NAME

Stream Management −

Functions

CUresult cuStreamAddCallback (**CUstream** hStream, **CUstreamCallback** callback, void *userData, unsigned int flags)
Add a callback to a compute stream.

CUresult cuStreamAttachMemAsync (**CUstream** hStream, **CUdeviceptr** dptr, size_t length, unsigned int flags)
Attach memory to a stream asynchronously.

CUresult cuStreamCreate (**CUstream** *phStream, unsigned int Flags)
Create a stream.

CUresult cuStreamCreateWithPriority (**CUstream** *phStream, unsigned int flags, int priority)
Create a stream with the given priority.

CUresult cuStreamDestroy (**CUstream** hStream)
Destroys a stream.

CUresult cuStreamGetFlags (**CUstream** hStream, unsigned int *flags)
Query the flags of a given stream.

CUresult cuStreamGetPriority (**CUstream** hStream, int *priority)
Query the priority of a given stream.

CUresult cuStreamQuery (**CUstream** hStream)
Determine status of a compute stream.

CUresult cuStreamSynchronize (**CUstream** hStream)
Wait until a stream's tasks are completed.

CUresult cuStreamWaitEvent (**CUstream** hStream, **CUevent** hEvent, unsigned int Flags)
Make a compute stream wait on an event.

Detailed Description

\brief stream management functions of the low-level CUDA driver API (**cuda.h**)

This section describes the stream management functions of the low-level CUDA driver application programming interface.

Function Documentation

CUresult cuStreamAddCallback (CUstream hStream, CUstreamCallback callback, void * userData, unsigned int flags)

Adds a callback to be called on the host after all currently enqueued items in the stream have completed. For each cuStreamAddCallback call, the callback will be executed exactly once. The callback will block later work in the stream until it is finished.

The callback may be passed **CUDA_SUCCESS** or an error code. In the event of a device error, all subsequently executed callbacks will receive an appropriate **CUresult**.

Callbacks must not make any CUDA API calls. Attempting to use a CUDA API will result in **CUDA_ERROR_NOT_PERMITTED**. Callbacks must not perform any synchronization that may depend on outstanding device work or other callbacks that are not mandated to run earlier. Callbacks without a mandated order (in independent streams) execute in undefined order and may be serialized.

This API requires compute capability 1.1 or greater. See **cuDeviceGetAttribute** or **cuDeviceGetProperties** to query compute capability. Attempting to use this API with earlier compute versions will return **CUDA_ERROR_NOT_SUPPORTED**.

For the purposes of Unified Memory, callback execution makes a number of guarantees:

- The callback stream is considered idle for the duration of the callback. Thus, for example, a callback may always use memory attached to the callback stream.
- The start of execution of a callback has the same effect as synchronizing an event recorded in the same stream immediately prior to the callback. It thus synchronizes streams which have been 'joined' prior to the callback.
- Adding device work to any stream does not have the effect of making the stream active until all preceding callbacks have executed. Thus, for example, a callback might use global attached memory even if work has been added to another stream, if it has been properly ordered with an event.

- Completion of a callback does not cause a stream to become active except as described above. The callback stream will remain idle if no device work follows the callback, and will remain idle across consecutive callbacks without device work in between. Thus, for example, stream synchronization can be done by signaling from a callback at the end of the stream.

Parameters:

hStream - Stream to add callback to
callback - The function to call once preceding stream operations are complete
userData - User specified data to be passed to the callback function
flags - Reserved for future use, must be 0

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_NOT_SUPPORTED**

Note:

This function uses standard semantics.
Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuStreamCreate, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamWaitEvent**, **cuStreamDestroy**, **cuMemAllocManaged**, **cuStreamAttachMemAsync**

CUresult cuStreamAttachMemAsync (CUstream hStream, CUdeviceptr dptr, size_t length, unsigned int flags)

Enqueues an operation in `hStream` to specify stream association of `length` bytes of memory starting from `dptr`. This function is a stream-ordered operation, meaning that it is dependent on, and will only take effect when, previous work in stream has completed. Any previous association is automatically replaced.
`dptr` must point to an address within managed memory space declared using the __managed__ keyword or allocated with **cuMemAllocManaged**.
`length` must be zero, to indicate that the entire allocation's stream association is being changed. Currently, it's not possible to change stream association for a portion of an allocation.
The stream association is specified using `flags` which must be one of **CUmemAttach_flags**. If the **CU_MEM_ATTACH_GLOBAL** flag is specified, the memory can be accessed by any stream on any device. If the **CU_MEM_ATTACH_HOST** flag is specified, the program makes a guarantee that it won't access the memory on the device from any stream. If the **CU_MEM_ATTACH_SINGLE** flag is specified, the program makes a guarantee that it will only access the memory on the device from `hStream`. It is illegal to attach singly to the NULL stream, because the NULL stream is a virtual global stream and not a specific stream. An error will be returned in this case.
When memory is associated with a single stream, the Unified Memory system will allow CPU access to this memory region so long as all operations in `hStream` have completed, regardless of whether other streams are active. In effect, this constrains exclusive ownership of the managed memory region by an active GPU to per-stream activity instead of whole-GPU activity.
Accessing memory on the device from streams that are not associated with it will produce undefined results. No error checking is performed by the Unified Memory system to ensure that kernels launched into other streams do not access this region.
It is a program's responsibility to order calls to **cuStreamAttachMemAsync** via events, synchronization or other means to ensure legal access to memory at all times. Data visibility and coherency will be changed appropriately for all kernels which follow a stream-association change.
If `hStream` is destroyed while data is associated with it, the association is removed and the association reverts to the default visibility of the allocation as specified at **cuMemAllocManaged**. For __managed__ variables, the default association is always **CU_MEM_ATTACH_GLOBAL**. Note that destroying a stream is an asynchronous operation, and as a result, the change to default association won't happen until all work in the stream has completed.

Parameters:

hStream - Stream in which to enqueue the attach operation
dptr - Pointer to memory (must be a pointer to managed memory)
length - Length of memory (must be zero)
flags - Must be one of **CUmemAttach_flags**

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**,

CUDA_ERROR_INVALID_HANDLE, **CUDA_ERROR_NOT_SUPPORTED**

Note:

This function uses standard semantics.

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

See also:

cuStreamCreate, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamWaitEvent**, **cuStreamDestroy**, **cuMemAllocManaged**

CUresult cuStreamCreate (CUstream * phStream, unsigned int Flags)

Creates a stream and returns a handle in `phStream`. The `Flags` argument determines behaviors of the stream. Valid values for `Flags` are:

- **CU_STREAM_DEFAULT**: Default stream creation flag.
- **CU_STREAM_NON_BLOCKING**: Specifies that work running in the created stream may run concurrently with work in stream 0 (the NULL stream), and that the created stream should perform no implicit synchronization with stream 0.

Parameters:

phStream - Returned newly created stream

Flags - Parameters for stream creation

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

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

See also:

cuStreamDestroy, **cuStreamCreateWithPriority**, **cuStreamGetPriority**, **cuStreamGetFlags**, **cuStreamWaitEvent**, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamAddCallback**

CUresult cuStreamCreateWithPriority (CUstream * phStream, unsigned int flags, int priority)

Creates a stream with the specified priority and returns a handle in `phStream`. This API alters the scheduler priority of work in the stream. Work in a higher priority stream may preempt work already executing in a low priority stream.

`priority` follows a convention where lower numbers represent higher priorities. ’0’ represents default priority. The range of meaningful numerical priorities can be queried using **cuCtxGetStreamPriorityRange**. If the specified priority is outside the numerical range returned by **cuCtxGetStreamPriorityRange**, it will automatically be clamped to the lowest or the highest number in the range.

Parameters:

phStream - Returned newly created stream

flags - Flags for stream creation. See **cuStreamCreate** for a list of valid flags

priority - Stream priority. Lower numbers represent higher priorities. See **cuCtxGetStreamPriorityRange** for more information about meaningful stream priorities that can be passed.

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

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

Stream priorities are supported only on Quadro and Tesla GPUs with compute capability 3.5 or higher.

In the current implementation, only compute kernels launched in priority streams are affected by the stream’s priority. Stream priorities have no effect on host-to-device and device-to-host memory operations.

See also:

cuStreamDestroy, **cuStreamCreate**, **cuStreamGetPriority**, **cuCtxGetStreamPriorityRange**, **cuStreamGetFlags**, **cuStreamWaitEvent**, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamAddCallback**

CUresult cuStreamDestroy (CUstream hStream)

Destroys the stream specified by `hStream`.

In case the device is still doing work in the stream `hStream` when **cuStreamDestroy()** is called, the function will return immediately and the resources associated with `hStream` will be released automatically once the device has completed all work in `hStream`.

Parameters:

hStream - Stream to destroy

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

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

See also:

cuStreamCreate, **cuStreamWaitEvent**, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamAddCallback**

CUresult cuStreamGetFlags (CUstream hStream, unsigned int * flags)

Query the flags of a stream created using **cuStreamCreate** or **cuStreamCreateWithPriority** and return the flags in `flags`.

Parameters:

hStream - Handle to the stream to be queried

flags - Pointer to an unsigned integer in which the stream's flags are returned The value returned in `flags` is a logical 'OR' of all flags that were used while creating this stream. See **cuStreamCreate** for the list of valid flags

Returns:

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**

Note:

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

See also:

cuStreamDestroy, **cuStreamCreate**, **cuStreamGetPriority**

CUresult cuStreamGetPriority (CUstream hStream, int * priority)

Query the priority of a stream created using **cuStreamCreate** or **cuStreamCreateWithPriority** and return the priority in `priority`. Note that if the stream was created with a priority outside the numerical range returned by **cuCtxGetStreamPriorityRange**, this function returns the clamped priority. See **cuStreamCreateWithPriority** for details about priority clamping.

Parameters:

hStream - Handle to the stream to be queried

priority - Pointer to a signed integer in which the stream's priority is returned

Returns:

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**

Note:

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

See also:

cuStreamDestroy, **cuStreamCreate**, **cuStreamCreateWithPriority**, **cuCtxGetStreamPriorityRange**, **cuStreamGetFlags**

CUresult cuStreamQuery (CUstream hStream)

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

For the purposes of Unified Memory, a return value of **CUDA_SUCCESS** is equivalent to having called **cuStreamSynchronize()**.

Parameters:

hStream - Stream to query status of

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**, **CUDA_ERROR_NOT_READY**

Note:

This function uses standard semantics.

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

See also:

cuStreamCreate, **cuStreamWaitEvent**, **cuStreamDestroy**, **cuStreamSynchronize**, **cuStreamAddCallback**

CUresult cuStreamSynchronize (CUstream hStream)

Waits until the device has completed all operations in the stream specified by `hStream`. If the context was created with the **CU_CTX_SCHED_BLOCKING_SYNC** flag, the CPU thread will block until the stream is finished with all of its tasks.

Parameters:

hStream - Stream to wait for

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**

Note:

This function uses standard semantics.

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

See also:

cuStreamCreate, **cuStreamDestroy**, **cuStreamWaitEvent**, **cuStreamQuery**, **cuStreamAddCallback**

CUresult cuStreamWaitEvent (CUstream hStream, CUevent hEvent, unsigned int Flags)

Makes all future work submitted to `hStream` wait until `hEvent` reports completion before beginning execution. This synchronization will be performed efficiently on the device. The event `hEvent` may be from a different context than `hStream`, in which case this function will perform cross-device synchronization.

The stream `hStream` will wait only for the completion of the most recent host call to **cuEventRecord()** on `hEvent`. Once this call has returned, any functions (including **cuEventRecord()** and **cuEventDestroy()**) may be called on `hEvent` again, and subsequent calls will not have any effect on `hStream`.

If **cuEventRecord()** has not been called on `hEvent`, this call acts as if the record has already completed, and so is a functional no-op.

Parameters:

hStream - Stream to wait

hEvent - Event to wait on (may not be NULL)

Flags - Parameters for the operation (must be 0)

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_HANDLE**,

Note:

This function uses standard semantics.

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

See also:

cuStreamCreate, **cuEventRecord**, **cuStreamQuery**, **cuStreamSynchronize**, **cuStreamAddCallback**, **cuStreamDestroy**

Author

Generated automatically by Doxygen from the source code.