NAME

Device Management −

Functions

cudaError_t cudaChooseDevice (int *device, const struct **cudaDeviceProp** *prop)
Select compute-device which best matches criteria.

__cudart_builtin__ **cudaError_t cudaDeviceGetAttribute** (int *value, enum **cudaDeviceAttr** attr, int device)
Returns information about the device.

cudaError_t cudaDeviceGetByPCIBusId (int *device, const char *pciBusId)
Returns a handle to a compute device.

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

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

cudaError_t cudaDeviceGetPCIBusId (char *pciBusId, int len, int device)
Returns a PCI Bus Id string for the device.

__cudart_builtin__ **cudaError_t cudaDeviceGetSharedMemConfig** (enum **cudaSharedMemConfig** *pConfig)
Returns the shared memory configuration for the current device.

__cudart_builtin__ **cudaError_t cudaDeviceGetStreamPriorityRange** (int *leastPriority, int *greatestPriority)
Returns numerical values that correspond to the least and greatest stream priorities.

cudaError_t cudaDeviceReset (void)
Destroy all allocations and reset all state on the current device in the current process.

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

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

cudaError_t cudaDeviceSetSharedMemConfig (enum **cudaSharedMemConfig** config)
Sets the shared memory configuration for the current device.

__cudart_builtin__ **cudaError_t cudaDeviceSynchronize** (void)
Wait for compute device to finish.

__cudart_builtin__ **cudaError_t cudaGetDevice** (int *device)
Returns which device is currently being used.

__cudart_builtin__ **cudaError_t cudaGetDeviceCount** (int *count)
Returns the number of compute-capable devices.

__cudart_builtin__ **cudaError_t cudaGetDeviceProperties** (struct **cudaDeviceProp** *prop, int device)
Returns information about the compute-device.

cudaError_t cudaIpcCloseMemHandle (void *devPtr)
Close memory mapped with cudaIpcOpenMemHandle.

cudaError_t cudaIpcGetEventHandle (**cudaIpcEventHandle_t** *handle, **cudaEvent_t** event)
Gets an interprocess handle for a previously allocated event.

cudaError_t cudaIpcGetMemHandle (**cudaIpcMemHandle_t** *handle, void *devPtr)
Gets an interprocess memory handle for an existing device memory allocation.

cudaError_t cudaIpcOpenEventHandle (**cudaEvent_t** *event, **cudaIpcEventHandle_t** handle)
Opens an interprocess event handle for use in the current process.

cudaError_t cudaIpcOpenMemHandle (void **devPtr, **cudaIpcMemHandle_t** handle, unsigned int flags)
Opens an interprocess memory handle exported from another process and returns a device pointer usable in the local process.

cudaError_t cudaSetDevice (int device)
Set device to be used for GPU executions.

cudaError_t cudaSetDeviceFlags (unsigned int flags)
Sets flags to be used for device executions.

cudaError_t cudaSetValidDevices (int *device_arr, int len)
Set a list of devices that can be used for CUDA.

Detailed Description

CUDART_DEVICE

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

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

Function Documentation

cudaError_t cudaChooseDevice (int * device, const struct cudaDeviceProp * prop)

Returns in `*device` the device which has properties that best match `*prop`.

Parameters:

device - Device with best match
prop - Desired device properties

Returns:

cudaSuccess, **cudaErrorInvalidValue**

Note:

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

See also:

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

__cudart_builtin__ cudaError_t cudaDeviceGetAttribute (int * value, enum cudaDeviceAttr attr, int device)

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

- **cudaDevAttrMaxThreadsPerBlock**: Maximum number of threads per block;
- **cudaDevAttrMaxBlockDimX**: Maximum x-dimension of a block;
- **cudaDevAttrMaxBlockDimY**: Maximum y-dimension of a block;
- **cudaDevAttrMaxBlockDimZ**: Maximum z-dimension of a block;
- **cudaDevAttrMaxGridDimX**: Maximum x-dimension of a grid;
- **cudaDevAttrMaxGridDimY**: Maximum y-dimension of a grid;
- **cudaDevAttrMaxGridDimZ**: Maximum z-dimension of a grid;
- **cudaDevAttrMaxSharedMemoryPerBlock**: Maximum amount of shared memory available to a thread block in bytes;
- **cudaDevAttrTotalConstantMemory**: Memory available on device for __constant__ variables in a CUDA C kernel in bytes;
- **cudaDevAttrWarpSize**: Warp size in threads;
- **cudaDevAttrMaxPitch**: Maximum pitch in bytes allowed by the memory copy functions that involve memory regions allocated through **cudaMallocPitch()**;
- **cudaDevAttrMaxTexture1DWidth**: Maximum 1D texture width;
- **cudaDevAttrMaxTexture1DLinearWidth**: Maximum width for a 1D texture bound to linear memory;
- **cudaDevAttrMaxTexture1DMipmappedWidth**: Maximum mipmapped 1D texture width;
- **cudaDevAttrMaxTexture2DWidth**: Maximum 2D texture width;
- **cudaDevAttrMaxTexture2DHeight**: Maximum 2D texture height;
- **cudaDevAttrMaxTexture2DLinearWidth**: Maximum width for a 2D texture bound to linear memory;
- **cudaDevAttrMaxTexture2DLinearHeight**: Maximum height for a 2D texture bound to linear memory;
- **cudaDevAttrMaxTexture2DLinearPitch**: Maximum pitch in bytes for a 2D texture bound to linear memory;

- **cudaDevAttrMaxTexture2DMipmappedWidth**: Maximum mipmapped 2D texture width;
- **cudaDevAttrMaxTexture2DMipmappedHeight**: Maximum mipmapped 2D texture height;
- **cudaDevAttrMaxTexture3DWidth**: Maximum 3D texture width;
- **cudaDevAttrMaxTexture3DHeight**: Maximum 3D texture height;
- **cudaDevAttrMaxTexture3DDepth**: Maximum 3D texture depth;
- **cudaDevAttrMaxTexture3DWidthAlt**: Alternate maximum 3D texture width, 0 if no alternate maximum 3D texture size is supported;
- **cudaDevAttrMaxTexture3DHeightAlt**: Alternate maximum 3D texture height, 0 if no alternate maximum 3D texture size is supported;
- **cudaDevAttrMaxTexture3DDepthAlt**: Alternate maximum 3D texture depth, 0 if no alternate maximum 3D texture size is supported;
- **cudaDevAttrMaxTextureCubemapWidth**: Maximum cubemap texture width or height;
- **cudaDevAttrMaxTexture1DLayeredWidth**: Maximum 1D layered texture width;
- **cudaDevAttrMaxTexture1DLayeredLayers**: Maximum layers in a 1D layered texture;
- **cudaDevAttrMaxTexture2DLayeredWidth**: Maximum 2D layered texture width;
- **cudaDevAttrMaxTexture2DLayeredHeight**: Maximum 2D layered texture height;
- **cudaDevAttrMaxTexture2DLayeredLayers**: Maximum layers in a 2D layered texture;
- **cudaDevAttrMaxTextureCubemapLayeredWidth**: Maximum cubemap layered texture width or height;
- **cudaDevAttrMaxTextureCubemapLayeredLayers**: Maximum layers in a cubemap layered texture;
- **cudaDevAttrMaxSurface1DWidth**: Maximum 1D surface width;
- **cudaDevAttrMaxSurface2DWidth**: Maximum 2D surface width;
- **cudaDevAttrMaxSurface2DHeight**: Maximum 2D surface height;
- **cudaDevAttrMaxSurface3DWidth**: Maximum 3D surface width;
- **cudaDevAttrMaxSurface3DHeight**: Maximum 3D surface height;
- **cudaDevAttrMaxSurface3DDepth**: Maximum 3D surface depth;
- **cudaDevAttrMaxSurface1DLayeredWidth**: Maximum 1D layered surface width;
- **cudaDevAttrMaxSurface1DLayeredLayers**: Maximum layers in a 1D layered surface;
- **cudaDevAttrMaxSurface2DLayeredWidth**: Maximum 2D layered surface width;
- **cudaDevAttrMaxSurface2DLayeredHeight**: Maximum 2D layered surface height;
- **cudaDevAttrMaxSurface2DLayeredLayers**: Maximum layers in a 2D layered surface;
- **cudaDevAttrMaxSurfaceCubemapWidth**: Maximum cubemap surface width;
- **cudaDevAttrMaxSurfaceCubemapLayeredWidth**: Maximum cubemap layered surface width;
- **cudaDevAttrMaxSurfaceCubemapLayeredLayers**: Maximum layers in a cubemap layered surface;
- **cudaDevAttrMaxRegistersPerBlock**: Maximum number of 32-bit registers available to a thread block;
- **cudaDevAttrClockRate**: Peak clock frequency in kilohertz;
- **cudaDevAttrTextureAlignment**: Alignment requirement; texture base addresses aligned to textureAlign bytes do not need an offset applied to texture fetches;
- **cudaDevAttrTexturePitchAlignment**: Pitch alignment requirement for 2D texture references bound to pitched memory;
- **cudaDevAttrGpuOverlap**: 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not;

- **cudaDevAttrMultiProcessorCount**: Number of multiprocessors on the device;
- **cudaDevAttrKernelExecTimeout**: 1 if there is a run time limit for kernels executed on the device, or 0 if not;
- **cudaDevAttrIntegrated**: 1 if the device is integrated with the memory subsystem, or 0 if not;
- **cudaDevAttrCanMapHostMemory**: 1 if the device can map host memory into the CUDA address space, or 0 if not;
- **cudaDevAttrComputeMode**: Compute mode is the compute mode that the device is currently in. Available modes are as follows:
  - **cudaComputeModeDefault**: Default mode - Device is not restricted and multiple threads can use **cudaSetDevice()** with this device.
  - **cudaComputeModeExclusive**: Compute-exclusive mode - Only one thread will be able to use **cudaSetDevice()** with this device.
  - **cudaComputeModeProhibited**: Compute-prohibited mode - No threads can use **cudaSetDevice()** with this device.
  - **cudaComputeModeExclusiveProcess**: Compute-exclusive-process mode - Many threads in one process will be able to use **cudaSetDevice()** with this device.
- **cudaDevAttrConcurrentKernels**: 1 if the device supports executing multiple kernels within the same context simultaneously, or 0 if not. It is not guaranteed that multiple kernels will be resident on the device concurrently so this feature should not be relied upon for correctness;
- **cudaDevAttrEccEnabled**: 1 if error correction is enabled on the device, 0 if error correction is disabled or not supported by the device;
- **cudaDevAttrPciBusId**: PCI bus identifier of the device;
- **cudaDevAttrPciDeviceId**: PCI device (also known as slot) identifier of the device;
- **cudaDevAttrTccDriver**: 1 if the device is using a TCC driver. TCC is only available on Tesla hardware running Windows Vista or later;
- **cudaDevAttrMemoryClockRate**: Peak memory clock frequency in kilohertz;
- **cudaDevAttrGlobalMemoryBusWidth**: Global memory bus width in bits;
- **cudaDevAttrL2CacheSize**: Size of L2 cache in bytes. 0 if the device doesn't have L2 cache;
- **cudaDevAttrMaxThreadsPerMultiProcessor**: Maximum resident threads per multiprocessor;
- **cudaDevAttrUnifiedAddressing**: 1 if the device shares a unified address space with the host, or 0 if not;
- **cudaDevAttrComputeCapabilityMajor**: Major compute capability version number;
- **cudaDevAttrComputeCapabilityMinor**: Minor compute capability version number;
- **cudaDevAttrStreamPrioritiesSupported**: 1 if the device supports stream priorities, or 0 if not;
- **cudaDevAttrGlobalL1CacheSupported**: 1 if device supports caching globals in L1 cache, 0 if not;
- **cudaDevAttrGlobalL1CacheSupported**: 1 if device supports caching locals in L1 cache, 0 if not;
- **cudaDevAttrMaxSharedMemoryPerMultiprocessor**: Maximum amount of shared memory available to a multiprocessor in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;
- **cudaDevAttrMaxRegistersPerMultiprocessor**: Maximum number of 32-bit registers available to a multiprocessor; this number is shared by all thread blocks simultaneously resident on a multiprocessor;
- cudaDevAttrManagedMemSupported: 1 if device supports allocating managed memory, 0 if not;
- **cudaDevAttrIsMultiGpuBoard**: 1 if device is on a multi-GPU board, 0 if not;
- **cudaDevAttrMultiGpuBoardGroupID**: Unique identifier for a group of devices on the same multi-GPU board;

Parameters:

value - Returned device attribute value
attr - Device attribute to query
device - Device number to query

Returns:
cudaSuccess, **cudaErrorInvalidDevice**, **cudaErrorInvalidValue**

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

See also:
cudaGetDeviceCount, **cudaGetDevice**, **cudaSetDevice**, **cudaChooseDevice**, **cudaGetDeviceProperties**

cudaError_t cudaDeviceGetByPCIBusId (int * device, const char * pciBusId)
Returns in `*device` a device ordinal given a PCI bus ID string.

Parameters:
device - Returned device ordinal
pciBusId - String in one of the following forms: [domain]:[bus]:[device].[function] [domain]:[bus]:[device] [bus]:[device].[function] where `domain`, `bus`, `device`, and `function` are all hexadecimal values

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevice**

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

See also:
cudaDeviceGetPCIBusId

__cudart_builtin__ cudaError_t cudaDeviceGetCacheConfig (enum cudaFuncCache * pCacheConfig)
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
- **cudaFuncCachePreferEqual**: prefer equal size L1 cache and 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:
cudaDeviceSetCacheConfig, **cudaFuncSetCacheConfig (C API)**, **cudaFuncSetCacheConfig (C++ API)**

__cudart_builtin__ cudaError_t cudaDeviceGetLimit (size_t * pValue, enum cudaLimit limit)
Returns in `*pValue` the current size of `limit`. The supported **cudaLimit** values are:

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

- **cudaLimitMallocHeapSize**: size in bytes of the heap used by the malloc() and free() device system calls;
- **cudaLimitDevRuntimeSyncDepth**: maximum grid depth at which a thread can isssue the device runtime call **cudaDeviceSynchronize()** to wait on child grid launches to complete.
- **cudaLimitDevRuntimePendingLaunchCount**: maximum number of outstanding device runtime launches.

Parameters:
limit - Limit to query
pValue - Returned size of the limit

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

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

See also:
cudaDeviceSetLimit

cudaError_t cudaDeviceGetPCIBusId (char * pciBusId, int len, int device)
Returns an ASCII string identifying the device `dev` in the NULL-terminated string pointed to by `pciBusId`. `len` specifies the maximum length of the string that may be returned.

Parameters:
pciBusId - Returned identifier string for the device in the following format [domain]:[bus]:[device].[function] where `domain`, `bus`, `device`, and `function` are all hexadecimal values. pciBusId should be large enough to store 13 characters including the NULL-terminator.
len - Maximum length of string to store in `name`
device - Device to get identifier string for

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevice**

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

See also:
cudaDeviceGetByPCIBusId

__cudart_builtin__ cudaError_t cudaDeviceGetSharedMemConfig (enum cudaSharedMemConfig * pConfig)
This function will return in `pConfig` the current size of shared memory banks on the current device. On devices with configurable shared memory banks, **cudaDeviceSetSharedMemConfig** can be used to change this setting, so that all subsequent kernel launches will by default use the new bank size. When **cudaDeviceGetSharedMemConfig** is called on devices without configurable shared memory, it will return the fixed bank size of the hardware.

The returned bank configurations can be either:

- cudaSharedMemBankSizeFourByte - shared memory bank width is four bytes.
- cudaSharedMemBankSizeEightByte - shared memory bank width is eight bytes.

Parameters:
pConfig - Returned cache configuration

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInitializationError**

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

See also:
cudaDeviceSetCacheConfig, **cudaDeviceGetCacheConfig**, **cudaDeviceSetSharedMemConfig**, **cudaFuncSetCacheConfig**

__cudart_builtin__ cudaError_t cudaDeviceGetStreamPriorityRange (int * leastPriority, int * greatestPriority)

Returns in `*leastPriority` and `*greatestPriority` the numerical values that correspond to the least and greatest stream priorities respectively. Stream priorities follow a convention where lower numbers imply greater priorities. The range of meaningful stream priorities is given by [`*greatestPriority`, `*leastPriority`]. If the user attempts to create a stream with a priority value that is outside the the meaningful range as specified by this API, the priority is automatically clamped down or up to either `*leastPriority` or `*greatestPriority` respectively. See **cudaStreamCreateWithPriority** for details on creating a priority stream. A NULL may be passed in for `*leastPriority` or `*greatestPriority` if the value is not desired.

This function will return '0' in both `*leastPriority` and `*greatestPriority` if the current context's device does not support stream priorities (see **cudaDeviceGetAttribute**).

Parameters:

leastPriority - Pointer to an int in which the numerical value for least stream priority is returned
greatestPriority - Pointer to an int in which the numerical value for greatest stream priority is returned

Returns:

cudaSuccess, **cudaErrorInvalidValue**

Note:

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

See also:

cudaStreamCreateWithPriority, **cudaStreamGetPriority**

cudaError_t cudaDeviceReset (void)

Explicitly destroys and 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:

cudaDeviceSynchronize

cudaError_t cudaDeviceSetCacheConfig (enum cudaFuncCache cacheConfig)

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
- **cudaFuncCachePreferEqual**: prefer equal size L1 cache and 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:
cudaDeviceGetCacheConfig, **cudaFuncSetCacheConfig (C API)**, **cudaFuncSetCacheConfig (C++ API)**

cudaError_t cudaDeviceSetLimit (enum cudaLimit limit, size_t value)

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 **cudaDeviceGetLimit()** 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 in bytes 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 in bytes 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 in bytes 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.
- **cudaLimitDevRuntimeSyncDepth** controls the maximum nesting depth of a grid at which a thread can safely call **cudaDeviceSynchronize()**. Setting this limit must be performed before any launch of a kernel that uses the device runtime and calls **cudaDeviceSynchronize()** above the default sync depth, two levels of grids. Calls to **cudaDeviceSynchronize()** will fail with error code **cudaErrorSyncDepthExceeded** if the limitation is violated. This limit can be set smaller than the default or up the maximum launch depth of 24. When setting this limit, keep in mind that additional levels of sync depth require the runtime to reserve large amounts of device memory which can no longer be used for user allocations. If these reservations of device memory fail, **cudaDeviceSetLimit** will return **cudaErrorMemoryAllocation**, and the limit can be reset to a lower value. This limit is only applicable to devices of compute capability 3.5 and higher. Attempting to set this limit on devices of compute capability less than 3.5 will result in the error **cudaErrorUnsupportedLimit** being returned.
- **cudaLimitDevRuntimePendingLaunchCount** controls the maximum number of outstanding device runtime launches that can be made from the current device. A grid is outstanding from the point of launch up until the grid is known to have been completed. Device runtime launches which violate this limitation fail and return **cudaErrorLaunchPendingCountExceeded** when **cudaGetLastError()** is called after launch. If more pending launches than the default (2048 launches) are needed for a module using the device runtime, this limit can be increased. Keep in mind that being able to sustain additional pending launches will require the runtime to reserve larger amounts of device memory upfront which can no longer be used for allocations. If these reservations fail, **cudaDeviceSetLimit** will return **cudaErrorMemoryAllocation**, and the limit can be reset to a lower value. This limit is only applicable to devices of compute capability 3.5 and higher. Attempting to set this limit on devices of compute capability less than 3.5 will result in the error **cudaErrorUnsupportedLimit** being returned.

Parameters:
limit - Limit to set

value - Size of limit

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

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

See also:
cudaDeviceGetLimit

cudaError_t cudaDeviceSetSharedMemConfig (enum cudaSharedMemConfig config)
On devices with configurable shared memory banks, this function will set the shared memory bank size which is used for all subsequent kernel launches. Any per-function setting of shared memory set via **cudaFuncSetSharedMemConfig** will override the device wide setting.

Changing the shared memory configuration between launches may introduce a device side synchronization point.

Changing the shared memory bank size will not increase shared memory usage or affect occupancy of kernels, but may have major effects on performance. Larger bank sizes will allow for greater potential bandwidth to shared memory, but will change what kinds of accesses to shared memory will result in bank conflicts.

This function will do nothing on devices with fixed shared memory bank size.

The supported bank configurations are:

- cudaSharedMemBankSizeDefault: set bank width the device default (currently, four bytes)
- cudaSharedMemBankSizeFourByte: set shared memory bank width to be four bytes natively.
- cudaSharedMemBankSizeEightByte: set shared memory bank width to be eight bytes natively.

Parameters:
config - Requested cache configuration

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInitializationError**

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

See also:
cudaDeviceSetCacheConfig, **cudaDeviceGetCacheConfig**, **cudaDeviceGetSharedMemConfig**, **cudaFuncSetCacheConfig**

__cudart_builtin__ cudaError_t cudaDeviceSynchronize (void)
Blocks until the device has completed all preceding requested tasks. **cudaDeviceSynchronize()** 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:
cudaDeviceReset

__cudart_builtin__ cudaError_t cudaGetDevice (int * device)
Returns in `*device` the current device for the calling host thread.

Parameters:
device - Returns the device on which the active host thread executes the device code.

Returns:
cudaSuccess

Note:

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

See also:
cudaGetDeviceCount, **cudaSetDevice**, **cudaGetDeviceProperties**, **cudaChooseDevice**

__cudart_builtin__ cudaError_t cudaGetDeviceCount (int * count)
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 then **cudaGetDeviceCount()** will return **cudaErrorNoDevice**. If no driver can be loaded to determine if any such devices exist then **cudaGetDeviceCount()** will return **cudaErrorInsufficientDriver**.

Parameters:
count - Returns the number of devices with compute capability greater or equal to 1.0

Returns:
cudaSuccess, **cudaErrorNoDevice**, **cudaErrorInsufficientDriver**

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

See also:
cudaGetDevice, **cudaSetDevice**, **cudaGetDeviceProperties**, **cudaChooseDevice**

__cudart_builtin__ cudaError_t cudaGetDeviceProperties (struct cudaDeviceProp * prop, int device)
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];
    int clockRate;
    size_t totalConstMem;
    int major;
    int minor;
    size_t textureAlignment;
    size_t texturePitchAlignment;
    int deviceOverlap;
    int multiProcessorCount;
    int kernelExecTimeoutEnabled;
    int integrated;
    int canMapHostMemory;
    int computeMode;
    int maxTexture1D;
    int maxTexture1DMipmap;
    int maxTexture1DLinear;
    int maxTexture2D[2];
    int maxTexture2DMipmap[2];
    int maxTexture2DLinear[3];
    int maxTexture2DGather[2];
    int maxTexture3D[3];
    int maxTexture3DAlt[3];
    int maxTextureCubemap;
    int maxTexture1DLayered[2];
    int maxTexture2DLayered[3];
    int maxTextureCubemapLayered[2];
    int maxSurface1D;
    int maxSurface2D[2];
```

```
    int maxSurface3D[3];
    int maxSurface1DLayered[2];
    int maxSurface2DLayered[3];
    int maxSurfaceCubemap;
    int maxSurfaceCubemapLayered[2];
    size_t surfaceAlignment;
    int concurrentKernels;
    int ECCEnabled;
    int pciBusID;
    int pciDeviceID;
    int pciDomainID;
    int tccDriver;
    int asyncEngineCount;
    int unifiedAddressing;
    int memoryClockRate;
    int memoryBusWidth;
    int l2CacheSize;
    int maxThreadsPerMultiProcessor;
    int streamPrioritiesSupported;
    int globalL1CacheSupported;
    int localL1CacheSupported;
    size_t sharedMemPerMultiprocessor;
    int regsPerMultiprocessor;
    int managedMemSupported;
    int isMultiGpuBoard;
    int multiGpuBoardGroupID;
}
```

where:

- **name[256]** 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;
- **regsPerBlock** is the maximum number of 32-bit registers available to a thread block;
- **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]** contains the maximum size of each dimension of a block;
- **maxGridSize[3]** contains the maximum size of each dimension of a grid;
- **clockRate** is the clock frequency in kilohertz;
- **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;
- **textureAlignment** is the alignment requirement; texture base addresses that are aligned to **textureAlignment** bytes do not need an offset applied to texture fetches;
- **texturePitchAlignment** is the pitch alignment requirement for 2D texture references that are bound to pitched memory;
- **deviceOverlap** is 1 if the device can concurrently copy memory between host and device while executing a kernel, or 0 if not. Deprecated, use instead asyncEngineCount.
- **multiProcessorCount** is the number of multiprocessors on the device;

- **kernelExecTimeoutEnabled** is 1 if there is a run time limit for kernels executed on the device, or 0 if not.
- **integrated** is 1 if the device is an integrated (motherboard) GPU and 0 if it is a discrete (card) component.
- **canMapHostMemory** is 1 if the device can map host memory into the CUDA address space for use with **cudaHostAlloc()**/**cudaHostGetDevicePointer()**, or 0 if not;
- **computeMode** is the compute mode that the device is currently in. Available modes are as follows:
  - cudaComputeModeDefault: Default mode - Device is not restricted and multiple threads can use **cudaSetDevice()** with this device.
  - cudaComputeModeExclusive: Compute-exclusive mode - Only one thread will be able to use **cudaSetDevice()** with this device.
  - cudaComputeModeProhibited: Compute-prohibited mode - No threads can use **cudaSetDevice()** with this device.
  - cudaComputeModeExclusiveProcess: Compute-exclusive-process mode - Many threads in one process will be able to use **cudaSetDevice()** with this device.
    If **cudaSetDevice()** is called on an already occupied `device` with computeMode **cudaComputeModeExclusive**, **cudaErrorDeviceAlreadyInUse** will be immediately returned indicating the device cannot be used. When an occupied exclusive mode device is chosen with **cudaSetDevice**, all subsequent non-device management runtime functions will return **cudaErrorDevicesUnavailable**.
- **maxTexture1D** is the maximum 1D texture size.
- **maxTexture1DMipmap** is the maximum 1D mipmapped texture texture size.
- **maxTexture1DLinear** is the maximum 1D texture size for textures bound to linear memory.
- **maxTexture2D[2]** contains the maximum 2D texture dimensions.
- **maxTexture2DMipmap[2]** contains the maximum 2D mipmapped texture dimensions.
- **maxTexture2DLinear[3]** contains the maximum 2D texture dimensions for 2D textures bound to pitch linear memory.
- **maxTexture2DGather[2]** contains the maximum 2D texture dimensions if texture gather operations have to be performed.
- **maxTexture3D[3]** contains the maximum 3D texture dimensions.
- **maxTexture3DAlt[3]** contains the maximum alternate 3D texture dimensions.
- **maxTextureCubemap** is the maximum cubemap texture width or height.
- **maxTexture1DLayered[2]** contains the maximum 1D layered texture dimensions.
- **maxTexture2DLayered[3]** contains the maximum 2D layered texture dimensions.
- **maxTextureCubemapLayered[2]** contains the maximum cubemap layered texture dimensions.
- **maxSurface1D** is the maximum 1D surface size.
- **maxSurface2D[2]** contains the maximum 2D surface dimensions.
- **maxSurface3D[3]** contains the maximum 3D surface dimensions.
- **maxSurface1DLayered[2]** contains the maximum 1D layered surface dimensions.
- **maxSurface2DLayered[3]** contains the maximum 2D layered surface dimensions.
- **maxSurfaceCubemap** is the maximum cubemap surface width or height.
- **maxSurfaceCubemapLayered[2]** contains the maximum cubemap layered surface dimensions.
- **surfaceAlignment** specifies the alignment requirements for surfaces.
- **concurrentKernels** is 1 if the device supports executing multiple kernels within the same context simultaneously, or 0 if not. It is not guaranteed that multiple kernels will be resident on the device concurrently so this feature should not be relied upon for correctness;

- **ECCEnabled** is 1 if the device has ECC support turned on, or 0 if not.
- **pciBusID** is the PCI bus identifier of the device.
- **pciDeviceID** is the PCI device (sometimes called slot) identifier of the device.
- **pciDomainID** is the PCI domain identifier of the device.
- **tccDriver** is 1 if the device is using a TCC driver or 0 if not.
- **asyncEngineCount** is 1 when the device can concurrently copy memory between host and device while executing a kernel. It is 2 when the device can concurrently copy memory between host and device in both directions and execute a kernel at the same time. It is 0 if neither of these is supported.
- **unifiedAddressing** is 1 if the device shares a unified address space with the host and 0 otherwise.
- **memoryClockRate** is the peak memory clock frequency in kilohertz.
- **memoryBusWidth** is the memory bus width in bits.
- **l2CacheSize** is L2 cache size in bytes.
- **maxThreadsPerMultiProcessor** is the number of maximum resident threads per multiprocessor.
- **streamPrioritiesSupported** is 1 if the device supports stream priorities, or 0 if it is not supported.
- **globalL1CacheSupported** is 1 if the device supports caching of globals in L1 cache, or 0 if it is not supported.
- **localL1CacheSupported** is 1 if the device supports caching of locals in L1 cache, or 0 if it is not supported.
- **sharedMemPerMultiprocessor** is the maximum amount of shared memory available to a multiprocessor in bytes; this amount is shared by all thread blocks simultaneously resident on a multiprocessor;
- **regsPerMultiprocessor** is the maximum number of 32-bit registers available to a multiprocessor; this number is shared by all thread blocks simultaneously resident on a multiprocessor;
- **managedMemSupported** is 1 if the device supports allocating managed memory, or 0 if it is not supported.
- **isMultiGpuBoard** is 1 if the device is on a multi-GPU board (e.g. Gemini cards), and 0 if not;
- **multiGpuBoardGroupID** is a unique identifier for a group of devices associated with the same board. Devices on the same multi-GPU board will share the same identifier;

Parameters:

prop - Properties for the specified device
device - Device number to get properties for

Returns:

cudaSuccess, **cudaErrorInvalidDevice**

See also:

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

cudaError_t cudaIpcCloseMemHandle (void * devPtr)

Unmaps memory returnd by **cudaIpcOpenMemHandle**. The original allocation in the exporting process as well as imported mappings in other processes will be unaffected.

Any resources used to enable peer access will be freed if this is the last mapping using them.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

devPtr - Device pointer returned by **cudaIpcOpenMemHandle**

Returns:

cudaSuccess, **cudaErrorMapBufferObjectFailed**, **cudaErrorInvalidResourceHandle**,

See also:

cudaMalloc, **cudaFree**, **cudaIpcGetEventHandle**, **cudaIpcOpenEventHandle**,

cudaIpcGetMemHandle, **cudaIpcOpenMemHandle**,

cudaError_t cudaIpcGetEventHandle (cudaIpcEventHandle_t * handle, cudaEvent_t event)

Takes as input a previously allocated event. This event must have been created with the **cudaEventInterprocess** and **cudaEventDisableTiming** flags set. This opaque handle may be copied into other processes and opened with **cudaIpcOpenEventHandle** to allow efficient hardware synchronization between GPU work in different processes.

After the event has been been opened in the importing process, **cudaEventRecord**, **cudaEventSynchronize**, **cudaStreamWaitEvent** and **cudaEventQuery** may be used in either process. Performing operations on the imported event after the exported event has been freed with **cudaEventDestroy** will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

handle - Pointer to a user allocated cudaIpcEventHandle in which to return the opaque event handle
event - Event allocated with **cudaEventInterprocess** and **cudaEventDisableTiming** flags.

Returns:

cudaSuccess, **cudaErrorInvalidResourceHandle**, **cudaErrorMemoryAllocation**, **cudaErrorMapBufferObjectFailed**

See also:

cudaEventCreate, **cudaEventDestroy**, **cudaEventSynchronize**, **cudaEventQuery**, **cudaStreamWaitEvent**, **cudaIpcOpenEventHandle**, **cudaIpcGetMemHandle**, **cudaIpcOpenMemHandle**, **cudaIpcCloseMemHandle**

cudaError_t cudaIpcGetMemHandle (cudaIpcMemHandle_t * handle, void * devPtr)

Takes a pointer to the base of an existing device memory allocation created with **cudaMalloc** and exports it for use in another process. This is a lightweight operation and may be called multiple times on an allocation without adverse effects.

If a region of memory is freed with **cudaFree** and a subsequent call to **cudaMalloc** returns memory with the same device address, **cudaIpcGetMemHandle** will return a unique handle for the new memory.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

handle - Pointer to user allocated cudaIpcMemHandle to return the handle in.
devPtr - Base pointer to previously allocated device memory

Returns:

cudaSuccess, **cudaErrorInvalidResourceHandle**, **cudaErrorMemoryAllocation**, **cudaErrorMapBufferObjectFailed**,

See also:

cudaMalloc, **cudaFree**, **cudaIpcGetEventHandle**, **cudaIpcOpenEventHandle**, **cudaIpcOpenMemHandle**, **cudaIpcCloseMemHandle**

cudaError_t cudaIpcOpenEventHandle (cudaEvent_t * event, cudaIpcEventHandle_t handle)

Opens an interprocess event handle exported from another process with **cudaIpcGetEventHandle**. This function returns a **cudaEvent_t** that behaves like a locally created event with the **cudaEventDisableTiming** flag specified. This event must be freed with **cudaEventDestroy**.

Performing operations on the imported event after the exported event has been freed with **cudaEventDestroy** will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:

event - Returns the imported event
handle - Interprocess handle to open

Returns:
cudaSuccess, **cudaErrorMapBufferObjectFailed**, **cudaErrorInvalidResourceHandle**

See also:
cudaEventCreate, **cudaEventDestroy**, **cudaEventSynchronize**, **cudaEventQuery**, **cudaStreamWaitEvent**, **cudaIpcGetEventHandle**, **cudaIpcGetMemHandle**, **cudaIpcOpenMemHandle**, **cudaIpcCloseMemHandle**

cudaError_t cudaIpcOpenMemHandle (void ** devPtr, cudaIpcMemHandle_t handle, unsigned int flags)

Maps memory exported from another process with **cudaIpcGetMemHandle** into the current device address space. For contexts on different devices **cudaIpcOpenMemHandle** can attempt to enable peer access between the devices as if the user called **cudaDeviceEnablePeerAccess**. This behavior is controlled by the **cudaIpcMemLazyEnablePeerAccess** flag. **cudaDeviceCanAccessPeer** can determine if a mapping is possible.

Contexts that may open cudaIpcMemHandles are restricted in the following way. cudaIpcMemHandles from each device in a given process may only be opened by one context per device per other process.

Memory returned from **cudaIpcOpenMemHandle** must be freed with **cudaIpcCloseMemHandle**.

Calling **cudaFree** on an exported memory region before calling **cudaIpcCloseMemHandle** in the importing context will result in undefined behavior.

IPC functionality is restricted to devices with support for unified addressing on Linux operating systems.

Parameters:
devPtr - Returned device pointer
handle - cudaIpcMemHandle to open
flags - Flags for this operation. Must be specified as **cudaIpcMemLazyEnablePeerAccess**

Returns:
cudaSuccess, **cudaErrorMapBufferObjectFailed**, **cudaErrorInvalidResourceHandle**, **cudaErrorTooManyPeers**

Note:
No guarantees are made about the address returned in `*devPtr`. In particular, multiple processes may not receive the same address for the same `handle`.

See also:
cudaMalloc, **cudaFree**, **cudaIpcGetEventHandle**, **cudaIpcOpenEventHandle**, **cudaIpcGetMemHandle**, **cudaIpcCloseMemHandle**, **cudaDeviceEnablePeerAccess**, **cudaDeviceCanAccessPeer**,

cudaError_t cudaSetDevice (int device)

Sets `device` as the current device for the calling host thread. Valid device id's are 0 to (**cudaGetDeviceCount()** - 1).

Any device memory subsequently allocated from this host thread using **cudaMalloc()**, **cudaMallocPitch()** or **cudaMallocArray()** will be physically resident on `device`. Any host memory allocated from this host thread using **cudaMallocHost()** or **cudaHostAlloc()** or **cudaHostRegister()** will have its lifetime associated with `device`. Any streams or events created from this host thread will be associated with `device`. Any kernels launched from this host thread using the <<<>>> operator or **cudaLaunch()** will be executed on `device`.

This call may be made from any host thread, to any device, and at any time. This function will do no synchronization with the previous or new device, and should be considered a very low overhead call.

Parameters:
device - Device on which the active host thread should execute the device code.

Returns:
cudaSuccess, **cudaErrorInvalidDevice**, **cudaErrorDeviceAlreadyInUse**

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

See also:

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

cudaError_t cudaSetDeviceFlags (unsigned int flags)

Records `flags` as the flags to use when initializing the current device. If no device has been made current to the calling thread then `flags` will be applied to the initialization of any device initialized by the calling host thread, unless that device has had its initialization flags set explicitly by this or any host thread.

If the current device has been set and that device has already been initialized then this call will fail with the error **cudaErrorSetOnActiveProcess**. In this case it is necessary to reset `device` using **cudaDeviceReset()** before the device's initialization flags may be set.

The two LSBs of the `flags` parameter can be used to control how the CPU thread interacts with the OS scheduler when waiting for results from the device.

- **cudaDeviceScheduleAuto**: 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 device, otherwise CUDA will not yield while waiting for results and actively spin on the processor.
- **cudaDeviceScheduleSpin**: Instruct CUDA to actively spin when waiting for results from the device. This can decrease latency when waiting for the device, but may lower the performance of CPU threads if they are performing work in parallel with the CUDA thread.
- **cudaDeviceScheduleYield**: Instruct CUDA to yield its thread when waiting for results from the device. This can increase latency when waiting for the device, but can increase the performance of CPU threads performing work in parallel with the device.
- **cudaDeviceScheduleBlockingSync**: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting for the device to finish work.
- **cudaDeviceBlockingSync**: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting for the device to finish work.
  Deprecated: This flag was deprecated as of CUDA 4.0 and replaced with **cudaDeviceScheduleBlockingSync**.
- **cudaDeviceMapHost**: This flag must be set in order to allocate pinned host memory that is accessible to the device. If this flag is not set, **cudaHostGetDevicePointer()** will always return a failure code.
- **cudaDeviceLmemResizeToMax**: Instruct CUDA to not reduce local memory after resizing local memory for a kernel. This can prevent thrashing by local memory allocations when launching many kernels with high local memory usage at the cost of potentially increased memory usage.

Parameters:

flags - Parameters for device operation

Returns:

cudaSuccess, **cudaErrorInvalidDevice**, **cudaErrorSetOnActiveProcess**

See also:

cudaGetDeviceCount, **cudaGetDevice**, **cudaGetDeviceProperties**, **cudaSetDevice**, **cudaSetValidDevices**, **cudaChooseDevice**

cudaError_t cudaSetValidDevices (int * device_arr, int len)

Sets a list of devices for CUDA execution in priority order using `device_arr`. The parameter `len` specifies the number of elements in the list. CUDA will try devices from the list sequentially until it finds one that works. If this function is not called, or if it is called with a `len` of 0, then CUDA will go back to its default behavior of trying devices sequentially from a default list containing all of the available CUDA devices in the system. If a specified device ID in the list does not exist, this function will return **cudaErrorInvalidDevice**. If `len` is not 0 and `device_arr` is NULL or if `len` exceeds the number of devices in the system, then **cudaErrorInvalidValue** is returned.

Parameters:

device_arr - List of devices to try
len - Number of devices in specified list

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevice**

Note:

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

See also:

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

Author

Generated automatically by Doxygen from the source code.