NAME

Memory Management −

Functions

cudaError_t cudaArrayGetInfo (struct **cudaChannelFormatDesc** *desc, struct **cudaExtent** *extent, unsigned int *flags, **cudaArray_t** array)
Gets info about the specified cudaArray.

__cudart_builtin__ **cudaError_t cudaFree** (void *devPtr)
Frees memory on the device.

cudaError_t cudaFreeArray (**cudaArray_t** array)
Frees an array on the device.

cudaError_t cudaFreeHost (void *ptr)
Frees page-locked memory.

cudaError_t cudaFreeMipmappedArray (**cudaMipmappedArray_t** mipmappedArray)
Frees a mipmapped array on the device.

cudaError_t cudaGetMipmappedArrayLevel (**cudaArray_t** *levelArray, **cudaMipmappedArray_const_t** mipmappedArray, unsigned int level)
Gets a mipmap level of a CUDA mipmapped array.

cudaError_t cudaGetSymbolAddress (void **devPtr, const void *symbol)
Finds the address associated with a CUDA symbol.

cudaError_t cudaGetSymbolSize (size_t *size, const void *symbol)
Finds the size of the object associated with a CUDA symbol.

cudaError_t cudaHostAlloc (void **pHost, size_t size, unsigned int flags)
Allocates page-locked memory on the host.

cudaError_t cudaHostGetDevicePointer (void **pDevice, void *pHost, unsigned int flags)
Passes back device pointer of mapped host memory allocated by cudaHostAlloc or registered by cudaHostRegister.

cudaError_t cudaHostGetFlags (unsigned int *pFlags, void *pHost)
Passes back flags used to allocate pinned host memory allocated by cudaHostAlloc.

cudaError_t cudaHostRegister (void *ptr, size_t size, unsigned int flags)
Registers an existing host memory range for use by CUDA.

cudaError_t cudaHostUnregister (void *ptr)
Unregisters a memory range that was registered with cudaHostRegister.

__cudart_builtin__ **cudaError_t cudaMalloc** (void **devPtr, size_t size)
Allocate memory on the device.

cudaError_t cudaMalloc3D (struct **cudaPitchedPtr** *pitchedDevPtr, struct **cudaExtent** extent)
Allocates logical 1D, 2D, or 3D memory objects on the device.

cudaError_t cudaMalloc3DArray (**cudaArray_t** *array, const struct **cudaChannelFormatDesc** *desc, struct **cudaExtent** extent, unsigned int flags=0)
Allocate an array on the device.

cudaError_t cudaMallocArray (**cudaArray_t** *array, const struct **cudaChannelFormatDesc** *desc, size_t width, size_t height=0, unsigned int flags=0)
Allocate an array on the device.

cudaError_t cudaMallocHost (void **ptr, size_t size)
Allocates page-locked memory on the host.

__cudart_builtin__ **cudaError_t cudaMallocManaged** (void **devPtr, size_t size, unsigned int flags)
Allocates memory that will be automatically managed by the Unified Memory system.

cudaError_t cudaMallocMipmappedArray (**cudaMipmappedArray_t** *mipmappedArray, const struct **cudaChannelFormatDesc** *desc, struct **cudaExtent** extent, unsigned int numLevels, unsigned int flags=0)
Allocate a mipmapped array on the device.

cudaError_t cudaMallocPitch (void **devPtr, size_t *pitch, size_t width, size_t height)
Allocates pitched memory on the device.

cudaError_t cudaMemcpy (void *dst, const void *src, size_t count, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpy2D (void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpy2DArrayToArray (**cudaArray_t** dst, size_t wOffsetDst, size_t

hOffsetDst, **cudaArray_const_t** src, size_t wOffsetSrc, size_t hOffsetSrc, size_t width, size_t height, enum **cudaMemcpyKind** kind=cudaMemcpyDeviceToDevice)
Copies data between host and device.

__cudart_builtin__ **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=0)
Copies data between host and device.

cudaError_t cudaMemcpy2DFromArray (void *dst, size_t dpitch, **cudaArray_const_t** src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpy2DFromArrayAsync (void *dst, size_t dpitch, **cudaArray_const_t** src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data between host and device.

cudaError_t cudaMemcpy2DToArray (**cudaArray_t** dst, size_t wOffset, size_t hOffset, const void *src, size_t spitch, size_t width, size_t height, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpy2DToArrayAsync (**cudaArray_t** dst, size_t wOffset, size_t hOffset, const void *src, size_t spitch, size_t width, size_t height, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data between host and device.

cudaError_t cudaMemcpy3D (const struct **cudaMemcpy3DParms** *p)
Copies data between 3D objects.

__cudart_builtin__ **cudaError_t cudaMemcpy3DAsync** (const struct **cudaMemcpy3DParms** *p, **cudaStream_t** stream=0)
Copies data between 3D objects.

cudaError_t cudaMemcpy3DPeer (const struct **cudaMemcpy3DPeerParms** *p)
Copies memory between devices.

cudaError_t cudaMemcpy3DPeerAsync (const struct **cudaMemcpy3DPeerParms** *p, **cudaStream_t** stream=0)
Copies memory between devices asynchronously.

cudaError_t cudaMemcpyArrayToArray (**cudaArray_t** dst, size_t wOffsetDst, size_t hOffsetDst, **cudaArray_const_t** src, size_t wOffsetSrc, size_t hOffsetSrc, size_t count, enum **cudaMemcpyKind** kind=cudaMemcpyDeviceToDevice)
Copies data between host and device.

__cudart_builtin__ **cudaError_t cudaMemcpyAsync** (void *dst, const void *src, size_t count, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data between host and device.

cudaError_t cudaMemcpyFromArray (void *dst, **cudaArray_const_t** src, size_t wOffset, size_t hOffset, size_t count, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpyFromArrayAsync (void *dst, **cudaArray_const_t** src, size_t wOffset, size_t hOffset, size_t count, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data between host and device.

cudaError_t cudaMemcpyFromSymbol (void *dst, const void *symbol, size_t count, size_t offset=0, enum **cudaMemcpyKind** kind=cudaMemcpyDeviceToHost)
Copies data from the given symbol on the device.

cudaError_t cudaMemcpyFromSymbolAsync (void *dst, const void *symbol, size_t count, size_t offset, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data from the given symbol on the device.

cudaError_t cudaMemcpyPeer (void *dst, int dstDevice, const void *src, int srcDevice, size_t count)
Copies memory between two devices.

cudaError_t cudaMemcpyPeerAsync (void *dst, int dstDevice, const void *src, int srcDevice, size_t count, **cudaStream_t** stream=0)
Copies memory between two devices asynchronously.

cudaError_t cudaMemcpyToArray (**cudaArray_t** dst, size_t wOffset, size_t hOffset, const void *src, size_t count, enum **cudaMemcpyKind** kind)
Copies data between host and device.

cudaError_t cudaMemcpyToArrayAsync (**cudaArray_t** dst, size_t wOffset, size_t hOffset, const

void *src, size_t count, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data between host and device.

cudaError_t cudaMemcpyToSymbol (const void *symbol, const void *src, size_t count, size_t offset=0, enum **cudaMemcpyKind** kind=cudaMemcpyHostToDevice)
Copies data to the given symbol on the device.

cudaError_t cudaMemcpyToSymbolAsync (const void *symbol, const void *src, size_t count, size_t offset, enum **cudaMemcpyKind** kind, **cudaStream_t** stream=0)
Copies data to the given symbol on the device.

cudaError_t cudaMemGetInfo (size_t *free, size_t *total)
Gets free and total device memory.

cudaError_t cudaMemset (void *devPtr, int value, size_t count)
Initializes or sets device memory to a value.

cudaError_t cudaMemset2D (void *devPtr, size_t pitch, int value, size_t width, size_t height)
Initializes or sets device memory to a value.

__cudart_builtin__ **cudaError_t cudaMemset2DAsync** (void *devPtr, size_t pitch, int value, size_t width, size_t height, **cudaStream_t** stream=0)
Initializes or sets device memory to a value.

cudaError_t cudaMemset3D (struct **cudaPitchedPtr** pitchedDevPtr, int value, struct **cudaExtent** extent)
Initializes or sets device memory to a value.

__cudart_builtin__ **cudaError_t cudaMemset3DAsync** (struct **cudaPitchedPtr** pitchedDevPtr, int value, struct **cudaExtent** extent, **cudaStream_t** stream=0)
Initializes or sets device memory to a value.

__cudart_builtin__ **cudaError_t cudaMemsetAsync** (void *devPtr, int value, size_t count, **cudaStream_t** stream=0)
Initializes or sets device memory to a value.

struct **cudaExtent make_cudaExtent** (size_t w, size_t h, size_t d)
Returns a **cudaExtent** *based on input parameters.*

struct **cudaPitchedPtr make_cudaPitchedPtr** (void *d, size_t p, size_t xsz, size_t ysz)
Returns a **cudaPitchedPtr** *based on input parameters.*

struct **cudaPos make_cudaPos** (size_t x, size_t y, size_t z)
Returns a **cudaPos** *based on input parameters.*

Detailed Description

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

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

Some functions have overloaded C++ API template versions documented separately in the **C++ API Routines** module.

Function Documentation

cudaError_t cudaArrayGetInfo (struct cudaChannelFormatDesc * desc, struct cudaExtent * extent, unsigned int * flags, cudaArray_t array)

Returns in `*desc`, `*extent` and `*flags` respectively, the type, shape and flags of `array`.

Any of `*desc`, `*extent` and `*flags` may be specified as NULL.

Parameters:

desc - Returned array type
extent - Returned array shape. 2D arrays will have depth of zero
flags - Returned array flags
array - The cudaArray to get info for

Returns:

cudaSuccess, **cudaErrorInvalidValue**

Note:

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

__cudart_builtin__ cudaError_t cudaFree (void * devPtr)

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.

Parameters:
devPtr - Device pointer to memory to free

Returns:
cudaSuccess, **cudaErrorInvalidDevicePointer**, **cudaErrorInitializationError**

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

See also:
cudaMalloc, **cudaMallocPitch**, **cudaMallocArray**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaHostAlloc**

cudaError_t cudaFreeArray (cudaArray_t array)
Frees the CUDA array `array`, which must have been * returned by a previous call to **cudaMallocArray()**. If **cudaFreeArray**(`array`) has already been called before, **cudaErrorInvalidValue** is returned. If `devPtr` is 0, no operation is performed.

Parameters:
array - Pointer to array to free

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

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

See also:
cudaMalloc, **cudaMallocPitch**, **cudaFree**, **cudaMallocArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**

cudaError_t cudaFreeHost (void * ptr)
Frees the memory space pointed to by `hostPtr`, which must have been returned by a previous call to **cudaMallocHost()** or **cudaHostAlloc()**.

Parameters:
ptr - Pointer to memory to free

Returns:
cudaSuccess, **cudaErrorInitializationError**

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

See also:
cudaMalloc, **cudaMallocPitch**, **cudaFree**, **cudaMallocArray**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaHostAlloc**

cudaError_t cudaFreeMipmappedArray (cudaMipmappedArray_t mipmappedArray)
Frees the CUDA mipmapped array `mipmappedArray`, which must have been returned by a previous call to **cudaMallocMipmappedArray()**. If **cudaFreeMipmappedArray**(`mipmappedArray`) has already been called before, **cudaErrorInvalidValue** is returned.

Parameters:
mipmappedArray - Pointer to mipmapped array to free

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

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

See also:
cudaMalloc, **cudaMallocPitch**, **cudaFree**, **cudaMallocArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**

cudaError_t cudaGetMipmappedArrayLevel (cudaArray_t * levelArray, cudaMipmappedArray_const_t mipmappedArray, unsigned int level)

Returns in `*levelArray` a CUDA array that represents a single mipmap level of the CUDA mipmapped array `mipmappedArray`.

If `level` is greater than the maximum number of levels in this mipmapped array, **cudaErrorInvalidValue** is returned.

Parameters:

levelArray - Returned mipmap level CUDA array
mipmappedArray - CUDA mipmapped array
level - Mipmap level

Returns:

cudaSuccess, **cudaErrorInvalidValue**

Note:

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

See also:

cudaMalloc3D, **cudaMalloc**, **cudaMallocPitch**, **cudaFree**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**, **make_cudaExtent**

cudaError_t cudaGetSymbolAddress (void ** devPtr, const void * symbol)

Returns in `*devPtr` the address of symbol `symbol` on the device. `symbol` is a variable that resides in global or constant memory space. If `symbol` cannot be found, or if `symbol` is not declared in the global or constant memory space, `*devPtr` is unchanged and the error **cudaErrorInvalidSymbol** is returned.

Parameters:

devPtr - Return device pointer associated with symbol
symbol - Device symbol address

Returns:

cudaSuccess, **cudaErrorInvalidSymbol**

Note:

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

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:

cudaGetSymbolAddress (C++ API), **cudaGetSymbolSize (C API)**

cudaError_t cudaGetSymbolSize (size_t * size, const void * symbol)

Returns in `*size` the size of symbol `symbol`. `symbol` is 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 the error **cudaErrorInvalidSymbol** is returned.

Parameters:

size - Size of object associated with symbol
symbol - Device symbol address

Returns:

cudaSuccess, **cudaErrorInvalidSymbol**

Note:

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

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:

cudaGetSymbolAddress (C API), **cudaGetSymbolSize (C++ API)**

cudaError_t cudaHostAlloc (void ** pHost, size_t size, unsigned int flags)

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 pinned memory 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.

The `flags` parameter enables different options to be specified that affect the allocation, as follows.

- **cudaHostAllocDefault**: This flag's value is defined to be 0 and causes **cudaHostAlloc()** to emulate **cudaMallocHost()**.
- **cudaHostAllocPortable**: The memory returned by this call will be considered as pinned memory by all CUDA contexts, not just the one that performed the allocation.
- **cudaHostAllocMapped**: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling **cudaHostGetDevicePointer()**.
- **cudaHostAllocWriteCombined**: Allocates the memory as write-combined (WC). WC memory can be transferred across the PCI Express bus more quickly on some system configurations, but cannot be read efficiently by most CPUs. WC memory is a good option for buffers that will be written by the CPU and read by the device via mapped pinned memory or host->device transfers.

All of these flags are orthogonal to one another: a developer may allocate memory that is portable, mapped and/or write-combined with no restrictions.

cudaSetDeviceFlags() must have been called with the **cudaDeviceMapHost** flag in order for the **cudaHostAllocMapped** flag to have any effect.

The **cudaHostAllocMapped** flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to **cudaHostGetDevicePointer()** because the memory may be mapped into other CUDA contexts via the **cudaHostAllocPortable** flag.

Memory allocated by this function must be freed with **cudaFreeHost()**.

Parameters:

pHost - Device pointer to allocated memory
size - Requested allocation size in bytes
flags - Requested properties of allocated memory

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaSetDeviceFlags, **cudaMallocHost (C API)**, **cudaFreeHost**

cudaError_t cudaHostGetDevicePointer (void ** pDevice, void * pHost, unsigned int flags)

Passes back the device pointer corresponding to the mapped, pinned host buffer allocated by **cudaHostAlloc()** or registered by **cudaHostRegister()**.

cudaHostGetDevicePointer() will fail if the **cudaDeviceMapHost** flag was not specified before deferred context creation occurred, or if called on a device that does not support mapped, pinned memory.

`flags` provides for future releases. For now, it must be set to 0.

Parameters:

pDevice - Returned device pointer for mapped memory
pHost - Requested host pointer mapping
flags - Flags for extensions (must be 0 for now)

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaSetDeviceFlags, **cudaHostAlloc**

cudaError_t cudaHostGetFlags (unsigned int * pFlags, void * pHost)

cudaHostGetFlags() will fail if the input pointer does not reside in an address range allocated by **cudaHostAlloc()**.

Parameters:

pFlags - Returned flags word
pHost - Host pointer

Returns:

cudaSuccess, **cudaErrorInvalidValue**

Note:

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

See also:

cudaHostAlloc

cudaError_t cudaHostRegister (void * ptr, size_t size, unsigned int flags)

Page-locks the memory range specified by `ptr` and `size` and maps it for the device(s) as specified by `flags`. This memory range also is added to the same tracking mechanism as **cudaHostAlloc()** to automatically accelerate 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 that has not been registered. Page-locking excessive amounts of memory 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 register staging areas for data exchange between host and device.

The `flags` parameter enables different options to be specified that affect the allocation, as follows.

- **cudaHostRegisterPortable**: The memory returned by this call will be considered as pinned memory by all CUDA contexts, not just the one that performed the allocation.
- **cudaHostRegisterMapped**: Maps the allocation into the CUDA address space. The device pointer to the memory may be obtained by calling **cudaHostGetDevicePointer()**. This feature is available only on GPUs with compute capability greater than or equal to 1.1.

All of these flags are orthogonal to one another: a developer may page-lock memory that is portable or mapped with no restrictions.

The CUDA context must have been created with the cudaMapHost flag in order for the **cudaHostRegisterMapped** flag to have any effect.

The **cudaHostRegisterMapped** flag may be specified on CUDA contexts for devices that do not support mapped pinned memory. The failure is deferred to **cudaHostGetDevicePointer()** because the memory may be mapped into other CUDA contexts via the **cudaHostRegisterPortable** flag.

The memory page-locked by this function must be unregistered with **cudaHostUnregister()**.

Parameters:

ptr - Host pointer to memory to page-lock
size - Size in bytes of the address range to page-lock in bytes
flags - Flags for allocation request

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaHostUnregister, **cudaHostGetFlags**, **cudaHostGetDevicePointer**

cudaError_t cudaHostUnregister (void * ptr)

Unmaps the memory range whose base address is specified by `ptr`, and makes it pageable again.

The base address must be the same one specified to **cudaHostRegister()**.

Parameters:

ptr - Host pointer to memory to unregister

Returns:

cudaSuccess, **cudaErrorInvalidValue**

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

See also:
cudaHostUnregister

__cudart_builtin__ cudaError_t cudaMalloc (void ** devPtr, size_t size)
Allocates `size` 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.

Parameters:
devPtr - Pointer to allocated device memory
size - Requested allocation size in bytes

Returns:
cudaSuccess, **cudaErrorMemoryAllocation**

See also:
cudaMallocPitch, **cudaFree**, **cudaMallocArray**, **cudaFreeArray**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**

cudaError_t cudaMalloc3D (struct cudaPitchedPtr * pitchedDevPtr, struct cudaExtent extent)
Allocates at least `width` * `height` * `depth` bytes of linear memory on the device and returns a **cudaPitchedPtr** 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 `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 and 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).

Parameters:
pitchedDevPtr - Pointer to allocated pitched device memory
extent - Requested allocation size (`width` field in bytes)

Returns:
cudaSuccess, **cudaErrorMemoryAllocation**

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

See also:
cudaMallocPitch, **cudaFree**, **cudaMemcpy3D**, **cudaMemset3D**, **cudaMalloc3DArray**, **cudaMallocArray**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**, **make_cudaPitchedPtr**, **make_cudaExtent**

cudaError_t cudaMalloc3DArray (cudaArray_t * array, const struct cudaChannelFormatDesc * desc, struct cudaExtent extent, unsigned int flags = `0`)
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**, or **cudaChannelFormatKindFloat**.

cudaMalloc3DArray() can allocate the following:

- A 1D array is allocated if the height and depth extents are both zero.
- A 2D array is allocated if only the depth extent is zero.
- A 3D array is allocated if all three extents are non-zero.
- A 1D layered CUDA array is allocated if only the height extent is zero and the cudaArrayLayered flag is set. Each layer is a 1D array. The number of layers is determined by the depth extent.
- A 2D layered CUDA array is allocated if all three extents are non-zero and the cudaArrayLayered flag is set. Each layer is a 2D array. The number of layers is determined by the depth extent.
- A cubemap CUDA array is allocated if all three extents are non-zero and the cudaArrayCubemap flag is set. Width must be equal to height, and depth must be six. A cubemap is a special type of 2D layered CUDA array, where the six layers represent the six faces of a cube. The order of the six layers in memory is the same as that listed in **cudaGraphicsCubeFace**.
- A cubemap layered CUDA array is allocated if all three extents are non-zero, and both, cudaArrayCubemap and cudaArrayLayered flags are set. Width must be equal to height, and depth must be a multiple of six. A cubemap layered CUDA array is a special type of 2D layered CUDA array that consists of a collection of cubemaps. The first six layers represent the first cubemap, the next six layers form the second cubemap, and so on.

The `flags` parameter enables different options to be specified that affect the allocation, as follows.

- **cudaArrayDefault**: This flag's value is defined to be 0 and provides default array allocation
- **cudaArrayLayered**: Allocates a layered CUDA array, with the depth extent indicating the number of layers
- **cudaArrayCubemap**: Allocates a cubemap CUDA array. Width must be equal to height, and depth must be six. If the cudaArrayLayered flag is also set, depth must be a multiple of six.
- **cudaArraySurfaceLoadStore**: Allocates a CUDA array that could be read from or written to using a surface reference.
- **cudaArrayTextureGather**: This flag indicates that texture gather operations will be performed on the CUDA array. Texture gather can only be performed on 2D CUDA arrays.

The width, height and depth extents must meet certain size requirements as listed in the following table. All values are specified in elements.

Note that 2D CUDA arrays have different size requirements if the **cudaArrayTextureGather** flag is set. In that case, the valid range for (width, height, depth) is ((1,maxTexture2DGather[0]), (1,maxTexture2DGather[1]), 0).

Parameters:

array - Pointer to allocated array in device memory
desc - Requested channel format
extent - Requested allocation size (`width` field in elements)
flags - Flags for extensions

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaMalloc3D, **cudaMalloc**, **cudaMallocPitch**, **cudaFree**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**, **make_cudaExtent**

cudaError_t cudaMallocArray (cudaArray_t * array, const struct cudaChannelFormatDesc * desc, size_t width, size_t height = `0`, unsigned int flags = `0`)

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**, or **cudaChannelFormatKindFloat**.

The `flags` parameter enables different options to be specified that affect the allocation, as follows.

- **cudaArrayDefault**: This flag's value is defined to be 0 and provides default array allocation
- **cudaArraySurfaceLoadStore**: Allocates an array that can be read from or written to using a surface reference
- **cudaArrayTextureGather**: This flag indicates that texture gather operations will be performed on the array.

`width` and `height` must meet certain size requirements. See **cudaMalloc3DArray()** for more details.

Parameters:

array - Pointer to allocated array in device memory
desc - Requested channel format
width - Requested array allocation width
height - Requested array allocation height
flags - Requested properties of allocated array

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaMalloc, **cudaMallocPitch**, **cudaFree**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaHostAlloc**

cudaError_t cudaMallocHost (void ** ptr, size_t size)

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.

Parameters:

ptr - Pointer to allocated host memory
size - Requested allocation size in bytes

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaMalloc, **cudaMallocPitch**, **cudaMallocArray**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaHostAlloc**, **cudaFree**, **cudaFreeArray**, **cudaMallocHost (C++ API)**, **cudaFreeHost**, **cudaHostAlloc**

__cudart_builtin__ cudaError_t cudaMallocManaged (void ** devPtr, size_t size, unsigned int flags)

Allocates `size` bytes of managed memory on the device and returns in `*devPtr` a pointer to the allocated memory. If the device doesn't support allocating managed memory, **cudaErrorNotSupported** is returned. Support for managed memory can be queried using the device attribute **cudaDevAttrManagedMemory**. The allocated memory is suitably aligned for any kind of variable. The memory is not cleared. If `size` is 0, **cudaMallocManaged** returns **cudaErrorInvalidValue**. The pointer is valid on the CPU and on all GPUs in the system that support

managed memory. All accesses to this pointer must obey the Unified Memory programming model.

`flags` specifies the default stream association for this allocation. `flags` must be one of **cudaMemAttachGlobal** or **cudaMemAttachHost**. If **cudaMemAttachGlobal** is specified, then this memory is accessible from any stream on any device. If **cudaMemAttachHost** is specified, then the allocation is created with initial visibility restricted to host access only; an explicit call to **cudaStreamAttachMemAsync** will be required to enable access on the device.

If the association is later changed via **cudaStreamAttachMemAsync** to a single stream, the default association, as specified during **cudaMallocManaged**, is restored when that stream is destroyed. For __managed__ variables, the default association is always **cudaMemAttachGlobal**. 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.

Memory allocated with **cudaMallocManaged** should be released with **cudaFree**.

On a multi-GPU system with peer-to-peer support, where multiple GPUs support managed memory, the physical storage is created on the GPU which is active at the time **cudaMallocManaged** is called. All other GPUs will reference the data at reduced bandwidth via peer mappings over the PCIe bus. The Unified Memory management system does not migrate memory between GPUs.

On a multi-GPU system where multiple GPUs support managed memory, but not all pairs of such GPUs have peer-to-peer support between them, the physical storage is created in 'zero-copy' or system memory. All GPUs will reference the data at reduced bandwidth over the PCIe bus. In these circumstances, use of the environment variable, CUDA_VISIBLE_DEVICES, is recommended to restrict CUDA to only use those GPUs that have peer-to-peer support. Alternatively, users can also set CUDA_MANAGED_FORCE_DEVICE_ALLOC to a non-zero value to force the driver to always use device memory for physical storage. When this environment variable is set to a non-zero value, all devices used in that process that support managed memory have to be peer-to-peer compatible with each other. The error **cudaErrorInvalidDevice** will be returned if a device that supports managed memory is used and it is not peer-to-peer compatible with any of the other managed memory supporting devices that were previously used in that process, even if **cudaDeviceReset** has been called on those devices. These environment variables are described in the CUDA programming guide under the 'CUDA environment variables' section.

Parameters:

devPtr - Pointer to allocated device memory
size - Requested allocation size in bytes
flags - Must be either **cudaMemAttachGlobal** or **cudaMemAttachHost**

Returns:

cudaSuccess, **cudaErrorMemoryAllocation cudaErrorNotSupported cudaErrorInvalidValue**

See also:

cudaMallocPitch, **cudaFree**, **cudaMallocArray**, **cudaFreeArray**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**, **cudaDeviceGetAttribute**, **cudaStreamAttachMemAsync**

cudaError_t cudaMallocMipmappedArray (cudaMipmappedArray_t * mipmappedArray, const struct cudaChannelFormatDesc * desc, struct cudaExtent extent, unsigned int numLevels, unsigned int flags = 0)

Allocates a CUDA mipmapped array according to the **cudaChannelFormatDesc** structure `desc` and returns a handle to the new CUDA mipmapped array in `*mipmappedArray`. `numLevels` specifies the number of mipmap levels to be allocated. This value is clamped to the range [1, 1 + floor(log2(max(width, height, depth)))].

The **cudaChannelFormatDesc** is defined as:

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

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

cudaMallocMipmappedArray() can allocate the following:

- A 1D mipmapped array is allocated if the height and depth extents are both zero.
- A 2D mipmapped array is allocated if only the depth extent is zero.
- A 3D mipmapped array is allocated if all three extents are non-zero.
- A 1D layered CUDA mipmapped array is allocated if only the height extent is zero and the cudaArrayLayered flag is set. Each layer is a 1D mipmapped array. The number of layers is determined by the depth extent.
- A 2D layered CUDA mipmapped array is allocated if all three extents are non-zero and the cudaArrayLayered flag is set. Each layer is a 2D mipmapped array. The number of layers is determined by the depth extent.
- A cubemap CUDA mipmapped array is allocated if all three extents are non-zero and the cudaArrayCubemap flag is set. Width must be equal to height, and depth must be six. The order of the six layers in memory is the same as that listed in **cudaGraphicsCubeFace**.
- A cubemap layered CUDA mipmapped array is allocated if all three extents are non-zero, and both, cudaArrayCubemap and cudaArrayLayered flags are set. Width must be equal to height, and depth must be a multiple of six. A cubemap layered CUDA mipmapped array is a special type of 2D layered CUDA mipmapped array that consists of a collection of cubemap mipmapped arrays. The first six layers represent the first cubemap mipmapped array, the next six layers form the second cubemap mipmapped array, and so on.

The `flags` parameter enables different options to be specified that affect the allocation, as follows.

- **cudaArrayDefault**: This flag's value is defined to be 0 and provides default mipmapped array allocation
- **cudaArrayLayered**: Allocates a layered CUDA mipmapped array, with the depth extent indicating the number of layers
- **cudaArrayCubemap**: Allocates a cubemap CUDA mipmapped array. Width must be equal to height, and depth must be six. If the cudaArrayLayered flag is also set, depth must be a multiple of six.
- **cudaArraySurfaceLoadStore**: This flag indicates that individual mipmap levels of the CUDA mipmapped array will be read from or written to using a surface reference.
- **cudaArrayTextureGather**: This flag indicates that texture gather operations will be performed on the CUDA array. Texture gather can only be performed on 2D CUDA mipmapped arrays, and the gather operations are performed only on the most detailed mipmap level.

The width, height and depth extents must meet certain size requirements as listed in the following table. All values are specified in elements.

Parameters:

mipmappedArray - Pointer to allocated mipmapped array in device memory
desc - Requested channel format
extent - Requested allocation size (`width` field in elements)
numLevels - Number of mipmap levels to allocate
flags - Flags for extensions

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaMalloc3D, **cudaMalloc**, **cudaMallocPitch**, **cudaFree**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaHostAlloc**, **make_cudaExtent**

cudaError_t cudaMallocPitch (void ** devPtr, size_t * pitch, size_t width, size_t height)

Allocates at least `width` (in bytes) * `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).

Parameters:

devPtr - Pointer to allocated pitched device memory
pitch - Pitch for allocation
width - Requested pitched allocation width (in bytes)
height - Requested pitched allocation height

Returns:

cudaSuccess, **cudaErrorMemoryAllocation**

Note:

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

See also:

cudaMalloc, **cudaFree**, **cudaMallocArray**, **cudaFreeArray**, **cudaMallocHost (C API)**, **cudaFreeHost**, **cudaMalloc3D**, **cudaMalloc3DArray**, **cudaHostAlloc**

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

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.

Parameters:

dst - Destination memory address
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMemcpy2D, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

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

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. `width` must not exceed either `dpitch` or `spitch`. 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` exceeds the maximum allowed.

Parameters:

dst - Destination memory address
dpitch - Pitch of destination memory
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

See also:

cudaMemcpy, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy2DArrayToArray (cudaArray_t dst, size_t wOffsetDst, size_t hOffsetDst, cudaArray_const_t src, size_t wOffsetSrc, size_t hOffsetSrc, size_t width, size_t height, enum cudaMemcpyKind kind = `cudaMemcpyDeviceToDevice`**)**

Copies a matrix (`height` rows of `width` bytes each) from the CUDA array `srcArray` starting at the upper left corner (`wOffsetSrc`, `hOffsetSrc`) to the CUDA array `dst` starting at the upper left corner (`wOffsetDst`, `hOffsetDst`), where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy. `wOffsetDst` + `width` must not exceed the width of the CUDA array `dst`. `wOffsetSrc` + `width` must not exceed the width of the CUDA array `src`.

Parameters:

dst - Destination memory address
wOffsetDst - Destination starting X offset
hOffsetDst - Destination starting Y offset
src - Source memory address
wOffsetSrc - Source starting X offset
hOffsetSrc - Source starting Y offset
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**,

cudaMemcpyToSymbolAsync, **cudaMemcpyFromSymbolAsync**

__cudart_builtin__ 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 = 0)
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. `width` must not exceed either `dpitch` or `spitch`. Calling **cudaMemcpy2DAsync()** with `dst` and `src` pointers that do not match the direction of the copy results in an undefined behavior. **cudaMemcpy2DAsync()** returns an error if `dpitch` or `spitch` is greater than the maximum allowed.

cudaMemcpy2DAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:
dst - Destination memory address
dpitch - Pitch of destination memory
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
stream - Stream identifier

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy2DFromArray (void * dst, size_t dpitch, cudaArray_const_t src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind)
Copies a matrix (`height` rows of `width` bytes each) from the CUDA array `srcArray` starting at the upper left corner (`wOffset`, `hOffset`) 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. `wOffset` + `width` must not exceed the width of the CUDA array `src`. `width` must not exceed `dpitch`. **cudaMemcpy2DFromArray()** returns an error if `dpitch` exceeds the maximum allowed.

Parameters:
dst - Destination memory address
dpitch - Pitch of destination memory
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
width - Width of matrix transfer (columns in bytes)

height - Height of matrix transfer (rows)
kind - Type of transfer

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy2DFromArrayAsync (void * dst, size_t dpitch, cudaArray_const_t src, size_t wOffset, size_t hOffset, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream = `0`)
Copies a matrix (`height` rows of `width` bytes each) from the CUDA array `srcArray` starting at the upper left corner (`wOffset`, `hOffset`) 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. `wOffset` + `width` must not exceed the width of the CUDA array `src`. `width` must not exceed `dpitch`. **cudaMemcpy2DFromArrayAsync()** returns an error if `dpitch` exceeds the maximum allowed.

cudaMemcpy2DFromArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:
dst - Destination memory address
dpitch - Pitch of destination memory
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer
stream - Stream identifier

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy2DToArray (cudaArray_t dst, size_t wOffset, size_t hOffset, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)

Copies a matrix (`height` rows of `width` bytes each) from the memory area pointed to by `src` to the CUDA array `dst` starting at the upper left corner (`wOffset`, `hOffset`) 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. `wOffset` + `width` must not exceed the width of the CUDA array `dst`. `width` must not exceed `spitch`. **cudaMemcpy2DToArray()** returns an error if `spitch` exceeds the maximum allowed.

Parameters:

dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy2DToArrayAsync (cudaArray_t dst, size_t wOffset, size_t hOffset, const void * src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind, cudaStream_t stream = 0)

Copies a matrix (`height` rows of `width` bytes each) from the memory area pointed to by `src` to the CUDA array `dst` starting at the upper left corner (`wOffset`, `hOffset`) 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. `wOffset` + `width` must not exceed the width of the CUDA array `dst`. `width` must not exceed `spitch`. **cudaMemcpy2DToArrayAsync()** returns an error if `spitch` exceeds the maximum allowed.

cudaMemcpy2DToArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:

dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
spitch - Pitch of source memory
width - Width of matrix transfer (columns in bytes)
height - Height of matrix transfer (rows)
kind - Type of transfer

stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpy3D (const struct cudaMemcpy3DParms * p)

```
struct cudaExtent {
  size_t width;
  size_t height;
  size_t depth;
};
struct cudaExtent make_cudaExtent(size_t w, size_t h, size_t d);

struct cudaPos {
  size_t x;
  size_t y;
  size_t z;
};
struct cudaPos make_cudaPos(size_t x, size_t y, size_t z);

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

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.

The source object must lie entirely within the region defined by `srcPos` and `extent`. The destination object must lie entirely within the region defined by `dstPos` and `extent`.

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

Parameters:

p - 3D memory copy parameters

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMalloc3D, **cudaMalloc3DArray**, **cudaMemset3D**, **cudaMemcpy3DAsync**, **cudaMemcpy**, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**, **make_cudaExtent**, **make_cudaPos**

__cudart_builtin__ cudaError_t cudaMemcpy3DAsync (const struct cudaMemcpy3DParms * p, cudaStream_t stream = 0)

```
struct cudaExtent {
  size_t width;
  size_t height;
  size_t depth;
};
struct cudaExtent make_cudaExtent(size_t w, size_t h, size_t d);

struct cudaPos {
  size_t x;
  size_t y;
  size_t z;
};
struct cudaPos make_cudaPos(size_t x, size_t y, size_t z);

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

cudaMemcpy3DAsync() 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 **cudaMemcpy3DAsync()** must specify one of `srcArray` or `srcPtr` and one of `dstArray` or `dstPtr`. Passing more than one non-zero source or destination will cause **cudaMemcpy3DAsync()** 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, **cudaMemcpy3DAsync()** 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.

The source object must lie entirely within the region defined by `srcPos` and `extent`. The destination object must lie entirely within the region defined by `dstPos` and `extent`.

cudaMemcpy3DAsync() returns an error if the pitch of `srcPtr` or `dstPtr` exceeds the maximum allowed. The pitch of a **cudaPitchedPtr** allocated with **cudaMalloc3D()** will always be valid.

cudaMemcpy3DAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:

p - 3D memory copy parameters
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidPitchValue**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cudaMalloc3D, **cudaMalloc3DArray**, **cudaMemset3D**, **cudaMemcpy3D**, **cudaMemcpy**, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**, **make_cudaExtent**, **make_cudaPos**

cudaError_t cudaMemcpy3DPeer (const struct cudaMemcpy3DPeerParms * p)

Perform a 3D memory copy according to the parameters specified in `p`. See the definition of the **cudaMemcpy3DPeerParms** structure for documentation of its parameters.

Note that this function is synchronous with respect to the host only if the source or destination of the transfer is host memory. Note also that this copy is serialized with respect to all pending and future asynchronous work in to the current device, the copy's source device, and the copy's destination device (use **cudaMemcpy3DPeerAsync** to avoid this synchronization).

Parameters:
p - Parameters for the memory copy

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

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

This function exhibits behavior for most use cases.

See also:
cudaMemcpy, **cudaMemcpyPeer**, **cudaMemcpyAsync**, **cudaMemcpyPeerAsync**, **cudaMemcpy3DPeerAsync**

cudaError_t cudaMemcpy3DPeerAsync (const struct cudaMemcpy3DPeerParms * p, cudaStream_t stream = `0`)
Perform a 3D memory copy according to the parameters specified in `p`. See the definition of the **cudaMemcpy3DPeerParms** structure for documentation of its parameters.

Parameters:
p - Parameters for the memory copy
stream - Stream identifier

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

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cudaMemcpy, **cudaMemcpyPeer**, **cudaMemcpyAsync**, **cudaMemcpyPeerAsync**, **cudaMemcpy3DPeerAsync**

cudaError_t cudaMemcpyArrayToArray (cudaArray_t dst, size_t wOffsetDst, size_t hOffsetDst, cudaArray_const_t src, size_t wOffsetSrc, size_t hOffsetSrc, size_t count, enum cudaMemcpyKind kind = `cudaMemcpyDeviceToDevice`**)**
Copies `count` bytes from the CUDA array `src` starting at the upper left corner (`wOffsetSrc`, `hOffsetSrc`) to the CUDA array `dst` starting at the upper left corner (`wOffsetDst`, `hOffsetDst`) where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

Parameters:
dst - Destination memory address
wOffsetDst - Destination starting X offset
hOffsetDst - Destination starting Y offset
src - Source memory address
wOffsetSrc - Source starting X offset
hOffsetSrc - Source starting Y offset
count - Size in bytes to copy
kind - Type of transfer

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidMemcpyDirection**

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

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

__cudart_builtin__ cudaError_t cudaMemcpyAsync (void * dst, const void * src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream = 0)

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 **cudaMemcpyAsync()** with `dst` and `src` pointers that do not match the direction of the copy results in an undefined behavior.

cudaMemcpyAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and the `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:

dst - Destination memory address
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyFromArray (void * dst, cudaArray_const_t src, size_t wOffset, size_t hOffset, size_t count, enum cudaMemcpyKind kind)

Copies `count` bytes from the CUDA array `src` starting at the upper left corner (`wOffset`, hOffset) 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.

Parameters:

dst - Destination memory address
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
count - Size in bytes to copy
kind - Type of transfer

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyFromArrayAsync (void * dst, cudaArray_const_t src, size_t wOffset, size_t hOffset, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream = `0`)

Copies `count` bytes from the CUDA array `src` starting at the upper left corner (`wOffset`, hOffset) 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 with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:

dst - Destination memory address
src - Source memory address
wOffset - Source starting X offset
hOffset - Source starting Y offset
count - Size in bytes to copy
kind - Type of transfer
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyFromSymbol (void * dst, const void * symbol, size_t count, size_t offset = `0`, enum cudaMemcpyKind kind = `cudaMemcpyDeviceToHost`)

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` is a variable that resides in global or constant memory space. `kind` can be either **cudaMemcpyDeviceToHost** or **cudaMemcpyDeviceToDevice**.

Parameters:

dst - Destination memory address
symbol - Device symbol address
count - Size in bytes to copy
offset - Offset from start of symbol in bytes
kind - Type of transfer

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidSymbol**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyFromSymbolAsync (void * dst, const void * symbol, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream_t stream = `0`)
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` is a variable that resides in global or constant memory space. `kind` can be either **cudaMemcpyDeviceToHost** or **cudaMemcpyDeviceToDevice**.

cudaMemcpyFromSymbolAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:
dst - Destination memory address
symbol - Device symbol address
count - Size in bytes to copy
offset - Offset from start of symbol in bytes
kind - Type of transfer
stream - Stream identifier

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidSymbol**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**

cudaError_t cudaMemcpyPeer (void * dst, int dstDevice, const void * src, int srcDevice, size_t count)
Copies memory from one device to memory on another device. `dst` is the base device pointer of the destination memory and `dstDevice` is the destination device. `src` is the base device pointer of the source memory and `srcDevice` is the source device. `count` specifies the number of bytes to copy.

Note that this function is asynchronous with respect to the host, but serialized with respect all pending

and future asynchronous work in to the current device, `srcDevice`, and `dstDevice` (use **cudaMemcpyPeerAsync** to avoid this synchronization).

Parameters:
dst - Destination device pointer
dstDevice - Destination device
src - Source device pointer
srcDevice - Source device
count - Size of memory copy in bytes

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

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

This function exhibits behavior for most use cases.

See also:
cudaMemcpy, **cudaMemcpyAsync**, **cudaMemcpyPeerAsync**, **cudaMemcpy3DPeerAsync**

cudaError_t cudaMemcpyPeerAsync (void * dst, int dstDevice, const void * src, int srcDevice, size_t count, cudaStream_t stream = `0`)
Copies memory from one device to memory on another device. `dst` is the base device pointer of the destination memory and `dstDevice` is the destination device. `src` is the base device pointer of the source memory and `srcDevice` is the source device. `count` specifies the number of bytes to copy.

Note that this function is asynchronous with respect to the host and all work on other devices.

Parameters:
dst - Destination device pointer
dstDevice - Destination device
src - Source device pointer
srcDevice - Source device
count - Size of memory copy in bytes
stream - Stream identifier

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

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:
cudaMemcpy, **cudaMemcpyPeer**, **cudaMemcpyAsync**, **cudaMemcpy3DPeerAsync**

cudaError_t cudaMemcpyToArray (cudaArray_t dst, size_t wOffset, size_t hOffset, const void * src, size_t count, enum cudaMemcpyKind kind)
Copies `count` bytes from the memory area pointed to by `src` to the CUDA array `dst` starting at the upper left corner (`wOffset`, `hOffset`), where `kind` is one of **cudaMemcpyHostToHost**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, or **cudaMemcpyDeviceToDevice**, and specifies the direction of the copy.

Parameters:
dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyToArrayAsync (cudaArray_t dst, size_t wOffset, size_t hOffset, const void * src, size_t count, enum cudaMemcpyKind kind, cudaStream_t stream = `0`)

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

cudaMemcpyToArrayAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToHost** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:

dst - Destination memory address
wOffset - Destination starting X offset
hOffset - Destination starting Y offset
src - Source memory address
count - Size in bytes to copy
kind - Type of transfer
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

Note:

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

See also:

cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyToSymbol (const void * symbol, const void * src, size_t count, size_t offset = `0`, enum cudaMemcpyKind kind = `cudaMemcpyHostToDevice`**)**

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` is a variable that resides in global or constant memory space. `kind` can be either **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToDevice**.

Parameters:

symbol - Device symbol address
src - Source memory address
count - Size in bytes to copy
offset - Offset from start of symbol in bytes
kind - Type of transfer

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidSymbol**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyToSymbolAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemcpyToSymbolAsync (const void * symbol, const void * src, size_t count, size_t offset, enum cudaMemcpyKind kind, cudaStream_t stream = `0`)
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` is a variable that resides in global or constant memory space. `kind` can be either **cudaMemcpyHostToDevice** or **cudaMemcpyDeviceToDevice**.

cudaMemcpyToSymbolAsync() is asynchronous with respect to the host, so the call may return before the copy is complete. The copy can optionally be associated to a stream by passing a non-zero `stream` argument. If `kind` is **cudaMemcpyHostToDevice** and `stream` is non-zero, the copy may overlap with operations in other streams.

Parameters:
symbol - Device symbol address
src - Source memory address
count - Size in bytes to copy
offset - Offset from start of symbol in bytes
kind - Type of transfer
stream - Stream identifier

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidSymbol**, **cudaErrorInvalidDevicePointer**, **cudaErrorInvalidMemcpyDirection**

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

This function exhibits behavior for most use cases.

This function uses standard semantics.

Use of a string naming a variable as the `symbol` parameter was deprecated in CUDA 4.1 and removed in CUDA 5.0.

See also:
cudaMemcpy, **cudaMemcpy2D**, **cudaMemcpyToArray**, **cudaMemcpy2DToArray**, **cudaMemcpyFromArray**, **cudaMemcpy2DFromArray**, **cudaMemcpyArrayToArray**, **cudaMemcpy2DArrayToArray**, **cudaMemcpyToSymbol**, **cudaMemcpyFromSymbol**, **cudaMemcpyAsync**, **cudaMemcpy2DAsync**, **cudaMemcpyToArrayAsync**, **cudaMemcpy2DToArrayAsync**, **cudaMemcpyFromArrayAsync**, **cudaMemcpy2DFromArrayAsync**, **cudaMemcpyFromSymbolAsync**

cudaError_t cudaMemGetInfo (size_t * free, size_t * total)
Returns in `*free` and `*total` respectively, the free and total amount of memory available for allocation by the device in bytes.

Parameters:
free - Returned free memory in bytes

total - Returned total memory in bytes

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

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

cudaError_t cudaMemset (void * devPtr, int value, size_t count)
Fills the first `count` bytes of the memory area pointed to by `devPtr` with the constant byte value `value`.

Note that this function is asynchronous with respect to the host unless `devPtr` refers to pinned host memory.

Parameters:
devPtr - Pointer to device memory
value - Value to set for each byte of specified memory
count - Size in bytes to set

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

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

See also .

See also:
cudaMemset2D, **cudaMemset3D**, **cudaMemsetAsync**, **cudaMemset2DAsync**, **cudaMemset3DAsync**

cudaError_t cudaMemset2D (void * devPtr, size_t pitch, int value, size_t width, size_t height)
Sets to the specified value `value` a matrix (`height` rows of `width` bytes each) pointed to by `dstPtr`. `pitch` is the width 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()**.

Note that this function is asynchronous with respect to the host unless `devPtr` refers to pinned host memory.

Parameters:
devPtr - Pointer to 2D device memory
pitch - Pitch in bytes of 2D device memory
value - Value to set for each byte of specified memory
width - Width of matrix set (columns in bytes)
height - Height of matrix set (rows)

Returns:
cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

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

See also .

See also:
cudaMemset, **cudaMemset3D**, **cudaMemsetAsync**, **cudaMemset2DAsync**, **cudaMemset3DAsync**

__cudart_builtin__ cudaError_t cudaMemset2DAsync (void * devPtr, size_t pitch, int value, size_t width, size_t height, cudaStream_t stream = `0`)
Sets to the specified value `value` a matrix (`height` rows of `width` bytes each) pointed to by `dstPtr`. `pitch` is the width 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()**.

cudaMemset2DAsync() is asynchronous with respect to the host, so the call may return before the

memset is complete. The operation can optionally be associated to a stream by passing a non-zero `stream` argument. If `stream` is non-zero, the operation may overlap with operations in other streams.

Parameters:

devPtr - Pointer to 2D device memory
pitch - Pitch in bytes of 2D device memory
value - Value to set for each byte of specified memory
width - Width of matrix set (columns in bytes)
height - Height of matrix set (rows)
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

Note:

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

See also .

This function uses standard semantics.

See also:

cudaMemset, **cudaMemset2D**, **cudaMemset3D**, **cudaMemsetAsync**, **cudaMemset3DAsync**

cudaError_t cudaMemset3D (struct cudaPitchedPtr pitchedDevPtr, int value, struct cudaExtent extent)

Initializes each element of a 3D array to the specified value `value`. The object to initialize is defined by `pitchedDevPtr`. The `pitch` field of `pitchedDevPtr` is the width in memory in bytes of the 3D array pointed to by `pitchedDevPtr`, 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 `pitchedDevPtr` may perform significantly faster than extents narrower than the `xsize`. Secondarily, extents with `height` equal to the `ysize` of `pitchedDevPtr` will perform faster than when the `height` is shorter than the `ysize`.

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

Note that this function is asynchronous with respect to the host unless `pitchedDevPtr` refers to pinned host memory.

Parameters:

pitchedDevPtr - Pointer to pitched device memory
value - Value to set for each byte of specified memory
extent - Size parameters for where to set device memory (`width` field in bytes)

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

Note:

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

See also .

See also:

cudaMemset, **cudaMemset2D**, **cudaMemsetAsync**, **cudaMemset2DAsync**, **cudaMemset3DAsync**, **cudaMalloc3D**, **make_cudaPitchedPtr**, **make_cudaExtent**

__cudart_builtin__ cudaError_t cudaMemset3DAsync (struct cudaPitchedPtr pitchedDevPtr, int value, struct cudaExtent extent, cudaStream_t stream = 0)

Initializes each element of a 3D array to the specified value `value`. The object to initialize is defined by `pitchedDevPtr`. The `pitch` field of `pitchedDevPtr` is the width in memory in bytes of the 3D array pointed to by `pitchedDevPtr`, 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 `pitchedDevPtr` may perform significantly faster than extents narrower than the `xsize`. Secondarily, extents with `height` equal to the `ysize` of `pitchedDevPtr` will perform faster than when the `height` is shorter than the `ysize`.

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

cudaMemset3DAsync() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero `stream` argument. If `stream` is non-zero, the operation may overlap with operations in other streams.

Parameters:

pitchedDevPtr - Pointer to pitched device memory
value - Value to set for each byte of specified memory
extent - Size parameters for where to set device memory (`width` field in bytes)
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

Note:

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

See also .

This function uses standard semantics.

See also:

cudaMemset, **cudaMemset2D**, **cudaMemset3D**, **cudaMemsetAsync**, **cudaMemset2DAsync**, **cudaMalloc3D**, **make_cudaPitchedPtr**, **make_cudaExtent**

__cudart_builtin__ cudaError_t cudaMemsetAsync (void * devPtr, int value, size_t count, cudaStream_t stream = `0`)

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

cudaMemsetAsync() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero `stream` argument. If `stream` is non-zero, the operation may overlap with operations in other streams.

Parameters:

devPtr - Pointer to device memory
value - Value to set for each byte of specified memory
count - Size in bytes to set
stream - Stream identifier

Returns:

cudaSuccess, **cudaErrorInvalidValue**, **cudaErrorInvalidDevicePointer**

Note:

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

See also .

This function uses standard semantics.

See also:

cudaMemset, **cudaMemset2D**, **cudaMemset3D**, **cudaMemset2DAsync**, **cudaMemset3DAsync**

struct cudaExtent make_cudaExtent (size_t w, size_t h, size_t d) `[read]`

Returns a **cudaExtent** based on the specified input parameters `w`, `h`, and `d`.

Parameters:

w - Width in bytes

h - Height in elements
d - Depth in elements

Returns:
cudaExtent specified by `w`, `h`, and `d`

See also:
make_cudaPitchedPtr, **make_cudaPos**

struct cudaPitchedPtr make_cudaPitchedPtr (void * d, size_t p, size_t xsz, size_t ysz) `[read]`
Returns a **cudaPitchedPtr** based on the specified input parameters `d`, `p`, `xsz`, and `ysz`.

Parameters:
d - Pointer to allocated memory
p - Pitch of allocated memory in bytes
xsz - Logical width of allocation in elements
ysz - Logical height of allocation in elements

Returns:
cudaPitchedPtr specified by `d`, `p`, `xsz`, and `ysz`

See also:
make_cudaExtent, **make_cudaPos**

struct cudaPos make_cudaPos (size_t x, size_t y, size_t z) `[read]`
Returns a **cudaPos** based on the specified input parameters `x`, `y`, and `z`.

Parameters:
x - X position
y - Y position
z - Z position

Returns:
cudaPos specified by `x`, `y`, and `z`

See also:
make_cudaExtent, **make_cudaPitchedPtr**

Author

Generated automatically by Doxygen from the source code.