

---

---

NVIDIA CUDA  
Linux Release Notes  
Version 2.0

---

---

---

New Features

---

Hardware Support

o Additional hardware support:

- GeForce GTX 280
- GeForce GTX 260
- GeForce 9800 GX2
- GeForce 9800 GTX
- GeForce 9600 GT
- GeForce 8800 GS
- GeForce 8600 GTS
- Quadro FX 3700
- Quadro NVS 130M
- Quadro NVS 135M
- Quadro NVS 140M
- Quadro NVS 140M
- Quadro NVS 135M
- Quadro NVS 130M
- Quadro FX 3600M

Platform Support

o Additional OS support

- Red Hat Enterprise Linux 4.6
- Red Hat Enterprise Linux 5.1
- SUSE Linux 10.3
- Fedora 8
- Ubuntu 7.10

New ISA Support SM 1.2

o Support for any() and all() intrinsics

o Support for atomic operations on shared memory

o Support for 64-bit atomic operations on global memory

- atomicAdd() (unsigned 64-bit int)
- atomicExch() (unsigned 64-bit int)
- atomicCAS() (unsigned 64-bit int)

Double Precision Computing with SM 1.3

o Compiler support for double-precision math

- Datatype 'double' compiles to native FP64 types when using SM 1.3

Note that math functions in the CUDA math library are overloaded.

In general, there are three prototypes for each math function:

- (1) double <func-name>(double), e.g. double log(double)
- (2) float <func-name>(float), e.g. float log(float)
- (3) float <func-name>f(float), e.g. float logf(float)

In particular, note that passing a float argument always results in a float result [variants (2) and (3) above].

o CUBLAS Library Support

- Added the BLAS1 functions:

- \* cublasIdamax()
- \* cublasIdamin()
- \* cublasDasum()
- \* cublasDaxpy()
- \* cublasDcopy()
- \* cublasDdot()
- \* cublasDnrm2()
- \* cublasDrot()
- \* cublasDrotg()

- \* cublasDrotm()
- \* cublasDrotmg()
- \* cublasDscal()
- \* cublasDswap()
- Added the BLAS2 functions:
  - \* cublasDgemv()
  - \* cublasDger()
  - \* cublasDsyr()
  - \* cublasDtrsv()
- Added the BLAS3 functions:
  - \* cublasDgemm()
  - \* cublasDsymm()
  - \* cublasDsyrk()
  - \* cublasDsyr2k()
  - \* cublasDtrmm()
  - \* cublasDtrsm()
  - \* cublasZgemm()

#### API Features

- o 3D texture API
  - cudaMalloc3D
  - cudaMalloc3DArray
  - cudaMemset3D
  - cu(da)Memcpy3D
  - cu(da)Memcpy3DAsync
  - cuArray3DCreate
  - cuArray3DGetDescriptor
  - CUDA\_MEMCPY3D and CUDA\_ARRAY3D\_DESCRIPTOR structures
- o Improved OpenGL interoperability API
  - cuGLCtxCreate
- o Context migration API
  - cuCtxAttach
  - cuCtxDestroy
  - cuCtxDetach
  - cuCtxSynchronize
  - cuCtxPushCurrent
  - cuCtxPopCurrent
- o Async constant memory update
  - cudaMemcpyToSymbolAsync
- o Improved device attribute query
  - cuDeviceGetAttribute

#### Performance Enhancements

- o Improved device->array memcpy performance

---

#### Known Issues

---

- o 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 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 run time restriction. For this reason it is recommended that CUDA is run on a GPU that is NOT attached to an X display.
- o While X does not need to be running in order to use CUDA, X must have been initialized at least once after booting in order to properly load the NVIDIA kernel module. The NVIDIA kernel module remains loaded even after X shuts down, allowing CUDA to continue to function.
- o 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.

- o 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.
- o The default compilation mode for host code is now C++. To restore the old behavior, use the option `--host-compilation=c`

---

#### Open64 Sources

---

The Open64 source files are controlled under terms of the GPL license. Current and previously released versions are located via anonymous ftp at [download.nvidia.com](http://download.nvidia.com) in the `CUDAOpen64` directory.

---

#### Revision History

---

06/2008 - Version 2.0  
05/2008 - Version 2.0 Beta  
11/2007 - Version 1.1  
06/2007 - Version 1.0  
06/2007 - Version 0.9  
02/2007 - Version 0.8 - Initial public Beta

---

#### More Information

---

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