NVIDIA CUDA Compute Unified Device Architecture

Reference Manual

Version 2.0

June 2008

Contents

1 RuntimeApiReference **1**

1.1 DeviceManagement RT . . . 2

1.1.1 cudaGetDeviceCount . . . 3

1.1.2 cudaSetDevice . . . 4

1.1.3 cudaGetDevice . . . 5

1.1.4 cudaGetDeviceProperties . . . 6

1.1.5 cudaChooseDevice . . . 8

1.2 ThreadManagement RT . . . 9

1.2.1 cudaThreadSynchronize . . . 10

1.2.2 cudaThreadExit . . . 11

1.3 StreamManagement RT . . . 12

1.3.1 cudaStreamCreate . . . 13

1.3.2 cudaStreamQuery . . . 14

1.3.3 cudaStreamSynchronize . . . 15

1.3.4 cudaStreamDestroy . . . 16

1.4 EventManagement RT . . . 17

1.4.1 cudaEventCreate . . . 18

1.4.2 cudaEventRecord . . . 19

1.4.3 cudaEventQuery . . . 20

1.4.4 cudaEventSynchronize . . . 21

1.4.5 cudaEventDestroy . . . 22

1.4.6 cudaEventElapsedTime . . . 23

1.5 MemoryManagement RT . . . 24

1.5.1 cudaMalloc . . . 25

1.5.2 cudaMallocPitch . . . 26

1.5.3 cudaFree . . . 27

1.5.4 cudaMallocArray . . . 28

1.5.5 cudaFreeArray . . . 29

1.5.6 cudaMallocHost . . . 30

1.5.7 cudaFreeHost . . . 31

1.5.8 cudaMemset . . . 32

1.5.9 cudaMemset2D . . . 33

1.5.10 cudaMemcpy . . . 34
1.5.11 cudaMemcpy2D . . . 35
1.5.12 cudaMemcpyToArray . . . 36
1.5.13 cudaMemcpy2DToArray . . . 37
1.5.14 cudaMemcpyFromArray . . . 38
1.5.15 cudaMemcpy2DFromArray . . . 39
1.5.16 cudaMemcpyArrayToArray . . . 40
1.5.17 cudaMemcpy2DArrayToArray . . . 41
1.5.18 cudaMemcpyToSymbol . . . 42
1.5.19 cudaMemcpyFromSymbol . . . 43
1.5.20 cudaGetSymbolAddress . . . 44
1.5.21 cudaGetSymbolSize . . . 45
1.5.22 cudaMalloc3D . . . 46
1.5.23 cudaMalloc3DArray . . . 48
1.5.24 cudaMemset3D . . . 50
1.5.25 cudaMemcpy3D . . . 52
1.6 TextureReferenceManagement RT . . . 54
1.6.1 LowLevelApi . . . 55
1.6.2 HighLevelApi . . . 63
1.7 ExecutionControl RT . . . 68
1.7.1 cudaConfigureCall . . . 69
1.7.2 cudaLaunch . . . 70
1.7.3 cudaSetupArgument . . . 71
1.8 OpenGlInteroperability RT . . . 72
1.8.1 cudaGLSetGLDevice . . . 73
1.8.2 cudaGLRegisterBufferObject . . . 74
1.8.3 cudaGLMapBufferObject . . . 75
1.8.4 cudaGLUnmapBufferObject . . . 76
1.8.5 cudaGLUnregisterBufferObject . . . 77
1.9 Direct3dInteroperability RT . . . 78
1.9.1 cudaD3D9GetDevice . . . 79
1.9.2 cudaD3D9SetDirect3DDevice . . . 80
1.9.3 cudaD3D9GetDirect3DDevice . . . 81
1.9.4 cudaD3D9RegisterResource . . . 82

1.9.5 cudaD3D9UnregisterResource . . . 84
1.9.6 cudaD3D9MapResources . . . 85
1.9.7 cudaD3D9UnmapResources . . . 86
1.9.8 cudaD3D9ResourceSetMapFlags . . . 87
1.9.9 cudaD3D9ResourceGetSurfaceDimensions . . . 89
1.9.10 cudaD3D9ResourceGetMappedPointer . . . 90
1.9.11 cudaD3D9ResourceGetMappedSize . . . 91
1.9.12 cudaD3D9ResourceGetMappedPitch . . . 92
1.10 ErrorHandling RT . . . 94
1.10.1 cudaGetLastError . . . 95
1.10.2 cudaGetErrorString . . . 97

2 DriverApiReference **98**
2.1 Initialization . . . 99
2.1.1 cuInit . . . 100
2.2 DeviceManagement . . . 101
2.2.1 cuDeviceComputeCapability . . . 102
2.2.2 cuDeviceGet . . . 103
2.2.3 cuDeviceGetAttribute . . . 104
2.2.4 cuDeviceGetCount . . . 106
2.2.5 cuDeviceGetName . . . 107
2.2.6 cuDeviceGetProperties . . . 108
2.2.7 cuDeviceTotalMem . . . 110
2.3 ContextManagement . . . 111
2.3.1 cuCtxAttach . . . 112
2.3.2 cuCtxCreate . . . 113
2.3.3 cuCtxDestroy . . . 115
2.3.4 cuCtxDetach . . . 116
2.3.5 cuCtxGetDevice . . . 117
2.3.6 cuCtxPopCurrent . . . 118
2.3.7 cuCtxPushCurrent . . . 119
2.3.8 cuCtxSynchronize . . . 120
2.4 ModuleManagement . . . 121
2.4.1 cuModuleGetFunction . . . 122

2.4.2 cuModuleGetGlobal . . . 123
2.4.3 cuModuleGetTexRef . . . 124
2.4.4 cuModuleLoad . . . 125
2.4.5 cuModuleLoadData . . . 126
2.4.6 cuModuleLoadFatBinary . . . 127
2.4.7 cuModuleUnload . . . 128
2.5 StreamManagement . . . 129
2.5.1 cuStreamCreate . . . 130
2.5.2 cuStreamDestroy . . . 131
2.5.3 cuStreamQuery . . . 132
2.5.4 cuStreamSynchronize . . . 133
2.6 EventManagement . . . 134
2.6.1 cuEventCreate . . . 135
2.6.2 cuEventDestroy . . . 136
2.6.3 cuEventElapsedTime . . . 137
2.6.4 cuEventQuery . . . 138
2.6.5 cuEventRecord . . . 139
2.6.6 cuEventSynchronize . . . 140
2.7 ExecutionControl . . . 141
2.7.1 cuLaunch . . . 142
2.7.2 cuLaunchGrid . . . 143
2.7.3 cuParamSetSize . . . 144
2.7.4 cuParamSetTexRef . . . 145
2.7.5 cuParamSetf . . . 146
2.7.6 cuParamSeti . . . 147
2.7.7 cuParamSetv . . . 148
2.7.8 cuFuncSetBlockShape . . . 149
2.7.9 cuFuncSetSharedSize . . . 150
2.8 MemoryManagement . . . 151
2.8.1 cuArrayCreate . . . 152
2.8.2 cuArray3DCreate . . . 154
2.8.3 cuArrayDestroy . . . 156
2.8.4 cuArrayGetDescriptor . . . 157
2.8.5 cuArray3DGetDescriptor . . . 158

2.8.6 cuMemAlloc . . . 159
2.8.7 cuMemAllocHost . . . 160
2.8.8 cuMemAllocPitch . . . 161
2.8.9 cuMemFree . . . 163
2.8.10 cuMemFreeHost . . . 164
2.8.11 cuMemGetAddressRange . . . 165
2.8.12 cuMemGetInfo . . . 166
2.8.13 cuMemcpy2D . . . 167
2.8.14 cuMemcpy3D . . . 170
2.8.15 cuMemcpyAtoA . . . 173
2.8.16 cuMemcpyAtoD . . . 174
2.8.17 cuMemcpyAtoH . . . 175
2.8.18 cuMemcpyDtoA . . . 176
2.8.19 cuMemcpyDtoD . . . 177
2.8.20 cuMemcpyDtoH . . . 178
2.8.21 cuMemcpyHtoA . . . 179
2.8.22 cuMemcpyHtoD . . . 180
2.8.23 cuMemset . . . 181
2.8.24 cuMemset2D . . . 182
2.9 TextureReferenceManagement . . . 183
2.9.1 cuTexRefCreate . . . 184
2.9.2 cuTexRefDestroy . . . 185
2.9.3 cuTexRefGetAddress . . . 186
2.9.4 cuTexRefGetAddressMode . . . 187
2.9.5 cuTexRefGetArray . . . 188
2.9.6 cuTexRefGetFilterMode . . . 189
2.9.7 cuTexRefGetFlags . . . 190
2.9.8 cuTexRefGetFormat . . . 191
2.9.9 cuTexRefSetAddress . . . 192
2.9.10 cuTexRefSetAddressMode . . . 193
2.9.11 cuTexRefSetArray . . . 194
2.9.12 cuTexRefSetFilterMode . . . 195
2.9.13 cuTexRefSetFlags . . . 196
2.9.14 cuTexRefSetFormat . . . 197

2.10 OpenGlInteroperability . . . 198
2.10.1 cuGLCtxCreate . . . 199
2.10.2 cuGLInit . . . 200
2.10.3 cuGLMapBufferObject . . . 201
2.10.4 cuGLRegisterBufferObject . . . 202
2.10.5 cuGLUnmapBufferObject . . . 203
2.10.6 cuGLUnregisterBufferObject . . . 204
2.11 Direct3dInteroperability . . . 205
2.11.1 cuD3D9GetDevice . . . 206
2.11.2 cuD3D9CtxCreate . . . 207
2.11.3 cuD3D9GetDirect3DDevice . . . 208
2.11.4 cuD3D9RegisterResource . . . 209
2.11.5 cuD3D9UnregisterResource . . . 211
2.11.6 cuD3D9MapResources . . . 212
2.11.7 cuD3D9UnmapResources . . . 213
2.11.8 cuD3D9ResourceSetMapFlags . . . 214
2.11.9 cuD3D9ResourceGetSurfaceDimensions . . . 216
2.11.10 cuD3D9ResourceGetMappedPointer . . . 217
2.11.11 cuD3D9ResourceGetMappedSize . . . 219
2.11.12 cuD3D9ResourceGetMappedPitch . . . 220

3 AtomicFunctions 222

3.1 ArithmeticFunctions . . . 223
3.1.1 atomicAdd . . . 224
3.1.2 atomicSub . . . 225
3.1.3 atomicExch . . . 226
3.1.4 atomicMin . . . 227
3.1.5 atomicMax . . . 228
3.1.6 atomicInc . . . 229
3.1.7 atomicDec . . . 230
3.1.8 atomicCAS . . . 231
3.2 BitwiseFunctions . . . 232
3.2.1 atomicAnd . . . 233
3.2.2 atomicOr . . . 234

3.2.3 atomicXor 235

1 RuntimeApiReference

NAME

Runtime API Reference

DESCRIPTION

There are two levels for the runtime API.

The low-level API (`cuda_runtime_api.h`) is a C-style interface that does not require compiling with nvcc.

The high-level API (`cuda_runtime.h`) is a C++-style interface built on top of the low-level API. It wraps some of the low level API routines, using overloading, references and default arguments. These wrappers can be used from C++ code and can be compiled with any C++ compiler. The high-level API also has some CUDA-specific wrappers that wrap low-level routines that deal with symbols, textures, and device functions. These wrappers require the use of nvcc because they depend on code being generated by the compiler. For example, the execution configuration syntax to invoke kernels is only available in source code compiled with `nvcc`.

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.1 DeviceManagement RT

NAME

Device Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaGetDeviceCount

cudaSetDevice

cudaGetDevice

cudaGetDeviceProperties

cudaChooseDevice

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.1.1 cudaGetDeviceCount

NAME

cudaGetDeviceCount - returns the number of compute-capable devices

SYNOPSIS

```
cudaError_t cudaGetDeviceCount( int* count )
```

DESCRIPTION

Returns in `*count` the number of devices with compute capability greater or equal to 1.0 that are available for execution. If there is no such device, `cudaGetDeviceCount()` returns 1 and device 0 only supports device emulation mode. Since this device will be able to emulate all hardware features, this device will report major and minor compute capability versions of 9999.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetDevice, *cudaSetDevice*, *cudaGetDeviceProperties*, *cudaChooseDevice*

1.1.2 cudaSetDevice

NAME

cudaSetDevice - sets device to be used for GPU executions

SYNOPSIS

```
cudaError_t cudaSetDevice( int dev )
```

DESCRIPTION

Records `dev` as the device on which the active host thread executes the device code.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDevice

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetDeviceCount, *cudaGetDevice*, *cudaGetDeviceProperties*, *cudaChooseDevice*

1.1.3 cudaGetDevice

NAME

cudaGetDevice - returns which device is currently being used

SYNOPSIS

```
cudaError_t cudaGetDevice( int* dev )
```

DESCRIPTION

Returns in `*dev` the device on which the active host thread executes the device code.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetDeviceCount, *cudaSetDevice*, *cudaGetDeviceProperties*, *cudaChooseDevice*

1.1.4 cudaGetDeviceProperties

NAME

cudaGetDeviceProperties - returns information on the compute-device

SYNOPSIS

```
cudaError_t cudaGetDeviceProperties( struct cudaDeviceProp* prop, int dev )
```

DESCRIPTION

Returns in ***prop** the properties of device **dev**. The **cudaDeviceProp** structure is defined as:

```
 struct cudaDeviceProp {
   char name[256];
   size_t totalGlobalMem;
   size_t sharedMemPerBlock;
   int regsPerBlock;
   int warpSize;
   size_t memPitch;
   int maxThreadsPerBlock;
   int maxThreadsDim[3];
   int maxGridSize[3];
   size_t totalConstMem;
   int major;
   int minor;
   int clockRate;
   size_t textureAlignment;
   int deviceOverlap;
   int multiProcessorCount;
 }
```

where:

name

is an ASCII string identifying the device;

totalGlobalMem

is the total amount of global memory available on the device in bytes;

sharedMemPerBlock

is the maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;

regsPerBlock

is the maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resident on a multiprocessor;

warpSize

is the warp size in threads;

memPitch

is the maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through **cudaMallocPitch()**;

maxThreadsPerBlock

is the maximum number of threads per block;

maxThreadsDim[3]

is the maximum sizes of each dimension of a block;

maxGridSize[3]

is the maximum sizes of each dimension of a grid;

totalConstMem

is the total amount of constant memory available on the device in bytes;

major, minor

are the major and minor revision numbers defining the device's compute capability;

clockRate

is the clock frequency in kilohertz;

textureAlignment

is the alignment requirement; texture base addresses that are aligned to **textureAlignment** bytes do not need an offset applied to texture fetches;

deviceOverlap

is 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not;

multiProcessorCount

is the number of multiprocessors on the device.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDevice

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetDeviceCount, *cudaGetDevice*, *cudaSetDevice*, *cudaChooseDevice*

1.1.5 cudaChooseDevice

NAME

cudaChooseDevice - select compute-device which best matches criteria

SYNOPSIS

```
cudaError_t cudaChooseDevice( int* dev, const struct cudaDeviceProp* prop )
```

DESCRIPTION

Returns in ***dev** the device which properties best match ***prop**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetDeviceCount, *cudaGetDevice*, *cudaSetDevice*, *cudaGetDeviceProperties*

1.2 ThreadManagement RT

NAME

Thread Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaThreadSynchronize

cudaThreadExit

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.2.1 cudaThreadSynchronize

NAME

cudaThreadSynchronize - wait for compute-device to finish

SYNOPSIS

```
cudaError_t cudaThreadSynchronize(void)
```

DESCRIPTION

Blocks until the device has completed all preceding requested tasks. **cudaThreadSynchronize()** returns an error if one of the preceding tasks failed.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaThreadExit

1.2.2 cudaThreadExit

NAME

cudaThreadExit - exit and clean-up from CUDA launches

SYNOPSIS

```
cudaError_t cudaThreadExit(void)
```

DESCRIPTION

Explicitly cleans up all runtime-related resources associated with the calling host thread. Any subsequent API call reinitializes the runtime. **cudaThreadExit()** is implicitly called on host thread exit.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaThreadSynchronize

1.3 StreamManagement RT

NAME

Stream Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaStreamCreate

cudaStreamQuery

cudaStreamSynchronize

cudaStreamDestroy

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.3.1 cudaStreamCreate

NAME

cudaStreamCreate - create an async stream

SYNOPSIS

```
cudaError_t cudaStreamCreate( cudaStream_t* stream )
```

DESCRIPTION

Creates a stream.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaStreamQuery, *cudaStreamSynchronize*, *cudaStreamDestroy*

1.3.2 cudaStreamQuery

NAME

cudaStreamQuery - queries a stream for completion-status

SYNOPSIS

```
cudaError_t cudaStreamQuery(cudaStream_t stream)
```

DESCRIPTION

Returns **cudaSuccess** if all operations in the stream have completed, or **cudaErrorNotReady** if not.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorNotReady

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaStreamCreate, *cudaStreamDestroy*, *cudaStreamSynchronize*

1.3.3 cudaStreamSynchronize

NAME

cudaStreamSynchronize - waits for stream tasks to complete

SYNOPSIS

```
cudaError_t cudaStreamSynchronize( cudaStream_t stream )
```

DESCRIPTION

Blocks until the device has completed all operations in the stream.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaStreamCreate, *cudaStreamDestroy*, *cudaStreamQuery*

1.3.4 cudaStreamDestroy

NAME

cudaStreamDestroy - destroys and cleans-up a stream object

SYNOPSIS

```
cudaError_t cudaStreamDestroy( cudaStream_t stream )
```

DESCRIPTION

Destroys a stream object.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaStreamCreate, *cudaStreamSychronize*, *cudaStreamDestroy*

1.4 EventManagement RT

NAME

Event Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaEventCreate

cudaEventRecord

cudaEventQuery

cudaEventSynchronize

cudaEventDestroy

cudaEventElapsedTime

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.4.1 cudaEventCreate

NAME

cudaEventCreate - creates an event-object

SYNOPSIS

```
cudaError_t cudaEventCreate( cudaEvent_t* event )
```

DESCRIPTION

Creates an event object.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidValue

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventRecord, *cudaEventQuery*, *cudaEventSynchronzie*, *cudaEventDestroy*, *cudaEventElapsedTime*

1.4.2 cudaEventRecord

NAME

cudaEventRecord - records an event

SYNOPSIS

```
cudaError_t cudaEventRecord( cudaEvent_t event, CUstream stream )
```

DESCRIPTION

Records an event. If **stream** is non-zero, the event is recorded after all preceding operations in the stream have been completed; otherwise, it is recorded after all preceding operations in the CUDA context have been completed. Since this operation is asynchronous, **cudaEventQuery()** and/or **cudaEventSynchronize()** must be used to determine when the event has actually been recorded.

If **cudaEventRecord()** has previously been called and the event has not been recorded yet, this function returns **cudaErrorInvalidValue**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventCreate, *cudaEventQuery*, *cudaEventSynchronize*, *cudaEventDestroy*, *cudaEventElapsedTime*

1.4.3 cudaEventQuery

NAME

cudaEventQuery - query if an event has been recorded

SYNOPSIS

```
cudaError_t cudaEventQuery( cudaEvent_t event )
```

DESCRIPTION

Returns **cudaSuccess** if the event has actually been recorded, or **cudaErrorNotReady** if not. If **cudaEventRecord()** has not been called on this event, the function returns **cudaErrorInvalidValue**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorNotReady

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidValue

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventCreate, *cudaEventRecord*, *cudaEventSynchronize*, *cudaEventDestroy*, *cudaEventElapsedTime*

1.4.4 cudaEventSynchronize

NAME

cudaEventSynchronize - wait for an event to be recorded

SYNOPSIS

```
cudaError_t cudaEventSynchronize( cudaEvent_t event )
```

DESCRIPTION

Blocks until the event has actually been recorded. If **cudaEventRecord()** has not been called on this event, the function returns **cudaErrorInvalidValue**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidValue

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventCreate, *cudaEventRecord*, *cudaEventQuery*, *cudaEventDestroy*, *cudaEventElapsedTime*

1.4.5 cudaEventDestroy

NAME

cudaEventDestroy - destroys an event-object

SYNOPSIS

```
cudaError_t cudaEventDestroy( cudaEvent_t event )
```

DESCRIPTION

Destroys the event-object.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidValue

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventCreate, *cudaEventQuery*, *cudaEventSynchronize*, *cudaEventRecord*, *cudaEventElapsedTime*

1.4.6 cudaEventElapsedTime

NAME

cudaEventElapsedTime - computes the elapsed time between events

SYNOPSIS

```
cudaError_t cudaEventElapsedTime( float* time, cudaEvent_t start, cudaEvent_t end );
```

DESCRIPTION

Computes the elapsed time between two events (in milliseconds with a resolution of around 0.5 microseconds). If either event has not been recorded yet, this function returns **cudaErrorInvalidValue**. If either event has been recorded with a non-zero stream, the result is undefined.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInitializationError

cudaErrorPriorLaunchFailure

cudaErrorInvalidValue

cudaErrorInvalidResourceHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaEventCreate, *cudaEventQuery*, *cudaEventSynchronize*, *cudaEventDestroy*, *cudaEventRecord*

1.5 MemoryManagement RT

NAME

Memory Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaMalloc

cudaMallocPitch

cudaFree

cudaMallocArray

cudaFreeArray

cudaMallocHost

cudaFreeHost

cudaMemset

cudaMemset2D

cudaMemcpy

cudaMemcpy2D

cudaMemcpyToArray

cudaMemcpy2DToArray

cudaMemcpyFromArray

cudaMemcpy2DFromArray

cudaMemcpyArrayToArray

cudaMemcpy2DArrayToArray

cudaMemcpyToSymbol

cudaMemcpyFromSymbol

cudaGetSymbolAddress

cudaGetSymbolSize

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.5.1 cudaMalloc

NAME

cudaMalloc - allocate memory on the GPU

SYNOPSIS

```
cudaError_t cudaMalloc( void** devPtr, size_t count )
```

DESCRIPTION

Allocates **count** bytes of linear memory on the device and returns in ***devPtr** a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. **cudaMalloc()** returns **cudaErrorMemoryAllocation** in case of failure.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMallocPitch, *cudaFree*, *cudaMallocArray*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.2 cudaMallocPitch

NAME

cudaMallocPitch - allocates memory on the GPU

SYNOPSIS

```
cudaError_t cudaMallocPitch( void** devPtr, size_t* pitch, size_t widthInBytes, size_t height
)
```

DESCRIPTION

Allocates at least **widthInBytes*height** bytes of linear memory on the device and returns in ***devPtr** a pointer to the allocated memory. The function may pad the allocation to ensure that corresponding pointers in any given row will continue to meet the alignment requirements for coalescing as the address is updated from row to row. The pitch returned in ***pitch** by **cudaMallocPitch()** is the width in bytes of the allocation. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type T, the address is computed as

```
T* pElement = (T*)((char*)BaseAddress + Row * pitch) + Column;
```

For allocations of 2D arrays, it is recommended that programmers consider performing pitch allocations using **cudaMallocPitch()**. Due to pitch alignment restrictions in the hardware, this is especially true if the application will be performing 2D memory copies between different regions of device memory (whether linear memory or CUDA arrays).

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaFree*, *cudaMallocArray*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.3 cudaFree

NAME

cudaFree - frees memory on the GPU

SYNOPSIS

```
cudaError_t cudaFree(void* devPtr)
```

DESCRIPTION

Frees the memory space pointed to by **devPtr**, which must have been returned by a previous call to **cudaMalloc()** or **cudaMallocPitch()**. Otherwise, or if **cudaFree(devPtr)** has already been called before, an error is returned. If **devPtr** is 0, no operation is performed. **cudaFree()** returns **cudaErrorInvalidDevicePointer** in case of failure.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDevicePointer

cudaErrorInitializationError

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaMallocPitch*, *cudaMallocArray*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.4 cudaMallocArray

NAME

cudaMallocArray - allocate an array on the GPU

SYNOPSIS

```
cudaError_t cudaMallocArray( struct cudaArray** array, const struct cudaChannelFormatDesc*
desc, size_t width, size_t height )
```

DESCRIPTION

Allocates a CUDA array according to the **cudaChannelFormatDesc** structure **desc** and returns a handle to the new CUDA array in ***array**. The **cudaChannelFormatDesc** is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where **cudaChannelFormatKind** is one of **cudaChannelFormatKindSigned**, **cudaChannelFormatKindUnsigned**, **cudaChannelFormatKindFloat**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaMallocPitch*, *cudaFree*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.5 cudaFreeArray

NAME

cudaFreeArray - frees an array on the GPU

SYNOPSIS

```
cudaError_t cudaFreeArray( struct cudaArray* array )
```

DESCRIPTION

Frees the CUDA array **array**. If **array** is 0, no operation is performed.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaMallocPitch*, *cudaFree*, *cudaMallocArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.6 cudaMallocHost

NAME

cudaMallocHost - allocates page-locked memory on the host

SYNOPSIS

```
cudaError_t cudaMallocHost( void** hostPtr, size_t size )
```

DESCRIPTION

Allocates **size** bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as **cudaMemcpy*()**. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as **malloc()**. Allocating excessive amounts of memory with **cudaMallocHost()** may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaMallocPitch*, *cudaFree*, *cudaMallocArray*, *cudaFreeArray*, *cudaFreeHost*

1.5.7 cudaFreeHost

NAME

cudaFreeHost - frees page-locked memory

SYNOPSIS

```
cudaError_t cudaFreeHost( void* hostPtr )
```

DESCRIPTION

Frees the memory space pointed to by **hostPtr**, which must have been returned by a previous call to **cudaMallocHost()**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc, *cudaMallocPitch*, *cudaFree*, *cudaMallocArray*, *cudaFreeArray*, *cudaMallocHost*

1.5.8 cudaMemset

NAME

cudaMemset - initializes or sets GPU memory to a value

SYNOPSIS

```
cudaError_t cudaMemset( void* devPtr, int value, size_t count )
```

DESCRIPTION

Fills the first **count** bytes of the memory area pointed to by **devPtr** with the constant byte value **value**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemset2D, *cudaMemset3D*

1.5.9 cudaMemset2D

NAME

cudaMemset2D - initializes or sets GPU memory to a value

SYNOPSIS

```
cudaError_t cudaMemset2D( void* dstPtr, size_t pitch, int value, size_t width, size_t height
)
```

DESCRIPTION

Sets to the specified value **value** a matrix (**height** rows of **width** bytes each) pointed to by **dstPtr**. **pitch** is the width in memory in bytes of the 2D array pointed to by **dstPtr**, including any padding added to the end of each row. This function performs fastest when the pitch is one that has been passed back by **cudaMallocPitch()**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemset, *cudaMemset3D*

1.5.10 cudaMemcpy

NAME

cudaMemcpy - copies data between GPU and host

SYNOPSIS

```
cudaError_t cudaMemcpy( void* dst, const void* src, size_t count, enum cudaMemcpyKind kind
)

cudaError_t cudaMemcpyAsync( void* dst, const void* src, size_t count, enum cudaMemcpyKind
kind, cudaStream_t stream )
```

DESCRIPTION

Copies **count** bytes from the memory area pointed to by **src** to the memory area pointed to by **dst**, where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. The memory areas may not overlap. Calling **cudaMemcpy()** with **dst** and **src** pointers that do not match the direction of the copy results in an undefined behavior.

cudaMemcpyAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy2D, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.11 cudaMemcpy2D

NAME

cudaMemcpy2D - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpy2D( void* dst, size_t dpitch, const void* src, size_t spitch, size_t
width, size_t height, enum cudaMemcpyKind kind )
```

```
cudaError_t cudaMemcpy2DAsync( void* dst, size_t dpitch, const void* src, size_t spitch, size_t
width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream )
```

DESCRIPTION

Copies a matrix (**height** rows of **width** bytes each) from the memory area pointed to by **src** to the memory area pointed to by **dst**, where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. **dpitch** and **spitch** are the widths in memory in bytes of the 2D arrays pointed to by **dst** and **src**, including any padding added to the end of each row. The memory areas may not overlap. Calling **cudaMemcpy2D()** with **dst** and **src** pointers that do not match the direction of the copy results in an undefined behavior. **cudaMemcpy2D()** returns an error if **dpitch** or **spitch** is greater than the maximum allowed.

cudaMemcpy2DAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidPitchValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.12 cudaMemcpyToArray

NAME

cudaMemcpyToArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpyToArray(struct cudaArray* dstArray, size_t dstX, size_t dstY, const void*
src, size_t count, enum cudaMemcpyKind kind)
```

```
cudaError_t cudaMemcpyToArrayAsync(struct cudaArray* dstArray, size_t dstX, size_t dstY, const
void* src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)
```

DESCRIPTION

Copies **count** bytes from the memory area pointed to by **src** to the CUDA array **dstArray** starting at the upper left corner (**dstX**, **dstY**), where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

cudaMemcpyToArrayAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.13 cudaMemcpy2DToArray

NAME

cudaMemcpy2DToArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpy2DToArray(struct cudaArray* dstArray, size_t dstX, size_t dstY, const
void* src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind); cudaError_t
cudaMemcpy2DToArrayAsync(struct cudaArray* dstArray, size_t dstX, size_t dstY, const void*
src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream);
```

DESCRIPTION

Copies a matrix (**height** rows of **width** bytes each) from the memory area pointed to by **src** to the CUDA array **dstArray** starting at the upper left corner (**dstX**, **dstY**), where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. **spitch** is the width in memory in bytes of the 2D array pointed to by **src**, including any padding added to the end of each row. **cudaMemcpy2D()** returns an error if **spitch** is greater than the maximum allowed.

cudaMemcpy2DToArrayAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidPitchValue

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.14 cudaMemcpyFromArray

NAME

cudaMemcpyFromArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpyFromArray(void* dst, const struct cudaArray* srcArray, size_t srcX, size_t srcY, size_t count, enum cudaMemcpyKind kind)
```

```
cudaError_t cudaMemcpyFromArrayAsync(void* dst, const struct cudaArray* srcArray, size_t srcX, size_t srcY, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream)
```

DESCRIPTION

Copies **count** bytes from the CUDA array **srcArray** starting at the upper left corner (**srcX**, **srcY**) to the memory area pointed to by **dst**, where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

cudaMemcpyFromArrayAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero stream argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.15 cudaMemcpy2DFromArray

NAME

cudaMemcpy2DFromArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpy2DFromArray(void* dst, size_t dpitch, const struct cudaArray* srcArray,
size_t srcX, size_t srcY, size_t width, size_t height, enum cudaMemcpyKind kind)
```

```
cudaError_t cudaMemcpy2DFromArrayAsync(void* dst, size_t dpitch, const struct cudaArray* srcArray,
size_t srcX, size_t srcY, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t
stream)
```

DESCRIPTION

Copies a matrix (**height** rows of **width** bytes each) from the CUDA array **srcArray** starting at the upper left corner (**srcX**, **srcY**) to the memory area pointed to by **dst**, where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. **dpitch** is the width in memory in bytes of the 2D array pointed to by **dst**, including any padding added to the end of each row. **cudaMemcpy2D()** returns an error if **dpitch** is greater than the maximum allowed.

cudaMemcpy2DFromArrayAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidPitchValue

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.16 cudaMemcpyArrayToArray

NAME

cudaMemcpyArrayToArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpyArrayToArray(struct cudaArray* dstArray, size_t dstX, size_t dstY, const
struct cudaArray* srcArray, size_t srcX, size_t srcY, size_t count, enum cudaMemcpyKind kind)
```

DESCRIPTION

Copies **count** bytes from the CUDA array **srcArray** starting at the upper left corner (**srcX**, **srcY**) to the CUDA array **dstArray** starting at the upper left corner (**dstX**, **dstY**), where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.17 cudaMemcpy2DArrayToArray

NAME

cudaMemcpy2DArrayToArray - copies data between host and device

SYNOPSIS

```
cudaError_t cudaMemcpy2DArrayToArray(struct cudaArray* dstArray, size_t dstX, size_t dstY,
const struct cudaArray* srcArray, size_t srcX, size_t srcY, size_t width, size_t height, enum
cudaMemcpyKind kind)
```

DESCRIPTION

Copies a matrix (**height** rows of **width** bytes each) from the CUDA array **srcArray** starting at the upper left corner (**srcX**, **srcY**) to the CUDA array **dstArray** starting at the upper left corner (**dstX**, **dstY**), where **kind** is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.5.18 cudaMemcpyToSymbol

NAME

cudaMemcpyToSymbol - copies data from host memory to GPU

SYNOPSIS

```
template < class T >

cudaError_t cudaMemcpyToSymbol( const T& symbol, const void* src, size_t count, size_t offset,
enum cudaMemcpyKind kind)
```

DESCRIPTION

Copies `count` bytes from the memory area pointed to by `src` to the memory area pointed to by `offset` bytes from the start of symbol `symbol`. The memory areas may not overlap. `symbol` can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. `kind` can be either `cudaMemcpyHostToDevice` or `cudaMemcpyDeviceToDevice`.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidSymbol

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyFromSymbol*

1.5.19 cudaMemcpyFromSymbol

NAME

cudaMemcpyFromSymbol - copies data from GPU to host memory

SYNOPSIS

```
template < class T >

cudaError_t cudaMemcpyFromSymbol( void *dst, const T& symbol, size_t count, size_t offset,
enum cudaMemcpyKind kind)
```

DESCRIPTION

Copies `count` bytes from the memory area pointed to by `offset` bytes from the start of symbol `symbol` to the memory area pointed to by `dst`. The memory areas may not overlap. `symbol` can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. `kind` can be either `cudaMemcpyDeviceToHost` or `cudaMemcpyDeviceToDevice`.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidSymbol

cudaErrorInvalidDevicePointer

cudaErrorInvalidMemcpyDirection

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemcpy, *cudaMemcpy2D*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*

1.5.20 cudaGetSymbolAddress

NAME

cudaGetSymbolAddress - finds the address associated with a CUDA symbol

SYNOPSIS

```
template < class T >

cudaError_t cudaGetSymbolAddress(void** devPtr, const T& symbol)
```

DESCRIPTION

Returns in ***devPtr** the address of symbol **symbol** on the device. **symbol** can either be a variable that resides in global memory space, or it can be a character string, naming a variable that resides in global memory space. If **symbol** cannot be found, or if **symbol** is not declared in global memory space, ***devPtr** is unchanged and an error is returned. **cudaGetSymbolAddress()** returns **cudaErrorInvalidSymbol** in case of failure

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidSymbol

cudaErrorAddressOfConstant

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetSymbolSize

1.5.21 cudaGetSymbolSize

NAME

cudaGetSymbolSize - finds the size of the object associated with a CUDA symbol

SYNOPSIS

```
template < class T >

cudaError_t cudaGetSymbolSize(size_t* size, const T& symbol)
```

DESCRIPTION

Returns in ***size** the size of symbol **symbol**. **symbol** can either be a variable that resides in global or constant memory space, or it can be a character string, naming a variable that resides in global or constant memory space. If **symbol** cannot be found, or if **symbol** is not declared in global or constant memory space, ***size** is unchanged and an error is returned. **cudaGetSymbolSize()** returns **cudaErrorInvalidSymbol** in case of failure.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidSymbol

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetSymbolAddress

1.5.22 cudaMalloc3D

NAME

cudaMalloc3D - allocates logical 1D, 2D, or 3D memory objects on the GPU

SYNOPSIS

```
struct cudaPitchedPtr {
  void *ptr;
  size_t pitch;
  size_t xsize;
  size_t ysize;
};

struct cudaExtent {
  size_t width;
  size_t height;
  size_t depth;
};

cudaError_t cudaMalloc3D( struct cudaPitchedPtr* pitchDevPtr, struct cudaExtent extent )
```

DESCRIPTION

Allocates at least *width*height*depth* bytes of linear memory on the device and returns a **pitchedDevPtr** in which *ptr* is a pointer to the allocated memory. The function may pad the allocation to ensure hardware alignment requirements are met. The pitch returned in the *pitch* field of the **pitchedDevPtr** is the width in bytes of the allocation.

The returned **cudaPitchedPtr** contains additional fields *xsize* and *ysize*, the logical width and height of the allocation, which are equivalent to the **width** and **height** extent parameters provided by the programmer during allocation.

For allocations of 2D, 3D objects, it is highly recommended that programmers perform allocations using **cudaMalloc3D()** or **cudaMallocPitch()**. Due to alignment restrictions in the hardware, this is especially true if the application will be performing memory copies involving 2D or 3D objects (whether linear memory or CUDA arrays).

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMallocPitch, *cudaFree*, *cudaMemcpy3D*, *cudaMemset3D*, *cudaMalloc3DArray*, *cudaMallocArray*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.23 cudaMalloc3DArray

NAME

cudaMalloc3DArray - allocate an array on the GPU

SYNOPSIS

```
struct cudaExtent { size_t width; size_t height; size_t depth; };
```

```
cudaError_t cudaMalloc3DArray( struct cudaArray** arrayPtr, const struct cudaChannelFormatDesc*
desc, struct cudaExtent extent )
```

DESCRIPTION

Allocates a CUDA array according to the **cudaChannelFormatDesc** structure **desc** and returns a handle to the new CUDA array in ***arrayPtr**. The **cudaChannelFormatDesc** is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where **cudaChannelFormatKind** is one of **cudaChannelFormatKindSigned**, **cudaChannelFormatKindUnsigned**, **cudaChannelFormatKindFloat**.

cudaMalloc3DArray is able to allocate 1D, 2D, or 3D arrays.

- A 1D array is allocated if the height and depth extent are both zero. For 1D arrays valid extents are {(1, 8192), 0, 0} .
- A 2D array is allocated if only the depth extent is zero. For 2D arrays valid extents are {(1, 65536), (1, 32768), 0}.
- A 3D array is allocate if all three extents are non-zero. For 3D arrays valid extents are {(1, 2048), (1, 2048), (1, 2048)}.

Note: That because of the differing extent limits it may be advantageous to use a degenerate array (with unused dimensions set to one) of higher dimensionality. For instance, a degenerate 2D array allows for significantly more linear storage than a 1D array.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMemoryAllocation

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMalloc3D, *cudaMalloc*, *cudaMallocPitch*, *cudaFree*, *cudaFreeArray*, *cudaMallocHost*, *cudaFreeHost*

1.5.24 cudaMemset3D

NAME

cudaMemset3D - initializes or sets GPU memory to a value

SYNOPSIS

```
struct cudaPitchedPtr {
  void *ptr;
  size_t pitch;
  size_t xsize;
  size_t ysize;
};

struct cudaExtent {
  size_t width;
  size_t height;
  size_t depth;
};

cudaError_t cudaMemset3D( struct cudaPitchedPtr dstPitchPtr, int value, struct cudaExtent extent
)
```

DESCRIPTION

Initializes each element of a 3D array to the specified value **value**. The object to initialize is defined by **dstPitchPtr**. The *pitch* field of **dstPitchPtr** is the width in memory in bytes of the 3D array pointed to by **dstPitchPtr**, including any padding added to the end of each row. The *xsize* field specifies the logical width of each row in bytes, while the *ysize* field specifies the height of each 2D slice in rows.

The extents of the initialized region are specified as a *width* in bytes, a *height* in rows, and a *depth* in slices.

Extents with *width* greater than or equal to the *xsize* of **dstPitchPtr** may perform significantly faster than extents narrower than the *xsize*. Secondarily, extents with *height* equal to the *ysize* of **dstPitchPtr** will perform faster than when the *hieght* is shorter than the *ysize*.

This function performs fastest when the **dstPitchPtr** has been allocated by **cudaMalloc3D()**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaMemset, *cudaMemset2D*, *cudaMalloc3D*

1.5.25 cudaMemcpy3D

NAME

cudaMemcpy3D - copies data between between 3D objects

SYNOPSIS

```
struct cudaExtent {
  size_t width, height, depth;
};

struct cudaPos {
  size_t x, y, z;
};

struct cudaMemcpy3DParms {
  struct cudaArray     *srcArray;
  struct cudaPos        srcPos;
  struct cudaPitchedPtr srcPtr;
  struct cudaArray     *dstArray;
  struct cudaPos        dstPos;
  struct cudaPitchedPtr dstPtr;
  struct cudaExtent     extent;
  enum cudaMemcpyKind   kind;
};

cudaError_t cudaMemcpy3D( const struct cudaMemcpy3DParms *p )

cudaError_t cudaMemcpy3DAsync( const struct cudaMemcpy3DParms *p, cudaStream_t stream )
```

DESCRIPTION

cudaMemcpy3D() copies data betwen two 3D objects. The source and destination objects may be in either host memory, device memory, or a CUDA array. The source, destination, extent, and kind of copy performed is specified by the **cudaMemcpy3DParms** struct which should be initialized to zero before use:

```
cudaMemcpy3DParms myParms = {0};
```

The struct passed to **cudaMemcpy3D()** must specify one of *srcArray* or *srcPtr* and one of *dstArray* or *dstPtr*. Passing more than one non-zero source or destination will cause **cudaMemcpy3D()** to return an error.

The *srcPos* and *dstPos* fields are optional offsets into the source and destination objects and are defined in units of each object's elements. The element for a host or device pointer is assumed to be **unsigned char**. For CUDA arrays, positions must be in the range [0, 2048) for any dimension.

The *extent* field defines the dimensions of the transferred area in elements. If a CUDA array is participating in the copy the extent is defined in terms of that array's elements. If no CUDA array is participating in the copy then the extents are defined in elements of **unsigned char**.

The *kind* field defines the direction of the copy. It must be one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**.

If the source and destination are both arrays **cudaMemcpy3D()** will return an error if they do not have the same element size.

The source and destination object may not overlap. If overlapping source and destination objects are specified undefined behavior will result.

cudaMemcpy3D() returns an error if the pitch of **srcPtr** or **dstPtr** is greater than the maximum allowed. The pitch of a **cudaPitchedPtr** allocated with **cudaMalloc3D()** will always be valid.

cudaMemcpy3DAsync() is an asynchronous copy operation and can optionally be associated to a stream by passing a non-zero stream argument. If either the source or destination is a host object it must be allocated in page-locked memory returned from **cudaMallocHost()**. It will return an error if a pointer to memory not allocated with **cudaMallocHost()** is passed as input.

RETURN VALUE

cudaSuccess

SEE ALSO

cudaMalloc3D, *cudaMalloc3DArray*, *cudaMemset3D*, *cudaMemcpy*, *cudaMemcpyToArray*, *cudaMemcpy2DToArray*, *cudaMemcpyFromArray*, *cudaMemcpy2DFromArray*, *cudaMemcpyArrayToArray*, *cudaMemcpy2DArrayToArray*, *cudaMemcpyToSymbol*, *cudaMemcpyFromSymbol*

1.6 TextureReferenceManagement RT

NAME

Texture Reference Management

DESCRIPTION

This section describes the CUDA runtime application programming interface.

Low-Level API

High-Level API

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.6.1 LowLevelApi

NAME

Low-Level Texture API

DESCRIPTION

This section describes the low-level CUDA run-time application programming interface for textures

cudaCreateChannelDesc

cudaGetChannelDesc

cudaGetTextureReference

cudaBindTexture

cudaBindTextureToArray

cudaUnbindTexture

cudaGetTextureAlignmentOffset

SEE ALSO

High-Level API

cudaCreateChannelDesc

NAME

cudaCreateChannelDesc - Low-level texture API

SYNOPSIS

```
struct cudaChannelFormatDesc cudaCreateChannelDesc(int x, int y, int z, int w, enum cudaChannelFormatKi
f);
```

DESCRIPTION

Returns a channel descriptor with format **f** and number of bits of each component **x**, **y**, **z**, and **w**. The **cudaChannelFormatDesc** is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where **cudaChannelFormatKind** is one of **cudaChannelFormatKindSigned**, **cudaChannelFormatKindUnsigned**, **cudaChannelFormatKindFloat**.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGetChannelDesc, *cudaGetTextureReference*, *cudaBindTexture*, *cudaBindTextureToArray*, *cudaUnbindTexture*, *cudaGetTextureAlignmentOffset*

cudaGetChannelDesc

NAME

cudaGetChannelDesc - Low-level texture API

SYNOPSIS

```
cudaError_t cudaGetChannelDesc(struct cudaChannelFormatDesc* desc, const struct cudaArray*
array);
```

DESCRIPTION

Returns in ***desc** the channel descriptor of the CUDA array **array**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetTextureReference*, *cudaBindTexture*, *cudaBindTextureToArray*, *cudaUnbindTexture*, *cudaGetTextureAlignmentOffset*

cudaGetTextureReference

NAME

cudaGetTextureReference - Low-level texture API

SYNOPSIS

```
cudaError_t cudaGetTextureReference( struct textureReference** texRef, const char* symbol)
```

DESCRIPTION

Returns in ***texRef** the structure associated to the texture reference defined by symbol **symbol**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidTexture

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetChannelDesc*, *cudaBindTexture*, *cudaBindTextureToArray*, *cudaUnbindTexture*, *cudaGetTextureAlignmentOffset*

cudaBindTexture

NAME

cudaBindTexture - Low-level texture API

SYNOPSIS

```
cudaError_t cudaBindTexture(size_t* offset, const struct textureReference* texRef, const void*
devPtr, const struct cudaChannelFormatDesc* desc, size_t size = UINT_MAX);
```

DESCRIPTION

Binds **size** bytes of the memory area pointed to by **devPtr** to the texture reference **texRef**. **desc** describes how the memory is interpreted when fetching values from the texture. Any memory previously bound to **texRef** is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, **cudaBindTexture()** returns in ***offset** a byte offset that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the **tex1Dfetch()** function. If the device memory pointer was returned from **cudaMalloc()**, the offset is guaranteed to be 0 and NULL may be passed as the **offset** parameter.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidTexture

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetChannelDesc*, *cudaGetTextureReference*, *cudaBindTextureToArray*, *cudaUnbindTexture*, *cudaGetTextureAlignmentOffset*

cudaBindTextureToArray

NAME

cudaBindTextureToArray - Low-level texture API

SYNOPSIS

```
cudaError_t cudaBindTextureToArray( const struct textureReference* texRef, const struct cudaArray*
array, const struct cudaChannelFormatDesc* desc);
```

DESCRIPTION

Binds the CUDA array **array** to the texture reference **texRef**. **desc** describes how the memory is interpreted when fetching values from the texture. Any CUDA array previously bound to **texRef** is unbound.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidTexture

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetChannelDesc*, *cudaGetTextureReference*, *cudaBindTexture*, *cudaUnbindTexture*, *cudaGetTextureAlignmentOffset*

cudaUnbindTexture

NAME

cudaUnbindTexture - Low-level texture API

SYNOPSIS

```
cudaError_t cudaUnbindTexture( const struct textureReference* texRef);
```

DESCRIPTION

Unbinds the texture bound to texture reference **texRef**.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetChannelDesc*, *cudaGetTextureReference*, *cudaBindTexture*, *cudaBindTextureToArray*, *cudaGetTextureAlignmentOffset*

cudaGetTextureAlignmentOffset

NAME

cudaGetTextureAlignmentOffset - Low-level texture API

SYNOPSIS

```
cudaError_t cudaGetTextureAlignmentOffset(size_t* offset, const struct textureReference* texRef);
```

DESCRIPTION

Returns in ***offset** the offset that was returned when texture reference **texRef** was bound.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidTexture

cudaErrorInvalidTextureBinding

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc, *cudaGetChannelDesc*, *cudaGetTextureReference*, *cudaBindTexture*, *cudaBindTextureToArray*, *cudaUnbindTexture*

1.6.2 HighLevelApi

NAME

High-Level Texture API

DESCRIPTION

This section describes the high-level CUDA run-time application programming interface for textures

cudaCreateChannelDesc

cudaBindTexture

cudaBindTextureToArray

cudaUnbindTexture

SEE ALSO

Low-Level API

cudaCreateChannelDesc HL

NAME

cudaCreateChannelDesc - High-level texture API

SYNOPSIS

```
template < class T >

struct cudaChannelFormatDesc cudaCreateChannelDesc <T >();
```

DESCRIPTION

Returns a channel descriptor with format **f** and number of bits of each component **x**, **y**, **z**, and **w**. The **cudaChannelFormatDesc** is defined as:

```
struct cudaChannelFormatDesc {
  int x, y, z, w;
  enum cudaChannelFormatKind f;
};
```

where **cudaChannelFormatKind** is one of **cudaChannelFormatKindSigned**, **cudaChannelFormatKindUnsigned**, **cudaChannelFormatKindFloat**.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaBindTexture HL

NAME

cudaBindTexture - High-level texture API

SYNOPSIS

```
template < class T, int dim, enum cudaTextureReadMode readMode >

static __inline__ __host__ cudaError_t cudaBindTexture(size_t* offset, const struct texture
< T, dim, readMode >& texRef, const void* devPtr, const struct cudaChannelFormatDesc& desc,
size_t size = UINT_MAX)
```

DESCRIPTION

Binds **size** bytes of the memory area pointed to by **devPtr** to texture reference **texRef**. **desc** describes how the memory is interpreted when fetching values from the texture. The **offset** parameter is an optional byte offset as with the low-level **cudaBindTexture()** function. Any memory previously bound to **texRef** is unbound.

```
template E<lt> class T, int dim, enum cudaTextureReadMode readMode E<gt>
static __inline__ __host__ cudaError_t cudaBindTexture(
   size_t* offset,
   const struct texture E<lt> T, dim, readMode E<gt>& texRef,
   const void* devPtr,
   size_t size = UINT_MAX);
```

binds size bytes of the memory area pointed to by devPtr to texture reference texRef. The channel descriptor is inherited from the texture reference type. The offset parameter is an optional byte offset as with the low-level cudaBindTexture() function described

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidTexture

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaBindTextureToArray HL

NAME

cudaBindTextureToArray - High-level texture API

SYNOPSIS

```
template < class T, int dim, enum cudaTextureReadMode readMode >

static __inline__ __host__ cudaError_t cudaBindTextureToArray( const struct texture < T, dim,
readMode >& texRef, const struct cudaArray* cuArray, const struct cudaChannelFormatDesc& desc)
```

DESCRIPTION

Binds the CUDA array **array** to texture reference **texRef**. **desc** describes how the memory is interpreted when fetching values from the texture. Any CUDA array previously bound to **texRef** is unbound.

```
template E<lt> class T, int dim, enum cudaTextureReadMode readMode E<gt>
static __inline__ __host__ cudaError_t cudaBindTextureToArray(
  const struct texture E<lt> T, dim, readMode E<gt>& texRef,
  const struct cudaArray* cuArray);
```

binds the CUDA array **array** to texture reference **texRef**. The channel descriptor is inherited from the CUDA array. Any CUDA array previously bound to **texRef** is unbound.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidTexture

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaCreateChannelDesc_HL

cudaUnbindTexture HL

NAME

cudaUnbindTexture - High-level texture API

SYNOPSIS

```
template < class T, int dim, enum cudaTextureReadMode readMode >

static __inline__ __host__ cudaError_t cudaUnbindTexture(const struct texture < T, dim, readMode
>& texRef)
```

DESCRIPTION

Unbinds the texture bound to texture reference **texRef**.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

1.7 ExecutionControl RT

NAME

Execution Control

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaConfigureCall

cudaLaunch

cudaSetupArgument

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.7.1 cudaConfigureCall

NAME

cudaConfigureCall - configure a device-launch

SYNOPSIS

```
cudaError_t cudaConfigureCall(dim3 gridDim, dim3 blockDim, size_t sharedMem = 0, int tokens
= 0)
```

DESCRIPTION

Specifies the grid and block dimensions for the device call to be executed similar to the execution configuration syntax. **cudaConfigureCall()** is stack based. Each call pushes data on top of an execution stack. This data contains the dimension for the grid and thread blocks, together with any arguments for the call.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidConfiguration

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaLaunch, *cudaSetupArgument*

1.7.2 cudaLaunch

NAME

cudaLaunch - launches a device function

SYNOPSIS

```
template < class T > cudaError_t cudaLaunch(T entry)
```

DESCRIPTION

Launches the function **entry** on the device. **entry** can either be a function that executes on the device, or it can be a character string, naming a function that executes on the device. **entry** must be declared as a **__global__** function. **cudaLaunch()** must be preceded by a call to **cudaConfigureCall()** since it pops the data that was pushed by **cudaConfigureCall()** from the execution stack.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDeviceFunction

cudaErrorInvalidConfiguration

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaConfigureCall, *cudaSetupArgument*

1.7.3 cudaSetupArgument

NAME

cudaSetupArgument - configure a device-launch

SYNOPSIS

```
cudaError_t cudaSetupArgument(void* arg, size_t count, size_t offset)
```

```
template < class T > cudaError_t cudaSetupArgument(T arg, size_t offset)
```

DESCRIPTION

Pushes **count** bytes of the argument pointed to by **arg** at **offset** bytes from the start of the parameter passing area, which starts at offset 0. The arguments are stored in the top of the execution stack. **cudaSetupArgument()** must be preceded by a call to **cudaConfigureCall()**.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaConfigureCall, *cudaLaunch*

1.8 OpenGlInteroperability RT

NAME

OpenGL Interoperability

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaGLSetGLDevice

cudaGLRegisterBufferObject

cudaGLMapBufferObject

cudaGLUnmapBufferObject

cudaGLUnregisterBufferObject

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.8.1 cudaGLSetGLDevice

NAME

cudaGLSetGLDevice - sets the CUDA device for use with GL Interopability

SYNOPSIS

```
cudaError_t cudaGLSetGLDevice(int device);
```

DESCRIPTION

Records dev as the device on which the active host thread executes the device code. Records the thread as using GL Interopability.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDevice

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGLRegisterBufferObject, *cudaGLMapBufferObject*, *cudaGLUnmapBufferObject*, *cudaGLUnregisterBufferObject*

1.8.2 cudaGLRegisterBufferObject

NAME

cudaGLRegisterBufferObject - OpenGL interoperability

SYNOPSIS

```
cudaError_t cudaGLRegisterBufferObject(GLuint bufferObj)
```

DESCRIPTION

Registers the buffer object of ID **bufferObj** for access by CUDA. This function must be called before CUDA can map the buffer object. While it is registered, the buffer object cannot be used by any OpenGL commands except as a data source for OpenGL drawing commands.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorNotInitialized

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGLSetGLDevice, *cudaGLMapBufferObject*, *cudaGLUnmapBufferObject*, *cudaGLUnregisterBufferObject*

1.8.3 cudaGLMapBufferObject

NAME

cudaGLMapBufferObject - OpenGL interoperability

SYNOPSIS

```
cudaError_t cudaGLMapBufferObject(void** devPtr, GLuint bufferObj);
```

DESCRIPTION

Maps the buffer object of ID **bufferObj** into the address space of CUDA and returns in ***devPtr** the base pointer of the resulting mapping.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorMapBufferObjectFailed

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGLSetGLDevice, *cudaGLRegisterBufferObject*, *cudaGLUnmapBufferObject*, *cudaGLUnregisterBufferObject*

1.8.4 cudaGLUnmapBufferObject

NAME

cudaGLUnmapBufferObject - OpenGL interoperability

SYNOPSIS

```
cudaError_t cudaGLUnmapBufferObject(GLuint bufferObj);
```

DESCRIPTION

Unmaps the buffer object of ID **bufferObj** for access by CUDA.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidDevicePointer

cudaErrorUnmapBufferObjectFailed

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGLSetGLDevice, *cudaGLRegisterBufferObject*, *cudaGLMapBufferObject*, *cudaGLUnregisterBufferObject*

1.8.5 cudaGLUnregisterBufferObject

NAME

cudaGLUnregisterBufferObject - OpenGL interoperability

SYNOPSIS

```
cudaError_t cudaGLUnregisterBufferObject(GLuint bufferObj);
```

DESCRIPTION

Unregisters the buffer object of ID **bufferObj** for access by CUDA.

RETURN VALUE

Relevant return values:

cudaSuccess

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaGLSetGLDevice, *cudaGLRegisterBufferObject*, *cudaGLMapBufferObject*, *cudaGLUnmapBufferObject*

1.9 Direct3dInteroperability RT

NAME

Direct3D Interoperability

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaD3D9GetDevice

cudaD3D9SetDirect3DDevice

cudaD3D9GetDirect3DDevice

cudaD3D9RegisterResource

cudaD3D9UnregisterResource

cudaD3D9MapResources

cudaD3D9UnmapResources

cudaD3D9ResourceGetSurfaceDimensions

cudaD3D9ResourceSetMapFlags

cudaD3D9ResourceGetMappedPointer

cudaD3D9ResourceGetMappedSize

cudaD3D9ResourceGetMappedPitch

As of CUDA 2.0 the following functions are deprecated. They should not be used in new development.

cudaD3D9Begin

cudaD3D9End

cudaD3D9RegisterVertexBuffer

cudaD3D9MapVertexBuffer

cudaD3D9UnmapVertexBuffer

cudaD3D9UnregisterVertexBuffer

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.9.1 cudaD3D9GetDevice

NAME

cudaD3D9GetDevice - gets the device number for an adapter

SYNOPSIS

```
cudaError_t cudaD3D9GetDevice(int* dev, const char* adapterName);
```

DESCRIPTION

Returns in ***dev** the CUDA-compatible device corresponding to the adapter name **adapterName** obtained from **EnumDisplayDevices** or **IDirect3D9::GetAdapterIdentifier()**. If no device on the adapter with name **adapterName** is CUDA-compatible then the call will fail.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9SetDirect3DDevice, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMap* *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.2 cudaD3D9SetDirect3DDevice

NAME

cudaD3D9SetDirect3DDevice - sets the Direct3D device to use for interoperability in this thread

SYNOPSIS

```
cudaError_t cudaD3D9SetDirect3DDevice(IDirect3DDevice9* pDxDevice);
```

DESCRIPTION

Records **pDxDevice** as the Direct3D device to use for Direct3D interoperability on this host thread. In order to use Direct3D interoperability, this call must be made before any other CUDA runtime calls on this thread.

Successful context creation on **pDxDevice** will increase the internal reference count on **pDxDevice**. This reference count will be decremented upon destruction of this context through *cudaThreadExit*.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

cudaErrorInvalidValue

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMap* *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.3 cudaD3D9GetDirect3DDevice

NAME

cudaD3D9GetDirect3DDevice - get the Direct3D device against which the current CUDA context was created

SYNOPSIS

```
cudaError_t cudaD3D9GetDirect3DDevice(IDirect3DDevice9** ppDxDevice);
```

DESCRIPTION

Returns in ***ppDxDevice** the Direct3D device against which this CUDA context was created in **cudaD3D9SetDirect3DDevice**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMap*
cudaD3D9ResourceGetMappedPointer, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.4 cudaD3D9RegisterResource

NAME

cudaD3D9RegisterResource - register a Direct3D resource for access by CUDA

SYNOPSIS

```
cudaError_t cudaD3D9RegisterResource(IDirect3DResource9* pResource, unsigned int Flags);
```

DESCRIPTION

Registers the Direct3D resource **pResource** for access by CUDA.

If this call is successful then the application will be able to map and unmap this resource until it is unregistered through *cudaD3D9UnregisterResource*. Also on success, this call will increase the internal reference count on **pResource**. This reference count will be decremented when this resource is unregistered through *cudaD3D9UnregisterResource*.

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of **pResource** must be one of the following.

- **IDirect3DVertexBuffer9**: No notes.
- **IDirect3DIndexBuffer9**: No notes.
- **IDirect3DSurface9**: Only stand-alone objects of type **IDirect3DSurface9** may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object.
- **IDirect3DBaseTexture9**: When a texture is registered all surfaces associated with the all mipmap levels of all faces of the texture will be accessible to CUDA.

The **Flags** argument specifies the mechanism through which CUDA will access the Direct3D resource. The following value is allowed.

- **cudaD3D9RegisterFlagsNone**: Specifies that CUDA will access this resource through a **void***. The pointer, size, and pitch for each subresource of this resource may be queried through *cudaD3D9ResourceGetMappedPointe* *cudaD3D9ResourceGetMappedSize*, and *cudaD3D9ResourceGetMappedPitch* respectively. This option is valid for all resource types.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Any resources allocated in **D3DPOOL_SYSTEMMEM** may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.

- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context then is returned. If **pResource** is of incorrect type (e.g, is a non-stand-alone **IDirect3DSurface9**) or is already registered then **cudaErrorInvalidHandle** is returned. If **pResource** cannot be registered then **cudaErrorUnknown** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMap* *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.5 cudaD3D9UnregisterResource

NAME

cudaD3D9UnregisterResource - unregister a Direct3D resource

SYNOPSIS

```
cudaError_t cudaD3D9UnregisterResource(IDirect3DResource9* pResource);
```

DESCRIPTION

Unregisters the Direct3D resource **pResource** so it is not accessable by CUDA unless registered again.

If **pResource** is not registered then **cudaErrorInvalidHandle** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMap* *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.6 cudaD3D9MapResources

NAME

cudaD3D9MapResources - map Direct3D resources for access by CUDA

SYNOPSIS

```
cudaError_t cudaD3D9MapResources(unsigned int count, IDirect3DResource9 **ppResources);
```

DESCRIPTION

Maps the **count** Direct3D resources in **ppResources** for access by CUDA.

The resources in **ppResources** may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before **cudaD3D9MapResources** will complete before any CUDA kernels issued after **cudaD3D9MapResources** begin.

If any of **ppResources** have not been registered for use with CUDA or if **ppResources** contains any duplicate entries then **cudaErrorInvalidHandle** is returned. If any of **ppResources** are presently mapped for access by CUDA then **cudaErrorUnknown** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetMapFlags*, *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.7 cudaD3D9UnmapResources

NAME

cudaD3D9UnmapResources - unmap Direct3D resources

SYNOPSIS

```
cudaError_t cudaD3D9UnmapResources(unsigned int count, IDirect3DResource9** ppResources);
```

DESCRIPTION

Unmaps the **count** Direct3D resources in **ppResources**.

This function provides the synchronization guarantee that any CUDA kernels issued before **cudaD3D9UnmapResources** will complete before any Direct3D calls issued after **cudaD3D9UnmapResources** begin.

If any of **ppResources** have not been registered for use with CUDA or if **ppResources** contains any duplicate entries then **cudaErrorInvalidHandle** is returned. If any of **ppResources** are not presently mapped for access by CUDA then **cudaErrorUnknown** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9ResourceGetSurfaceDimensions*, *cudaD3D9ResourceSetM*
cudaD3D9ResourceGetMappedPointer, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.8 cudaD3D9ResourceSetMapFlags

NAME

cudaD3D9ResourceSetMapFlags - set usage flags for mapping a Direct3D resource

SYNOPSIS

```
cudaError_t cudaD3D9ResourceSetMapFlags(IDirect3DResource9 *pResource, unsigned int Flags);
```

DESCRIPTION

Set flags for mapping the Direct3D resource **pResource**.

Changes to flags will take effect the next time **pResource** is mapped. The **Flags** argument may be any of the following.

- **cudaD3D9MapFlagsNone**: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.
- **cudaD3D9MapFlagsReadOnly**: Specifies that CUDA kernels which access this resource will not write to this resource.
- **cudaD3D9MapFlagsWriteDiscard**: Specifies that CUDA kernels which access this resource will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

If **pResource** has not been registered for use with CUDA then **cudaErrorInvalidHandle** is returned. If **pResource** is presently mapped for access by CUDA then **cudaErrorUnknown** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensio *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.9 cudaD3D9ResourceGetSurfaceDimensions

NAME

cudaD3D9ResourceGetSurfaceDimensions - get the dimensions of a registered surface

SYNOPSIS

```
cudaError_t cudaD3D9ResourceGetSurfaceDimensions(size_t* pWidth, size_t* pHeight, size_t *pDepth,
IDirect3DResource9* pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pWidth**, ***pHeight**, and ***pDepth** the dimensions of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**.

Because anti-aliased surfaces may have multiple samples per pixel it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters **pWidth**, **pHeight**, and **pDepth** are optional. For 2D surfaces, the value returned in ***pDepth** will be 0.

If **pResource** is not of type **IDirect3DBaseTexture9** or **IDirect3DSurface9** or if **pResource** has not been registered for use with CUDA then **cudaErrorInvalidHandle** is returned.

For usage requirements of **Face** and **Level** parameters see *cudaD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceSetMapFlags*, *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.10 cudaD3D9ResourceGetMappedPointer

NAME

cudaD3D9ResourceGetMappedPointer - get a pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
cudaError_t cudaD3D9ResourceGetMappedPointer(void** pPointer, IDirect3DResource9* pResource,
unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pPointer** the base pointer of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The value set in **pPointer** may change every time that **pResource** is mapped.

If **pResource** is not registered then **cudaErrorInvalidHandle** is returned. If **pResource** was not registered with usage flags **cudaD3D9RegisterFlagsNone** then **cudaErrorInvalidHandle** is returned. If **pResource** is not mapped then **cudaErrorUnknown** is returned.

If **pResource** is of type **IDirect3DCubeTexture9** then **Face** must one of the values enumerated by type **D3DCUBEMAP_FACES**. For all other types **Face** must be 0. If **Face** is invalid then **cudaErrorInvalidValue** is returned.

If **pResource** is of type **IDirect3DBaseTexture9** then **Level** must correspond to a valid mipmap level. Only mipmap level 0 is supported for now. For all other types **Level** must be 0. If **Level** is invalid then **cudaErrorInvalidValue** is returned.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensio*
cudaD3D9ResourceSetMapFlags, *cudaD3D9ResourceGetMappedSize*, *cudaD3D9ResourceGetMappedPitch*

1.9.11 cudaD3D9ResourceGetMappedSize

NAME

cudaD3D9ResourceGetMappedSize - get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
cudaError_t cudaD3D9ResourceGetMappedSize(size_t* pSize, IDirect3DResource9* pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pSize** the size of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The value set in **pSize** may change every time that **pResource** is mapped.

If **pResource** has not been registered for use with CUDA then **cudaErrorInvalidHandle** is returned. If **pResource** was not registered with usage flags **cudaD3D9RegisterFlagsNone** then **cudaErrorInvalidHandle** is returned. If **pResource** is not mapped for access by CUDA then **cudaErrorUnknown** is returned.

For usage requirements of **Face** and **Level** parameters see *cudaD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensio*
cudaD3D9ResourceSetMapFlags, *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedPitch*

1.9.12 cudaD3D9ResourceGetMappedPitch

NAME

cudaD3D9ResourceGetMappedPitch - get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
cudaError_t cudaD3D9ResourceGetMappedPitch(size_t* pPitch, size_t* pPitchSlice, IDirect3DResource9*
pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pPitch** and ***pPitchSlice** the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The values set in **pPitch** and **pPitchSlice** may change every time that **pResource** is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

y***pitch** + (**bytes per pixel**)***x**

For a 3D surface the byte offset of the sample of at position **x**,**y**,**z** from the base pointer of the surface is

z***slicePitch** + **y*****pitch** + (**bytes per pixel**)***x**

Both parameters **pPitch** and **pPitchSlice** are optional and may be set to NULL.

For a 2D surface the byte offset of the sample of at position **x**,**y** from the base pointer of the surface is

If **pResource** is not of type **IDirect3DBaseTexture9** or one of its sub-types or if **pResource** has not been registered for use with CUDA then **cudaErrorInvalidHandle** is returned. If **pResource** was not registered with usage flags **cudaD3D9RegisterFlagsNone** then **cudaErrorInvalidHandle** is returned. If **pResource** is not mapped for access by CUDA then **cudaErrorUnknown** is returned.

For usage requirements of **Face** and **Level** parameters see *cudaD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInvalidValue

cudaErrorInvalidHandle

cudaErrorUnknown

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cudaD3D9GetDevice, *cudaD3D9SetDirect3DDevice*, *cudaD3D9GetDirect3DDevice*, *cudaD3D9RegisterResource*, *cudaD3D9UnregisterResource*, *cudaD3D9MapResources*, *cudaD3D9UnmapResources*, *cudaD3D9ResourceGetSurfaceDimensio* *cudaD3D9ResourceSetMapFlags*, *cudaD3D9ResourceGetMappedPointer*, *cudaD3D9ResourceGetMappedSize*

1.10 ErrorHandling RT

NAME

Error Handling

DESCRIPTION

This section describes the CUDA runtime application programming interface.

cudaGetLastError

cudaGetErrorString

SEE ALSO

Device Management, Thread Management, Stream Management, Event Management, Execution Management, Memory Management, Texture Reference Management, OpenGL Interoperability, Direct3D Interoperability, Error Handling

1.10.1 cudaGetLastError

NAME

cudaGetLastError - returns the last error from a run-time call

SYNOPSIS

```
cudaError_t cudaGetLastError( void )
```

DESCRIPTION

Returns the last error that was returned from any of the runtime calls in the same host thread and resets it to **cudaSuccess**.

RETURN VALUE

Relevant return values:

cudaSuccess

cudaErrorInitializationError

cudaErrorLaunchFailure

cudaErrorPriorLaunchFailure

cudaErrorLaunchTimeout

cudaErrorLaunchOutOfResources

cudaErrorInvalidDeviceFunction

cudaErrorInvalidConfiguration

cudaErrorInvalidDevice

cudaErrorInvalidValue

cudaErrorInvalidDevicePointer

cudaErrorInvalidTexture

cudaErrorInvalidTextureBinding

cudaErrorInvalidChannelDescriptor

cudaErrorTextureFetchFailed

cudaErrorTextureNotBound

cudaErrorSynchronizationError

cudaErrorUnknown

cudaErrorInvalidResourceHandle

cudaErrorNotReady

Note that this function may also return error codes from previous asynchronous launches.

SEE ALSO

cudaGetErrorString, *cudaError*

1.10.2 cudaGetErrorString

NAME

cudaGetErrorString - returns the message string from an error

SYNOPSIS

```
const char* cudaGetErrorString(cudaError_t error);
```

DESCRIPTION

Returns a message string from an error code.

RETURN VALUE

`char*` pointer to a NULL-terminated string

SEE ALSO

cudaGetLastError

2 DriverApiReference

NAME

Driver API Reference

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.1 Initialization

NAME

Driver Initialization

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuInit

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.1.1 cuInit

NAME

cuInit - initialize the CUDA driver API

SYNOPSIS

```
CUresult cuInit( unsigned int Flags );
```

DESCRIPTION

Initializes the driver API and must be called before any other function from the driver API. Currently, the **Flags** parameters must be 0. If **cuInit()** has not been called, any function from the driver API will return **CUDA_ERROR_NOT_INITIALIZED**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NO_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

2.2 DeviceManagement

NAME

Device Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuDeviceComputeCapability

cuDeviceGet

cuDeviceGetAttribute

cuDeviceGetCount

cuDeviceGetName

cuDeviceGetProperties

cuDeviceTotalMem

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.2.1 cuDeviceComputeCapability

NAME

cuDeviceComputeCapability - returns the compute capability of the device

SYNOPSIS

```
CUresult cuDeviceComputeCapability(int* major, int* minor, CUdevice dev);
```

DESCRIPTION

Returns in ***major** and ***minor** the major and minor revision numbers that define the compute capability of device **dev**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceGetAttribute, *cuDeviceGetCount*, *cuDeviceGetName*, *cuDeviceGet*, *cuDeviceGetProperties*, *cuDeviceTotalMem*

2.2.2 cuDeviceGet

NAME

cuDeviceGet - returns a device-handle

SYNOPSIS

```
CUresult cuDeviceGet(CUdevice* dev, int ordinal);
```

DESCRIPTION

Returns in ***dev** a device handle given an ordinal in the range **[0, cuDeviceGetCount()-1]**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetAttribute*, *cuDeviceGetCount*, *cuDeviceGetName*, *cuDeviceGetProperties*, *cuDeviceTotalMem*

2.2.3 cuDeviceGetAttribute

NAME

cuDeviceGetAttribute - returns information about the device

SYNOPSIS

```
CUresult cuDeviceGetAttribute(int* value, CUdevice_attribute attrib, CUdevice dev);
```

DESCRIPTION

Returns in ***value** the integer value of the attribute **attrib** on device **dev**. The supported attributes are:

- **CU_DEVICE_ATTRIBUTE_MAX_THREADS_PER_BLOCK**: maximum number of threads per block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_X**: maximum x-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Y**: maximum y-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_BLOCK_DIM_Z**: maximum z-dimension of a block;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_X**: maximum x-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Y**: maximum y-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_GRID_DIM_Z**: maximum z-dimension of a grid;
- **CU_DEVICE_ATTRIBUTE_MAX_SHARED_MEMORY_PER_BLOCK**: maximum amount of shared memory available to a thread block in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;
- **CU_DEVICE_ATTRIBUTE_TOTAL_CONSTANT_MEMORY**: total amount of constant memory available on the device in bytes;
- **CU_DEVICE_ATTRIBUTE_WARP_SIZE**: warp size in threads;
- **CU_DEVICE_ATTRIBUTE_MAX_PITCH**: maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through **cuMemAllocPitch()**;
- **CU_DEVICE_ATTRIBUTE_MAX_REGISTERS_PER_BLOCK**: maximum number of 32-bit registers available to a thread block; this number is shared by all thread blocks simultaneously resident on a multiprocessor;
- **CU_DEVICE_ATTRIBUTE_CLOCK_RATE**: clock frequency in kilohertz;
- **CU_DEVICE_ATTRIBUTE_TEXTURE_ALIGNMENT**: alignment requirement; texture base addresses aligned to **textureAlign** bytes do not need an offset applied to texture fetches;
- **CU_DEVICE_ATTRIBUTE_GPU_OVERLAP**: 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not;
- **CU_DEVICE_ATTRIBUTE_MULTIPROCESSOR_COUNT**: number of multiprocessors on the device.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetCount*, *cuDeviceGetName*, *cuDeviceGet*, *cuDeviceGetProperties*, *cuDeviceTotalMem*

2.2.4 cuDeviceGetCount

NAME

cuDeviceGetCount - returns the number of compute-capable devices

SYNOPSIS

```
CUresult cuDeviceGetCount(int* count);
```

DESCRIPTION

Returns in ***count** the number of devices with compute capability greater or equal to 1.0 that are available for execution. If there is no such device, **cuDeviceGetCount()** returns 0.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetAttribute*, *cuDeviceGetName*, *cuDeviceGet*, *cuDeviceGetProperties*, *cuDeviceTotalMem*

2.2.5 cuDeviceGetName

NAME

cuDeviceGetName - returns an identifier string

SYNOPSIS

```
CUresult cuDeviceGetName(char* name, int len, CUdevice dev);
```

DESCRIPTION

Returns an ASCII string identifying the device **dev** in the NULL-terminated string pointed to by **name**. **len** specifies the maximum length of the string that may be returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetAttribute*, *cuDeviceGetCount*, *cuDeviceGet*, *cuDeviceGetProperties*, *cuDeviceTotalMem*

2.2.6 cuDeviceGetProperties

NAME

cuDeviceGetProperties - get device properties

SYNOPSIS

```
CUresult cuDeviceGetProperties(CUdevprop* prop, CUdevice dev);
```

DESCRIPTION

Returns in ***prop** the properties of device **dev**. The **CUdevprop** structure is defined as:

```
typedef struct CUdevprop_st {
   int maxThreadsPerBlock;
   int maxThreadsDim[3];
   int maxGridSize[3];
   int sharedMemPerBlock;
   int totalConstantMemory;
   int SIMDWidth;
   int memPitch;
   int regsPerBlock;
   int clockRate;
   int textureAlign
} CUdevprop;
```

where:

- **maxThreadsPerBlock** is the maximum number of threads per block;
- **maxThreadsDim[3]** is the maximum sizes of each dimension of a block;
- **maxGridSize[3]** is the maximum sizes of each dimension of a grid;
- **sharedMemPerBlock** is the total amount of shared memory available per block in bytes;
- **totalConstantMemory** is the total amount of constant memory available on the device in bytes;
- **SIMDWidth** is the warp size;
- **memPitch** is the maximum pitch allowed by the memory copy functions that involve memory regions allocated through **cuMemAllocPitch()**;
- **regsPerBlock** is the total number of registers available per block;
- **clockRate** is the clock frequency in kilohertz;
- **textureAlign** is the alignment requirement; texture base addresses that are aligned to **textureAlign** bytes do not need an offset applied to texture fetches.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetAttribute*, *cuDeviceGetCount*, *cuDeviceGetName*, *cuDeviceGet*, *cuDeviceTotalMem*

2.2.7 cuDeviceTotalMem

NAME

cuDeviceTotalMem - returns the total amount of memory on the device

SYNOPSIS

```
CUresult cuDeviceTotalMem( unsigned int* bytes, CUdevice dev );
```

DESCRIPTION

Returns in ***bytes** the total amount of memory available on the device **dev** in bytes.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_DEVICE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuDeviceComputeCapability, *cuDeviceGetAttribute*, *cuDeviceGetCount*, *cuDeviceGetName*, *cuDeviceGet*, *cuDeviceGetProperties*

2.3 ContextManagement

NAME

Context Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuCtxAttach

cuCtxCreate

cuCtxDestroy

cuCtxDetach

cuCtxGetDevice

cuCtxPopCurrent

cuCtxPushCurrent

cuCtxSynchronize

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.3.1 cuCtxAttach

NAME

cuCtxAttach - increment context usage-count

SYNOPSIS

```
CUresult cuCtxAttach(CUcontext* pCtx, unsigned int Flags);
```

DESCRIPTION

Increments the usage count of the context and passes back a context handle in ***pCtx** that must be passed to **cuCtxDetach()** when the application is done with the context. **cuCtxAttach()** fails if there is no context current to the thread.

Currently, the **Flags** parameter must be 0.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxDetach*, *cuCtxGetDevice*, *cuCtxSynchronize*

2.3.2 cuCtxCreate

NAME

cuCtxCreate - create a CUDA context

SYNOPSIS

```
CUresult cuCtxCreate(CUcontext* pCtx, unsigned int Flags, CUdevice dev);
```

DESCRIPTION

Creates a new CUDA context and associates it with the calling thread. The **Flags** parameter is described below. The context is created with a usage count of 1 and the caller of **cuCtxCreate()** must call **cuCtxDestroy()** or **cuCtxDetach()** when done using the context. If a context is already current to the thread, it is supplanted by the newly created context and may be restored by a subsequent call to **cuCtxPopCurrent()**.

The two LSBs of the **Flags** parameter can be used to control how the OS thread which owns the CUDA context at the time of an API call interacts with the OS scheduler when waiting for results from the GPU.

- **CU_CTX_SCHED_AUTO**: The default value if the **Flags** parameter is zero, uses a heuristic based on the number of active CUDA contexts in the process C and the number of logical processors in the system P. If $C > P$ then CUDA will yield to other OS threads when waiting for the GPU, otherwise CUDA will not yield while waiting for results and actively spin on the processor.
- **CU_CTX_SCHED_SPIN**: Instruct CUDA to actively spin when waiting for results from the GPU. This can decrease latency when waiting for the GPU, but may lower the performance of CPU threads if they are performing work in parallel with the CUDA thread.
- **CU_CTX_SCHED_YIELD**: Instruct CUDA to yield its thread when waiting for results from the GPU. This can increase latency when waiting for the GPU, but can increase the performance of CPU threads performing work in parallel with the GPU.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_DEVICE

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxAttach, *cuCtxDetach*, *cuCtxDestroy*, *cuCtxPushCurrent*, *cuCtxPopCurrent*

2.3.3 cuCtxDestroy

NAME

cuCtxDestroy - destroy the current context context or a floating CuDA context

SYNOPSIS

```
CUresult cuCtxDestroy(CUcontext ctx);
```

DESCRIPTION

Destroys the given CUDA context. If the context usage count is not equal to 1, or the context is current to any CPU thread other than the current one, this function fails. Floating contexts (detached from a CPU thread via **cuCtxPopCurrent()**) may be destroyed by this function.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDetach*, *cuCtxPushCurrent*, *cuCtxPopCurrent*

2.3.4 cuCtxDetach

NAME

cuCtxDetach - decrement a context's usage-count

SYNOPSIS

```
CUresult cuCtxDetach(CUcontext ctx);
```

DESCRIPTION

Decrements the usage count of the context, and destroys the context if the usage count goes to 0. The context must be a handle that was passed back by **cuCtxCreate()** or **cuCtxAttach()**, and must be current to the calling thread.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDestroy*, *cuCtxPushCurrent*, *cuCtxPopCurrent*

2.3.5 cuCtxGetDevice

NAME

cuCtxGetDevice - return device-ID for current context

SYNOPSIS

```
CUresult cuCtxGetDevice(CUdevice* device);
```

DESCRIPTION

Returns in ***device** the ordinal of the current context's device.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDetach*, *cuCtxSynchronize*

2.3.6 cuCtxPopCurrent

NAME

cuCtxPopCurrent - pops the current CUDA context from the current CPU thread

SYNOPSIS

```
CUresult cuCtxPopCurrent(CUcontext *pctx);
```

DESCRIPTION

Pops the current CUDA context from the CPU thread. The CUDA context must have a usage count of 1. CUDA contexts have a usage count of 1 upon creation; the usage count may be incremented with **cuCtxAttach()** and decremented with **cuCtxDetach()**.

If successful, **cuCtxPopCurrent()** passes back the context handle in ***pctx**. The context may then be made current to a different CPU thread by calling **cuCtxPushCurrent()**.

Floating contexts may be destroyed by calling **cuCtxDestroy()**.

If a context was current to the CPU thread before **cuCtxCreate** or **cuCtxPushCurrent** was called, this function makes that context current to the CPU thread again.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDetach*, *cuCtxDestroy*, *cuCtxPushCurrent*

2.3.7 cuCtxPushCurrent

NAME

cuCtxPushCurrent - attach floating context to CPU thread

SYNOPSIS

```
CUresult cuCtxPushCurrent(CUcontext ctx);
```

DESCRIPTION

Pushes the given context onto the CPU thread's stack of current contexts. The specified context becomes the CPU thread's current context, so all CUDA functions that operate on the current context are affected.

The previous current context may be made current again by calling **cuCtxDestroy()** or **cuCtxPopCurrent()**.

The context must be "floating," i.e. not attached to any thread. Contexts are made to float by calling **cuCtxPopCurrent()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDetach*, *cuCtxDestroy*, *cuCtxPopCurrent*

2.3.8 cuCtxSynchronize

NAME

cuCtxSynchronize - block for a context's tasks to complete

SYNOPSIS

```
CUresult cuCtxSynchronize(void);
```

DESCRIPTION

Blocks until the device has completed all preceding requested tasks. **cuCtxSynchronize()** returns an error if one of the preceding tasks failed.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuCtxAttach*, *cuCtxDetach*, *cuCtxGetDevice*

2.4 ModuleManagement

NAME

Module Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuModuleGetFunction

cuModuleGetGlobal

cuModuleGetTexRef

cuModuleLoad

cuModuleLoadData

cuModuleLoadFatBinary

cuModuleUnload

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.4.1 cuModuleGetFunction

NAME

cuModuleGetFunction - returns a function handle

SYNOPSIS

```
CUresult cuModuleGetFunction(CUfunction* func, CUmodule mod, const char* funcname);
```

DESCRIPTION

Returns in ***func** the handle of the function of name **funcname** located in module **mod**. If no function of that name exists, **cuModuleGetFunction()** returns **CUDA_ERROR_NOT_FOUND**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NOT_FOUND

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadData*, *cuModuleLoadFatBinary*, *cuModuleUnload*, *cuModuleGetGlobal*, *cuModuleGetTexRef*

2.4.2 cuModuleGetGlobal

NAME

cuModuleGetGlobal - returns a global pointer from a module

SYNOPSIS

```
CUresult cuModuleGetGlobal(CUdeviceptr* devPtr, unsigned int* bytes, CUmodule mod, const char*
globalname);
```

DESCRIPTION

Returns in ***devPtr** and ***bytes** the base pointer and size of the global of name **globalname** located in module **mod**. If no variable of that name exists, **cuModuleGetGlobal()** returns **CUDA_ERROR_NOT_FOUND**. Both parameters devPtr and bytes are optional. If one of them is null, it is ignored.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NOT_FOUND

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadData*, *cuModuleLoadFatBinary*, *cuModuleUnload*, *cuModuleGetFunction*, *cuModuleGetTexRef*

2.4.3 cuModuleGetTexRef

NAME

cuModuleGetTexRef - gets a handle to a texture-reference

SYNOPSIS

```
CUresult cuModuleGetTexRef(CUtexref* texRef, CUmodule hmod, const char* texrefname);
```

DESCRIPTION

Returns in ***texref** the handle of the texture reference of name **texrefname** in the module **mod**. If no texture reference of that name exists, **cuModuleGetTexRef()** returns **CUDA_ERROR_NOT_FOUND**. This texture reference handle should not be destroyed, since it will be destroyed when the module is unloaded.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NOT_FOUND

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadData*, *cuModuleLoadFatBinary*, *cuModuleUnload*, *cuModuleGetFunction*, *cuModuleGetGlobal*

2.4.4 cuModuleLoad

NAME

cuModuleLoad - loads a compute module

SYNOPSIS

```
CUresult cuModuleLoad(CUmodule* mod, const char* filename);
```

DESCRIPTION

Takes a file name **filename** and loads the corresponding module **mod** into the current context. The CUDA driver API does not attempt to lazily allocate the resources needed by a module; if the memory for functions and data (constant and global) needed by the module cannot be allocated, **cuModuleLoad()** fails. The file should be a *cubin* file as output by **nvcc**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NOT_FOUND

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_FILE_NOT_FOUND

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoadData, *cuModuleLoadFatBinary*, *cuModuleUnload*, *cuModuleGetFunction*, *cuModuleGetGlobal*, *cuModuleGetTexRef*

2.4.5 cuModuleLoadData

NAME

cuModuleLoadData - loads a module's data

SYNOPSIS

```
CUresult cuModuleLoadData(CUmodule* mod, const void* image);
```

DESCRIPTION

Takes a pointer **image** and loads the corresponding module **mod** into the current context. The pointer may be obtained by mapping a *cubin* file, passing a *cubin* file as a text string, or incorporating a *cubin* object into the executable resources and using operation system calls such as WindowsâĂŹ **FindResource()** to obtain the pointer.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadFatBinary*, *cuModuleUnload*, *cuModuleGetFunction*, *cuModuleGetGlobal*, *cuModuleGetTexRef*

2.4.6 cuModuleLoadFatBinary

NAME

cuModuleLoadFatBinary - loads a fat-binary object

SYNOPSIS

```
CUresult cuModuleLoadFatBinary(CUmodule* mod, const void* fatBin);
```

DESCRIPTION

Takes a pointer **fatBin** and loads the corresponding module **mod** into the current context. The pointer represents a *fat binary* object, which is a collection of different *cubin* files, all representing the same device code but compiled and optimized for different architectures. There is currently no documented API for constructing and using fat binary objects by programmers, and therefore this function is an internal function in this version of CUDA. More information can be found in the **nvcc** document.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_NOT_FOUND

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_NO_BINARY_FOR_GPU

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadData*, *cuModuleUnload*, *cuModuleGetFunction*, *cuModuleGetGlobal*, *cuModuleGetTexRef*

2.4.7 cuModuleUnload

NAME

cuModuleUnload - unloads a module

SYNOPSIS

```
CUresult cuModuleUnload(CUmodule mod);
```

DESCRIPTION

Unloads a module mod from the current context.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuModuleLoad, *cuModuleLoadData*, *cuModuleLoadFatBinary*, *cuModuleGetFunction*, *cuModuleGetGlobal*, *cuModuleGetTexRef*

2.5 StreamManagement

NAME

Stream Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuStreamCreate

cuStreamDestroy

cuStreamQuery

cuStreamSynchronize

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.5.1 cuStreamCreate

NAME

cuStreamCreate - create a stream

SYNOPSIS

```
CUresult cuStreamCreate(CUstream* stream, unsigned int flags);
```

DESCRIPTION

Creates a stream. At present, **flags** is required to be 0.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuStreamQuery, *cuStreamSynchronize*, *cuStreamDestroy*

2.5.2 cuStreamDestroy

NAME

cuStreamDestroy - destroys a stream

SYNOPSIS

```
CUresult cuStreamDestroy(CUstream stream);
```

DESCRIPTION

Destroys the stream.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuStreamCreate, *cuStreamQuery*, *cuStreamSynchronize*

2.5.3 cuStreamQuery

NAME

cuStreamQuery - determine status of a compute stream

SYNOPSIS

```
CUresult cuStreamQuery(CUstream stream);
```

DESCRIPTION

Returns **CUDA_SUCCESS** if all operations in the stream have completed, or **CUDA_ERROR_NOT_READY** if not.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_READY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuStreamCreate, *cuStreamSynchronize*, *cuStreamDestroy*

2.5.4 cuStreamSynchronize

NAME

cuStreamSynchronize - block until a stream's tasks are completed

SYNOPSIS

```
CUresult cuStreamSynchronize(CUstream stream);
```

DESCRIPTION

Blocks until the device has completed all operations in the stream.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuStreamCreate, *cuStreamQuery*, *cuStreamDestroy*

2.6 EventManagement

NAME

Event Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuEventCreate

cuEventDestroy

cuEventElapsedTime

cuEventQuery

cuEventRecord

cuEventSynchronize

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.6.1 cuEventCreate

NAME

cuEventCreate - creates an event

SYNOPSIS

```
CUresult cuEventCreate(CUevent* event, unsigned int flags);
```

DESCRIPTION

Creates an event. At present, **flags** is required to be 0.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventRecord, *cuEventQuery*, *cuEventSynchronize*, *cuEventDestroy*, *cuEventElapsedTime*

2.6.2 cuEventDestroy

NAME

cuEventDestroy - destroys an event

SYNOPSIS

```
CUresult cuEventDestroy(CUevent event);
```

DESCRIPTION

Destroys the event.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventCreate, *cuEventRecord*, *cuEventQuery*, *cuEventSynchronize*, *cuEventElapsedTime*

2.6.3 cuEventElapsedTime

NAME

cuEventElapsedTime - computes the elapsed time between two events

SYNOPSIS

```
CUresult cuEventDestroy(float* time, CUevent start, CUevent end);
```

DESCRIPTION

Computes the elapsed time between two events (in milliseconds with a resolution of around 0.5 microseconds). If either event has not been recorded yet, this function returns **CUDA_ERROR_INVALID_VALUE**. If either event has been recorded with a non-zero stream, the result is undefined.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventCreate, *cuEventRecord*, *cuEventQuery*, *cuEventSynchronize*, *cuEventDestroy*

2.6.4 cuEventQuery

NAME

cuEventQuery - queries an event's status

SYNOPSIS

```
CUresult cuEventQuery(CUevent event);
```

DESCRIPTION

Returns **CUDA_SUCCESS** if the event has actually been recorded, or **CUDA_ERROR_NOT_READY** if not. If **cuEventRecord()** has not been called on this event, the function returns **CUDA_ERROR_INVALID_VALU**

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_READY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventCreate, *cuEventRecord*, *cuEventSynchronize*, *cuEventDestroy*, *cuEventElapsedTime*

2.6.5 cuEventRecord

NAME

cuEventRecord - records an event

SYNOPSIS

```
CUresult cuEventRecord(CUevent event, CUstream stream);
```

DESCRIPTION

Records an event. If **stream** is non-zero, the event is recorded after all preceding operations in the stream have been completed; otherwise, it is recorded after all preceding operations in the CUDA context have been completed. Since this operation is asynchronous, **cuEventQuery()** and/or **cuEventSynchronize()** must be used to determine when the event has actually been recorded.

If **cuEventRecord()** has previously been called and the event has not been recorded yet, this function returns **CUDA_ERROR_INVALID_VALUE**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventCreate, *cuEventQuery*, *cuEventSynchronize*, *cuEventDestroy*, *cuEventElapsedTime*

2.6.6 cuEventSynchronize

NAME

cuEventSynchronize - waits for an event to complete

SYNOPSIS

```
CUresult cuEventSynchronize(CUevent event);
```

DESCRIPTION

Blocks until the event has actually been recorded. If **cuEventRecord()** has not been called on this event, the function returns **CUDA_ERROR_INVALID_VALUE**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuEventCreate, *cuEventRecord*, *cuEventQuery*, *cuEventDestroy*, *cuEventElapsedTime*

2.7 ExecutionControl

NAME

Execution Control

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuLaunch

cuLaunchGrid

cuParamSetSize

cuParamSetTexRef

cuParamSetf

cuParamSeti

cuParamSetv

cuFuncSetBlockShape

cuFuncSetSharedSize

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.7.1 cuLaunch

NAME

cuLaunch - launches a CUDA function

SYNOPSIS

```
CUresult cuLaunch(CUfunction func);
```

DESCRIPTION

Invokes the kernel **func** on a 1ÃŮ1 grid of blocks. The block contains the number of threads specified by a previous call to **cuFuncSetBlockShape()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunchGrid*

2.7.2 cuLaunchGrid

NAME

cuLaunchGrid - launches a CUDA function

SYNOPSIS

```
CUresult cuLaunchGrid(CUfunction func, int grid_width, int grid_height);

CUresult cuLaunchGridAsync(CUfunction func, int grid_width, int grid_height, CUstream stream);
```

DESCRIPTION

Invokes the kernel on a **grid_width** x **grid_height** grid of blocks. Each block contains the number of threads specified by a previous call to **cuFuncSetBlockShape()**.

cuLaunchGridAsync() can optionally be associated to a stream by passing a non-zero **stream** argument.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_LAUNCH_INCOMPATIBLE_TEXTURING

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*

2.7.3 cuParamSetSize

NAME

cuParamSetSize - sets the parameter-size for the function

SYNOPSIS

```
CUresult cuParamSetSize(CUfunction func, unsigned int numbytes);
```

DESCRIPTION

Sets through **numbytes** the total size in bytes needed by the function parameters of function **func**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.7.4 cuParamSetTexRef

NAME

cuParamSetTexRef - adds a texture-reference to the function's argument list

SYNOPSIS

```
CUresult cuParamSetTexRef(CUfunction func, int texunit, CUtexref texRef);
```

DESCRIPTION

Makes the CUDA array or linear memory bound to the texture reference **texRef** available to a device program as a texture. In this version of CUDA, the texture reference must be obtained via **cuModuleGetTexRef()** and the **texunit** parameter must be set to **CU_PARAM_TR_DEFAULT**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuLaunch*, *cuLaunchGrid*

2.7.5 cuParamSetf

NAME

cuParamSetf - adds a floating-point parameter to the function's argument list

SYNOPSIS

```
CUresult cuParamSetf(CUfunction func, int offset, float value);
```

DESCRIPTION

Sets a floating point parameter that will be specified the next time the kernel corresponding to **func** will be invoked. **offset** is a byte offset.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSeti*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.7.6 cuParamSeti

NAME

cuParamSeti - adds an integer parameter to the function's argument list

SYNOPSIS

```
CUresult cuParamSeti(CUfunction func, int offset, unsigned int value);
```

DESCRIPTION

Sets an integer parameter that will be specified the next time the kernel corresponding to **func** will be invoked. **offset** is a byte offset.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.7.7 cuParamSetv

NAME

cuParamSetv - adds arbitrary data to the function's argument list

SYNOPSIS

```
CUresult cuParamSetv(CUfunction func, int offset, void* ptr, unsigned int numbytes);
```

DESCRIPTION

Copies an arbitrary amount of data into the parameter space of the kernel corresponding to **func**. **offset** is a byte offset.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuFuncSetSharedSize*, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.7.8 cuFuncSetBlockShape

NAME

cuFuncSetBlockShape - sets the block-dimensions for the function

SYNOPSIS

```
CUresult cuFuncSetBlockShape(CUfunction func, int x, int y, int z);
```

DESCRIPTION

Specifies the X, Y and Z dimensions of the thread blocks that are created when the kernel given by **func** is launched.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetSharedSize, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.7.9 cuFuncSetSharedSize

NAME

cuFuncSetSharedSize - sets the shared-memory size for the function

SYNOPSIS

```
CUresult cuFuncSetSharedSize(CUfunction func, unsigned int bytes);
```

DESCRIPTION

Sets through **bytes** the amount of shared memory that will be available to each thread block when the kernel given by **func** is launched.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuFuncSetBlockShape, *cuParamSetSize*, *cuParamSeti*, *cuParamSetf*, *cuParamSetv*, *cuParamSetTexRef*, *cuLaunch*, *cuLaunchGrid*

2.8 MemoryManagement

NAME

Memory Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuArrayCreate

cuArray3DCreate

cuArrayDestroy

cuArrayGetDescriptor

cuArray3DGetDescriptor

cuMemAlloc

cuMemAllocHost

cuMemAllocPitch

cuMemFree

cuMemFreeHost

cuMemGetAddressRange

cuMemGetInfo

cuMemcpy2D

cuMemcpy3D

cuMemcpyAtoA

cuMemcpyAtoD

cuMemcpyAtoH

cuMemcpyDtoA

cuMemcpyDtoD

cuMemcpyDtoH

cuMemcpyHtoA

cuMemcpyHtoD

cuMemset

cuMemset2D

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.8.1 cuArrayCreate

NAME

cuArrayCreate - creates a 1D or 2D CUDA array

SYNOPSIS

```
CUresult cuArrayCreate(CUarray* array, const CUDA_ARRAY_DESCRIPTOR* desc);
```

DESCRIPTION

Creates a CUDA array according to the **CUDA_ARRAY_DESCRIPTOR** structure **desc** and returns a handle to the new CUDA array in ***array**. The **CUDA_ARRAY_DESCRIPTOR** structure is defined as such:

```
typedef struct {
   unsigned int Width;
   unsigned int Height;
   CUarray_format Format;
   unsigned int NumChannels;
} CUDA_ARRAY_DESCRIPTOR;
```

where:

- **Width** and **Height** are the width and height of the CUDA array (in elements); the CUDA array is one-dimensional if height is 0, two-dimensional, otherwise;
- **NumChannels** specifies the number of packed components per CUDA array element.; it may be 1, 2 or 4;
- **Format** specifies the format of the elements; **CUarray_format** is defined as such:

```
typedef enum CUarray_format_enum {
   CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
   CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
   CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
   CU_AD_FORMAT_SIGNED_INT8 = 0x08,
   CU_AD_FORMAT_SIGNED_INT16 = 0x09,
   CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
   CU_AD_FORMAT_HALF = 0x10,
   CU_AD_FORMAT_FLOAT = 0x20
} CUarray_format;
```

Here are examples of CUDA array descriptions:

- Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
```

```
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 1;
```

- Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
```

- Description for a **width x height** CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
desc.Height = height;
```

- Description for a **width x height** CUDA array of 16-bit elements, each of which is two 8-bit unsigned chars:

```
CUDA_ARRAY_DESCRIPTOR arrayDesc;
desc.FormatFlags = CU_AD_FORMAT_UNSIGNED_INTS;
desc.NumChannels = 2;
desc.Width = width;
desc.Height = height;
```

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.2 cuArray3DCreate

NAME

cuArray3DCreate - creates a CUDA array

SYNOPSIS

```
CUresult cuArray3DCreate(CUarray* array, const CUDA_ARRAY3D_DESCRIPTOR* desc);
```

DESCRIPTION

Creates a CUDA array according to the **CUDA_ARRAY3D_DESCRIPTOR** structure **desc** and returns a handle to the new CUDA array in ***array**. The **CUDA_ARRAY3D_DESCRIPTOR** structure is defined as such:

```
typedef struct {
   unsigned int Width;
   unsigned int Height;
   unsigned int Depth;
   CUarray_format Format;
   unsigned int NumChannels;
   unsigned int Flags;
} CUDA_ARRAY3D_DESCRIPTOR;
```

where:

- **Width**, **Height** and **Depth** are the width, height and depth of the CUDA array (in elements); the CUDA array is one-dimensional if height and depth are 0, two-dimensional if depth is 0, and three-dimensional otherwise;
- **NumChannels** specifies the number of packed components per CUDA array element.; it may be 1, 2 or 4;
- **Format** specifies the format of the elements; **CUarray_format** is defined as such:

  ```
  typedef enum CUarray_format_enum {
     CU_AD_FORMAT_UNSIGNED_INT8 = 0x01,
     CU_AD_FORMAT_UNSIGNED_INT16 = 0x02,
     CU_AD_FORMAT_UNSIGNED_INT32 = 0x03,
     CU_AD_FORMAT_SIGNED_INT8 = 0x08,
     CU_AD_FORMAT_SIGNED_INT16 = 0x09,
     CU_AD_FORMAT_SIGNED_INT32 = 0x0a,
     CU_AD_FORMAT_HALF = 0x10,
     CU_AD_FORMAT_FLOAT = 0x20
  } CUarray_format;
  ```

- **Flags** provides for future features. For now, it must be set to 0.

Here are examples of CUDA array descriptions:

- Description for a CUDA array of 2048 floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 2048;
desc.Height = 0;
desc.Depth = 0;
```

- Description for a 64 x 64 CUDA array of floats:

```
CUDA_ARRAY3D_DESCRIPTOR desc;
desc.Format = CU_AD_FORMAT_FLOAT;
desc.NumChannels = 1;
desc.Width = 64;
desc.Height = 64;
desc.Depth = 0;
```

- Description for a **width x height x depth** CUDA array of 64-bit, 4x16-bit float16's:

```
CUDA_ARRAY_DESCRIPTOR desc;
desc.FormatFlags = CU_AD_FORMAT_HALF;
desc.NumChannels = 4;
desc.Width = width;
desc.Height = height;
desc.Depth = depth;
```

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArray3DGetDescriptor*, *cuArrayDestroy*

2.8.3 cuArrayDestroy

NAME

cuArrayDestroy - destroys a CUDA array

SYNOPSIS

```
CUresult cuArrayDestroy(CUarray array);
```

DESCRIPTION

Destroys the CUDA array **array**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_ARRAY_IS_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuMemset*, *cuMemset2D*

2.8.4 cuArrayGetDescriptor

NAME

cuArrayGetDescriptor - get a 1D or 2D CUDA array descriptor

SYNOPSIS

```
CUresult cuArrayGetDescriptor(CUDA_ARRAY_DESCRIPTOR* arrayDesc, CUarray array)
```

DESCRIPTION

Returns in ***arrayDesc** a descriptor of the format and dimensions of the 1D or 2D CUDA array **array**. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuArrayCreate, *cuArray3DCreate*, *cuArray3DGetDescriptor*, *cuArrayDestroy*

2.8.5 cuArray3DGetDescriptor

NAME

cuArray3DGetDescriptor - get a 3D CUDA array descriptor

SYNOPSIS

```
CUresult cuArray3DGetDescriptor(CUDA_ARRAY3D_DESC *arrayDesc, CUarray array);
```

DESCRIPTION

Returns in ***arrayDesc** a descriptor containing information on the format and dimensions of the CUDA array **array**. It is useful for subroutines that have been passed a CUDA array, but need to know the CUDA array parameters for validation or other purposes.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuArrayCreate, *cuArray3DCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*

2.8.6 cuMemAlloc

NAME

cuMemAlloc - allocates device memory

SYNOPSIS

```
CUresult cuMemAlloc(CUdeviceptr* devPtr, unsigned int count);
```

DESCRIPTION

Allocates **count** bytes of linear memory on the device and returns in ***devPtr** a pointer to the allocated memory. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. If **count** is 0, **cuMemAlloc()** returns **CUDA_ERROR_INVALID_VALUE**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.7 cuMemAllocHost

NAME

cuMemAllocHost - allocates page-locked host memory

SYNOPSIS

```
CUresult cuMemAllocHost(void** hostPtr, unsigned int count);
```

DESCRIPTION

Allocates count bytes of host memory that is page-locked and accessible to the device. The driver tracks the virtual memory ranges allocated with this function and automatically accelerates calls to functions such as **cuMemcpy()**. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as **malloc()**. Allocating excessive amounts of memory with **cuMemAllocHost()** may degrade system performance, since it reduces the amount of memory available to the system for paging. As a result, this function is best used sparingly to allocate staging areas for data exchange between host and device.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.8 cuMemAllocPitch

NAME

cuMemAllocPitch - allocates device memory

SYNOPSIS

```
CUresult cuMemAllocPitch(CUdeviceptr* devPtr, unsigned int* pitch, unsigned int widthInBytes,
unsigned int height, unsigned int elementSizeBytes);
```

DESCRIPTION

Allocates at least **widthInBytes*height** bytes of linear memory on the device and returns in ***devPtr** a pointer to the allocated memory. The function may pad the allocation to ensure that corresponding pointers in any given row will continue to meet the alignment requirements for coalescing as the address is updated from row to row. **elementSizeBytes** specifies the size of the largest reads and writes that will be performed on the memory range. **elementSizeBytes** may be 4, 8 or 16 (since coalesced memory transactions are not possible on other data sizes). If **elementSizeBytes** is smaller than the actual read/write size of a kernel, the kernel will run correctly, but possibly at reduced speed. The pitch returned in ***pitch** by **cuMemAllocPitch()** is the width in bytes of the allocation. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type **T**, the address is computed as

```
T* pElement = (T*)((char*)BaseAddress + Row * Pitch) + Column;
```

The pitch returned by **cuMemAllocPitch()** is guaranteed to work with **cuMemcpy2D()** under all circumstances. For allocations of 2D arrays, it is recommended that programmers consider performing pitch allocations using **cuMemAllocPitch()**. Due to alignment restrictions in the hardware, this is especially true if the application will be performing 2D memory copies between different regions of device memory (whether linear memory or CUDA arrays).

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.9 cuMemFree

NAME

cuMemFree - frees device memory

SYNOPSIS

```
CUresult cuMemFree(CUdeviceptr devPtr);
```

DESCRIPTION

Frees the memory space pointed to by **devPtr**, which must have been returned by a previous call to **cuMemMalloc()** or **cuMemMallocPitch()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.10 cuMemFreeHost

NAME

cuMemFreeHost - frees page-locked host memory

SYNOPSIS

```
CUresult cuMemFreeHost(void* hostPtr);
```

DESCRIPTION

Frees the memory space pointed to by **hostPtr**, which must have been returned by a previous call to **cuMemAllocHost()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.11 cuMemGetAddressRange

NAME

cuMemGetAddressRange - get information on memory allocations

SYNOPSIS

```
CUresult cuMemGetAddressRange(CUdeviceptr* basePtr, unsigned int* size, CUdeviceptr devPtr);
```

DESCRIPTION

Returns the base address in ***basePtr** and size and ***size** of the allocation by **cuMemAlloc()** or **cuMemAllocPitch()** that contains the input pointer **devPtr**. Both parameters **basePtr** and **size** are optional. If one of them is null, it is ignored.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.12 cuMemGetInfo

NAME

cuMemGetInfo - gets free and total memory

SYNOPSIS

```
CUresult cuMemGetInfo(unsigned int* free, unsigned int* total);
```

DESCRIPTION

Returns in ***free** and ***total** respectively, the free and total amount of memory available for allocation by the CUDA context, in bytes.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemAlloc, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*, *cuMemset2D*

2.8.13 cuMemcpy2D

NAME

cuMemcpy2D - copies memory for 2D arrays

SYNOPSIS

```
CUresult cuMemcpy2D(const CUDA_MEMCPY2D* copyParam);

CUresult cuMemcpy2DUnaligned(const CUDA_MEMCPY2D* copyParam);

CUresult cuMemcpy2DAsync(const CUDA_MEMCPY2D* copyParam, CUstream stream);
```

DESCRIPTION

Perform a 2D memory copy according to the parameters specified in **copyParam**. The **CUDA_MEMCPY2D** structure is defined as such:

```
typedef struct CUDA_MEMCPY2D_st {
   unsigned int srcXInBytes, srcY;
   CUmemorytype srcMemoryType;
   const void *srcHost;
   CUdeviceptr srcDevice;
   CUarray srcArray;
   unsigned int srcPitch;
   unsigned int dstXInBytes, dstY;
   CUmemorytype dstMemoryType;
   void *dstHost;
   CUdeviceptr dstDevice;
   CUarray dstArray;
   unsigned int dstPitch;
   unsigned int WidthInBytes;
   unsigned int Height;
} CUDA_MEMCPY2D;
```

where:

- **srcMemoryType** and **dstMemoryType** specify the type of memory of the source and destination, respectively; **CUmemorytype_enum** is defined as such:

  ```
  typedef enum CUmemorytype_enum {
     CU_MEMORYTYPE_HOST = 0x01,
     CU_MEMORYTYPE_DEVICE = 0x02,
     CU_MEMORYTYPE_ARRAY = 0x03
  } CUmemorytype;
  ```

  If **srcMemoryType** is **CU_MEMORYTYPE_HOST**, **srcHost** and **srcPitch** specify the (host) base address of the source data and the bytes per row to apply. **srcArray** is ignored.

  If **srcMemoryType** is **CU_MEMORYTYPE_DEVICE**, **srcDevice** and **srcPitch** specify the (device) base address of the source data and the bytes per row to apply. **srcArray** is ignored.

If **srcMemoryType** is **CU_MEMORYTYPE_ARRAY**, **srcArray** specifies the handle of the source data. **srcHost**, **srcDevice** and **srcPitch** are ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_HOST**, **dstHost** and **dstPitch** specify the (host) base address of the destination data and the bytes per row to apply. **dstArray** is ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_DEVICE**, **dstDevice** and **dstPitch** specify the (device) base address of the destination data and the bytes per row to apply. **dstArray** is ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_ARRAY**, **dstArray** specifies the handle of the destination data. **dstHost**, **dstDevice** and **dstPitch** are ignored.

- **srcXInBytes** and **srcY** specify the base address of the source data for the copy.

  For host pointers, the starting address is

  ```
  void* Start = (void*)((char*)srcHost+srcY*srcPitch + srcXInBytes);
  ```

  For device pointers, the starting address is

  ```
  CUdeviceptr Start = srcDevice+srcY*srcPitch+srcXInBytes;
  ```

  For CUDA arrays, **srcXInBytes** must be evenly divisible by the array element size.

- **dstXInBytes** and **dstY** specify the base address of the destination data for the copy.

  For host pointers, the base address is

  ```
  void* dstStart = (void*)((char*)dstHost+dstY*dstPitch + dstXInBytes);
  ```

  For device pointers, the starting address is

  ```
  CUdeviceptr dstStart = dstDevice+dstY*dstPitch+dstXInBytes;
  ```

  For CUDA arrays, **dstXInBytes** must be evenly divisible by the array element size.

- **WidthInBytes** and **Height** specify the width (in bytes) and height of the 2D copy being performed. Any pitches must be greater than or equal to **WidthInBytes**.

  cuMemcpy2D() returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_**
  cuMemAllocPitch() passes back pitches that always work with **cuMemcpy2D()**. On intra-device memory copies (device ? device, CUDA array ? device, CUDA array ? CUDA array), **cuMemcpy2D()** may fail for pitches not computed by **cuMemAllocPitch()**. **cuMemcpy2DUnaligned()** does not have this restriction, but may run significantly slower in the cases where **cuMemcpy2D()** would have returned an error code.

  cuMemcpy2DAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy3D*

2.8.14 cuMemcpy3D

NAME

cuMemcpy3D - copies memory for 3D arrays

SYNOPSIS

```
CUresult cuMemcpy3D(const CUDA_MEMCPY3D* copyParam);

CUresult cuMemcpy3DAsync(const CUDA_MEMCPY3D* copyParam, CUstream stream);
```

DESCRIPTION

Perform a 3D memory copy according to the parameters specified in **copyParam**. The **CUDA_MEMCPY3D** structure is defined as such:

```
typedef struct CUDA_MEMCPY3D_st {

    unsigned int srcXInBytes, srcY, srcZ;
    unsigned int srcLOD;
    CUmemorytype srcMemoryType;
        const void *srcHost;
        CUdeviceptr srcDevice;
        CUarray srcArray;
        unsigned int srcPitch;  // ignored when src is array
        unsigned int srcHeight; // ignored when src is array; may be 0 if Depth==1

    unsigned int dstXInBytes, dstY, dstZ;
    unsigned int dstLOD;
    CUmemorytype dstMemoryType;
        void *dstHost;
        CUdeviceptr dstDevice;
        CUarray dstArray;
        unsigned int dstPitch;  // ignored when dst is array
        unsigned int dstHeight; // ignored when dst is array; may be 0 if Depth==1

    unsigned int WidthInBytes;
    unsigned int Height;
    unsigned int Depth;
} CUDA_MEMCPY3D;
CUresult  CUDAAPI cuMemcpy3D( const CUDA_MEMCPY3D *pCopy );
```

where:

- **srcMemoryType** and **dstMemoryType** specify the type of memory of the source and destination, respectively; **CUmemorytype_enum** is defined as such:

  ```
  typedef enum CUmemorytype_enum {
  ```

```
    CU_MEMORYTYPE_HOST = 0x01,
    CU_MEMORYTYPE_DEVICE = 0x02,
    CU_MEMORYTYPE_ARRAY = 0x03
} CUmemorytype;
```

If **srcMemoryType** is **CU_MEMORYTYPE_HOST**, **srcHost**, **srcPitch** and **srcHeight** specify the (host) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. **srcArray** is ignored.

If **srcMemoryType** is **CU_MEMORYTYPE_DEVICE**, **srcDevice**, **srcPitch** and **srcHeight** specify the (device) base address of the source data, the bytes per row, and the height of each 2D slice of the 3D array. **srcArray** is ignored.

If **srcMemoryType** is **CU_MEMORYTYPE_ARRAY**, **srcArray** specifies the handle of the source data. **srcHost**, **srcDevice**, **srcPitch** and **srcHeight** are ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_HOST**, **dstHost** and **dstPitch** specify the (host) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. **dstArray** is ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_DEVICE**, **dstDevice** and **dstPitch** specify the (device) base address of the destination data, the bytes per row, and the height of each 2D slice of the 3D array. **dstArray** is ignored.

If **dstMemoryType** is **CU_MEMORYTYPE_ARRAY**, **dstArray** specifies the handle of the destination data. **dstHost**, **dstDevice**, **dstPitch** and **dstHeight** are ignored.

- **srcXInBytes**, **srcY** and **srcZ** specify the base address of the source data for the copy.

  For host pointers, the starting address is

  ```
  void* Start = (void*)((char*)srcHost+(srcZ*srcHeight+srcY)*srcPitch + srcXInBytes);
  ```

  For device pointers, the starting address is

  ```
  CUdeviceptr Start = srcDevice+(srcZ*srcHeight+srcY)*srcPitch+srcXInBytes;
  ```

  For CUDA arrays, **srcXInBytes** must be evenly divisible by the array element size.

- **dstXInBytes**, **dstY** and **dstZ** specify the base address of the destination data for the copy.

  For host pointers, the base address is

  ```
  void* dstStart = (void*)((char*)dstHost+(dstZ*dstHeight+dstY)*dstPitch + dstXInBytes);
  ```

  For device pointers, the starting address is

  ```
  CUdeviceptr dstStart = dstDevice+(dstZ*dstHeight+dstY)*dstPitch+dstXInBytes;
  ```

  For CUDA arrays, **dstXInBytes** must be evenly divisible by the array element size.

- **WidthInBytes**, **Height** and **Depth** specify the width (in bytes), height and depth of the 3D copy being performed. Any pitches must be greater than or equal to **WidthInBytes**.

  cuMemcpy3D() returns an error if any pitch is greater than the maximum allowed (**CU_DEVICE_ATTRIBUTE_**

  cuMemcpy3DAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

  The **srcLOD** and **dstLOD** members of the **CUDA_MEMCPY3D** structure must be set to 0.

RETURN VALUE

CUDA_SUCCESS

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*, *cuMemcpy2DAsync*

2.8.15 cuMemcpyAtoA

NAME

cuMemcpyAtoA - copies memory from Array to Array

SYNOPSIS

```
CUresult cuMemcpyAtoA(CUarray dstArray, unsigned int dstIndex, CUarray srcArray, unsigned int
srcIndex, unsigned int count);
```

DESCRIPTION

Copies from one 1D CUDA array to another. **dstArray** and **srcArray** specify the handles of the destination and source CUDA arrays for the copy, respectively. **dstIndex** and **srcIndex** specify the destination and source indices into the CUDA array. These values are in the range **[0, Width-1]** for the CUDA array; they are not byte offsets. **count** is the number of bytes to be copied. The size of the elements in the CUDA arrays need not be the same format, but the elements must be the same size; and count must be evenly divisible by that size.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpy2D*

2.8.16 cuMemcpyAtoD

NAME

cuMemcpyAtoD - copies memory from Array to Device

SYNOPSIS

```
CUresult cuMemcpyAtoD(CUdeviceptr dstDevPtr, CUarray srcArray, unsigned int srcIndex, unsigned
int count);
```

DESCRIPTION

Copies from a 1D CUDA array to device memory. **dstDevPtr** specifies the base pointer of the destination and must be naturally aligned with the CUDA array elements. **srcArray** and **srcIndex** specify the CUDA array handle and the index (in array elements) of the array element where the copy is to begin. **count** specifies the number of bytes to copy and must be evenly divisible by the array element size.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.17 cuMemcpyAtoH

NAME

cuMempcyAtoH - copies memory from Array to Host

SYNOPSIS

```
CUresult cuMemcpyAtoH(void* dstHostPtr, CUarray srcArray, unsigned int srcIndex, unsigned int
count);
```

```
CUresult cuMemcpyAtoHAsync(void* dstHostPtr, CUarray srcArray, unsigned int srcIndex, unsigned
int count, CUstream stream);
```

DESCRIPTION

Copies from a 1D CUDA array to host memory. **dstHostPtr** specifies the base pointer of the destination. **srcArray** and **srcIndex** specify the CUDA array handle and starting index of the source data. **count** specifies the number of bytes to copy.

cuMemcpyAtoHAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.18 cuMemcpyDtoA

NAME

cuMemcpyDtoA - copies memory from Device to Array

SYNOPSIS

```
CUresult cuMemcpyDtoA(CUarray dstArray, unsigned int dstIndex, CUdeviceptr srcDevPtr, unsigned
int count);
```

DESCRIPTION

Copies from device memory to a 1D CUDA array. **dstArray** and **dstIndex** specify the CUDA array handle and starting index of the destination data. **srcDevPtr** specifies the base pointer of the source. **count** specifies the number of bytes to copy.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.19 cuMemcpyDtoD

NAME

cuMemcpyDtoD - copies memory from Device to Device

SYNOPSIS

```
CUresult cuMemcpyDtoD(CUdeviceptr dstDevPtr, CUdeviceptr srcDevPtr, unsigned int count);
```

DESCRIPTION

Copies from device memory to device memory. **dstDevice** and **srcDevPtr** are the base pointers of the destination and source, respectively. **count** specifies the number of bytes to copy.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.20 cuMemcpyDtoH

NAME

cuMemcpyDtoH - copies memory from Device to Host

SYNOPSIS

```
CUresult cuMemcpyDtoH(void* dstHostPtr, CUdeviceptr srcDevPtr, unsigned int count);

CUresult cuMemcpyDtoHAsync(void* dstHostPtr, CUdeviceptr srcDevPtr, unsigned int count, CUstream
stream);
```

DESCRIPTION

Copies from device to host memory. **dstHostPtr** and **srcDevPtr**specify the base addresses of the source and destination, respectively. **count**specifies the number of bytes to copy.

MemcpyDtoHAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.21 cuMemcpyHtoA

NAME

cuMemcpyHtoA - copies memory from Host to Array

SYNOPSIS

```
CUresult cuMemcpyHtoA(CUarray dstArray, unsigned int dstIndex, const void *srcHostPtr, unsigned int count);

CUresult cuMemcpyHtoAAsync(CUarray dstArray, unsigned int dstIndex, const void *srcHostPtr, unsigned int count, CUstream stream);
```

DESCRIPTION

Copies from host memory to a 1D CUDA array. **dstArray** and **dstIndex** specify the CUDA array handle and starting index of the destination data. **srcHostPtr** specify the base addresse of the source. **count** specifies the number of bytes to copy.

cuMemcpyHtoAAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyHtoD, *cuMemcpyDtoH*, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.22 cuMemcpyHtoD

NAME

cuMemcpyHtoD - copy memory from Host to Device

SYNOPSIS

```
CUresult cuMemcpyHtoD(CUdeviceptr dstDevPtr, const void *srcHostPtr, unsigned int count);

CUresult cuMemcpyHtoDAsync(CUdeviceptr dstDevPtr, const void *srcHostPtr, unsigned int count,
CUstream stream);
```

DESCRIPTION

Copies from host memory to device memory. **dstDevPtr** and **srcHostPtr** specify the base addresses of the destination and source, respectively. **count** specifies the number of bytes to copy.

cuMemcpyHtoDAsync() is asynchronous and can optionally be associated to a stream by passing a non-zero **stream** argument. It only works on page-locked host memory and returns an error if a pointer to pageable memory is passed as input.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemcpyDtoH, *cuMemcpyDtoD*, *cuMemcpyDtoA*, *cuMemcpyAtoD*, *cuMemcpyAtoH*, *cuMemcpyHtoA*, *cuMemcpyAtoA*, *cuMemcpy2D*

2.8.23 cuMemset

NAME

cuMemset - initializes device memory

SYNOPSIS

```
CUresult cuMemsetD8(CUdeviceptr dstDevPtr, unsigned char value, unsigned int count );
CUresult cuMemsetD16(CUdeviceptr dstDevPtr, unsigned short value, unsigned int count );
CUresult cuMemsetD32(CUdeviceptr dstDevPtr, unsigned int value, unsigned int count );
```

DESCRIPTION

Sets the memory range of count 8-, 16-, or 32-bit values to the specified value **value**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset2D*

2.8.24 cuMemset2D

NAME

cuMemset2D - initializes device memory

SYNOPSIS

```
CUresult cuMemsetD2D8(CUdeviceptr dstDevPtr, unsigned int dstPitch, unsigned char value, unsigned int width, unsigned int height );

CUresult cuMemsetD2D16(CUdeviceptr dstDevPtr, unsigned int dstPitch, unsigned short value, unsigned int width, unsigned int height );

CUresult cuMemsetD2D32(CUdeviceptr dstDevPtr, unsigned int dstPitch, unsigned int value, unsigned int width, unsigned int height );
```

DESCRIPTION

Sets the 2D memory range of **width** 8-, 16-, or 32-bit values to the specified value **value**. **height** specifies the number of rows to set, and **dstPitch** specifies the number of bytes between each row. These functions perform fastest when the pitch is one that has been passed back by **cuMemAllocPitch()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuMemGetInfo, *cuMemAlloc*, *cuMemAllocPitch*, *cuMemFree*, *cuMemAllocHost*, *cuMemFreeHost*, *cuMemGetAddressRange*, *cuArrayCreate*, *cuArrayGetDescriptor*, *cuArrayDestroy*, *cuMemset*

2.9 TextureReferenceManagement

NAME

Texture Reference Management

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuTexRefCreate

cuTexRefDestroy

cuTexRefGetAddress

cuTexRefGetAddressMode

cuTexRefGetArray

cuTexRefGetFilterMode

cuTexRefGetFlags

cuTexRefGetFormat

cuTexRefSetAddress

cuTexRefSetAddressMode

cuTexRefSetArray

cuTexRefSetFilterMode

cuTexRefSetFlags

cuTexRefSetFormat

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGlInteroperability*, *Direct3dInteroperability*

2.9.1 cuTexRefCreate

NAME

cuTexRefCreate - creates a texture-reference

SYNOPSIS

```
CUresult cuTexRefCreate(CUtexref* texRef);
```

DESCRIPTION

Creates a texture reference and returns its handle in ***texRef**. Once created, the application must call **cuTexRefSetArray()** or **cuTexRefSetAddress()** to associate the reference with allocated memory. Other texture reference functions are used to specify the format and interpretation (addressing, filtering, etc.) to be used when the memory is read through this texture reference. To associate the texture reference with a texture ordinal for a given function, the application should call **cuParamSetTexRef()**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefDestroy, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.2 cuTexRefDestroy

NAME

cuTexRefDestroy - destroys a texture-reference

SYNOPSIS

```
CUresult cuTexRefDestroy(CUtexref texRef);
```

DESCRIPTION

Destroys the texture reference.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.3 cuTexRefGetAddress

NAME

cuTexRefGetAddress - gets the address associated with a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetAddress(CUdeviceptr* devPtr, CUtexref texRef);
```

DESCRIPTION

Returns in ***devPtr** the base address bound to the texture reference **texRef**, or returns **CUDA_ERROR_INVALID_VA** if the texture reference is not bound to any device memory range.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.4 cuTexRefGetAddressMode

NAME

cuTexRefGetAddressMode - gets the addressing mode used by a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetAddressMode(CUaddress_mode* mode, CUtexref texRef, int dim);
```

DESCRIPTION

Returns in ***mode** the addressing mode corresponding to the dimension **dim** of the texture reference **texRef**. Currently the only valid values for **dim** are 0 and 1.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.5 cuTexRefGetArray

NAME

cuTexRefGetArray - gets the array bound to a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetArray(CUarray* array, CUtexref texRef);
```

DESCRIPTION

Returns in ***array** the CUDA array bound by the texture reference **texRef**, or returns **CUDA_ERROR_INVALID_VA** if the texture reference is not bound to any CUDA array.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.6 cuTexRefGetFilterMode

NAME

cuTexRefGetFilterMode - gets the filter-mode used by a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetFilterMode(CUfilter_mode* mode, CUtexref texRef);
```

DESCRIPTION

Returns in ***mode** the filtering mode of the texture reference **texRef**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.7 cuTexRefGetFlags

NAME

cuTexRefGetFlags - gets the flags used by a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetFlags(unsigned int* flags, CUtexref texRef);
```

DESCRIPTION

Returns in ***flags** the flags of the texture reference **texRef**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*

2.9.8 cuTexRefGetFormat

NAME

cuTexRefGetFormat - gets the format used by a texture-reference

SYNOPSIS

```
CUresult cuTexRefGetFormat(CUarray_format* format, int* numPackedComponents, CUtexref texRef);
```

DESCRIPTION

Returns in ***format** and ***numPackedComponents** the format and number of components of the CUDA array bound to the texture reference **texRef**. If **format** or **numPackedComponents** is null, it will be ignored.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFlags*

2.9.9 cuTexRefSetAddress

NAME

cuTexRefSetAddress - binds an address as a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetAddress(unsigned int* byteOffset, CUtexref texRef, CUdeviceptr devPtr,
int bytes);
```

DESCRIPTION

Binds a linear address range to the texture reference **texRef**. Any previous address or CUDA array state associated with the texture reference is superseded by this function. Any memory previously bound to **texRef** is unbound.

Since the hardware enforces an alignment requirement on texture base addresses, **cuTexRefSetAddress()** passes back a byte offset in ***byteOffset** that must be applied to texture fetches in order to read from the desired memory. This offset must be divided by the texel size and passed to kernels that read from the texture so they can be applied to the **tex1Dfetch()** function.

If the device memory pointer was returned from **cuMemAlloc()**, the offset is guaranteed to be 0 and NULL may be passed as the **ByteOffset** parameter.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.10 cuTexRefSetAddressMode

NAME

cuTexRefSetAddressMode - set the addressing mode for a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetAddressMode(CUtexref texRef, int dim, CUaddress_mode mode);
```

DESCRIPTION

Specifies the addressing mode **mode** for the given dimension of the texture reference **texRef**. If **dim** is zero, the addressing mode is applied to the first parameter of the functions used to fetch from the texture; if **dim** is 1, the second, and so on. **CUaddress_mode** is defined as such:

```
typedef enum CUaddress_mode_enum {
   CU_TR_ADDRESS_MODE_WRAP = 0,
   CU_TR_ADDRESS_MODE_CLAMP = 1,
   CU_TR_ADDRESS_MODE_MIRROR = 2,
} CUaddress_mode;
```

Note that this call has no effect if **texRef** is bound to linear memory.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.11 cuTexRefSetArray

NAME

cuTexRefSetArray - binds an array to a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetArray(CUtexref texRef, CUarray array, unsigned int flags);
```

DESCRIPTION

Binds the CUDA array **array** to the texture reference **texRef**. Any previous address or CUDA array state associated with the texture reference is superseded by this function. **flags** must be set to **CU_TRSA_OVERRIDE_FORMAT**. Any CUDA array previously bound to **texRef** is unbound.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.12 cuTexRefSetFilterMode

NAME

cuTexRefSetFilterMode - sets the mode for a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetFilterMode(CUtexref texRef, CUfilter_mode mode);
```

DESCRIPTION

Specifies the filtering mode **mode** to be used when reading memory through the texture reference texRef. **CUfilter_mode_enum** is defined as such:

```
typedef enum CUfilter_mode_enum {
   CU_TR_FILTER_MODE_POINT = 0,
   CU_TR_FILTER_MODE_LINEAR = 1
} CUfilter_mode;
```

Note that this call has no effect if **texRef** is bound to linear memory.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.13 cuTexRefSetFlags

NAME

cuTexRefSetFlags - sets flags for a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetFlags(CUtexref texRef, unsigned int Flags);
```

DESCRIPTION

Specifies optional flags to control the behavior of data returned through the texture reference. The valid flags are:

- **CU_TRSF_READ_AS_INTEGER**, which suppresses the default behavior of having the texture promote integer data to floating point data in the range [0, 1];
- **CU_TRSF_NORMALIZED_COORDINATES**, which suppresses the default behavior of having the texture coordinates range from [0, Dim) where Dim is the width or height of the CUDA array. Instead, the texture coordinates [0, 1.0) reference the entire breadth of the array dimension

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetFormat*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.9.14 cuTexRefSetFormat

NAME

cuTexRefSetFormat - sets the format for a texture-reference

SYNOPSIS

```
CUresult cuTexRefSetFormat(CUtexref texRef, CUarray_format format, int numPackedComponents)
```

DESCRIPTION

Specifies the format of the data to be read by the texture reference **texRef**. **format** and **numPackedComponents** are exactly analogous to the **Format** and **NumChannels** members of the **CUDA_ARRAY_DESCRIPTOR** structure: They specify the format of each component and the number of components per array element.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuTexRefCreate, *cuTexRefDestroy*, *cuTexRefSetArray*, *cuTexRefSetAddress*, *cuTexRefSetAddressMode*, *cuTexRefSetFilterMode*, *cuTexRefSetFlags*, *cuTexRefGetAddress*, *cuTexRefGetArray*, *cuTexRefGetAddressMode*, *cuTexRefGetFilterMode*, *cuTexRefGetFormat*, *cuTexRefGetFlags*

2.10 OpenGlInteroperability

NAME

OpenGL Interoperability

DESCRIPTION

This section describes the low-level CUDA driver application programming interface.

cuGLCtxCreate

cuGLInit

cuGLMapBufferObject

cuGLRegisterBufferObject

cuGLUnmapBufferObject

cuGLUnregisterBufferObject

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGLInteroperability*, *Direct3dInteroperability*

2.10.1 cuGLCtxCreate

NAME

cuGLCtxCreate - create a CUDA context for interoperability with OpenGL

SYNOPSIS

```
CUresult cuGLCtxCreate(CUcontext *pCtx, unsigned int Flags, CUdevice device);
```

DESCRIPTION

Creates a new CUDA context, initializes OpenGL interoperability, and associates the CUDA context with the calling thread. It must be called before performing any other OpenGL interoperability operations. It may fail if the needed OpenGL driver facilities are not available. For usage of the **Flags** parameter, see **cuCtxCreate**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuCtxCreate, *cuGLInit cuGLRegisterBufferObject*, *cuGLMapBufferObject*, *cuGLUnmapBufferObject*, *cuGLUnregisterBufferObject*

2.10.2 cuGLInit

NAME

cuGLInit - initializes GL interoperability

SYNOPSIS

```
CUresult cuGLInit(void);
```

DESCRIPTION

Initializes OpenGL interoperability. It must be called before performing any other OpenGL interoperability operations. It may fail if the needed OpenGL driver facilities are not available.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuGLCtxCreate, *cuGLRegisterBufferObject*, *cuGLMapBufferObject*, *cuGLUnmapBufferObject*, *cuGLUnregisterBufferObject*

2.10.3 cuGLMapBufferObject

NAME

cuGLMapBufferObject - maps a GL buffer object

SYNOPSIS

```
CUresult cuGLMapBufferObject(CUdeviceptr* devPtr, unsigned int* size, GLuint bufferObj);
```

DESCRIPTION

Maps the buffer object of ID **bufferObj** into the address space of the current CUDA context and returns in ***devPtr** and ***size** the base pointer and size of the resulting mapping.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_MAP_FAILED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuGLCtxCreate, *cuGLInit*, *cuGLRegisterBufferObject*, *cuGLUnmapBufferObject*, *cuGLUnregisterBufferObject*

2.10.4 cuGLRegisterBufferObject

NAME

cuGLRegisterBufferObject - registers a GL buffer object

SYNOPSIS

```
CUresult cuGLRegisterBufferObject(GLuint bufferObj);
```

DESCRIPTION

Registers the buffer object of ID **bufferObj** for access by CUDA. This function must be called before CUDA can map the buffer object. While it is registered, the buffer object cannot be used by any OpenGL commands except as a data source for OpenGL drawing commands.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_ALREADY_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuGLCtxCreate, *cuGLInit*, *cuGLMapBufferObject*, *cuGLUnmapBufferObject*, *cuGLUnregisterBufferObject*

2.10.5 cuGLUnmapBufferObject

NAME

cuGLUnmapBufferObject - unmaps a GL buffer object

SYNOPSIS

```
CUresult cuGLUnmapBufferObject(GLuint bufferObj);
```

DESCRIPTION

Unmaps the buffer object of ID **bufferObj** for access by CUDA.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuGLCtxCreate, *cuGLInit*, *cuGLRegisterBufferObject*, *cuGLMapBufferObject*, *cuGLUnregisterBufferObject*

2.10.6 cuGLUnregisterBufferObject

NAME

cuGLUnregisterBufferObject - unregister a GL buffer object

SYNOPSIS

```
CUresult cuGLUnregisterBufferObject(GLuint bufferObj);
```

DESCRIPTION

Unregisters the buffer object of ID **bufferObj** for access by CUDA.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuGLCtxCreate, *cuGLInit*, *cuGLRegisterBufferObject*, *cuGLMapBufferObject*, *cuGLUnmapBufferObject*

2.11 Direct3dInteroperability

NAME

Direct3D Interoperability

DESCRIPTION

This section describes Direct3D interoperability in the the low-level CUDA driver application programming interface.

cuD3D9GetDevice

cuD3D9CtxCreate

cuD3D9GetDirect3DDevice

cuD3D9RegisterResource

cuD3D9UnregisterResource

cuD3D9MapResources

cuD3D9UnmapResources

cuD3D9ResourceGetSurfaceDimensions

cuD3D9ResourceSetMapFlags

cuD3D9ResourceGetMappedPointer

cuD3D9ResourceGetMappedSize

cuD3D9ResourceGetMappedPitch

As of CUDA 2.0 the following functions are deprecated. They should not be used in new development.

cuD3D9Begin

cuD3D9End

cuD3D9MapVertexBuffer

cuD3D9RegisterVertexBuffer

cuD3D9UnmapVertexBuffer

cuD3D9UnregisterVertexBuffer

SEE ALSO

Initialization, *DeviceManagement*, *ContextManagement*, *ModuleManagement*, *StreamManagement*, *EventManagement*, *ExecutionControl*, *MemoryManagement*, *TextureReferenceManagement*, *OpenGLInteroperability*, *Direct3dInteroperability*

2.11.1 cuD3D9GetDevice

NAME

cuD3D9GetDevice - gets the device number for an adapter

SYNOPSIS

```
CUresult cuD3D9GetDevice(CUdevice* dev, const char* adapterName);
```

DESCRIPTION

Returns in ***dev** the CUDA-compatible device corresponding to the adapter name **adapterName** obtained from **EnumDisplayDevices** or **IDirect3D9::GetAdapterIdentifier()**. If no device on the adapter with name **adapterName** is CUDA-compatible then the call will fail.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9CtxCreate, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*
cuD3D9UnmapResources, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.2 cuD3D9CtxCreate

NAME

cuD3D9CtxCreate - create a CUDA context for interoperability with Direct3D

SYNOPSIS

```
CUresult cuD3D9CtxCreate(CUcontext* pCtx, CUdevice* pCuDevice, unsigned int Flags, IDirect3DDevice9*
pDxDevice);
```

DESCRIPTION

Creates a new CUDA context, enables interoperability for that context with the Direct3D device **pDxDevice**, and associates the created CUDA context with the calling thread. The **CUcontext** created will be returned in ***pCtx**. If **pCuDevice** is non-NULL then the **CUdevice** on which this CUDA context was created will be returned in ***pCuDevice**. For usage of the **Flags** parameter, see *cuCtxCreate*. Direct3D resources from this device may be registered and mapped through the lifetime of this CUDA context.

This context will function only until its Direct3D device is destroyed. On success, this call will increase the internal reference count on **pDxDevice**. This reference count will be decremented upon destruction of this context through *cuCtxDestroy*.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*
cuD3D9UnmapResources, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.3 cuD3D9GetDirect3DDevice

NAME

cuD3D9GetDirect3DDevice - get the Direct3D device against which the current CUDA context was created

SYNOPSIS

```
CUresult cuD3D9GetDirect3DDevice(IDirect3DDevice9** ppDxDevice);
```

DESCRIPTION

Returns in ***ppDxDevice** the Direct3D device against which this CUDA context was created in **cuD3D9CtxCreate**.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.4 cuD3D9RegisterResource

NAME

cuD3D9RegisterResource - register a Direct3D resource for access by CUDA

SYNOPSIS

```
CUresult cuD3D9RegisterResource(IDirect3DResource9* pResource, unsigned int Flags);
```

DESCRIPTION

Registers the Direct3D resource **pResource** for access by CUDA.

If this call is successful then the application will be able to map and unmap this resource until it is unregistered through *cuD3D9UnregisterResource*. Also on success, this call will increase the internal reference count on **pResource**. This reference count will be decremented when this resource is unregistered through *cuD3D9UnregisterResource*.

This call is potentially high-overhead and should not be called every frame in interactive applications.

The type of **pResource** must be one of the following.

- **IDirect3DVertexBuffer9**: No notes.
- **IDirect3DIndexBuffer9**: No notes.
- **IDirect3DSurface9**: Only stand-alone objects of type **IDirect3DSurface9** may be explicitly shared. In particular, individual mipmap levels and faces of cube maps may not be registered directly. To access individual surfaces associated with a texture, one must register the base texture object.
- **IDirect3DBaseTexture9**: When a texture is registered all surfaces associated with the all mipmap levels of all faces of the texture will be accessible to CUDA.

The **Flags** argument specifies the mechanism through which CUDA will access the Direct3D resource. The following value is allowed.

- **CU_D3D9_REGISTER_FLAGS_NONE**: Specifies that CUDA will access this resource through a **CUdeviceptr**. The pointer, size, and pitch for each subresource of this resource may be queried through *cuD3D9ResourceGetMappedPointer*, *cuD3D9ResourceGetMappedSize*, and *cuD3D9ResourceGetMappedPitch* respectively. This option is valid for all resource types.

Not all Direct3D resources of the above types may be used for interoperability with CUDA. The following are some limitations.

- The primary rendertarget may not be registered with CUDA.
- Resources allocated as shared may not be registered with CUDA.
- Any resources allocated in **D3DPOOL_SYSTEMMEM** may not be registered with CUDA.
- Textures which are not of a format which is 1, 2, or 4 channels of 8, 16, or 32-bit integer or floating-point data cannot be shared.

- Surfaces of depth or stencil formats cannot be shared.

If Direct3D interoperability is not initialized on this context then **CUDA_ERROR_INVALID_CONTEXT** is returned. If **pResource** is of incorrect type (e.g, is a non-stand-alone **IDirect3DSurface9**) or is already registered then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** cannot be registered then **CUDA_ERROR_UNKNOWN** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_OUT_OF_MEMORY

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.5 cuD3D9UnregisterResource

NAME

cuD3D9UnregisterResource - unregister a Direct3D resource

SYNOPSIS

```
CUresult cuD3D9UnregisterResource(IDirect3DResource9* pResource);
```

DESCRIPTION

Unregisters the Direct3D resource **pResource** so it is not accessable by CUDA unless registered again.

If **pResource** is not registered then **CUDA_ERROR_INVALID_HANDLE** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.6 cuD3D9MapResources

NAME

cuD3D9MapResources - map Direct3D resources for access by CUDA

SYNOPSIS

```
CUresult cuD3D9MapResources(unsigned int count, IDirect3DResource9 **ppResources);
```

DESCRIPTION

Maps the **count** Direct3D resources in **ppResources** for access by CUDA.

The resources in **ppResources** may be accessed in CUDA kernels until they are unmapped. Direct3D should not access any resources while they are mapped by CUDA. If an application does so the results are undefined.

This function provides the synchronization guarantee that any Direct3D calls issued before **cuD3D9MapResources** will complete before any CUDA kernels issued after **cuD3D9MapResources** begin.

If any of **ppResources** have not been registered for use with CUDA or if **ppResources** contains any duplicate entries then **CUDA_ERROR_INVALID_HANDLE** is returned. If any of **ppResources** are presently mapped for access by CUDA then **CUDA_ERROR_ALREADY_MAPPED** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_ALREADY_MAPPED

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappe*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.7 cuD3D9UnmapResources

NAME

cuD3D9UnmapResources - unmap Direct3D resources

SYNOPSIS

```
CUresult cuD3D9UnmapResources(unsigned int count, IDirect3DResource9** ppResources);
```

DESCRIPTION

Unmaps the **count** Direct3D resources in **ppResources**.

This function provides the synchronization guarantee that any CUDA kernels issued before **cuD3D9UnmapResources** will complete before any Direct3D calls issued after **cuD3D9UnmapResources** begin.

If any of **ppResources** have not been registered for use with CUDA or if **ppResources** contains any duplicate entries then **CUDA_ERROR_INVALID_HANDLE** is returned. If any of **ppResources** are not presently mapped for access by CUDA then **CUDA_ERROR_NOT_MAPPED** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_MAPPED

CUDA_ERROR_UNKNOWN

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappedP*
cuD3D9ResourceGetMappedSize, *cuD3D9ResourceGetMappedPitch*

2.11.8 cuD3D9ResourceSetMapFlags

NAME

cuD3D9ResourceSetMapFlags - set usage flags for mapping a Direct3D resource

SYNOPSIS

```
CUresult cuD3D9ResourceSetMapFlags(IDirect3DResource9 *pResource, unsigned int Flags);
```

DESCRIPTION

Set flags for mapping the Direct3D resource **pResource**.

Changes to flags will take effect the next time **pResource** is mapped. The **Flags** argument may be any of the following.

- **CU_D3D9_MAPRESOURCE_FLAGS_NONE**: Specifies no hints about how this resource will be used. It is therefore assumed that this resource will be read from and written to by CUDA kernels. This is the default value.
- **CU_D3D9_MAPRESOURCE_FLAGS_READONLY**: Specifies that CUDA kernels which access this resource will not write to this resource.
- **CU_D3D9_MAPRESOURCE_FLAGS_WRITEDISCARD**: Specifies that CUDA kernels which access this resource will not read from this resource and will write over the entire contents of the resource, so none of the data previously stored in the resource will be preserved.

If **pResource** has not been registered for use with CUDA then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** is presently mapped for access by CUDA then **CUDA_ERROR_ALREADY_MAPPED** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_ALREADY_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceGetMappedPointe* *cuD3D9ResourceGetMappedSize*, *cuD3D9ResourceGetMappedPitch*

2.11.9 cuD3D9ResourceGetSurfaceDimensions

NAME

cuD3D9ResourceGetSurfaceDimensions - get the dimensions of a registered surface

SYNOPSIS

```
CUresult cuD3D9ResourceGetSurfaceDimensions(unsigned int* pWidth, unsigned int* pHeight, unsigned int *pDepth, IDirect3DResource9* pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pWidth**, ***pHeight**, and ***pDepth** the dimensions of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**.

Because anti-aliased surfaces may have multiple samples per pixel it is possible that the dimensions of a resource will be an integer factor larger than the dimensions reported by the Direct3D runtime.

The parameters **pWidth**, **pHeight**, and **pDepth** are optional. For 2D surfaces, the value returned in ***pDepth** will be 0.

If **pResource** is not of type **IDirect3DBaseTexture9** or **IDirect3DSurface9** or if **pResource** has not been registered for use with CUDA then **CUDA_ERROR_INVALID_HANDLE** is returned.

For usage requirements of **Face** and **Level** parameters see *cuD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappedPointer*, *cuD3D9ResourceGetMappedSize*, *cuD3D9ResourceGetMappedPitch*

2.11.10 cuD3D9ResourceGetMappedPointer

NAME

cuD3D9ResourceGetMappedPointer - get a pointer through which to access a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
CUresult cuD3D9ResourceGetMappedPointer(CUdeviceptr* pDevPtr, IDirect3DResource9* pResource,
unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pDevPtr** the base pointer of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The value set in **pDevPtr** may change every time that **pResource** is mapped.

If **pResource** is not registered then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** was not registered with usage flags **CU_D3D9_REGISTER_FLAGS_NONE** then **CUDA_ERROR_INVA** is returned. If **pResource** is not mapped then **CUDA_ERROR_NOT_MAPPED** is returned.

If **pResource** is of type **IDirect3DCubeTexture9** then **Face** must one of the values enumerated by type **D3DCUBEMAP_FACES**. For all other types **Face** must be 0. If **Face** is invalid then **CUDA_ERROR_INVALID_V** is returned.

If **pResource** is of type **IDirect3DBaseTexture9** then **Level** must correspond to a valid mipmap level. At present only mipmap level 0 is supported. For all other types **Level** must be 0. If **Level** is invalid then **CUDA_ERROR_INVALID_VALUE** is returned.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappedSize*, *cuD3D9ResourceGetMappedPitch*

2.11.11 cuD3D9ResourceGetMappedSize

NAME

cuD3D9ResourceGetMappedSize - get the size of a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
CUresult cuD3D9ResourceGetMappedSize(unsigned int* pSize, IDirect3DResource9* pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pSize** the size of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The value set in **pSize** may change every time that **pResource** is mapped.

If **pResource** has not been registered for use with CUDA then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** was not registered with usage flags **CU_D3D9_REGISTER_FLAGS_NONE** then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** is not mapped for access by CUDA then **CUDA_ERROR_NOT_MAPPED** is returned.

For usage requirements of **Face** and **Level** parameters see *cuD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappedPointer*, *cuD3D9ResourceGetMappedPitch*

2.11.12 cuD3D9ResourceGetMappedPitch

NAME

cuD3D9ResourceGetMappedPitch - get the pitch of a subresource of a Direct3D resource which has been mapped for access by CUDA

SYNOPSIS

```
CUresult cuD3D9ResourceGetMappedPitch(unsigned int* pPitch, unsigned int* pPitchSlice, IDirect3DResourc
pResource, unsigned int Face, unsigned int Level);
```

DESCRIPTION

Returns in ***pPitch** and ***pPitchSlice** the pitch and Z-slice pitch of the subresource of the mapped Direct3D resource **pResource** which corresponds to **Face** and **Level**. The values set in **pPitch** and **pPitchSlice** may change every time that **pResource** is mapped.

The pitch and Z-slice pitch values may be used to compute the location of a sample on a surface as follows.

y*__pitch__ + (**bytes per pixel**)***x**

For a 3D surface the byte offset of the sample of at position **x**,**y**,**z** from the base pointer of the surface is

z*__slicePitch__ + **y***__pitch__ + (**bytes per pixel**)***x**

Both parameters **pPitch** and **pPitchSlice** are optional and may be set to NULL.

For a 2D surface the byte offset of the sample of at position **x**,**y** from the base pointer of the surface is

If **pResource** is not of type **IDirect3DBaseTexture9** or one of its sub-types or if **pResource** has not been registered for use with CUDA then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** was not registered with usage flags **CU_D3D9_REGISTER_FLAGS_NONE** then **CUDA_ERROR_INVALID_HANDLE** is returned. If **pResource** is not mapped for access by CUDA then **CUDA_ERROR_NOT_MAPPED** is returned.

For usage requirements of **Face** and **Level** parameters see *cuD3D9ResourceGetMappedPointer*.

RETURN VALUE

Relevant return values:

CUDA_SUCCESS

CUDA_ERROR_DEINITIALIZED

CUDA_ERROR_NOT_INITIALIZED

CUDA_ERROR_INVALID_CONTEXT

CUDA_ERROR_INVALID_VALUE

CUDA_ERROR_INVALID_HANDLE

CUDA_ERROR_NOT_MAPPED

Note that this function may also return error codes from previous, asynchronous launches.

SEE ALSO

cuD3D9GetDevice, *cuD3D9CtxCreate*, *cuD3D9GetDirect3DDevice*, *cuD3D9RegisterResource*, *cuD3D9UnregisterResource*, *cuD3D9MapResources*, *cuD3D9UnmapResources*, *cuD3D9ResourceGetSurfaceDimensions*, *cuD3D9ResourceSetMapFlags*, *cuD3D9ResourceGetMappedPointer*, *cuD3D9ResourceGetMappedSize*

3 AtomicFunctions

NAME

Atomic Functions

DESCRIPTION

Atomic functions can only be used in device functions.

NOTES

32-bit atomic operations are only supported on devices of compute capability 1.1 and higher. 64-bit atomic operations are only supported on devices of compute capability 1.2 and higher.

SEE ALSO

ArithmeticFunctions, *BitwiseFunctions*

3.1 ArithmeticFunctions

NAME

Arithmetic Functions

DESCRIPTION

This section describes the atomic arithmetic functions.

atomicAdd

atomicSub

atomicExch

atomicMin

atomicMax

atomicInc

atomicDec

atomicCAS

SEE ALSO

BitwiseFunctions

3.1.1 atomicAdd

NAME

atomicAdd - atomic addition

SYNOPSIS

```
int atomicAdd(int* address, int val);

unsigned int atomicAdd(unsigned int* address, unsigned int val);

unsigned long long int atomicAdd(unsigned long long int* address, unsigned long long int val);
```

DESCRIPTION

Reads the 32- or 64-bit word **old** located at the address **address** in global memory, computes **(old + val)**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

32-bit atomic operations are only supported on devices of compute capability 1.1 and higher. 64-bit atomic operations are only supported on devices of compute capability 1.2 and higher.

SEE ALSO

atomicSub, *atomicExch*, *atomicMin*, *atomicMax*, *atomicInc*, *atomicDec*, *atomicCAS*

3.1.2 atomicSub

NAME

atomicSub - atomic subtraction

SYNOPSIS

```
int atomicSub(int* address, int val);
```

```
unsigned int atomicSub(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word **old** located at the address **address** in global memory, computes **(old - val)**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAdd, *atomicExch*, *atomicMin*, *atomicMax*, *atomicInc*, *atomicDec*, *atomicCAS*

3.1.3 atomicExch

NAME

atomicExch - atomic exchange

SYNOPSIS

```
int atomicExch(int* address, int val);

unsigned int atomicExch(unsigned int* address, unsigned int val);

unsigned long long int atomicExch(unsigned long long int* address, unsigned long long int val);
```

DESCRIPTION

Reads the 32- or 64-bit word **old** located at the address **address** in global memory and stores **val** back to global memory at the same address. These two operations are performed in one atomic transaction. The function returns **old**.

NOTES

32-bit atomic operations are only supported on devices of compute capability 1.1 and higher. 64-bit atomic operations are only supported on devices of compute capability 1.2 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicMin*, *atomicMax*, *atomicInc*, *atomicDec*, *atomicCAS*

3.1.4 atomicMin

NAME

atomicMin - atomic minimum

SYNOPSIS

```
int atomicMin(int* address, int val);
```

```
unsigned int atomicMin(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes the minimum of **old** and **val**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns b<old>.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicExch*, *atomicMax*, *atomicInc*, *atomicDec*, *atomicCAS*

3.1.5 atomicMax

NAME

atomicMax - atomic maximum

SYNOPSIS

```
int atomicMax(int* address, int val);
```

```
unsigned int atomicMax(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes the maximum of **old** and **val**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicExch*, *atomicMin*, *atomicInc*, *atomicDec*, *atomicCAS*

3.1.6 atomicInc

NAME

atomicInc - atomic increment

SYNOPSIS

```
unsigned int atomicInc(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes **((old >= val) ? 0 : (old+1))**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicExch*, *atomicMin*, *atomicMax*, *atomicDec*, *atomicCAS*

3.1.7 atomicDec

NAME

atomicDec - atomic decrement

SYNOPSIS

```
unsigned int atomicDec(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes **(((old == 0)** , and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicExch*, *atomicMin*, *atomicMax*, *atomicInc*, *atomicCAS*

3.1.8 atomicCAS

NAME

atomicCAS - atomic compare-and-swap

SYNOPSIS

```
int atomicCAS(int* address, int compare, int val);

unsigned int atomicCAS(unsigned int* address, unsigned int compare, unsigned int val);

unsigned long long int atomicCAS(unsigned long long int* address, unsigned long long int compare,
unsigned long long int val);
```

DESCRIPTION

Reads the 32- or 64-bit word old located at the address **address** in global memory, computes **(old == compare ? val : old)**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old** (Compare And Swap).

NOTES

32-bit atomic operations are only supported on devices of compute capability 1.1 and higher. 64-bit atomic operations are only supported on devices of compute capability 1.2 and higher.

SEE ALSO

atomicAdd, *atomicSub*, *atomicExch*, *atomicMin*, *atomicMax*, *atomicInc*, *atomicDec*

3.2 BitwiseFunctions

NAME

Bitwise Functions

DESCRIPTION

This section describes the atomic bitwise functions.

atomicAnd

atomicOr

atomicXor

SEE ALSO

ArithmeticFunctions

3.2.1 atomicAnd

NAME

atomicAnd - atomic bitwise-and

SYNOPSIS

```
int atomicAnd(int* address, int val);
```

```
unsigned int atomicAnd(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes **(old & val)**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicOr, *atomicXor*

3.2.2 atomicOr

NAME

atomicOr - atomic bitwise-or

SYNOPSIS

```
int atomicOr(int* address, int val);
```

```
unsigned int atomicOr(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes **(old** , and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAnd, *atomicXor*

3.2.3 atomicXor

NAME

atomicXor - atomic bitwise-xor

SYNOPSIS

```
int atomicXor(int* address, int val);
```

```
unsigned int atomicXor(unsigned int* address, unsigned int val);
```

DESCRIPTION

Reads the 32-bit word old located at the address **address** in global memory, computes **(old ^ val)**, and stores the result back to global memory at the same address. These three operations are performed in one atomic transaction. The function returns **old**.

NOTES

Atomic operations are only supported on devices of compute capability 1.1 and higher.

SEE ALSO

atomicAnd, *atomicOr*

Index

ArithmeticFunctions, 223
atomicAdd, 224
atomicAnd, 233
atomicCAS, 231
atomicDec, 230
atomicExch, 226
AtomicFunctions, 222
atomicInc, 229
atomicMax, 228
atomicMin, 227
atomicOr, 234
atomicSub, 225
atomicXor, 235

BitwiseFunctions, 232

ContextManagement, 111
cuArray3DCreate, 154
cuArray3DGetDescriptor, 158
cuArrayCreate, 152
cuArrayDestroy, 156
cuArrayGetDescriptor, 157
cuCtxAttach, 112
cuCtxCreate, 113
cuCtxDestroy, 115
cuCtxDetach, 116
cuCtxGetDevice, 117
cuCtxPopCurrent, 118
cuCtxPushCurrent, 119
cuCtxSynchronize, 120
cuD3D9CtxCreate, 207
cuD3D9GetDevice, 206
cuD3D9GetDirect3DDevice, 208
cuD3D9MapResources, 212
cuD3D9RegisterResource, 209
cuD3D9ResourceGetMappedPitch, 220
cuD3D9ResourceGetMappedPointer, 217
cuD3D9ResourceGetMappedSize, 219
cuD3D9ResourceGetSurfaceDimensions, 216
cuD3D9ResourceSetMapFlags, 214
cuD3D9UnmapResources, 213
cuD3D9UnregisterResource, 211
cudaBindTexture, 59
cudaBindTexture HL, 65
cudaBindTextureToArray, 60
cudaBindTextureToArray HL, 66
cudaChooseDevice, 8
cudaConfigureCall, 69
cudaCreateChannelDesc, 56
cudaCreateChannelDesc HL, 64
cudaD3D9GetDevice, 79
cudaD3D9GetDirect3DDevice, 81
cudaD3D9MapResources, 85
cudaD3D9RegisterResource, 82
cudaD3D9ResourceGetMappedPitch, 92
cudaD3D9ResourceGetMappedPointer, 90
cudaD3D9ResourceGetMappedSize, 91
cudaD3D9ResourceGetSurfaceDimensions, 89
cudaD3D9ResourceSetMapFlags, 87
cudaD3D9SetDirect3DDevice, 80
cudaD3D9UnmapResources, 86
cudaD3D9UnregisterResource, 84
cudaEventCreate, 18
cudaEventDestroy, 22
cudaEventElapsedTime, 23
cudaEventQuery, 20
cudaEventRecord, 19
cudaEventSynchronize, 21
cudaFree, 27
cudaFreeArray, 29
cudaFreeHost, 31
cudaGetChannelDesc, 57
cudaGetDevice, 5
cudaGetDeviceCount, 3
cudaGetDeviceProperties, 6
cudaGetErrorString, 97
cudaGetLastError, 95
cudaGetSymbolAddress, 44
cudaGetSymbolSize, 45
cudaGetTextureAlignmentOffset, 62
cudaGetTextureReference, 58
cudaGLMapBufferObject, 75
cudaGLRegisterBufferObject, 74
cudaGLSetGLDevice, 73
cudaGLUnmapBufferObject, 76
cudaGLUnregisterBufferObject, 77
cudaLaunch, 70
cudaMalloc, 25
cudaMalloc3D, 46
cudaMalloc3DArray, 48
cudaMallocArray, 28
cudaMallocHost, 30
cudaMallocPitch, 26
cudaMemcpy, 34
cudaMemcpy2D, 35
cudaMemcpy2DArrayToArray, 41
cudaMemcpy2DFromArray, 39
cudaMemcpy2DToArray, 37
cudaMemcpy3D, 52
cudaMemcpyArrayToArray, 40
cudaMemcpyFromArray, 38
cudaMemcpyFromSymbol, 43

cudaMemcpyToArray, 36
cudaMemcpyToSymbol, 42
cudaMemset, 32
cudaMemset2D, 33
cudaMemset3D, 50
cudaSetDevice, 4
cudaSetupArgument, 71
cudaStreamCreate, 13
cudaStreamDestroy, 16
cudaStreamQuery, 14
cudaStreamSynchronize, 15
cudaThreadExit, 11
cudaThreadSynchronize, 10
cudaUnbindTexture, 61
cudaUnbindTexture HL, 67
cuDeviceComputeCapability, 102
cuDeviceGet, 103
cuDeviceGetAttribute, 104
cuDeviceGetCount, 106
cuDeviceGetName, 107
cuDeviceGetProperties, 108
cuDeviceTotalMem, 110
cuEventCreate, 135
cuEventDestroy, 136
cuEventElapsedTime, 137
cuEventQuery, 138
cuEventRecord, 139
cuEventSynchronize, 140
cuFuncSetBlockShape, 149
cuFuncSetSharedSize, 150
cuGLCtxCreate, 199
cuGLInit, 200
cuGLMapBufferObject, 201
cuGLRegisterBufferObject, 202
cuGLUnmapBufferObject, 203
cuGLUnregisterBufferObject, 204
cuInit, 100
cuLaunch, 142
cuLaunchGrid, 143
cuMemAlloc, 159
cuMemAllocHost, 160
cuMemAllocPitch, 161
cuMemcpy2D, 167
cuMemcpy3D, 170
cuMemcpyAtoA, 173
cuMemcpyAtoD, 174
cuMemcpyAtoH, 175
cuMemcpyDtoA, 176
cuMemcpyDtoD, 177
cuMemcpyDtoH, 178
cuMemcpyHtoA, 179
cuMemcpyHtoD, 180
cuMemFree, 163
cuMemFreeHost, 164
cuMemGetAddressRange, 165
cuMemGetInfo, 166
cuMemset, 181
cuMemset2D, 182
cuModuleGetFunction, 122
cuModuleGetGlobal, 123
cuModuleGetTexRef, 124
cuModuleLoad, 125
cuModuleLoadData, 126
cuModuleLoadFatBinary, 127
cuModuleUnload, 128
cuParamSetf, 146
cuParamSeti, 147
cuParamSetSize, 144
cuParamSetTexRef, 145
cuParamSetv, 148
cuStreamCreate, 130
cuStreamDestroy, 131
cuStreamQuery, 132
cuStreamSynchronize, 133
cuTexRefCreate, 184
cuTexRefDestroy, 185
cuTexRefGetAddress, 186
cuTexRefGetAddressMode, 187
cuTexRefGetArray, 188
cuTexRefGetFilterMode, 189
cuTexRefGetFlags, 190
cuTexRefGetFormat, 191
cuTexRefSetAddress, 192
cuTexRefSetAddressMode, 193
cuTexRefSetArray, 194
cuTexRefSetFilterMode, 195
cuTexRefSetFlags, 196
cuTexRefSetFormat, 197

DeviceManagement, 101
DeviceManagement RT, 2
Direct3dInteroperability, 205
Direct3dInteroperability RT, 78
DriverApiReference, 98

ErrorHandling RT, 94
EventManagement, 134
EventManagement RT, 17
ExecutionControl, 141
ExecutionControl RT, 68

HighLevelApi, 63

Initialization, 99

LowLevelApi, 55

MemoryManagement, 151
MemoryManagement RT, 24

ModuleManagement, 121

OpenGlInteroperability, 198
OpenGlInteroperability RT, 72

RuntimeApiReference, 1

StreamManagement, 129
StreamManagement RT, 12

TextureReferenceManagement, 183
TextureReferenceManagement RT, 54
ThreadManagement RT, 9