** function is considered unused if it has been fully inlined into its callers and has no other references.
- Within the body of a **__device__**, **__global__**, or **__device__ __host__** function, variables without any device memory qualifiers can be declared as static storage. They have the same restrictions as **__device__** variables defined in namespace scope.
- The **nvstd::function** implementation has been enhanced to allow **operator()** to be invoked from **__host__** and **__host__ __device__** functions as well as from **__device__** functions. The intent is to allow the **nvstd::function** to be usable in both host and device code. Please see the "C/C++ Language Support" section of the *CUDA Programming Guide* for more information.
- The compiler now supports instantiating **__global__** function templates with closure types of extended **__host__ __device__** lambdas defined in host code. This functionality is enabled when the **--expt-extended-lambda** flag is passed to **nvcc**. Please see the "C/C++ Language Support" section of the *CUDA Programming Guide* for more information.

- The compiler now provides the following type traits to detect closure types of extended **`__device__`** and extended **`__host__ __device__`** lambdas:

  - **`__nv_is_extended_device_lambda_closure_type(T)`**
  - **`__nv_is_extended_host_device_lambda_closure_type(T)`**

  Please see the "C/C++ Language Support" section of the *CUDA C Programming Guide* for more information.
- The NVRTC API has been enhanced for easier integration with templated host code. The following functions have been added:

  - **`nvrtcAddNameExpression()`** and **`nvrtcGetLoweredName()`**. This pair of functions can be used to extract the mangled (lowered) names of **`__global__`** functions and function template instantiations, given high-level source expressions denoting the addresses of the functions. This is useful in using the CUDA Driver API to look up the kernel functions in the generated PTX.
  - **`template <typename T> nvrtcResult nvrtcGetTypeName()`**. This function uses host platform mechanisms such as **`abi::__cxa_demangle()`** to extract the string name corresponding to the type argument **`T`**. The type name string can be incorporated into a **`__global__`** template instantiationin the NVRTC source string, thus allowing templated host code functions to create customized **`__global__`** template instantiations at run time.

  Please see the NVRTC documentation for details and runnable examples.
- The limit on the number of variables that can be captured by extended lambdas has been increased from 30 to 1023.
- The CUDA compiler now implements the C++17 **`*this`** capture specification for extended **`__device__`** lambdas and lambdas defined in device code. The support is enabled with the experimental **`--expt-extended-lambda`** flag in **`nvcc`**. Further information about extended lambdas can be found the *CUDA C Programming Guide*.
- The new **`nvcc`** flag **`--ftemplate-depth <limit>`** has been added to set the maximum instantiation depth for template classes to **`<limit>`**. This value is also passed to the host compiler if it provides an equivalent flag.

2.2.2. CUDA Profiler

- CUDA 8.0 provides CPU profiling to identify hot-spot regions in the code, along with call-graph and source-level drill-down.
- The dependency analysis feature enables optimization of the program runtime and the concurrency of applications using multiple CPU threads and CUDA streams. It allows computing the critical path of a specific execution, detecting waiting time, and inspecting dependencies among activities executing in different threads or streams.
- Enabled support for mixed precision (FP16) in the CUDA debugger and profiler.
- Enabled profiling of NVLink, including topology, bandwidth, and throughput.
- Visual Profiler and **`nvprof`** now support NVLink analysis for devices with compute capability 6.0.
- Visual Profiler and **`nvprof`** now support dependency analysis, which enables optimization of the program runtime and concurrency of applications utilizing

multiple CPU threads and CUDA streams. It allows computing the critical path of a specific execution, detecting waiting time, and inspecting dependencies between functions executing in different threads or streams.

- Visual Profiler and **`nvprof`** now support OpenACC profiling.
- Visual Profiler now supports CPU profiling.
- Unified Memory profiling now provides GPU page fault information on devices with compute capability 6.0 on supported platforms.
- Unified Memory profiling now provides CPU page fault information.
- Unified Memory profiling support is extended to the Mac OS platform.
- The Visual Profiler source-disassembly view now has a single integrated view for the different source level analysis results collected for a kernel instance, and results of different analysis steps can be viewed together.
- The PC sampling feature has been enhanced to point out the true latency issues for devices with compute capability 6.0 and higher.
- Support has been added for 16-bit floating point (FP16) data format profiling.
- If the new NVIDIA Tools Extension API(NVTX) feature of domains is used, then Visual Profiler and **`nvprof`** will show the NVTX markers and ranges grouped by domain. The Visual Profiler now adds a default file extension **`.nvvp`** if an extension is not specified when saving or opening a session file.

2.2.3. CUDA Profiling Tools Interface (CUPTI)

- Sampling of the program counter (PC) was enhanced to point out true latency issues: it indicates if the stall reasons for warps are actually causing stalls in the issue pipeline. Field **`latencySamples`** of the new activity record **`CUpti_ActivityPCSampling2`** provides true latency samples. This field is valid for devices with compute capability 6.0 and higher.
- Support for NVLink topology information—such as the pair of devices connected via NVLink, peak bandwidth, memory access permissions, and so on—is provided through the new activity record **`CUpti_ActivityNvLink`**. NVLink performance metrics for data transmitted and received, throughput for transmit and receive, and header overhead for each physical link is provided.
- CUPTI now supports profiling of OpenACC applications. OpenACC profiling information is provided in the form of the new activity records **`CUpti_ActivityOpenAccData`**, **`CUpti_ActivityOpenAccLaunch`**, and **`CUpti_ActivityOpenAccOther`**. This aids in correlating OpenACC constructs on the CPU with the corresponding activity taking place on the GPU and mapping it back to the source code. New routine **`cuptiOpenACCInitialize()`** is used to initialize profiling for supported OpenACC runtimes.
- Unified memory profiling now provides GPU page fault events on devices with compute capability 6.0 on supported platforms.
- Unified Memory profiling support is extended to the Mac OS platform.
- Support for 16-bit floating point (FP16) data format profiling was added. New metrics **`inst_fp_16`**, **`flop_count_hp_add`**, **`flop_count_hp_mul`**, **`flop_count_hp_fma`**, **`flop_count_hp`**, **`flop_hp_efficiency`**, and **`half_precision_fu_utilization`** are supported. Peak FP16 flops per cycle for

device can be queried using the enum **CUPTI_DEVICE_ATTR_FLOP_HP_PER_CYCLE** added to **CUpti_DeviceAttribute**.

- Added new activity kinds **CUPTI_ACTIVITY_KIND_SYNCHRONIZATION**, **CUPTI_ACTIVITY_KIND_STREAM**, and **CUPTI_ACTIVITY_KIND_CUDA_EVENT** to support the tracing of CUDA synchronization constructs, such as context, stream and CUDA event synchronization. Synchronization details are provided in the form of the new activity record **CUpti_ActivitySynchronization**. Enum **CUpti_ActivitySynchronizationType** lists different types of CUDA synchronization constructs.
- Added routines **cuptiSetThreadIdType()** and **cuptiGetThreadIdType()** to set and get the mechanism used to fetch the thread ID used in CUPTI records. Enum **CUpti_ActivityThreadIdType** lists all supported mechanisms.
- Added routine **cuptiComputeCapabilitySupported()** to check the support for a specific compute capability by the CUPTI.
- Added support to establish a correlation between an external API (such as OpenACC or OpenMP) and CUPTI API activity records. Routines **cuptiActivityPushExternalCorrelationId()** and **cuptiActivityPopExternalCorrelationId()** should be used to push and pop external correlation IDs for the calling thread. Generated records of type **CUpti_ActivityExternalCorrelation** contain both external and CUPTI assigned correlation IDs.
- Added containers to store information of events and metrics in the form of the activity records **CUpti_ActivityInstantaneousEvent**, **CUpti_ActivityInstantaneousEventInstance**, **CUpti_ActivityInstantaneousMetric**, and **CUpti_ActivityInstantaneousMetricInstance**. These activity records are not produced by the CUPTI; they are included for completeness and ease of use. Profilers built on top of CUPTI that sample events may choose to use these records to store the collected event data.
- Support was added for the domains and annotation of synchronization objects in NVTX v2. New activity record **CUpti_ActivityMarker2** and enums to indicate various stages of a synchronization object—**CUPTI_ACTIVITY_FLAG_MARKER_SYNC_ACQUIRE**, **CUPTI_ACTIVITY_FLAG_MARKER_SYNC_ACQUIRE_SUCCESS**, **CUPTI_ACTIVITY_FLAG_MARKER_SYNC_ACQUIRE_FAILED**, and **CUPTI_ACTIVITY_FLAG_MARKER_SYNC_RELEASE**—were added.
- Unused field **runtimeCorrelationId** of the activity record **CUpti_ActivityMemset** was broken into two fields, **flags** and **memoryKind**, to indicate the asynchronous behavior and the kind of the memory used for the memset operation. It is supported by the new flag **CUPTI_ACTIVITY_FLAG_MEMSET_ASYNC** added to the enum **CUpti_ActivityFlag**.
- Added flag **CUPTI_ACTIVITY_MEMORY_KIND_MANAGED** to the enum **CUpti_ActivityMemoryKind** to indicate managed memory.

2.3. CUDA Libraries

2.3.1. cuBLAS Library

cuBLAS 8.0.61.2 New Features

This update contains performance enhancements and bug-fixes to the cuBLAS library in CUDA Toolkit 8. Deep Learning applications based on Recurrent Neural Networks (RNNs) and Fully Connected Networks (FCNs) will benefit from new GEMM kernels and improved heuristics in this release.

This update supports the x86_64 architecture on Linux, Windows, and Mac OS operating systems, and the ppc64le architecture on Linux only.

The highlights of this update are as follows:

- Performance enhancements for GEMM matrices used in speech and natural language processing
- Integration of OpenAI GEMM kernels
- Improved GEMM heuristics to select optimized algorithms for given input sizes
- Heuristic fixes for batched GEMMs
- GEMM performance bug fixes for Pascal and Kepler platforms

CUDA 8.0.61 (CUDA 8.0 Production) New Features

- The cuBLAS library added a new function **`cublasGemmEx()`**, which is an extension of **`cublas<t/>gemm()`**. It allows the user to specify the algorithm, as well as the precision of the computation and of the input and output matrices. The function can be used to perform matrix-matrix multiplication at lower precision.
- The cuBLAS library now supports a Gaussian implementation for the GEMM, SYRK, and HERK operations on complex matrices.
- New routines for batched GEMMs, **`cublas<T>gemmStridedBatch()`**, have been added. These routines implement a new batch API for GEMMs that is easier to set up. The routines are optimized for performance on GPU architectures **`sm_5x`** or greater.
- The cublasXt API now accepts matrices that are resident in GPU memory.

2.3.2. cuFFT Library

- The cuFFT library now includes half-precision floating point datatype (FP16) FFT functions for transform sizes that are powers of two. For FFT size 2, real-to-complex and complex-to-real transforms in FP16 are currently not supported.
- Improvements were made to cuFFT to take advantage of multi-GPU configurations. The cuFFT library was also optimized for different NVLink topologies.

- In cuFFT 8.0, compatibility modes different from **CUFFT_COMPATIBILITY_FFT_PADDING** are no longer supported. Function **cufftSetCompatibilityMode()** no longer accepts the following values for the mode parameter: **CUFFT_COMPATIBILITY_NATIVE**, **CUFFT_COMPATIBILITY_FFTW_ALL**, and **CUFFT_COMPATIBILITY_FFT_ASYMMETRIC**. The error code **CUFFT_NOT_SUPPORTED** is returned in each case.
- The cuFFT library now includes multi-GPU support for up to 8 GPUs.

2.3.3. CUDA Math Library

- Support for half-precision floating point (FP16) has been added to the CUDA math library.
- CUDA 8.0 introduces a new built-in for **fp64 atomicAdd()**. Note that this built-in cannot be overridden with a custom function declared by the user (even if the code is not specifically being compiled for targets other than **sm_60**).

2.3.4. CUDA nvGRAPH Library

- CUDA 8.0 introduces nvGRAPH, a new library that is a collection of routines to process graph problems on GPUs. It includes implementations of the semi-ring SPMV, single source shortest path (SSSP), Widest Path, and PageRank algorithms. The nvGraph library will undergo some changes for the CUDA 8.0 final release:
    - The **naga.h** header will be renamed **nvgraph.h**.
    - Library names (***.so, *.a, *.dll**) will be changed from **libnaga*** to **libnvgraph***.
    - The signature of **nvgraphGetGraphStructure()** will be changed to

      ```
      nvgraphStatus_t NVGRAPH_API nvgraphGetGraphStructure (
          nvgraphHandle_t handle, nvgraphGraphDescr_t descrG,
          void* topologyData, nvgraphTopologyType_t* TType);
      ```

      Its functionality will be updated to return more information to the user.

2.4. CUDA Samples

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

Chapter 3. UNSUPPORTED FEATURES

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

General CUDA

- **cuFFT Compatibility Modes.** Compatibility modes different from **`CUFFT_COMPATIBILITY_FFT_PADDING`** are no longer supported.

CUDA Tools

- **Legacy Command Line Profiler.** The legacy Command Line Profiler was deprecated in CUDA 7.0 and is now removed in CUDA Toolkit 8.0. This notice only applies to the legacy Command Line Profiler controlled by the **`COMPUTE_PROFILE`** environment variable; there is no impact on the newer **`nvprof`** profiler, which is also controlled from the command line, or on the Visual Profiler.

Chapter 4. DEPRECATED FEATURES

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

General CUDA

- **Redundant Device Functions.** Five redundant device functions —**`syncthreads()`**, **`trap()`**, **`brkpt()`**, **`prof_trigger()`**, and **`threadfence()`**— have been deprecated because their respective functionalites are identical to **`__syncthreads()`**, **`__trap()`**, **`__brkpt()`**, **`__prof_trigger()`**, and **`__threadfence()`**.
- **Fermi Architecture Support.** Fermi architecture support is being deprecated in the CUDA 8.0 Toolkit, which will be the last toolkit release to support it. Future versions of the CUDA Toolkit will not support the architecture and are not guaranteed to work on that platform. Note that support for Fermi is being deprecated in the CUDA Toolkit but not in the driver. Applications compiled with CUDA 8.0 or older will continue to work on Fermi with newer NVIDIA drivers.
- **Windows Server 2008 R2 Support.** Support for Windows Server 2008 R2 is now deprecated and will be removed in a future version of the CUDA Toolkit.

CUDA Tools

- **Fermi Causes Compiler Warning.** The CUDA compiler driver (**`nvcc`**) now emits a warning when Fermi is chosen as a code generation target (**`compute_2x`** or **`sm_2x`**).
- **32-bit Linux CUDA Applications.** CUDA Toolkit support for 32-bit Linux CUDA applications has been dropped. Existing 32-bit applications will continue to work with the 64-bit driver, but support is deprecated.

Chapter 5. PERFORMANCE IMPROVEMENTS

5.1. CUDA Tools

5.1.1. CUDA Compilers

- In CUDA 8.0, some 64-bit integer divisions by zero are converted to 32-bit divisions to improve performance. A CUDA 8.0 result may now differ from a CUDA 7.5 one and is still undefined.
- The performance of single-precision square root (**`sqrt`**), single-precision reciprocal (**`rcp`**), and double-precision division (**`div`**) has nearly doubled.

5.2. CUDA Libraries

5.2.1. cuBLAS Library

- The cuBLAS library now supports high-performance SGEMM routines on Maxwell for handling problem sizes where **`m`** and **`n`** are not necessarily a multiple of the computation tile size. This leads to much smoother and more predictable performance.

5.2.2. CUDA Math Library

- Performance for more than 15 double-precision instructions was improved, with the most significant improvements in division, exponents, and logarithms. Performance and accuracy were improved for following single-precision functions: **`log1pf()`**, **`log2f()`**, **`logf()`**, **`acoshf()`**, **`asinhf()`**, and **`atanhf()`**.

Chapter 6.
RESOLVED ISSUES

6.1. General CUDA

- On some SLES or openSUSE system configurations, the NVIDIA GL library package may need to be locked before the steps for a GL-less installation are performed. The NVIDIA GL library package can be locked with this command:

  ```
  sudo zypper addlock nvidia-glG04
  ```
- On GP10x systems, applications that use **cudaMallocManaged()** and attempt to use **cuda-gdb** will incur random spurious MMU faults that will take down the application.
- Functions **cudaMallocHost()** and **cudaHostRegister()** don't work correctly on multi-GPU systems with the IOMMU enabled on Linux. The only workaround is to disable unified memory support with the **CUDA_DISABLE_UNIFIED_MEMORY=1** environment variable.
- Fixed an issue where **cuda-gdb** or **cuda-memcheck** would crash when used on an application that calls **cudaMemPrefetchAsync()**.
- Fixed a potential issue that can cause an application to hang when using **cudaMemPrefetchAsync()**.
- The device attributes **cudaDevAttrComputePreemptionSupported** and **cudaDevAttrCanUseHostPointerForRegisteredMem** do not have counterparts in **cudaDeviceProp**.

6.2. CUDA Tools

6.2.1. CUDA Compilers

- Fixed an issue with wrong code generation for computing the address of an array when using a 64-bit index.

- When a program is compiled with whole program optimization, applying launch bounds to recursive functions or to indirect function calls may have unpredictable results.
- The alignment of the built-in `long2` and `ulong2` types on 64-bit Linux and Mac OS X systems has been changed to 16 bytes (from 8 bytes). It now matches the alignment computed when compiling CUDA code with `nvcc` on these systems.
- When C++11 code (`-std=c++11`) is compiled on Linux with `gcc` as the host compiler, invoking `pow()` or `std::pow()` from device code with `(float, int)` or `(double, int)` arguments now compiles successfully.
- Because support for Fermi GPUs is deprecated in the CUDA 8.0 Toolkit, the CUDA compiler driver (`nvcc`) now emits a warning when Fermi is chosen as a code generation target (`compute_2x` or `sm_2x`).
- Dynamic Parallelism is supported with NVRTC when `compute >= 35` is specified as the compilation target. Generated PTX code that uses Dynamic Parallelism needs to be linked against the CUDA device runtime library (`cudadevrt`) before being loaded by the CUDA Driver API. The NVRTC documentation has a simple example that demonstrates generating PTX code, linking against the CUDA device runtime library, and executing the linked module.

6.2.2. CUDA Profiler

- The PC sampling warp state counts were incorrect in some cases.
- Profiling applications using `nvprof` or Visual Profiler on systems without an NVIDIA driver resulted in an error. This is now reported as a warning.

6.2.3. cuSOLVER Library

- Fixed an issue with the cuSOLVER library where some of its functions were not exposed, resulting in link errors

6.2.4. NVIDIA Tools Extension (NVTX)

- The NVIDIA Tools Extension SDK (NVTX) function `nvtxGetExportTable()` was missing from the export table list.

6.3. CUDA Libraries

6.3.1. cuBLAS Library

- Fixed GEMM performance issues on Kepler and Pascal for different matrix sizes, including small batches. Note that this fix is available **only** in the cuBLAS packages on the CUDA network repository.
- Updated the cuBLAS headers to use comments that are in compliance with ANSI C standards.
- Made optimizations for mixed-precision (`FP16`, `INT8`) matrix-matrix multiplication of matrices with a small number of columns `(n)`.

- Fixed an issue with the **`trsm()`** function for large-sized matrices.

Chapter 7.
KNOWN ISSUES

7.1. General CUDA

- Function **`cuDeviceGetP2PAttribute()`** was not published in the cuda library (**`libcuda.so`**). Until a new build of the toolkit is issued, users can either use the driver version, **`cudaDeviceGetP2PAttribute()`**, or perform the link to use **`libcuda`** directly instead of the stub (usually it can be done by adding **`-L/usr/lib64`**).
- Installation of CUDA 8.0 on an Ubuntu 14.04.4 system requires a system reboot to avoid an **`nvidia-nvlink`** error and system sluggishness.
- Enabling per-thread synchronization behavior [http://docs.nvidia.com/cuda/cuda-runtime-api/stream-sync-behavior.html#stream-sync-behavior__per-thread-default-stream] does not work correctly with the following CUDA runtime routines, which use the legacy synchronization behavior [http://docs.nvidia.com/cuda/cuda-runtime-api/stream-sync-behavior.html#stream-sync-behavior__legacy-default-stream]:

  - **`cudaMemcpyPeer()`**
  - **`cudaMemcpyPeerAsync()`** with a **`NULL`** stream argument
  - **`cudaGraphicsMapResources()`** with a **`NULL`** stream argument
  - **`cudaGraphicsUnmapResources()`** with a **`NULL`** stream argument

  To work around this issue, use **`cudaMemcpyPeerAsync()`** with the **`cudaPerThreadStream`** stream argument instead of **`cudaMemcpyPeer()`**, and pass the **`cudaPerThreadStream`** argument instead of **`NULL`** to the other routines.
- If the Windows toolkit installation fails, it may be because Visual Studio, **`Nvda.Launcher.exe`**, **`Nsight.Monitor.exe`**, or **`Nvda.CrashReporter.exe`** is running. Make sure these programs are closed and try to install again.
- Peer access is disabled between two devices if either of them is in SLI mode.
- Unified memory is not currently supported with IOMMU. The workaround is to disable IOMMU in the BIOS. Please refer to the vendor documentation for the steps to disable it in the BIOS.

7.2. CUDA Tools

7.2.1. CUDA Compiler

- **`Extended __device__`** and **`__host__ __device__`** lambdas are enabled by the experimental compiler flag **`--expt-extended-lambda`**. When the closure type of such a lambda is used in the creating the mangled name for an entity (for example, a function), the compiler generated mangled name does not conform to the IA64 ABI. As a result, such names cannot be demangled using tools like c++filt. This issue will be fixed in a future CUDA Toolkit release.

7.2.2. CUDA Profiler

- When a device is in the "exclusive" process compute mode, the profiler may fail to collect events or metrics in "application replay" mode. In this case, use "kernel replay" mode.
- The timestamp and duration for some memory copies on some devices with compute capability 6.1 are incorrect. This can result in errors, and dependency analysis results may not be available. This issue can impact **`nvprof`**, **`nvvp`**, and **`cupti`**.
- In the Visual Profiler, the NVLink Analysis diagram may be incorrect after the diagram is scrolled. This can be corrected by horizontally resizing the diagram panel.
- In the Visual Profiler, the **Run->Configure Metrics and Events...** dialog does not work for the device that has NVLink support. It's suggested to collect all metrics and events using **`nvprof`** and then import into **`nvvp`**.
- Some devices with compute capability 6.1 don't support multi-context scope collection for metrics. This issue affects **`nvprof`**, Visual Profiler, and CUPTI.

7.2.3. CUDA Profiling Tools Interface (CUPTI)

- Some devices with compute capability 6.1 don't support multi-context scope collection for metrics. This issue affects **`nvprof`**, Visual Profiler, and CUPTI.

Acknowledgments

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

NVIDIA acknowledges Scott Gray for his work on small-tile GEMM kernels for Pascal. These kernels were originally developed for OpenAI and included in cuBLAS 8.0.61.2.

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

www.nvidia.com