NAME

Thread Management [DEPRECATED] −

Functions

cudaError_t cudaThreadExit (void)
Exit and clean up from CUDA launches.

cudaError_t cudaThreadGetCacheConfig (enum **cudaFuncCache** *pCacheConfig)
Returns the preferred cache configuration for the current device.

cudaError_t cudaThreadGetLimit (size_t *pValue, enum **cudaLimit** limit)
Returns resource limits.

cudaError_t cudaThreadSetCacheConfig (enum **cudaFuncCache** cacheConfig)
Sets the preferred cache configuration for the current device.

cudaError_t cudaThreadSetLimit (enum **cudaLimit** limit, size_t value)
Set resource limits.

cudaError_t cudaThreadSynchronize (void)
Wait for compute device to finish.

Detailed Description

\brief deprecated thread management functions of the CUDA runtime API (cuda_runtime_api.h)

This section describes deprecated thread management functions of the CUDA runtime application programming interface.

Function Documentation

cudaError_t cudaThreadExit (void)

Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is identical to the non-deprecated function **cudaDeviceReset()**, which should be used instead.

Explicitly destroys all cleans up all resources associated with the current device in the current process. Any subsequent API call to this device will reinitialize the device.

Note that this function will reset the device immediately. It is the caller's responsibility to ensure that the device is not being accessed by any other host threads from the process when this function is called.

Returns:
cudaSuccess

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

See also:
cudaDeviceReset

cudaError_t cudaThreadGetCacheConfig (enum cudaFuncCache * pCacheConfig)

Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is identical to the non-deprecated function **cudaDeviceGetCacheConfig()**, which should be used instead.

On devices where the L1 cache and shared memory use the same hardware resources, this returns through `pCacheConfig` the preferred cache configuration for the current device. This is only a preference. The runtime will use the requested configuration if possible, but it is free to choose a different configuration if required to execute functions.

This will return a `pCacheConfig` of **cudaFuncCachePreferNone** on devices where the size of the L1 cache and shared memory are fixed.

The supported cache configurations are:

- **cudaFuncCachePreferNone**: no preference for shared memory or L1 (default)
- **cudaFuncCachePreferShared**: prefer larger shared memory and smaller L1 cache
- **cudaFuncCachePreferL1**: prefer larger L1 cache and smaller shared memory

Parameters:
pCacheConfig - Returned cache configuration

Returns:

cudaSuccess, **cudaErrorInitializationError**

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

See also:
cudaDeviceGetCacheConfig

cudaError_t cudaThreadGetLimit (size_t * pValue, enum cudaLimit limit)
Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is identical to the non-deprecated function **cudaDeviceGetLimit()**, which should be used instead.

Returns in `*pValue` the current size of `limit`. The supported **cudaLimit** values are:

- **cudaLimitStackSize**: stack size of each GPU thread;
- **cudaLimitPrintfFifoSize**: size of the shared FIFO used by the printf() and fprintf() device system calls.
- **cudaLimitMallocHeapSize**: size of the heap used by the malloc() and free() device system calls;

Parameters:
limit - Limit to query
pValue - Returned size in bytes of limit

Returns:
cudaSuccess, **cudaErrorUnsupportedLimit**, **cudaErrorInvalidValue**

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

See also:
cudaDeviceGetLimit

cudaError_t cudaThreadSetCacheConfig (enum cudaFuncCache cacheConfig)
Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is identical to the non-deprecated function **cudaDeviceSetCacheConfig()**, which should be used instead.

On devices where the L1 cache and shared memory use the same hardware resources, this sets through `cacheConfig` the preferred cache configuration for the current device. This is only a preference. The runtime will use the requested configuration if possible, but it is free to choose a different configuration if required to execute the function. Any function preference set via **cudaFuncSetCacheConfig (C API)** or **cudaFuncSetCacheConfig (C++ API)** will be preferred over this device-wide setting. Setting the device-wide cache configuration to **cudaFuncCachePreferNone** will cause subsequent kernel launches to prefer to not change the cache configuration unless required to launch the kernel.

This setting does nothing on devices where the size of the L1 cache and shared memory are fixed.

Launching a kernel with a different preference than the most recent preference setting may insert a device-side synchronization point.

The supported cache configurations are:

- **cudaFuncCachePreferNone**: no preference for shared memory or L1 (default)
- **cudaFuncCachePreferShared**: prefer larger shared memory and smaller L1 cache
- **cudaFuncCachePreferL1**: prefer larger L1 cache and smaller shared memory

Parameters:
cacheConfig - Requested cache configuration

Returns:
cudaSuccess, **cudaErrorInitializationError**

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

See also:
cudaDeviceSetCacheConfig

cudaError_t cudaThreadSetLimit (enum cudaLimit limit, size_t value)

Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is identical to the non-deprecated function **cudaDeviceSetLimit()**, which should be used instead.

Setting `limit` to `value` is a request by the application to update the current limit maintained by the device. The driver is free to modify the requested value to meet h/w requirements (this could be clamping to minimum or maximum values, rounding up to nearest element size, etc). The application can use **cudaThreadGetLimit()** to find out exactly what the limit has been set to.

Setting each **cudaLimit** has its own specific restrictions, so each is discussed here.

- **cudaLimitStackSize** controls the stack size of each GPU thread. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **cudaErrorUnsupportedLimit** being returned.
- **cudaLimitPrintfFifoSize** controls the size of the shared FIFO used by the printf() and fprintf() device system calls. Setting **cudaLimitPrintfFifoSize** must be performed before launching any kernel that uses the printf() or fprintf() device system calls, otherwise **cudaErrorInvalidValue** will be returned. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **cudaErrorUnsupportedLimit** being returned.
- **cudaLimitMallocHeapSize** controls the size of the heap used by the malloc() and free() device system calls. Setting **cudaLimitMallocHeapSize** must be performed before launching any kernel that uses the malloc() or free() device system calls, otherwise **cudaErrorInvalidValue** will be returned. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **cudaErrorUnsupportedLimit** being returned.

Parameters:

limit - Limit to set
value - Size in bytes of limit

Returns:

cudaSuccess, **cudaErrorUnsupportedLimit**, **cudaErrorInvalidValue**

Note:

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

See also:

cudaDeviceSetLimit

cudaError_t cudaThreadSynchronize (void)

Deprecated

Note that this function is deprecated because its name does not reflect its behavior. Its functionality is similar to the non-deprecated function **cudaDeviceSynchronize()**, which should be used instead.

Blocks until the device has completed all preceding requested tasks. **cudaThreadSynchronize()** returns an error if one of the preceding tasks has failed. If the **cudaDeviceScheduleBlockingSync** flag was set for this device, the host thread will block until the device has finished its work.

Returns:

cudaSuccess

Note:

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

See also:

cudaDeviceSynchronize

Author

Generated automatically by Doxygen from the source code.