曦云®系列通用计算 GPU

MXMACA®运行时 API 参考

CSRD-23021-020-F3_V03　　2024-05-07

沐曦专有和三级保密信息　　本文档受 NDA 管控

更新记录

版本	日期	更新说明
V03	2024-05-07	更新以下章节： 2.4.3.7 mcError_t mcFuncGetAttributes (struct mcFuncAttributes * attr, const void * func) 2.12.3.1 mcError_t mcFuncGetAttribute (int * value, mcFunction_attribute attrib, mcFunction_t hfunc)
V02	2024-03-15	新增曦云®系列 GPU 产品信息
V01	2023-10-16	正式版本首次发布

目录

1. MXMACA 运行时简介

MXMACA®运行时（Runtime）API 提供在主机上执行的 C 和 C++函数，用于分配和释放设备内存，在主机内存和设备内存之间传输数据，以及管理具有多个设备的系统。本文档介绍了 MXMACA 驱动提供的所有运行时 API。

1.1 常规行为-流同步

1.1.1 默认流

当 0 作为 mcStream_t 传递或由隐式操作流的 API 传递时，使用的默认流可以配置为具有遗留（legacy）同步行为或每线程（per-thread）同步行为，详细信息请参见 1.1.1.1 遗留默认流和 1.1.1.2 每线程默认流。

可以使用以下任一选项控制行为。

默认流的行为有两种控制方法。

- 编译时使用 mxcc 选项--default-stream：

```
$ mxcc -x maca --default-stream {legacy|null|per-thread} -my code.cpp -o my_app
```

- 在包含任意 MXMACA 头文件之前，可通过定义 MACA_API_Per_thread_DEFAULT_STREAM 宏来启用每线程行为：

```
#define MACA_API_PER_THREAD_DEFAULT_STREAM
#include <mc_runtime_api.h>
```

1.1.1.1 遗留默认流

遗留默认流是一个隐式流，它与同一个 mcContext 中除非阻塞流之外的所有流同步，如下所述。对于仅使用运行时 API 的应用程序，每个设备将有一个上下文。当在遗留流中执行操作（如内核启动或 mcStreamWaitEvent()）时，遗留流先等待所有阻塞流，该操作在遗留流中排队进行，然后所有阻塞流等待遗留流。

例如，运行以下代码，先在流 my_stream 中启动内核 my_kernel_1，在遗留流中启动内核 my_kernel_2，然后在流 my_stream 中启动内核 my_kernel_3：

```
my_kernel_1<<<1, 1, 0, my_stream >>>();
my_kernel_2<<<1, 1>>>();
my_kernel_3<<<1, 1, 0, my_stream >>>()
```

产生的结果行为是：my_kernel_2 会阻塞 my_kernel_1，且 my_kernel_3 会阻塞 my_kernel_2。

可以使用 mcStreamNonBlocking 标志和流创建 API 来创建不与遗留流同步的非阻塞流。

可以通过 mcStream_t 句柄 mcStreamLegacy 显式使用遗留默认流。

1.1.1.2 每线程默认流

每线程默认流是线程和 mcContext 本地的隐式流，不与其他流同步（正如显式创建的流）。每线程默认流不是非阻塞流，如果在程序中使用遗留默认流和每线程默认流，它将与遗留默认流同步。

可以通过 mcStream_t 句柄 mcStreamPerThread 显式使用每线程默认流。

说明

MXMACA 运行时 API 当前版本仅部分支持每线程默认流。

1.2 常规行为-API 同步

运行时 API 提供同步和异步形式的 memcpy 和 memset 函数，后者具有 Async 后缀。因为每个函数都可能会表现出同步或异步行为，具体取决于传递给函数的参数。

1.2.1 Memcpy

在运行时 API 参考文档中，每个 memcpy 函数都被分为同步或异步，定义如下。

1.2.1.1 同步

- 所有涉及同一内存区域的传输相对于主机都是完全同步的。
- 对于从分页（pageable）主机内存到设备内存的传输，启动复制前执行流同步。一旦分页缓冲区被复制到分级存储，用于到设备内存的 DMA 传输，该函数就会返回，但 DMA 到目标内存的传输可能尚未完成。
- 对于从固页（pinned）主机内存到设备内存的传输，该函数相对于主机是同步的。
- 对于从设备内存到分页或固页主机内存的传输，该函数仅在复制完成后返回。
- 对于从设备内存到设备内存的传输，不执行主机端同步。
- 对于从任意主机内存到任意主机内存的传输，该函数相对于主机是完全同步的。

1.2.1.2 异步

- 对于从设备内存到分页主机内存的传输，该函数仅在复制完成后返回。
- 对于从任意主机内存到任意主机内存的传输，该函数相对于主机是完全同步的。
- 如果必须先将分页内存暂存到固页内存，MXMACA 驱动可以与流同步，并将副本暂存到固页内存。
- 对于其他所有传输，该函数是完全异步的。

1.2.2 Memset

同步 memset 函数相对于主机是异步的，除非目标是固页主机内存或统一内存区域，在这种情况下，它们是完全同步的。异步版本相对于主机始终是异步的。

1.2.3 内核启动

内核启动相对于主机是异步的。有关并发内核执行和数据传输的详细信息，参见《曦云®系列通用计算 GPU 运行时 API 编程指南》。

2. MXMACA 运行时 API 模块

2.1 初始化

2.1.1 模块描述

本章节介绍 MXMACA 运行时 API 的初始化函数。

2.1.2 函数

`mcError_t` **`mcInit`** `(unsigned int flags)`

显式初始化 MC 运行时。

2.1.3 函数介绍

2.1.3.1 mcError_t mcInit (unsigned int *flags*)

显式初始化 MC 运行时。

参数

in	*flags*	MC 的初始化标志

返回值

`mcErrorInvalidValue, mcSuccess`

初始化 MXMACA 运行时设备和上下文，并且必须在其他 MC API 之前调用。如果未调用 `mcInit`，则大多数 MC API 将隐式调用 `mcInit`。

2.2 设备管理

2.2.1 模块描述

本章节介绍 MXMACA 运行时 API 的设备管理函数。

2.2.2 函数

- `mcError_t` **`mcDeviceGetUuid`** `(mcUuid_t *uuid, mcDevice_t device)`

  返回设备的 UUID。

- `mcError_t` **`mcChooseDevice`** `(int *deviceId, const` **`mcDeviceProp_t`** `*prop)`

  选择最符合条件的设备。

- `mcError_t` **`mcDeviceGetAttribute`** `(int *value, enum mcDeviceAttribute_t attr, int deviceId)`

  查询特定的设备属性。

- `mcError_t` **`mcDeviceGetLimit`** `(size_t *pValue, enum mcLimit_t limit)`

  获取当前设备的资源限制。

- `mcError_t` **`mcDeviceGetP2PAttribute`** `(int *value, enum mcDeviceP2PAttr attr, int srcDevice, int dstDevice)`

  查询两个设备之间链接的属性。

- `mcError_t` **`mcDeviceGetPCIBusId`** `(char *pciBusId, int len, int device)`

  返回设备的 PCI 总线 ID 字符串。

- `mcError_t` **`mcDeviceGetSharedMemConfig`** `(enum mcSharedMemConfig *pConfig)`

  返回当前设备的共享内存配置。

- `mcError_t` **`mcDeviceGetStreamPriorityRange`** `(int *leastPriority, int *greatestPriority)`

  返回优先级范围。

- `mcError_t` **`mcDeviceReset`** `(void)`

  复位当前设备到默认状态。

- `mcError_t` **`mcDeviceSetCacheConfig`** `(enum mcFuncCache_t cacheConfig)`

  设置当前设备的首选缓存配置。

- `mcError_t` **`mcDeviceSetSharedMemConfig`** `(enum mcSharedMemConfig config)`

  设置当前设备的共享内存配置。

- `mcError_t mcDeviceSynchronize (void)`

  等待当前设备上的所有活跃流完成。

- `mcError_t` **`mcGetDevice`** `(int *deviceId)`

  返回当前正在使用的设备 ID。

- `mcError_t` **`mcGetDeviceCount`** `(int *count)`

  返回具有计算能力的设备的数量。

- `mcError_t` **`mcGetDeviceFlags`** `(unsigned int *flags)`

  获取为当前设备设置的标志。

- `mcError_t` **`mcGetDeviceProperties`** `(mcDeviceProp_t *prop, int deviceId)`

  返回计算设备的设备属性。

- `mcError_t` **`mcSetDevice`** `(int deviceId)`

  将默认设备设置为用于此线程的后续 MC API 调用。

- `mcError_t` **`mcSetDeviceFlags`** `(unsigned int flag)`

  当前设备行为根据传递的标志进行变更。

- `mcError_t` **`mcSetValidDevices`** `(int *deviceArray, int len)`

  从指定的设备列表中选择一个有效设备作为当前设备。

- `mcError_t` **`mcDeviceGet`** `(mcDevice_t *device, int ordinal)`

  返回计算设备的句柄。

- `mcError_t` **`mcDeviceGetName`** `(char *name, int len, mcDevice_t device)`

  返回设备指定的 GPU 设备的产品名称。

- `mcError_t` **`mcDeviceTotalMem`** `(size_t *totmem, mcDevice_t deivce)`

  返回设备上的内存总量。

- `mcError_t` **`mcDeviceGetDefaultMemPool`** `(mcMemPool_t *memPool, mcDevice_t deivce)`

  返回设备的默认内存池（mempool）。

- `mcError_t` **`mcDeviceGetMemPool`** `(mcMemPool_t *memPool, mcDevice_t deivce)`

  获取设备的当前内存池。

- `mcError_t` **`mcDeviceSetLimit`** `(enum mcLimit_t limit, size_t value)`

  设置当前设备的资源限制。

- `mcError_t` **`mcDeviceGetLuid`** `(char *luid, unsigned int *deviceNodeMask, mcDevice_t dev)`

  返回设备的 LUID 和设备节点掩码。

- `mcError_t` **`mcDeviceSetMemPool`** `(mcDevice_t dev, mcMemPool_t pool)`

  设置设备的当前内存池。

2.2.3 函数介绍

2.2.3.1 mcError_t mcChooseDevice (int * *deviceId*, const mcDeviceProp_t * *prop*)

选择最符合条件的设备。

参数

out	*deviceId*	设备 ID： 0, 1, 2
in	*prop*	设备属性指针

返回值

`mcSuccess, mcErrorInvalidValue`

返回与 deviceID 匹配的设备的详细属性信息，属性详细信息项包括：

`totalGlobalMem, sharedMemPerBlock, regsPerBlock, warpSize, maxThreadsPerBlock, totalConstMem, multiProcessorCount, l2CacheSize, maxThreadsPerMultiProcessor, memoryClockRate, memoryBusWidth`

2.2.3.2 mcError_t mcDeviceGet (mcDevice_t * *device*, int *ordinal*)

返回计算设备的句柄。

参数

out	*device*	设备句柄
in	*ordinal*	设备 ID，范围为：0...mcGetDeviceCount()

返回值

`mcSuccess, #mcErrorInavlidDevice`

在`*device` 中返回一个设备句柄，由范围[0, mcGetDeviceCount()-1]内的序号给定。

2.2.3.3 mcError_t mcDeviceGetAttribute (int * *value*, enum mcDeviceAttribute_t *attr*, int *deviceId*)

查询特定的设备属性。

参数

out	*value*	指向要返回的值的指针

in	*attr*	要查询的属性
in	*deviceId*	要查询信息的设备 ID

返回值

`mcSuccess, mcErrorInvalidDevice, mcErrorInvalidValue`

在`*value`中返回设备`device`上属性`attr`的整数值。支持的属性在`mcDeviceAttribute_t`中指定：

- `::mcDevAttrMaxThreadsPerBlock`: 每个块的最大线程数
- `::mcDevAttrMaxBlockDimX`: 块的最大 x 维度
- `::mcDevAttrMaxBlockDimY`: 块的最大 y 维度
- `::mcDevAttrMaxBlockDimZ`: 块的最大 z 维度
- `::mcDevAttrMaxGridDimX`: 网格的最大 x 维度
- `::mcDevAttrMaxGridDimY`: 网格的最大 y 维度
- `::mcDevAttrMaxGridDimZ`: 网格的最大 z 维度
- `::mcDevAttrMaxSharedMemoryPerBlock`: 线程块可用的最大共享内存容量，以字节（byte）为单位
- `::mcDevAttrTotalConstantMemory`: 设备上可用于 MC C 内核中 **constant** 变量的内存，以字节为单位
- `::mcDevAttrWarpSize`: 线程中的扭曲大小
- `::mcDevAttrMaxPitch`: 内存复制函数允许的最大间距（pitch），以字节为单位，这些函数涉及通过 **mcMallocPitch()**分配的内存区域
- `::mcDevAttrMaxTexture1DWidth`: 最大一维（1D）纹理宽度
- `::mcDevAttrMaxTexture1DLinearWidth`: 绑定到线性存储器的一维纹理的最大宽度
- `::mcDevAttrMaxTexture1DMipmappedWidth`: 最大 mipmapped 一维纹理宽度
- `::mcDevAttrMaxTexture2DWidth`: 最大二维（2D）纹理宽度
- `::mcDevAttrMaxTexture2DHeight`: 最大二维纹理高度
- `::mcDevAttrMaxTexture2DLinearWidth`: 绑定到线性存储器的二维纹理的最大宽度
- `::mcDevAttrMaxTexture2DLinearHeight`: 绑定到线性存储器的二维纹理的最大高度
- `::mcDevAttrMaxTexture2DLinearPitch`: 绑定到线性存储器的二维纹理的最大间距，以字节为单位
- `::mcDevAttrMaxTexture2DMipmappedWidth`: 最大 mipmapped 二维纹理宽度

- ::mcDevAttrMaxTexture2DMipmappedHeight：最大 mipmapped 二维纹理高度
- ::mcDevAttrMaxTexture3DWidth：最大三维（3D）纹理宽度
- ::mcDevAttrMaxTexture3DHeight：最大三维纹理高度
- ::mcDevAttrMaxTexture3DDepth：最大三维纹理深度
- ::mcDevAttrMaxTexture3DWidthAlt：备用最大三维纹理宽度，如果不支持备用最大三维理尺寸，则为 0
- ::mcDevAttrMaxTexture3DHeightAlt：备用最大三维纹理高度，如果不支持备用最大三维理尺寸，则为 0
- ::mcDevAttrMaxTexture3DDepthAlt：备用最大三维纹理深度，如果不支持备用最大三维理尺寸，则为 0
- ::mcDevAttrMaxTextureCubemapWidth：最大立方体贴图（cubemap）纹理宽度或高度
- ::mcDevAttrMaxTexture1DLayeredWidth：最大一维分层纹理宽度
- ::mcDevAttrMaxTexture1DLayeredLayers：一维分层纹理中的最大层数
- ::mcDevAttrMaxTexture2DLayeredWidth：最大二维分层纹理宽度
- ::mcDevAttrMaxTexture2DLayeredHeight：最大二维分层纹理高度
- ::mcDevAttrMaxTexture2DLayeredLayers：二维分层纹理中的最大层数
- ::mcDevAttrMaxTextureCubemapLayeredWidth：最大立方体贴图分层纹理宽度或高度
- ::mcDevAttrMaxTextureCubemapLayeredLayers：立方体贴图分层纹理中的最大层数
- ::mcDevAttrMaxSurface1DWidth：最大一维表面宽度
- ::mcDevAttrMaxSurface2DWidth：最大二维表面宽度
- ::mcDevAttrMaxSurface2DHeight：最大二维表面高度
- ::mcDevAttrMaxSurface3DWidth：最大三维表面宽度
- ::mcDevAttrMaxSurface3DHeight：最大三维表面高度
- ::mcDevAttrMaxSurface3DDepth：最大三维表面深度
- ::mcDevAttrMaxSurface1DLayeredWidth：最大一维分层表面宽度
- ::mcDevAttrMaxSurface1DLayeredLayers：一维分层表面中的最大层数
- ::mcDevAttrMaxSurface2DLayeredWidth：最大二维分层表面宽度
- ::mcDevAttrMaxSurface2DLayeredHeight：最大二维分层表面高度
- ::mcDevAttrMaxSurface2DLayeredLayers：二维分层表面中的最大层数
- ::mcDevAttrMaxSurfaceCubemapWidth：最大立方体贴图表面宽度

- ::mcDevAttrMaxSurfaceCubemapLayeredWidth：最大立方体贴图分层表面宽度
- ::mcDevAttrMaxSurfaceCubemapLayeredLayers：立方体贴图分层表面中的最大层数
- ::mcDevAttrMaxRegistersPerBlock：线程块可用的 32 位寄存器的最大数量
- ::mcDevAttrClockRate：峰值时钟频率，以千赫兹（kilohertz）为单位
- ::mcDevAttrTextureAlignment：对齐要求，与::textureAlign 字节对齐的纹理基址不需要应用于纹理提取的偏移量（offset）
- ::mcDevAttrTexturePitchAlignment：对绑定到 pitched 内存的二维纹理引用的间距对齐要求
- ::mcDevAttrGpuOverlap：如果设备可以在执行内核时在主机和设备之间并发复制内存，则为 1；如果不能，则为 0
- ::mcDevAttrMultiProcessorCount：设备上的多处理器数量
- ::mcDevAttrKernelExecTimeout：如果在设备上执行的内核有运行时间限制，则为 1；如果没有，则为 0
- ::mcDevAttrIntegrated：如果设备与内存子系统集成，则为 1；如果没有，则为 0
- ::mcDevAttrCanMapHostMemory：如果设备可以将主机内存映射到 MC 地址空间，则为 1；如果不能，则为 0
- ::mcDevAttrComputeMode：设备当前所处的计算模式。可用模式如下：
  - mcComputeModeDefault：默认（default）模式，设备不受限制，多个线程可以对此设备使用 **mcSetDevice**()
  - mcComputeModeExclusive：计算独占（compute-exclusive）模式，只有一个线程能够对此设备使用 **mcSetDevice()**
  - mcComputeModeProhibited：禁止计算（compute-prohibited）模式，任何线程都不能对此设备使用 **mcSetDevice()**
  - mcComputeModeExclusiveProcess：计算独占进程（compute-exclusive-process）模式，一个进程中的众多线程能够对此设备使用 **mcSetDevice()**
- ::mcDevAttrConcurrentKernels：如果设备支持在同一个上下文中同时执行多个内核，则为 1；如果不能，则为 0。不保证多个内核将同时驻留在设备上，因此不应依赖此功能来确保正确性
- ::mcDevAttrEccEnabled：如果在设备上启用了纠错，则为 1；如果设备禁用或不支持纠错，则为 0
- ::mcDevAttrPciBusId：设备的 PCI 总线标识符
- ::mcDevAttrPciDeviceId：设备的 PCI 设备（也称为插槽）标识符
- ::mcDevAttrMemoryClockRate：峰值内存时钟频率，以千赫兹为单位
- ::mcDevAttrGlobalMemoryBusWidth：全局内存总线宽度，以位（bit）为单位

- ::mcDevAttrL2CacheSize: L2 缓存的大小，以字节为单位。如果设备没有 L2 缓存，则为 0
- ::mcDevAttrMaxThreadsPerMultiProcessor: 每个多处理器的最大线程数
- ::mcDevAttrUnifiedAddressing: 如果设备与主机共享统一地址空间，则为 1；如果不是，则为 0
- ::mcDevAttrComputeCapabilityMajor: 计算能力主版本号
- ::mcDevAttrComputeCapabilityMinor: 计算能力次版本号
- ::mcDevAttrStreamPrioritiesSupported: 如果设备支持流优先级，则为 1；如果不支持，则为 0
- ::mcDevAttrGlobalL1CacheSupported: 如果设备支持在 L1 缓存中缓存全局变量，则为 1；如果不支持，则为 0
- ::mcDevAttrLocalL1CacheSupported: 如果设备支持在 L1 缓存中缓存本地变量，则为 1；如果不支持，则为 0
- ::mcDevAttrMaxSharedMemoryPerMultiprocessor: 多处理器可用的最大共享内存容量，以字节为单位。此数量由所有驻留在多处理器上的线程块同时共享
- ::mcDevAttrMaxRegistersPerMultiprocessor: 多处理器可用的 32 位寄存器的最大数量。此数量由所有驻留在多处理器上的线程块同时共享
- ::mcDevAttrManagedMemory: 如果设备支持分配托管内存（managed memory），则为 1；如果不支持，则为 0
- ::mcDevAttrIsMultiGpuBoard: 如果设备位于多 GPU 板上，则为 1；如果不是，则为 0
- ::mcDevAttrMultiGpuBoardGroupID: 同一个多 GPU 板上一组设备的唯一标识符
- ::mcDevAttrHostNativeAtomicSupported: 如果设备和主机之间的链接支持本机原子操作（atomic operation），则为 1
- ::mcDevAttrSingleToDoublePrecisionPerfRatio: 单精度性能（每秒浮点运算）与双精度性能的比例
- ::mcDevAttrPageableMemoryAccess: 如果设备无需在其上调用 mcHostRegister 就支持连贯访问分页内存，则为 1；否则为 0
- ::mcDevAttrConcurrentManagedAccess: 如果设备可以与 CPU 同时连贯访问托管内存，则为 1；否则为 0
- ::mcDevAttrComputePreemptionSupported: 如果设备支持计算抢占（Compute Preemption），则为 1；如果不支持，则为 0
- ::mcDevAttrCanUseHostPointerForRegisteredMem: 如果设备可以访问与 CPU 位于相同虚拟地址的主机注册内存，则为 1；否则为 0

- ::mcDevAttrCooperativeLaunch: 如果设备支持通过 **mcLaunchCooperativeKernel** 启动合作内核，则为 1；否则为 0
- ::mcDevAttrCooperativeMultiDeviceLaunch: 如果设备支持通过 **mcLaunchCooperativeKernelMultiDevice** 启动合作内核，则为 1；否则为 0
- ::mcDevAttrCanFlushRemoteWrites: 如果设备支持刷新未完成的远程写入，则为 1；否则为 0
- ::mcDevAttrHostRegisterSupported: 如果设备支持通过 **mcHostRegister** 进行主机内存注册，则为 1；否则为 0
- ::mcDevAttrPageableMemoryAccessUsesHostPageTables: 如果设备通过主机的页表访问分页内存，则为 1；否则为 0
- ::mcDevAttrDirectManagedMemAccessFromHost: 如果主机无需迁移就可直接访问设备上的托管内存，则为 1；否则为 0
- ::mcDevAttrMaxSharedMemoryPerBlockOptin: 设备上每个块共享内存的最大容量。使用 **mcFuncSetAttribute** 时可以选择此值
- ::mcDevAttrMaxBlocksPerMultiprocessor: 多处理器上可存在的线程块的最大数量
- ::mcDevAttrMaxPersistingL2CacheSize: 最大 L2 持久性线路容量设置，以字节为单位
- ::mcDevAttrMaxAccessPolicyWindowSize: mcAccessPolicyWindow::num_bytes 的最大值
- ::mcDevAttrReservedSharedMemoryPerBlock: MC 驱动为每个块预留的共享内存，以字节为单位
- ::mcDevAttrSparseMcArraySupported: 如果设备支持稀疏 MC 数组和稀疏 MC mipmapped 数组，则为 1
- ::mcDevAttrHostRegisterReadOnlySupported: 设备支持使用 **mcHostRegister** 标志 mcHostRegisterReadOnly 来注册内存，此内存必须以只读方式映射到 GPU
- ::mcDevAttrMemoryPoolsSupported: 如果设备支持使用 mcMallocAsync 和 mcMemPool 系列 API，则为 1；否则为 0
- ::mcDevAttrGPUDirectRDMASupported: 如果设备支持 GPUDirect RDMA API，则为 1；否则为 0
- ::mcDevAttrGPUDirectRDMAFlushWritesOptions: 要根据 mcFlushGPUDirectRDMAWritesOptions 枚举进行解释的位掩码
- ::mcDevAttrGPUDirectRDMAWritesOrdering: 有关数值，请参见 mcGPUDirectRDMAWritesOrdering 枚举
- ::mcDevAttrMemoryPoolSupportedHandleTypes: 基内存池的 IPC 支持的句柄类型的位掩码

- ::mcDevAttrDeferredMappingMcArraySupported: 如果设备支持延迟映射 MC 数组和 MC mipmapped 数组，则为 1

2.2.3.4 mcError_t mcDeviceGetDefaultMemPool (mcMemPool_t * *memPool*, mcDevice_t *deivce*)

返回设备的默认内存池。

参数

out	*memPool*	返回的默认内存池
in	*device*	设备句柄

返回值

mcSuccess, #mcErrorInavlidDevice, #mcErrorInavlidValue, mcErrorNotSupported

设备的默认内存池包含该设备的设备内存。

2.2.3.5 mcError_t mcDeviceGetLimit (size_t * *pValue*, enum mcLimit_t *limit*)

获取当前设备的资源限制。

参数

out	*pValue*	返回查询待限制类型的当前大小
in	*limit*	要查询的限制类型

返回值

mcSuccess, mcErrorUnsupportedLimit, mcErrorInvalidValue

在*pValue 中返回限制的当前大小。支持的::mcLimit_t 值为：

- mcLimitStackSize：每个 GPU 线程的堆栈大小，以字节为单位
- mcLimitPrintfFifoSize：::printf()设备系统调用使用的共享 FIFO 的大小，以字节为单位
- mcLimitMallocHeapSize：::malloc()和::free()设备系统调用使用的堆的大小，以字节为单位
- mcLimitDevRuntimeSyncDepth：线程可以调用设备运行时 **mcDeviceSynchronize()**以等待子网格启动完成的最大网格深度
- mcLimitDevRuntimePendingLaunchCount：未完成的设备运行时启动的最大数量

- `mcLimitMaxL2FetchGranularity`: L2 缓存获取粒度（fetch granularity）
- `mcLimitPersistingL2CacheSize`: 持久性 L2 缓存大小，以字节为单位

说明

目前，仅 `mcLimitMallocHeapSize` 可用。

2.2.3.6 mcError_t mcDeviceGetLuid (char * *luid*, unsigned int * *deviceNodeMask*, mcDevice_t *dev*)

返回设备的 LUID 和设备节点掩码。

参数

luid	返回的 LUID
deviceNodeMask	返回的设备节点掩码
dev	要获取标识符字符串的设备 ID

返回值

`mcSuccess, mcErrorInvalidValue, mcErrorInvalidDevice`

返回标识信息（`luid` 和 `deviceNodeMask`），以允许匹配设备与图形 API。

2.2.3.7 mcError_t mcDeviceGetMemPool (mcMemPool_t * *memPool*, mcDevice_t *deivce*)

获取设备的当前内存池。

参数

out	*memPool*	返回的内存池
in	*device*	设备句柄

返回值

`mcSuccess, #mcErrorInavlidValue, mcErrorNotSupported`

返回为该设备提供给 **`mcDeviceSetMemPool`** 的最后一个池；如果从未调用过 **`mcDeviceSetMemPool`**，则返回设备的默认内存池。默认情况下，当前内存池是设备的默认内存池，否则必须使用 **`mcDeviceSetMemPool`** 设置返回的池。

2.2.3.8 mcError_t mcDeviceGetName (char * *name*, int *len*, mcDevice_t *device*)

返回设备指定的 GPU 设备的产品名称。

参数

out	*name*	设备的产品名称
in	*len*	设备的产品名称长度
in	*device*	设备 ID，范围为：0...mcGetDeviceCount()

返回值

`mcSuccess, mcErrorInvalidValue, #mcErrorInavlidDevice`

返回一个 ASCII 字符串，该字符串在 `name` 指向的以'\0'结尾的字符串中标识设备 `dev`。`len` 指定返回字符串的最大长度。

2.2.3.9 mcError_t mcDeviceGetP2PAttribute (int * *value*, enum mcDeviceP2PAttr *attr*, int *srcDevice*, int *dstDevice*)

查询两个设备之间链接的属性。

参数

in	*value*	请求属性的返回值
in	*attrib*	`srcDevice` 和 `dstDevice` 之间链接的请求属性
in	*srcDevice*	目标链接的源设备
in	*dstDevice*	目标链接的目标设备

返回值

`mcSuccess, #mcErrorInavlidDevice, mcErrorInvalidValue`

在`*value` 中返回 `srcDevice` 和 `dstDevice` 之间链接的请求属性 `attrib` 的值。支持的属性如下：

- `mcDevP2PAttrPerformanceRank`：指示两个设备之间链路性能的相对值。值越低，表示性能越好。0 表示性能最高的链接。
- `mcDevP2PAttrAccessSupported`：如果启用了端对端访问，则为 1。
- `mcDevP2PAttrNativeAtomicSupported`：如果支持通过链路执行本机原子操作，则为 1。
- `mcDevP2PAttrMcArrayAccessSupported`：如果支持通过链接访问 MC 数组，则为 1。

返回值如下：

- 如果 srcDevice 或 dstDevice 无效，或者它们表示相同的设备，返回 mcErrorInvalidDevice。
- 如果 attr 无效，或者 value 为空指针，返回 mcErrorInvalidValue。
- 如果成功获取 attr 信息，返回 mcSuccess。

2.2.3.10 mcError_t mcDeviceGetPCIBusId (char * *pciBusId*, int *len*, int *device*)

返回设备的 PCI 总线 ID 字符串。

参数

out	*pciBusId*	返回的设备的标识符字符串，格式如下：[domain]:[bus]:[device].[function]，其中 domain、bus、device 和 function 都是十六进制值。pciBusId 应该足够大以存储 13 个字符，包括 NULL 终止符。
in	*len*	要存储在名称中的字符串的最大长度
in	*device*	要获取标识符字符串的设备

返回值

mcSuccess, #mcErrorInavlidDevice, #mcErrorInavlidValue

返回一个 ASCII 字符串，该字符串在 pciBusId 指向的以'\0'结尾的字符串中标识设备 dev。len 指定返回字符串的最大长度。

2.2.3.11 mcError_t mcDeviceGetSharedMemConfig (enum mcSharedMemConfig * *pConfig*)

返回当前设备的共享内存配置。

参数

out	*pConfig*	返回的缓存配置

返回值

mcSuccess, mcErrorInvalidValue

此函数将在 pConfig 中返回当前设备上共享内存存储体（bank）的当前大小。

返回的存储体配置可以是：

- mcSharedMemBankSizeFourByte：共享内存存储体宽为 4 个字节

- `mcSharedMemBankSizeEightByte`：共享内存存储体宽度为 8 字节

说明

当前设备唯一支持的共享内存配置是 `mcSharedMemBankSizeFourByte`。

2.2.3.12 mcError_t mcDeviceGetStreamPriorityRange (int * *leastPriority*, int * *greatestPriority*)

返回优先级范围。

参数

in,out	*leastPriority*	返回指向与最低优先级相对应的值的指针
in,out	*greatestPriority*	返回指向与最高优先级相对应的值的指针

返回值

`mcSuccess`

在`*leastPriority`和`*greatestPriority`中返回分别对应于最低和最高流优先级的数值。流优先级遵循约定：数字越低，优先级越高。

有意义的流优先级范围由[*greatestPriority, *leastPriority]给出。如果用户尝试创建优先级值超出由此 API 指定的有意义范围的流，优先级会自动降低或提高到*leastPriority 或*greatestPriority。有关创建优先级流的详情，参见 `mcStreamCreateWithPriority`。如果不需要该值，则可以将 NULL 传入 *leastPriority 或*greatestPriority。

如果当前上下文的设备不支持流优先级，则此函数将在`*leastPriority`和`*greatestPriority`中返回 `0`，详情参见 `mcDeviceGetAttribute`。

2.2.3.13 mcError_t mcDeviceGetUuid (mcUuid_t * *uuid*, mcDevice_t *device*)

返回设备的 UUID。

参数

out	*uuid*	返回的 UUID
in	*device*	要获取标识符字符串的设备

返回值

`mcSuccess, #mcErrorInavlidValue, #mcErrorInavlidDevice`

返回 16 个八位字节（16-octets），用于标识结构中 UUID 指向的设备。如果设备处于 MIG 模式，则返回

其唯一标识订阅的 MIG 计算实例的 MIG UUID。

2.2.3.14 mcError_t mcDeviceReset (void)

复位当前设备到默认状态。

返回值

`mcSuccess`

显式销毁和清理当前进程中与当前设备关联的所有资源。调用方有责任确保在后续 API 调用中不会访问或传递资源，这样做会导致未定义的行为。这些资源包括 MXMACA 类型，例如：

`::mcStream_t, mcEvent_t, mcArray_t, mcMipmappedArray_t, ::mcTextureObject_t, ::mcSurfaceObject_t, ::textureReference, ::surfaceReference, mcExternalMemory_t, mcExternalSemaphore_t` 和 `mcGraphicsResource_t`。

对该设备的任何后续 API 调用都将重新初始化该设备。

说明

此函数将立即重置设备。调用方负责确保在调用此函数时，进程中的任何其他主机线程都不会访问设备。

另请参阅

mcDeviceSynchronize

2.2.3.15 mcError_t mcDeviceSetCacheConfig (enum mcFuncCache_t *cacheConfig*)

设置当前设备的首选缓存配置。

参数

in	*cacheConfig*	请求的缓存配置

返回值

`mcSuccess, mcErrorInvalidValue`

在 L1 缓存和共享内存使用相同硬件资源的设备上，将通过 `cacheConfig` 设置当前设备的首选缓存配置。这只是一种偏好。如果可能，运行时将使用请求的配置，但如果需要执行函数，可以自由选择不同的配置。

在 L1 缓存和共享内存大小固定的设备上，此设置不起任何作用。

2.2.3.16 mcError_t mcDeviceSetLimit (enum mcLimit_t *limit*, size_t *value*)

设置当前设备的资源限制。

参数

in	*limit*	要设置的限制类型
in	*value*	限制类型的目标设置值

返回值

`mcSuccess, mcErrorUnsupportedLimit, mcErrorInvalidValue, mcErrorMemoryAllocation`

将 `limit` 设置为 `value` 是应用程序更新设备维护的当前限制的请求。驱动可以自由修改请求的值以满足硬件（h/w）要求：固定到最小值或最大值、四舍五入到最接近的元素大小等。应用程序可以使用 **mcDeviceGetLimit()**来确定限制的确切设置。

设置每个::mcLimit 都有自己的特定限制，详情如下：

- `mcLimitStackSize`：控制每个 GPU 线程的堆栈大小，以字节为单位。
- `mcLimitPrintfFifoSize`：控制::printf()设备系统调用使用的共享 FIFO 的大小，以字节为单位。在启动任何使用`::printf()`设备系统调用的内核后，都不得设置 mcLimitPrintfFifoSize。在这种情况下，会返回 mcErrorInvalidValue。
- `::mcimitMallocHeapSize`：控制`::malloc()`和`::free()`设备系统调用使用的堆的大小，以字节为单位。在启动任何使用::malloc()或::free()设备系统调用的内核后，都不得设置 mcLimitMallocHeapSize。在这种情况下，会返回 mcErrorInvalidValue。
- `mcLimitDevRuntimeSyncDepth`：控制线程可以安全调用 **mcDeviceSynchronize()**的网格的最大嵌套深度在启动使用设备运行时并在两级网格的默认同步深度以上调用 **mcDeviceSynchronize()** 的内核之前，必须设置此限制。如果违反限制，对 **mcDeviceSynchronize()**的调用将失败，错误码为 mcErrorSyncDepthExceeded。此限制可以设置为小于默认值或最大发射深度 24。设置此限制时，请记住，额外级别的同步深度需要运行时预留大量设备内存，这些内存无法再用于用户分配。如果这些设备内存预留失败，**mcDeviceSetLimit** 会返回 mcErrorMemoryAllocation，并且可以将限制重置为较低的值。此限制仅适用于计算能力为 3.5 或更高的设备。试图在计算能力低于 3.5 的设备上设置此限制将导致返回错误 mcErrorUnsupportedLimit。
- `mcLimitDevRuntimePendingLaunchCount`：控制可以从当前设备进行的未完成设备运行时启动的最大数量。从启动那一刻到已知网格已完成，网格都是未完成的。启动后调用 **mcGetLastError()** ，违反此限制的设备运行时启动失败并返回

mcErrorLaunchPendingCountExceeded。如果使用设备运行时的模块需要比默认值（2048 次启动）更多的挂起启动，则可以增加此限制。请记住，要想维持额外的挂起启动，需要运行时预留大量的设备内存，这些内存无法再用于分配。如果这些预留失败时，**mcDeviceSetLimit** 会返回 mcErrorMemoryAllocation，并且可以将限制重置为较低的值。此限制仅适用于计算能力为 3.5 或更高的设备。试图在计算能力低于 3.5 的设备上设置此限制将导致返回错误 mcErrorUnsupportedLimit。

- `mcLimitMaxL2FetchGranularity`：控制 L2 缓存获取粒度。值的范围从 0B 到 128B。这纯粹是一个性能提示，可以根据平台忽略或固定。
- `mcLimitPersistingL2CacheSize`：控制可用于持久性 L2 缓存的大小，以字节为单位。这纯粹是一个性能提示，可以根据平台忽略或固定。

说明

目前，仅 `mcLimitMallocHeapSize` 可用。

2.2.3.17 mcError_t mcDeviceSetMemPool (mcDevice_t *dev*, mcMemPool_t *pool*)

设置设备的当前内存池。

返回值

`mcSuccess, mcErrorInvalidValue`

内存池必须是指定设备的本地内存池。**mcMemAllocAsync** 从提供的流设备的当前内存池中分配。默认情况下，设备的当前内存池是其默认内存池。

说明

使用 **mcMemAllocFromPoolAsync** 从不同于流运行的设备指定异步分配。

2.2.3.18 mcError_t mcDeviceSetSharedMemConfig (enum mcSharedMemConfig *config*)

设置当前设备的共享内存配置。

参数

in	*config*	要设置的共享内存配置

返回值

`mcSuccess, mcErrorInvalidValue`

在具有可配置共享内存存储体的设备上，此函数将设置用于所有后续内核启动的共享内存存储体大小。

支持的存储体配置为：

- mcSharedMemBankSizeDefault：将存储体宽度设置为默认的初始设置（当前为 4 个字节）
- mcSharedMemBankSizeFourByte：将共享内存存储体宽度设置为本机 4 个字节
- mcSharedMemBankSizeEightByte：将共享内存存储体宽度设置为本机 8 个字节。此函数在具有固定共享内存存储体大小的设备上不执行任何操作。

说明

当前设备唯一支持的共享内存配置是 `mcSharedMemBankSizeFourByte`。

2.2.3.19 mcError_t mcDeviceSynchronize (void)

等待当前设备上的所有活跃流完成。

返回值

`mcSuccess, mcErrorMemoryAllocation`

阻塞，直到设备完成所有先前请求的任务。如果先前的某个任务失败，mcDeviceSynchronize()会返回错误。如果为该设备设置了 mcDeviceScheduleBlockingSync 标志，则主机线程直到设备完成其工作都将阻塞。

另请参阅

mcSetDevice, mcDeviceReset

2.2.3.20 mcError_t mcDeviceTotalMem (size_t * *totmem*, mcDevice_t *deivce*)

返回设备上的内存总量。

参数

out	*totmem*	返回的设备上可用的内存大小，以字节为单位
in	*device*	设备句柄

返回值

`mcSuccess, #mcErrorInavlidDevice, mcErrorInvalidValue`

2.2.3.21 mcError_t mcGetDevice (int * *deviceId*)

返回当前正在使用的设备 ID。

参数

out	*deviceId*	当前正在使用的设备 ID

返回值

`mcSuccess, mcErrorInvalidDevice, mcErrorInvalidValue`

使用线程局部存储（thread-local-storage）为每个线程维护一个默认设备。此设备隐式用于此线程调用的 MXMACA 运行时 API。mcGetDevice 在`*device` 中返回调用主机线程的默认设备。

另请参阅

mcSetDevice, mcGetDevicesizeBytes

2.2.3.22 mcError_t mcGetDeviceCount (int * *count*)

返回具有计算能力的设备的数量。

参数

out	*count*	返回的具有计算能力的设备的数量

返回值

`mcSuccess`

在`*count` 中返回能够运行计算命令的设备的数量。

2.2.3.23 mcError_t mcGetDeviceFlags (unsigned int * *flags*)

获取为当前设备设置的标志。

参数

out	*flags*	用于存储设备标志的指针

返回值

`mcSuccess, mcErrorInvalidValue`

2.2.3.24 mcError_t mcGetDeviceProperties (mcDeviceProp_t * *prop*, int *deviceId*)

返回计算设备的设备属性。

参数

out	*prop*	设备的属性
in	*deviceId*	要查询信息的设备 ID

返回值

`mcSuccess, mcErrorInvalidDevice`

在`*prop` 中返回设备 `dev` 的属性。使用 *`deviceId`* 指定的设备属性信息填充`*prop`。

2.2.3.25 mcError_t mcSetDevice (int *deviceId*)

将默认设备设置为用于此线程的后续 MC API 调用。

参数

in	*deviceId*	范围 0...**mcGetDeviceCount()**内的有效设备

返回值

`mcSuccess, mcErrorInvalidDevice, mcErrorDeviceAlreadyInUse`

将 `device` 设置为调用主机线程的默认设备。有效设备 ID 为：0...(mcGetDeviceCount()-1)。

众多 MC API 隐式使用**默认设备**：

- 后续从此主机线程（使用 MCMalloc）分配的任何设备内存都将在 device 上进行分配。
- 从此主机线程创建的任何流或事件都将与 device 关联。
- 从该主机线程启动的任何内核都将在 device 上执行，除非指定了特定的流，在这种情况下，将使用与该流关联的设备。

此函数可以从任何主机线程调用。多个主机线程可以使用同一设备。此函数不与旧设备或新设备同步，并且运行时消耗非常小。在进行使用默认设备的 MXMACA 运行时调用之前，应用程序可以使用 mcSetDevice 快速切换默认设备。

默认设备存储在每个线程的线程局部存储中。线程池实现可以继承上一个线程的默认设备。好的做法是始终在 MC 编码序列开始时调用 mcSetDevice，以建立已知的标准设备

另请参阅

mcGetDevice, mcGetDeviceCount

2.2.3.26 mcError_t mcSetDeviceFlags (unsigned int *flag*)

当前设备行为根据传递的标志进行变更。

参数

in	*flags*	设置当前设备行为的标志

返回值

`mcSuccess, mcErrorInvalidDevice, mcErrorSetOnActiveProcess`

调度标志（schedule flag）会影响 MXMACA 等待设备上运行的命令完成的方式。

- mcDeviceScheduleSpin：MXMACA 运行时将在提交工作的线程中主动自旋（spin），直到命令完成。提供了最低的延迟，但会消耗 CPU 核心，并可能增加功率。
- mcDeviceScheduleYield：MXMACA 运行时将 CPU 让给系统，以便其他任务可以使用 CPU。这可能会增加检测完成的延迟，但会消耗更少的功率，并且对系统中的其他任务更友好。
- mcDeviceScheduleBlockingSync：在 MXMACA 平台上，这是 mcDeviceScheduleYield 的同义词。
- mcDeviceScheduleAuto：使用启发式方法在 Spin 和 Yield 模式之间进行选择。如果 MXMACA 上下文的数量大于系统中的逻辑处理器数量，请使用 Spin 进行调度。否则使用 Yield 进行调度。
- mcDeviceMapHost：允许映射主机内存。在 MXMACA 上，这始终是允许的，并且会忽略该标志。
- mcDeviceLmemResizeToMax：MXMACA 会默默忽略该标志。

2.2.3.27 mcError_t mcSetValidDevices (int * *deviceArray*, int *len*)

从指定的设备列表中选择一个有效设备作为当前设备。

参数

in	*deviceArray*	尝试的设备列表
in	*len*	指定列表中的设备数量

返回值

`mcSuccess, mcErrorInvalidValue, mcErrorInvalidDevice`

使用 deviceArray 按优先级顺序为 MXMACA 执行设置设备列表。参数 len 指定列表中的元素数。MXMACA 将按顺序尝试列表中的设备，直到找到有效的设备。

如果未调用此函数，或者如果调用该函数的 len 为 0，则 MXMACA 将返回到其默认行为，即从包含系统中所有可用 MXMACA 设备的默认列表中按顺序尝试设备。

如果列表中指定的设备 ID 不存在，此函数会返回 mcErrorInvalidDevice。

如果 len 不为 0 且 deviceArray 为空（NULL），或者 len 超过系统中的设备数，则返回 mcErrorInvalidValue。

2.3 流管理

2.3.1 模块描述

本章节介绍 MXMACA 运行时 API 的流管理函数。MC 运行时中暂不支持以下流 API：

- mcStreamAttachMemAsync

2.3.2 函数

- `mcError_t` **`mcStreamCreate`** `(mcStream_t *stream)`

  创建异步流。

- `mcError_t` **`mcStreamCreateWithFlags`** `(mcStream_t *pStream, unsigned int flags)`

  创建异步流。

- `mcError_t` **`mcStreamCreateWithPriority`** `(mcStream_t *stream, unsigned int flags, int priority)`

  创建具有指定优先级的异步流。

- `mcError_t` **`mcStreamDestroy`** `(mcStream_t stream)`

  销毁和清理异步流。

- `mcError_t` **`mcStreamGetFlags`** `(mcStream_t hStream, unsigned int *flags)`

  查询流的标志。

- `mcError_t` **`mcStreamGetPriority`** `(mcStream_t hStream, int *priority)`

  查询流的优先级。

- `mcError_t` **`mcStreamGetCtx`** `(mcStream_t hStream, mcCtx_t *ctx)`

  查询与流相关联的上下文。

- `mcError_t` **`mcStreamSetAttribute`** `(mcStream_t hStream, mcStreamAttrID attr, const` **`mcStreamAttrValue`** `*value)`

  设置流属性。

- `mcError_t` **`mcStreamGetAttribute`** `(mcStream_t hStream, mcStreamAttrID attr,` **`mcStreamAttrValue`** `*value_out)`

  查询流属性。

- `mcError_t` **`mcStreamCopyAttributes`** `(mcStream_t dst, mcStream_t src)`

  将属性从源流复制到目标流。

- `mcError_t` **`mcStreamQuery`** `(mcStream_t stream)`

查询异步流的完成状态。

- `mcError_t` **`mcStreamSynchronize`** `(mcStream_t stream)`

  等待流任务完成。

- `mcError_t` **`mcStreamWaitEvent`** `(mcStream_t stream, mcEvent_t event, unsigned int flags __dparm(0))`

  使计算流等待事件。

- `mcError_t` **`mcStreamAddCallback`** `(mcStream_t stream, mcStreamCallback_t callback, void *userData, unsigned int flags)`

  向计算流添加回调。

- `mcError_t` **`mcStreamAttachMemAsync`** `(mcStream_t stream, void *devPtr, size_t length __dparm(0), unsigned int flags __dparm(mcMemAttachSingle))`

  将内存异步附加到流。

- `mcError_t` **`mcStreamBeginCapture`** `(mcStream_t stream, mcStreamCaptureMode mode)`

  开始对流进行图形捕获。

- `mcError_t` **`mcStreamEndCapture`** `(mcStream_t stream, mcGraph_t *pGraph)`

  结束对流的捕获，返回捕获的图形。

- `mcError_t` **`mcStreamIsCapturing`** `(mcStream_t stream, mcStreamCaptureStatus *pCaptureStatus)`

  返回流的捕获状态。

- `mcError_t` **`mcStreamGetCaptureInfo`** `(mcStream_t stream, mcStreamCaptureStatus *captureStatus_out, unsigned long long *id_out __dparm(0), mcGraph_t *graph_out __dparm(0), const mcGraphNode_t **dependencies_out __dparm(0), size_t *numDependencies_out __dparm(0))`

  查询流的捕获状态。

- `mcError_t` **`mcStreamUpdateCaptureDependencies`** `(mcStream_t stream, mcGraphNode_t *dependencies, size_t numDependencies, unsigned int flags __dparm(0))`

  更新正在进行捕获的流中的依赖集合。

- `mcError_t` **`mcThreadExchangeStreamCaptureMode`** `(mcStreamCaptureMode *mode)`

  交换线程的流捕获交互模式。

2.3.3 函数介绍

2.3.3.1 mcError_t mcStreamAddCallback (mcStream_t *stream*, mcStreamCallback_t *callback*, void * *userData*, unsigned int *flags*)

向计算流添加回调。

参数

in	*stream*	要添加回调的流
in	*callback*	先前的流操作完成后要调用的函数
in	*userData*	要传递给回调函数的用户指定数据
in	*flags*	预留供将来使用，必须为 0

返回值

`mcSuccess`, `mcErrorInvalidValue`, `mcErrorInvalidResourceHandle`, `mcErrorNotSupported`

添加一个回调，以便在流中所有当前排队的项目完成后在主机上调用。对于每个 mcStreamAddCallback 调用，回调将只执行一次。回调将阻塞流中后续的工作，直到完成为止。

回调可能会传递 mcSuccess 或错误码。如果发生发生设备错误，所有后续执行的回调都将收到相应的 mcError_t。

回调不得进行任何 MC API 调用。尝试使用 MC API 可能会导致 mcErrorNotPermitted。回调不得执行任何同步，同步可能依赖于未完成的设备工作或其他未被强制提前运行的回调。没有强制顺序的回调（在独立流中）以未定义的顺序执行，可能是序列化的。

对于统一内存而言，回调执行做出多项保证：

- 回调流在回调期间是空闲的。因此，例如，回调可能一直使用附加到回调流的内存。
- 开始执行回调与同步回调前记录在同一流中的事件具有相同的效果。因此，此函数同步在回调之前已经“加入”的流。
- 在执行了所有先前的回调之前，将设备工作添加到任意流不会使流处于活跃状态。因此，例如，如果它已经与事件一起正确排序，即使工作已经添加到另一个流中，回调可能会使用全局附加内存。
- 除上述情况外，完成回调不会导致流变为活跃状态。如果回调之后没有设备工作，则回调流将保持空闲，并且在没有设备工作的连续回调中保持空闲。因此，例如，流同步可以通过在流结束时从回调发出信号来完成。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamQuery, mcStreamSynchronize, mcStreamWaitEvent, mcStreamDestroy, mcMallocManaged, mcStreamAttachMemAsync, mcLaunchHostFunc, mcDrvStreamAddCallback

2.3.3.2 mcError_t mcStreamAttachMemAsync (mcStream_t *stream*, void * *devPtr*, size_t length __*dparm*0, unsigned int flags __*dparm*mcMemAttachSingle)

将内存异步附加到流。

参数

in	*stream*	要在其中排队附加操作的流
in	*devPtr*	指向内存的指针，必须指向托管内存或系统分配的内存中的有效主机可访问区域
in	*length*	内存长度，默认为零
in	*flags*	必须是 mcMemAttachGlobal、mcMemAttachHost 或 mcMemAttachSingle，默认为 mcMemAttachSingle

返回值

`mcSuccess`, `mcErrorInvalidValue`, `mcErrorInvalidValue`, `mcErrorInvalidResourceHandle`

对流中的操作进行排队，以指定从 devPtr 开始的内存长度字节的流关联。此函数是流排序操作，这意味着它依赖于流中的前一项工作，并且只有在该工作完成时才会生效。任何以前的关联都会被自动替换。

devPtr 必须指向以下类型的内存之一：

- 使用 **managed** 关键字声明或使用 mcMallocManaged 分配的托管内存。
- 系统分配的分页内存中的有效主机可访问区域。只有当与流关联的设备报告设备属性 mcDevAttrPageableMemoryAccess 的非零值时，才能指定此类型的内存。

对于托管分配，长度必须为零或整个分配的大小。两者都表示整个分配的流关联正在改变。目前，无法更改部分托管分配的流关联。

对于分页分配，长度必须不为零。

流关联是使用标志指定的，这些标志必须是 mcMemAttachGlobal、mcMemAttachHost 或 cMemAttachSingle。标志的默认值为 mcMemAttachSingle。

- 如果指定了 mcMemAttachGlobal 标志，则任何设备上的任何流都可以访问内存。
- 如果指定了 mcMemAttachHost 标志，则程序保证不会从设备属性 mcDevAttrConcurrentManagedAccess 为零值的设备上的任何流访问设备上的内存。
- 如果指定了 mcMemAttachSingle 标志，并且流与设备属性 mcDevAttrConcurrentManagedAccess 的值为零的设备相关联，则程序保证只从流访问设备上的内存。

单独附加到空流（NULL stream）是非法的，因为空流是虚拟全局流，而不是特定流。在这种情况下，会返回错误。

当内存与单个流关联时，统一内存系统将允许 CPU 访问该内存区域，只要流中的所有操作都已完成，而不管其他流是否处于活跃状态。实际上，这将活跃 GPU 对托管内存区域的独占所有权限制为每个流的活动，而不是整个 GPU 的活动。

从与设备无关的流访问设备上的内存将产生未定义的结果。统一内存系统不执行任何错误检查，以确保启动到其他流中的内核不会访问此区域。

程序有责任通过事件、同步或其他方式对 mcStreamAttachMemAsync 进行排序调用，以确保始终合法访问内存。对于遵循流关联更改的所有内核，数据可见性和一致性将适当更改。

如果流在数据与其关联时被销毁，则会删除关联，并且关联将恢复为 mcMallocManaged 中指定分配的默认可见性。对于 **managed** 变量，默认关联始终为 mcMemAttachGlobal。请注意，销毁流是异步操作，因此，在流中的所有工作完成之前，不会更改为默认关联。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamWaitEvent, mcStreamSynchronize, mcStreamAddCallback, mcStreamDestroy, mcMallocManaged, mcDrvStreamAttachMemAsync

2.3.3.3 mcError_t mcStreamBeginCapture (mcStream_t *stream*, mcStreamCaptureMode *mode*)

开始对流进行图形捕获。

开始对流进行图形捕获。当流处于捕获模式时，推送到流中的所有操作都不会执行，而是会被捕获到一个图形中，该图形将通过 **mcStreamEndCapture** 返回。如果流是::mcStreamLegacy，则可能不会启动捕获。必须在启动捕获的同一流上结束它，并且只有当该流尚未处于捕获模式时才可以发起捕获。可以通过 **mcStreamIsCapturing** 查询捕获模式。可以通过 **mcStreamGetCaptureInfo** 查询表示捕获序列的唯一 id。

如果 `mode` 不是 mcStreamCaptureModeRelaxed，则必须从同一线程对此流调用 **mcStreamEndCapture**。

参数

in	*stream*	要在其中启动捕获的流
in	*mode*	控制此捕获序列与其他可能不安全的 API 调用的交互。更多详细信息，参见 **mcThreadExchangeStreamCaptureMode**。

返回值

`mcSuccess, mcErrorInvalidValue`

另请参阅

mcStreamCreate, mcStreamIsCapturing, mcStreamEndCapture, mcThreadExchangeStreamCaptureMode

2.3.3.4 mcError_t mcStreamCopyAttributes (mcStream_t *dst*, mcStream_t *src*)

将属性从源流复制到目标流。

参数

in	*dst*	目标流
in	*src*	源流。相关属性，参见 mcStreamAttrID

返回值

`mcSuccess, mcErrorNotSupported`

将属性从源流 src 复制到目标流 dst。两个流必须具有相同的上下文。

另请参阅

mcAccessPolicyWindow

2.3.3.5 mcError_t mcStreamCreate (mcStream_t * *stream*)

创建异步流。

参数

in,out	stream	指向 mcStream_t 的有效指针

返回值

`mcSuccess, mcErrorInvalidValue`

创建新的异步流。`stream`返回一个不透明（opaque）句柄，可使用该句柄在后续的`mcStream*`命令中引用新创建的流。流是在堆上分配流的，即使句柄超出范围，也会保持分配状态。要释放流使用的内存，应用程序必须调用 mcStreamDestroy。

另请参阅

mcStreamCreateWithFlags, mcStreamCreateWithPriority, mcStreamSynchronize, mcStreamWaitEvent, mcStreamDestroy

2.3.3.6 mcError_t mcStreamCreateWithFlags (mcStream_t * *pStream*, unsigned int *flags*)

创建异步流。

参数

in,out	*pStream*	指向新流标识符的指针
in	*flags*	流创建的参数

返回值

mcSuccess, mcErrorInvalidValue

创建新的异步流。`flags`参数确定流的行为。支持的`flags`值有：

- mcStreamDefault：默认流创建标志。
- mcStreamNonBlocking：指定在创建的流中运行的工作可以与流 0（空流）中的工作同时运行，并且创建的流不应与流 0 执行隐式同步。

另请参阅

mcStreamCreate, mcStreamCreateWithPriority, mcStreamGetFlags, mcStreamQuery, mcStreamSynchronize, mcStreamWaitEvent, mcStreamAddCallback, mcStreamDestroy, mcDrvStreamCreate

2.3.3.7 mcError_t mcStreamCreateWithPriority (mcStream_t * *stream*, unsigned int *flags*, int *priority*)

创建具有指定优先级的异步流。

参数

in,out	*stream*	指向新流标识符的流指针

in	*flags*	流创建的标志有关可以传递的有效标志的列表，参见 mcStreamCreateWithFlags
in	*priority*	流的优先级。数字越低表示优先级越高。有关可以传递的有意义的流优先级的详细信息，参见 mcDeviceGetStreamPriorityRange。

返回值

mcSuccess, mcErrorInvalidValue

创建具有指定优先级的流，并在 `pStream` 中返回句柄。此 API 更改流中工作的调度程序优先级。较高优先级流中的工作可能会抢占已在低优先级流中执行的工作。

流优先级遵循约定：数字越低，优先级越高。0 表示默认优先级。可以使用 mcDeviceGetStreamPriorityRange 查询有意义范围的数字优先级。如果指定的优先级超出 mcDeviceGetStreamPriorityRange 返回的数字范围，它将自动被固定为该范围内的最低或最高数字。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcDeviceGetStreamPriorityRange, mcStreamGetPriority, mcStreamQuery, mcStreamWaitEvent, mcStreamAddCallback, mcStreamSynchronize, mcStreamDestroy, mcDrvStreamCreateWithPriority

2.3.3.8 mcError_t mcStreamDestroy (mcStream_t *stream*)

销毁和清理异步流。

参数

in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

销毁和清理流指定的异步流。

如果调用 **mcStreamDestroy()**时设备仍在流流中工作，则该函数将立即返回，并且一旦设备完成流中的所有工作，与流关联的资源将自动释放。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamQuery, mcStreamWaitEvent, mcStreamSynchronize, mcStreamAddCallback, mcDrvStreamDestroy

2.3.3.9 mcError_t mcStreamEndCapture (mcStream_t stream, mcGraph_t * *pGraph*)

结束对流的捕获，返回捕获的图形。

结束对流的捕获，通过 `pGraph` 返回捕获的图形。必须通过调用 **mcStreamBeginCapture** 在流中启动捕获。如果由于违反流捕获规则而导致捕获无效，则会返回 NULL 图形。

如果 **mcStreamBeginCapture** 的 `mode` 参数不是::mcstreamCaptureModeRelaxed，则此调用必须与::mcstreamBeginCapture 来自相同的线程。

参数

in	*stream*	要结束捕获的流句柄
out	*pGraph*	捕获的图形

返回值

mcSuccess, mcErrorInvalidValue, mcErrorStreamCaptureWrongThread

另请参阅

mcStreamCreate, mcStreamBeginCapture, mcStreamIsCapturing

2.3.3.10 mcError_t mcStreamGetAttribute (mcStream_t *hStream*, mcStreamAttrID *attr*, mcStreamAttrValue * *value_out*)

查询流属性。

参数

in	*hStream*	要查询的流句柄
in	*attr*	要查询的流属性参数名称
in,out	*value_out*	要查询的流属性参数值

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

从 `hStream` 中查询属性 `attr`，并将其存储在 `value_out` 的相应成员中。

另请参阅

mcAccessPolicyWindow

2.3.3.11 mcError_t mcStreamGetCaptureInfo (mcStream_t *stream*, mcStreamCaptureStatus * *captureStatus_out*, unsigned long long *id_out __dparm0, mcGraph_t *graph_out __dparm0, const mcGraphNode_t **dependencies_out __dparm0, size_t *numDependencies_out __dparm0)

查询流的捕获状态。

查询与流捕获相关的流状态。

如果在对未使用 mcStreamNonBlocking 创建的流进行捕获时在::mcStreamAllThread（空流）上调用此函数，则返回 mcErrorStreamCaptureImplicit。

只有当以下两项均为真时，才会返回有效数据（捕获状态除外）：

- 调用返回 mcSuccess
- 返回的捕获状态为 mcStreamCaptureStatusActive

参数

out	*stream*	要查询的流
out	*captureStatus_out*	返回流的捕获状态所需的位置
out	*id_out*	返回捕获序列 ID 的可选位置，该 ID 在进程的整个生命周期内是唯一的
out	*graph_out*	将要捕获的图形返回到的可选位置在捕获序列进行期间，允许在图形上执行除销毁和节点删除之外的所有操作。此 API 不转移图形的所有权，在 **mcStreamEndCapture** 处转移或销毁该图形。 请注意，对于某些错误，图形句柄可能会在捕获结束前失效。由于图形上的直接操作而无法从 **mcStreamEndCapture** 的原始流访问或变得无法访问的节点不会触发 mcErrorStreamCaptureUnjoined。
out	*dependencies_out*	存储指向节点数组的指针的可选位置流中要捕获的下一个节点将取决于这组节点，缺少修改这组节点的操作，如事件等待。数组指针在对流进行操作的下一个 API 调用或捕获结束之前都有效。节点句柄可能会被复制出去，并且在它们或图形被销毁之前一直有效。驱动拥有的数组也可以直接传递给在图形（而不是流）上操作的 API，而无需复制。
out	*numDependencies_out*	存储 dependencies_out 中返回的数组大小的可选位置

返回值

mcSuccess, mcErrorInvalidValue, mcErrorStreamCaptureImplicit

另请参阅

mcStreamGetCaptureInfo, mcStreamBeginCapture, mcStreamIsCapturing, mcStreamUpdateCaptureDependencies

2.3.3.12 mcError_t mcStreamGetCtx (mcStream_t *hStream*, mcCtx_t * *ctx*)

查询与流相关联的上下文。

参数

in	*hStream*	要查询流的句柄
out	*ctx*	返回的与流关联的上下文

返回值

mcSuccess, mcErrorInvalidHandle

返回流关联的上下文。

另请参阅

mcStreamCreateWithPriority, mcDeviceGetStreamPriorityRange, mcStreamGetFlags, mcDrvStreamGetPriority

2.3.3.13 mcError_t mcStreamGetFlags (mcStream_t *hStream*, unsigned int * *flags*)

查询流的标志。

参数

in	*hStream*	要查询流的句柄
in,out	*flags*	指向无符号整数的指针，在该整数中返回流的标志

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

查询流的标志。标志在 flag 中返回。有效标志的列表，参见 mcStreamCreateWithFlags。

另请参阅

mcStreamCreateWithPriority, mcStreamCreateWithFlags, mcStreamGetPriority, mcDrvStreamGetFlags

2.3.3.14 mcError_t mcStreamGetPriority (mcStream_t *hStream*, int * *priority*)

查询流的优先级。

参数

in	*hStream*	要查询流的句柄
in,out	*priority*	指向带符号整数的指针，在该整数中返回流的优先级

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

查询流的优先级。优先级在 priority 中返回。请注意，如果创建的流的优先级超出了 mcDeviceGetStreamPriorityRange 返回的有意义的数字范围，则此函数返回固定的优先级。有关优先级固定的详细信息，参见 mcStreamCreateWithPriority。

另请参阅

mcStreamCreateWithPriority, mcDeviceGetStreamPriorityRange, mcStreamGetFlags, mcDrvStreamGetPriority

2.3.3.15 mcError_t mcStreamIsCapturing (mcStream_t *stream*, mcStreamCaptureStatus * *pCaptureStatus*)

返回流的捕获状态。

通过 `pCaptureStatus` 返回流的捕获状态。成功调用后，`*pCaptureStatus` 将包含以下内容之一：

- mcStreamCaptureStatusNone：流未进行捕获。
- mcStreamCaptureStatusActive：流正在进行捕获。
- mcStreamCaptureStatusInvalidated：流正在进行捕获，但错误使捕获序列无效。必须在启动捕获序列的流上使用 **mcStreamEndCapture** 终止它，才能继续使用流。

请注意，如果在对同一设备上的阻塞流进行捕获时在::mcStreamLegacy（空流）上调用此函数，它会返

回 mcErrorStreamCaptureImplicit，并且在调用后未指定*pCaptureStatus。阻塞流的捕获不会失效。

对阻塞流进行捕获时，legacy 流将处于不可用状态，直到阻塞流的捕获终止。不支持对 legacy 流进行捕获，且尝试操作会对正在进行捕获的流产生隐式依赖。

参数

in	*stream*	要查询的流
out	*pCaptureStatus*	返回流的捕获状态

返回值

mcSuccess, mcErrorInvalidValue, mcErrorStreamCaptureImplicit

另请参阅

mcStreamCreate, mcStreamBeginCapture, mcStreamEndCapture

2.3.3.16 mcError_t mcStreamQuery (mcStream_t *stream*)

查询异步流的完成状态。

参数

in	*stream*	流标识符

返回值

mcSuccess, mcErrorNotReady

如果流中的所有操作都已完成，则返回 mcSuccess；如果未完成，则返回 mcErrorNotReady。对于统一内存而言，mcSuccess 的返回值相当于调用了 **mcStreamSynchronize()**。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamWaitEvent, mcStreamSynchronize, mcStreamAddCallback, mcStreamDestroy, mcDrvStreamQuery

2.3.3.17 mcError_t mcStreamSetAttribute (mcStream_t *hStream*, mcStreamAttrID *attr*, const mcStreamAttrValue * *value*)

设置流属性。

参数

in	*hStream*	要为其设置属性的流的句柄
in	*attr*	要为流设置的属性
in,out	*value*	要为流设置的属性值

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

根据值的相应属性设置 hStream 上的属性 attr。更新后的属性将应用于提交到流的后续工作。它不会影响以前提交的工作。

另请参阅

mcAccessPolicyWindow

2.3.3.18 mcError_t mcStreamSynchronize (mcStream_t *stream*)

等待流任务完成。

参数

in	*Stream*	标识符

返回值

mcSuccess, mcErrorInvalidResourceHandle

阻塞，直到流完成所有操作。如果为该设备设置了 mcDeviceScheduleBlockingSync 标志，则主机线程直到流完成其所有任务都将阻塞。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamQuery, mcStreamWaitEvent, mcStreamAddCallback, mcStreamDestroy, mcDrvStreamSynchronize

2.3.3.19 mcError_t mcStreamUpdateCaptureDependencies (mcStream_t *stream*, mcGraphNode_t * *dependencies*, size_t *numDependencies*, unsigned int flags *__dparm*0)

更新正在进行捕获的流中的依赖集合。

修改正在进行捕获的流的依赖集合。依赖集合是流中下一个捕获的节点将依赖的节点集合。

有效的标志是 mcStreamAddCaptureDependencies 和 mcStreamSetCaptureDependencies。这些控制是将传递给 API 的集合添加到现有集合或替换现有集合。标志值 0 默认为 mcStreamAddCaptureDependencies。

如果无法从 **mcStreamEndCapture** 的流访问通过此 API 从依赖集合中删除的节点，则不会导致 mcErrorStreamCaptureUnjoined。

参数

in	*stream*	要进行捕获的流
in	*dependencies*	指向要添加/替换的节点数组的指针
in	*numDependencies*	依赖中数组的大小
in	*flags*	指定依赖集合的修改方法（添加/替换）

返回值

mcSuccess, mcErrorInvalidValue, mcErrorIllegalState

2.3.3.20 mcError_t mcStreamWaitEvent (mcStream_t *stream*, mcEvent_t *event*, unsigned int flags *__dparm*0)

使计算流等待事件。

参数

in	*stream*	要进行等待的流
in	*event*	要等待的事件
in	*flags*	操作的参数

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle

使提交到流的所有未来工作等待事件中捕获的所有工作。有关事件捕获的内容的详细信息，参见 **mcEventRecord()**。如果适用，将在设备上高效地执行同步。事件可能来自与流不同的设备。*flags* 的有效值为：

- mcEventWaitDefault：默认事件创建标志。
- mcEventWaitExternal：在执行流捕获时，事件在图中被捕获为外部事件节点。

另请参阅

mcStreamCreate, mcStreamCreateWithFlags, mcStreamQuery, mcStreamSynchronize, mcStreamAddCallback, mcStreamDestroy, mcDrvStreamWaitEvent

2.3.3.21 mcError_t mcThreadExchangeStreamCaptureMode (mcStreamCaptureMode * *mode*)

交换线程的流捕获交互模式。

将调用线程的流捕获交互模式设置为*mode 中包含的值，并用线程的上一个模式覆盖*mode。为了促进跨函数或模块边界的确定性行为，鼓励调用方以推送方式使用此 API：

```
mcStreamCaptureMode mode = desiredMode;
mcThreadExchangeStreamCaptureMode(&mode);
...
mcThreadExchangeStreamCaptureMode(&mode); // restore previous mode
```

在流捕获期间（参见 mcStreamBeginCapture），某些操作，例如对 mcMalloc 的调用，可能不安全。对于 mcMalloc，操作不会异步排队到流，也不会被流捕获观察到。因此，如果通过 mcStreamBeginCapture 捕获的操作序列取决于每当启动图形时重演的分配，则捕获的图形将无效。

因此，流捕获对可以在 mcStreamBeginCapture-mcStreamEndCapture 序列内或同时进行的 API 调用进行了限制。此行为可以通过此 API 和 mcStreamBeginCapture 的标志进行控制。

线程的模式有：

- mcStreamCaptureModeGlobal：这是默认模式。如果本地线程具有未在 cuStreamBeginCapture 使用 mcStreamCaptureModeRelaxed 启动的持续捕获序列，或者如果任何其他线程具有使用 mcStreamCaptureModeGlobal 启动的并发捕获序列，则禁止此线程进行可能不安全的 API 调用。
- mcStreamCaptureModeThreadLocal：如果本地线程具有未使用 mcStreamCaptureModeRelaxed 启动的持续捕获序列，则禁止其进行可能不安全的 API 调用。忽略其他线程中的并发捕获序列。
- mcStreamCaptureModeRelaxed：不禁止本地线程进行可能不安全的 API 调用。请注意，仍然禁止线程进行必然与流捕获冲突的 API 调用，例如，尝试对捕获序列中最后记录的事件执行 mcEventQuery。

参数

mode	指向要与当前模式交换的模式值的指针

返回值

mcSuccess, mcErrorInvalidValue

2.4 执行控制

2.4.1 模块描述

本章节介绍 MXMACA 运行时 API 的执行控制函数。

2.4.2 函数

- `mcError_t` **`mcLaunchKernel`** `(const void *function_address, dim3 numBlocks, dim3 dimBlocks, void **args, size_t sharedMemBytes __dparm(0), mcStream_t stream __dparm(0))`

  启动设备函数。

- `mcError_t` **`mcModuleLaunchKernel`** `(mcFunction_t f, unsigned int gridDimX, unsigned int gridDimY, unsigned int gridDimZ, unsigned int blockDimX, unsigned int blockDimY, unsigned int blockDimZ, unsigned int sharedMemBytes, mcStream_t stream, void **kernelParams, void **extra)`

  使用启动参数和流上的共享内存启动内核 f，参数传递给 kernelparams 或 extra。

- `mcError_t` **`mcLaunchHostFunc`** `(mcStream_t stream, mcHostFn_t fn, void *userData)`

  将主机函数调用加入流中的队列。

- `mcError_t` **`mcLaunchCooperativeKernel`** `(const void *f, dim3 gridDim, dim3 blockDim, void **kernelParams, unsigned int sharedMemBytes, mcStream_t stream)`

  使用启动参数和流上的共享内存启动内核 f，参数传递给 kernelparams 或 extra，线程块可以在执行时进行协作和同步。

- `mcError_t` **`mcLaunchCooperativeKernelMultiDevice`** `(`**`mcLaunchParams`** `*launchParamsList, int numDevices, unsigned int flags)`

  在多个设备上启动内核，线程块可以在执行时进行协作和同步。

- `mcError_t` **`mcExtLaunchMultiKernelMultiDevice`** `(`**`mcLaunchParams`** `*launchParamsList, int numDevices, unsigned int flags)`

  在多个设备上启动内核，并确保在从其他线程对指定流上的其他工作进行排队之前，所有指定的内核都在各自的流上调度。

- `mcError_t` **`mcFuncGetAttributes`** `(struct` **`mcFuncAttributes`** `*attr, const void *func)`

  找出给定函数的属性。

- `mcError_t mcFuncSetAttribute (const void *func, mcFuncAttribute attr, int value)`

  设置特定函数的属性。

2.4.3 函数介绍

2.4.3.1 mcError_t mcLaunchKernel (const void * *function_address*, dim3 *numBlocks*, dim3 *dimBlocks*, void ** *args*, size_t sharedMemBytes *__dparm*0, mcStream_t stream *__dparm*0)

启动设备功能。

参数

in	*function_address*	内核存根函数指针
in	*numBlocks*	块的数量
in	*dimBlocks*	块尺寸
in	*args*	内核参数
in	*sharedMemBytes*	要分配给此内核的动态共享内存容量。MACA-Clang 编译器支持外部共享声明
in	*stream*	应该调度内核的流。可以是 0，在这种情况下，默认流与关联的同步规则一起使用

返回值

mcSuccess, mcErrorInvalidValue, mcInvalidDevice

2.4.3.2 mcError_t mcModuleLaunchKernel (mcFunction_t *f*, unsigned int *gridDimX*, unsigned int *gridDimY*, unsigned int *gridDimZ*, unsigned int *blockDimX*, unsigned int *blockDimY*, unsigned int *blockDimZ*, unsigned int *sharedMemBytes*, mcStream_t *stream*, void ** *kernelParams*, void ** *extra*)

使用启动参数和流上的共享内存启动内核 f，参数传递给 kernelparams 或 extra。

参数

in	*f*	要启动的内核
in	*gridDimX*	指定为 blockDimX 倍数的 X 网格尺寸

in	*gridDimY*	指定为 blockDimY 倍数的 Y 网格尺寸
in	*gridDimZ*	指定为 blockDimZ 倍数的 Z 网格尺寸
in	*blockDimX*	工作项中指定的 X 块尺寸
in	*blockDimY*	工作项中指定的 Y 块尺寸
in	*blockDimZ*	工作项中指定的 Z 块尺寸
in	*sharedMemBytes*	要分配给此内核的动态共享内存容量。mc-Clang 编译器支持外部共享声明
in	*stream*	应该调度内核的流。可以是 0，在这种情况下，默认流与关联的同步规则一起使用
in	*kernelParams*	用户核函数要传入的参数
in	*extra*	指向内核参数的指针。直接传递给内核，并且必须在内核所期望的内存布局和对齐中

返回值

mcSuccess, mcInvalidDevice, mcErrorInitializationError, mcErrorInvalidValue

说明

MC 暂不支持 kernellParams 参数。请使用其他参数。有关示例用法，请参见 mc_porting_driver_api.md。

2.4.3.3 mcError_t mcLaunchHostFunc (mcStream_t *stream*, mcHostFn_t *fn*, void * *userData*)

将主机函数调用加入流中的队列。

参数

in	*stream*	应该调度主机函数的流
in	*fn*	先前的流操作完成后要调用的函数
in	*userData*	要传递给函数的用户指定数据

返回值

mcSuccess, mcErrorInvalidResourceHandle, mcErrorInvalidValue, mcErrorNotSupported

将主机函数加入队列，以便在流中运行。该函数将在当前排队的工作之后调用，并将阻塞在其之后添加的工作。主机函数不得进行任何 MXMACA 运行时 API 调用。尝试使用 MXMACA API 可能会导致 mcErrorNotPermitted，但这不是必需的。主机函数不得执行任何同步，同步可能依赖于未被强制提前运行的未完成的 MXMACA 工作的同步。没有强制顺序的主机函数（例如在独立流中）以未定义的顺序执

行，可能是序列化的。

对于统一内存而言，执行做出多项保证：

- 流在函数执行期间是空闲的。因此，例如，函数可能总是使用附加到它所在队列的流的内存。
- 函数执行的开始与同步在函数之前、记录在同一流中的事件具有相同的效果。因此，此函数同步在函数之前已经“加入”的流。
- 在所有前面的主机函数和流回调执行之前，向任何流添加设备工作都不会使流处于活跃状态。因此，例如，如果工作已在带有事件的函数调用之后排序，即使工作已添加到另一个流，函数也可能使用全局附加内存。
- 除上述情况外，完成函数不会导致流变为活跃状态。如果函数后面没有设备工作，则流将保持空闲，并且在没有设备工作的情况下，流将在连续的主机函数或流回调之间保持空闲。因此，例如，流同步可以通过在流结束时从主机函数发出信号来完成。

说明

与 mcStreamAddCallback 相比，如果 mcruntine 上下文中出现错误，则不会调用该函数。

2.4.3.4 mcError_t mcLaunchCooperativeKernel (const void * *f*, dim3 *gridDim*, dim3 *blockDim*, void ** *kernelParams*, unsigned int *sharedMemBytes*, mcStream_t *stream*)

使用启动参数和流上的共享内存启动内核 f，参数传递给 kernelparams 或 extra，线程块可以在执行时进行协作和同步。

参数

in	*f*	要启动的内核
in	*gridDim*	指定为 blockDim 倍数的网格尺寸
in	*blockDim*	工作项（work-item）中指定的块尺寸
in	*kernelParams*	内核参数列表
in	*sharedMemBytes*	要分配给此内核的动态共享内存容量。Clang 编译器支持外部共享声明
in	*stream*	应该调度内核的流。可以是 0，在这种情况下，默认流与关联的同步规则一起使用

返回值

mcSuccess, mcInvalidDevice, mcErrorInitializationError, mcErrorInvalidValue, mcErrorCooperativeLaunchTooLarge

该函数调用块的 gridDim（gridDim.x gridDim.y gridDim.z）网格上的内核函数。每个块包含 blockDim（blockDim.x blockDim.y blockDim.z）线程。调用此内核的设备的设备属性 mcDevAttrCooperativeLaunch 必须为非零值。

启动的块总数不能超过 mcOccupancyMaxActiveBlocksPerMultiprocessor（或 mcOccupancyMaxActiveBlocksPerMultiprocessorWithFlags）返回的每个多处理器的最大块数乘以设备属性 mcDevAttrMultiProcessorCount 指定的多处理器数。

内核无法利用动态并行性。

如果内核有 N 个参数，那么 args 应该指向 N 个指针的数组。从 args[0]到 args[N-1]的每个指针都指向将从中复制实际参数的内存区域。

对于模板化函数，按如下方式传递函数符号：

```
func_name<template_arg_0,...,template_arg_N>
```

sharedMem 设置每个线程块可用的动态共享内存容量。

stream 指定与调用关联的流。

2.4.3.5 mcError_t mcLaunchCooperativeKernelMultiDevice (mcLaunchParams * *launchParamsList*, int *numDevices*, unsigned int *flags*)

在多个设备上启动内核，线程块可以在执行时进行协作和同步。

参数

in	*mcLaunchParams*	启动参数列表，每个设备一个
in	*numDevices*	launchParamsList 数组的大小
in	*flags*	用于控制启动行为的标志

返回值

mcSuccess, mcInvalidDevice, mcErrorInitializationError, mcErrorInvalidValue, mcErrorCooperativeLaunchTooLarge

2.4.3.6 mcError_t mcExtLaunchMultiKernelMultiDevice (mcLaunchParams * *launchParamsList*, int *numDevices*, unsigned int *flags*)

在多个设备上启动内核，并确保在从其他线程对指定流上的其他工作进行排队之前，所有指定的内核都在各自的流上调度。

参数

in	*mcLaunchParams*	启动参数列表，每个设备一个
in	*numDevices*	launchParamsList 数组的大小
in	*flags*	用于控制启动行为的标志

返回值

mcSuccess, mcInvalidDevice, mcErrorInitializationError, mcErrorInvalidValue

2.4.3.7 mcError_t mcFuncGetAttributes (struct mcFuncAttributes * *attr*, const void * *func*)

找出给定函数的属性。

参数

out	*attr*	返回的指向函数属性的指针
in	*func*	设备函数符号

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidDeviceFunction

此函数获取通过 `func` 指定的函数的属性。`func` 是一个设备函数符号，必须声明为**全局（global）**函数。获取的属性放置在 `attr` 中。如果指定的函数不存在，则返回 mcErrorInvalidDeviceFunction。对于模板化函数，按如下方式传递函数符号：func_name<template_arg_0,...,template_arg_N>。

说明

某些函数属性，可能会因当前使用的设备而异。如属性 maxThreadsPerBlock，根据用户设定的 launch_bounds 返回 maxThreadsPerBlock 的值；若未设定，则默认返回 512，以优化当前设备性能。但需注意，硬件支持的最大值为 1024。

2.4.3.8 mcError_t mcFuncSetAttribute (const void * *func*, mcFuncAttribute *attr*, int *value*)

设置特定函数的属性。

参数

in	*func*	要设置属性的函数
in	*attr*	请求的属性

in	*value*	要设置的值

返回值

mcSuccess, mcErrorInvalidDeviceFunction, mcErrorInvalidValue

此函数设置通过 func 指定的函数的属性。参数 func 必须是指向在设备上执行的函数的指针。func 指定的参数必须声明为全局函数。将 attr 定义的枚举设置为 value 定义的值。如果指定的函数不存在，则返回 mcErrorInvalidDeviceFunction 。 如 果 指 定 的 属 性 无 法 写 入 ， 或 者 值 不 正 确 ， 则 返 回 mcErrorInvalidValue。

支持的属性有效值为：

- mcFuncAttributeMaxDynamicSharedMemorySize：请求的动态分配共享内存的最大大小，以字节为单位。此值与函数属性 sharedSizeBytes 之和不能超过设备属性 mcDeviceAttributeMaxSharedMemoryPerBlock。可请求的动态共享内存的最大大小可能因 GPU 架构而异。
- mcFuncAttributePreferredSharedMemoryCarveout：在 L1 缓存和共享内存使用相同硬件资源的设备上，这将设置共享内存划分首选项，占总共享内存的百分比。这只是一个提示，如果需要执行此函数，驱动可以选择不同的比例。

2.5 事件管理

2.5.1 模块描述

本章节介绍 MXMACA 运行时 API 的事件管理函数。

2.5.2 函数

- `mcError_t` **`mcEventCreate`** `(mcEvent_t *event)`
- `mcError_t` **`mcEventCreateWithFlags`** `(mcEvent_t *event, unsigned int flags)`

  创建具有指定标志的事件。

- `mcError_t` **`mcEventRecord`** `(mcEvent_t event, mcStream_t stream __dparm(nullptr))`

  在指定流中记录事件。

- `mcError_t` **`mcEventRecordWithFlags`** `(mcEvent_t event, mcStream_t stream __dparm(nullptr), unsigned int flags __dparm(0))`

  在指定流中记录事件。

- `mcError_t` **`mcEventSynchronize`** `(mcEvent_t event)`

等待事件完成。

- `mcError_t` **`mcEventElapsedTime`** `(float *ms, mcEvent_t start, mcEvent_t stop)`

  返回两个事件运行的持续时间。

- `mcError_t` **`mcEventElapsedTimeWithFlags`** `(float *ms, mcEvent_t start, mcEvent_t stop, unsigned int flag)`

  返回两个事件运行的持续时间。

- `mcError_t` **`mcEventQuery`** `(mcEvent_t event)`

  查询事件状态。

- `mcError_t` **`mcEventDestroy`** `(mcEvent_t event)`

  销毁指定事件。

2.5.3 函数介绍

2.5.3.1 mcError_t mcEventCreate (mcEvent_t * *event*)

创建事件。

参数

in,out	*event*	返回新创建的事件

返回值

mcSuccess, #mcErrorNotInitialized, mcErrorInvalidValue, mcErrorLaunchFailure, #mcErrorOutOfMemory

另请参阅

mcEventCreateWithFlags, mcEventRecord, mcEventQuery, mcEventSynchronize, mcEventDestroy, mcEventElapsedTime

2.5.3.2 mcError_t mcEventCreateWithFlags (mcEvent_t * *event*, unsigned int *flags*)

创建具有指定标志的事件。

参数

in,out	*event*	返回新创建的事件

in	*flags*	用于控制事件行为的标志。有效值为：cEventDefault、mcEventBlockingSync、mcEventDisableTiming、mcEventInterprocess

- mcEventDefault：默认标志。该事件将使用主动同步（active synchronization）并支持计时。阻塞同步提供了尽可能低的延迟，但代价是花费一个 CPU 对事件进行轮询（poll）。
- mcEventBlockingSync：该事件将使用阻塞同步。如果对此事件调用 mcEventSynchronize，在事件完成之前，线程都将阻塞。这会增加同步的延迟，但会导致其他 CPU 线程的功耗更低，资源更多。
- mcEventDisableTiming：禁用定时信息的记录。使用此标志创建的事件不会记录分析数据，且如果用于同步，则会提供最佳性能。

返回值

mcSuccess, #mcErrorNotInitialized, mcErrorInvalidValue, mcErrorLaunchFailure, #mcErrorOutOfMemory

另请参阅

mcEventCreate, mcEventSynchronize, mcEventDestroy, mcEventElapsedTime

2.5.3.3 mcError_t mcEventDestroy (mcEvent_t *event*)

销毁指定事件。

参数

in	*event*	要销毁的事件

返回值

mcSuccess, #mcErrorNotInitialized, mcErrorInvalidValue, mcErrorLaunchFailure

释放与事件关联的内存。如果在调用 **mcEventDestroy()**时事件正在记录，但尚未完成记录，则函数将立即返回，并在稍后同步 mcDevice 时释放 completion_future 资源。

另请参阅

mcEventCreate, mcEventCreateWithFlags, mcEventQuery, mcEventSynchronize, mcEventRecord, mcEventElapsedTime

返回值

mcSuccess

2.5.3.4 mcError_t mcEventElapsedTime (float * *ms*, mcEvent_t *start*, mcEvent_t *stop*)

返回两个事件运行的持续时间。

参数

out	*ms*	返回启动和停止之间的时间，以 ms 为单位
in	*start*	启动事件
in	*stop*	停止事件

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotReady, mcErrorInvalidHandle, #mcErrorNotInitialized, mcErrorLaunchFailure

计算两个事件运行的持续时间。时间以 ms 为单位计算，分辨率约为 1 us。

直到其他所有流上的所有命令完成执行之前，记录在空流中的事件都将阻塞，然后记录时间戳。

记录在非空流中的事件将在到达指定流的头部时记录其时间戳。因此，记录事件的时间可能明显在主机调用 mcEventRecord()之后。

如果尚未对任一事件调用 mcEventRecord()，则返回 mcErrorInvalidHandle。如果对两个事件都调用了 mcEventRecord()，但尚未在一个或两个事件上记录时间戳（即，mcEventQuery()将在至少一个事件上返回 mcErrorNotReady），则返回 mcErrorNotReady。

另请参阅

mcEventCreate, mcEventCreateWithFlags, mcEventQuery, mcEventDestroy, mcEventRecord, mcEventSynchronize

2.5.3.5 mcError_t mcEventElapsedTimeWithFlags (float * *ms*, mcEvent_t *start*, mcEvent_t *stop*, unsigned int *flag*)

返回两个事件运行的持续时间。

参数

out	*ms*	返回启动和停止之间的时间，以 ms 为单位
in	*start*	启动事件
in	*stop*	停止事件

in	*flag*	操作的参数

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotReady, mcErrorInvalidHandle, #mcErrorNotInitialized, mcErrorLaunchFailure

计算两个事件运行的持续时间。时间以 ms 为单位计算，分辨率约为 1 us。使用标志 mcEventElapsedTimeInterDevice，允许设备间时间流逝。

直到其他所有流上的所有命令完成执行之前，记录在空流中的事件都将阻塞，然后记录时间戳。

在该流中的先前所有命令都已执行完毕之后，记录在非空流中的事件将在到达指定流的头部时记录其时间戳。因此，记录事件的时间可能明显在主机调用 **mcEventRecord()**之后。

如果尚未对任一事件调用 **mcEventRecord()**，则返回 mcErrorInvalidHandle。如果对两个事件都调用了 **mcEventRecord()**，但尚未在一个或两个事件上记录时间戳（也就是说，**mcEventQuery()**将在至少一个事件上返回 mcErrorNotReady），则返回 mcErrorNotReady。

另请参阅

mcEventCreate, mcEventCreateWithFlags, mcEventQuery, mcEventDestroy, mcEventRecord, mcEventSynchronize

2.5.3.6 mcError_t mcEventQuery (mcEvent_t *event*)

查询事件状态。

参数

in	*event*	要查询的事件

返回值

mcSuccess, mcErrorNotReady, mcErrorInvalidHandle, mcErrorInvalidValue, #mcErrorNotInitialized, mcErrorLaunchFailure

查询指定事件的状态。如果相应流（指定给 **mcEventRecord()**）中的所有命令都已完成，此函数会返回 mcErrorNotReady。如果该工作尚未完成，或者没有对事件调用 **mcEventRecord()**，则返回 mcSuccess。

另请参阅

mcEventCreate, mcEventCreateWithFlags, mcEventRecord, mcEventDestroy, mcEventSynchronize, mcEventElapsedTime

2.5.3.7 mcError_t mcEventRecord (mcEvent_t *event*, mcStream_t stream __*dparm*nullptr)

在指定流中记录事件。

参数

in	*event*	要记录的事件
in	*stream*	在其中要记录事件的流句柄

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorNotInitialized, mcErrorInvalidHandle, mcErrorLaunchFailure

必须使用 **mcEventQuery()** 或 **mcEventSynchronize()** 来确定事件何时从 recording（在调用 **mcEventRecord()**之后）转换为 recorded（如果请求，设置时间戳时）。

在该流中所有先前的命令都已执行完毕之后，记录在非空流中的事件在到达指定流的头部时，将从 recording 状态转换为 recorded 状态。

如果之前在此事件上调用过 **mcEventRecord()**，则此调用将覆盖事件中的任何现有状态。

如果对当前正在记录的事件调用此函数，则结果未定义：

任何一个未完成的记录都可能将状态保存到事件中，并且顺序不受保证。

另请参阅

mcEventCreate, **mcEventCreateWithFlags**, **mcEventQuery**, **mcEventSynchronize**, **mcEventDestroy**, **mcEventElapsedTime**

2.5.3.8 mcError_t mcEventRecordWithFlags (mcEvent_t *event*, mcStream_t stream __*dparm*nullptr, unsigned int flags __*dparm*0)

在指定流中记录事件。

参数

in	*event*	要记录的事件
in	*stream*	在其中要记录事件的流
in	*flags*	操作的参数

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorNotInitialized, mcErrorInvalidHandle, mcErrorLaunchFailure

必须使用 **mcEventQuery()** 或 **mcEventSynchronize()** 来确定事件何时从 recording（在调用 **mcEventRecordWithFlags()**之后）转换为 recorded（如果请求，设置时间戳时）。

在该流中所有先前的命令都已执行完毕之后，记录在非空流中的事件在到达指定流的头部时，将从 recording 状态转换为 recorded 状态。

如果之前在此事件上调用过 **mcEventRecordWithFlags()**，则此调用将覆盖事件中的任何现有状态。

如果对当前正在记录的事件调用此函数，则结果未定义：

任何一个未完成的记录都可能将状态保存到事件中，并且顺序不受保证。

标志包括：

- mcEventRecordDefault：默认事件记录标志。
- mcEventRecordExternal：在执行流捕获时，事件在图中被捕获为外部事件节点。

另请参阅

mcEventCreate, mcEventCreateWithFlags, mcEventQuery, mcEventSynchronize, mcEventDestroy, mcEventElapsedTime

2.5.3.9 mcError_t mcEventSynchronize (mcEvent_t *event*)

等待事件完成。

此函数将阻塞，直到事件准备就绪，等待在使用 **mcEventRecord()**记录事件时指定的流中先前所有的工作。

如果尚未对事件调用 **mcEventRecord()**，此函数将立即返回。

参数

in	*event*	要等待的事件

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorNotInitialized, mcErrorInvalidHandle, mcErrorLaunchFailure

另请参阅

mcEventCreate，mcEventCreateWithFlags，mcEventQuery，mcEventDestroy，mcEventRecord，mcEventElapsedTime

2.6 占用（Occupancy）

2.6.1 模块描述

本章节介绍 MXMACA 运行时 API 的占用函数。

2.6.2 函数

- `mcError_t` **`mcOccupancyAvailableDynamicSMemPerBlock`** `(size_t *dynamicSmemSize, const void *f, int numBlocks, int blockSize)`

  在 SM 上启动 numBlocks 块时，返回每个块可用的动态共享内存。

- `mcError_t` **`mcOccupancyMaxPotentialBlockSize`** `(int *gridSize, int *blockSize, const void *f, size_t dynSharedMemPerBlk, int blockSizeLimit)`

  确定网格和块大小以实现内核的最大占用率。

- `mcError_t` **`mcOccupancyMaxPotentialBlockSizeWithFlags`** `(int *gridSize, int *blockSize, const void *f, size_t dynSharedMemPerBlk, int blockSizeLimit, unsigned int flags)`

  返回具有指定标志的设备函数达到最大潜在占用率的网格和块大小。

- `mcError_t` **`mcOccupancyMaxActiveBlocksPerMultiprocessor`** `(int *numBlocks, const void *f, int blockSize, size_t dynamicSMemSize)`

  在*numBlocks 中返回设备函数的每个流式多处理器的最大活跃块数。

- `mcError_t` **`mcOccupancyMaxActiveBlocksPerMultiprocessorWithFlags`** `(int *numBlocks, const void *f, int blockSize, size_t dynamicSMemSize, unsigned int flags)`

  返回具有指定标志的设备函数的占用率。

- `mcError_t` **`mcModuleOccupancyMaxActiveBlocksPerMultiprocessor`** `(int *numBlocks, mcFunction_t f, int blockSize, size_t dynamicSMemSize)`

  在*numBlocks 中返回设备函数句柄的每个流式多处理器的最大活跃块数。

2.6.3 函数介绍

2.6.3.1 mcError_t mcModuleOccupancyMaxActiveBlocksPerMultiprocessor (int * *numBlocks*, mcFunction_t *f*, int *blockSize*, size_t *dynamicSMemSize*)

在*numBlocks 中返回设备函数句柄的每个流式多处理器的最大活跃块数。

参数

out	*numBlocks*	返回的占用率
in	*f*	计算占用率的内核函数句柄
in	*blockSize*	内核要启动的块大小
in	*dynamicSMemSize*	每个块的预期动态共享内存使用量，以字节为单位

返回值

mcSuccess, mcErrorInvalidDevice, mcErrorInvalidDeviceFunction, mcErrorInvalidValue

2.6.3.2 mcError_t mcOccupancyAvailableDynamicSMemPerBlock (size_t * *dynamicSmemSize*, const void * *f*, int *numBlocks*, int *blockSize*)

在 SM 上启动 numBlocks 块时，返回每个块可用的动态共享内存。

参数

out	*dynamicSmemSize*	返回的最大动态共享内存
in	*f*	计算占用率的内核函数
in	*numBlocks*	SM 上要启动的块数
in	*blockSize*	块的大小

返回值

mcSuccess, mcInvalidDevice, mcErrorInvalidValue

2.6.3.3 mcError_t mcOccupancyMaxActiveBlocksPerMultiprocessor (int * *numBlocks*, const void * *f*, int *blockSize*, size_t *dynamicSMemSize*)

在*numBlocks 中返回设备函数的每个流式多处理器的最大活跃块数。

参数

out	*numBlocks*	返回的占用率
in	*f*	计算占用率的内核函数
in	*blockSize*	内核要启动的块大小
in	*dynamicSMemSize*	每个块的预期动态共享内存使用量，以字节为单位

返回值

mcSuccess, mcErrorInvalidDevice, mcErrorInvalidDeviceFunction, mcErrorInvalidValue

2.6.3.4 mcError_t mcOccupancyMaxActiveBlocksPerMultiprocessorWithFlags (int * *numBlocks*, const void * *f*, int *blockSize*, size_t *dynamicSMemSize*, unsigned int *flags*)

返回具有指定标志的设备函数的占用率。

参数

out	*numBlocks*	返回的占用率
in	*f*	计算占用率的内核函数
in	*blockSize*	内核要启动的块大小
in	*dynamicSMemSize*	每个块的预期动态共享内存使用量，以字节为单位
in	*flags*	占用率计算器的请求行为

在*numBlocks 中返回设备函数的每个流式多处理器的最大活跃块数。flags 参数控制特殊情况的处理方式。有效的 flags 有：

- mcOccupancyDefault：将默认行为保持为 mcOccupancyMaxPotentialBlockSize。
- mcOccupancyDisableCachingOverride：此标志将抑制平台上全局缓存影响占用率的默认行为。在这样的平台上，如果启用了缓存，但每个块 SM 资源的使用为零占用，则占用率计算器将计算占用率，就好像禁用了缓存一样。在这种情况下，设置此标志会使占用率计算器返回 0。

返回值

mcSuccess, mcErrorInvalidDevice, mcErrorInvalidDeviceFunction, mcErrorInvalidValue

2.6.3.5 mcError_t mcOccupancyMaxPotentialBlockSize (int * *gridSize*, int * *blockSize*, const void * *f*, size_t *dynSharedMemPerBlk*, int *blockSizeLimit*)

确定网格和块大小以实现内核的最大占用率。

参数

out	*gridSize*	最大潜在占用率的最小网格尺寸
out	*blockSize*	最大潜在占用率的块大小
in	*f*	计算占用率的内核函数
in	*dynSharedMemPerBlk*	每个块的动态共享内存使用情况，以字节为单位
in	*blockSizeLimit*	内核的最大块大小，使用 0 表示没有限制

返回值

mcSuccess, mcErrorInvalidDevice, mcErrorInvalidValue, mcErrorInvalidDeviceFuntion

2.6.3.6 mcError_t mcOccupancyMaxPotentialBlockSizeWithFlags (int * *gridSize*, int * *blockSize*, const void * *f*, size_t *dynSharedMemPerBlk*, int *blockSizeLimit*, unsigned int *flags*)

返回具有指定标志的设备函数达到最大潜在占用率的网格和块大小。

参数

out	*gridSize*	最大潜在占用率的最小网格尺寸
out	*blockSize*	最大潜在占用率的块大小
in	*f*	计算占用率的内核函数
in	*dynSharedMemPerBlk*	每个块的动态共享内存使用情况，以字节为单位
in	*blockSizeLimit*	内核的最大块大小，使用 0 表示没有限制
in	*flags*	占用率计算器的请求行为

在*gridSize 和*blockSize 中返回一个建议的网格/块大小对，可实现最佳潜在占用率（即具有最小块数的最大活跃扭曲数）。flags 参数控制特殊情况的处理方式。有效的 flags 有：

- mcOccupancyDefault：将默认行为保持为 mcOccupancyMaxPotentialBlockSize。
- mcOccupancyDisableCachingOverride：此标志将抑制平台上全局缓存影响占用率的默认行为。在这样的平台上，如果启用了缓存，但每个块 SM 资源的使用为零占用，则占用率计算器将计算占用率，就好像禁用了缓存一样。在这种情况下，设置此标志会使占用率计算器返回 0。

返回值

mcSuccess, mcErrorInvalidDevice, mcErrorInvalidValue, mcErrorInvalidDeviceFuntion

2.7 版本管理

2.7.1 模块描述

本章节介绍 MXMACA 运行时 API 的版本管理函数。

2.7.2 函数

- `mcError_t` **`mcRuntimeGetVersion`** `(int *version)`

  返回 MXMACA 运行时版本。

- `mcError_t` **`mcDriverGetVersion`** `(int *version)`

  返回 MXMACA 驱动版本。

2.7.3 函数介绍

2.7.3.1 mcError_t mcDriverGetVersion (int * *version*)

返回 MXMACA 驱动版本。

参数

in	*version*	返回 MC 驱动版本

返回值

`mcSuccess, mcErrorInvalidValue`

在*version 中返回当前 MC 驱动实例的版本号。版本返回格式为：1000 主版本+10 次版本。例如，MC 驱动 1.2 表示为：1020。

如果版本参数为空，此函数会自动返回 mcErrorInvalidValue。

另请参阅

mcRuntimeGetVersion

2.7.3.2 mcError_t mcRuntimeGetVersion (int * *version*)

返回 MXMACA 运行时版本。

参数

in	*version*	返回 MXMACA 运行时版本

返回值

`mcSuccess, mcErrorInvalidValue`

在*version 中返回当前 MXMACA 运行时实例的版本号。版本返回格式为：1000 主版本+10 次版本。例如，MXMACA 运行时 1.2 表示为：1020。

如果版本参数为空，此函数会自动返回 mcErrorInvalidValue。

另请参阅

mcDriverGetVersion

2.8 错误处理

2.8.1 模块描述

本章节介绍 MXMACA 运行时 API 的错误处理函数。

2.8.2 函数

- `mcError_t` **`mcGetLastError`** `(void)`

  返回 MXMACA 运行时 API 调用返回的最后一个错误。

- `mcError_t` **`mcPeekAtLastError`** `(void)`

  返回 MXMACA 运行时 API 调用返回的最后一个错误。

- `const char *` **`mcGetErrorName`** `(mcError_t mc_error)`

  以文本形式返回指定错误码的名称。

- `const char *` **`mcGetErrorString`** `(mcError_t mc_error)`

  返回易用的文本字符串消息来解释错误。

2.8.3 函数介绍

2.8.3.1 const char * mcGetErrorName (mcError_t *mc_error*)

以文本形式返回指定错误码的名称。

参数

in	*mc_error*	要转换为名称的错误码

返回值

指向以'\0'结尾的错误名称的 `const char` 指针

返回一个包含枚举中错误码名称的字符串。如果错误代码无法识别，则返回“unrecoginzed error code”。

另请参阅

mcGetErrorString, mcGetLastError, #mcPeakAtLastError

2.8.3.2 const char * mcGetErrorString (mcError_t *mc_error*)

返回易用的文本字符串消息来解释错误。

参数

in	*mc_error*	要转换为字符串的错误码

返回值

指向以'\0'结尾的错误字符串的 `const char` 指针

返回一个枚举中错误码的描述字符串。如果错误代码无法识别，则返回“unrecoginzed error code”。

另请参阅

mcGetErrorName, mcGetLastError, mcPeakAtLastError

2.8.3.3 mcError_t mcGetLastError (void)

返回 MXMACA 运行时 API 调用返回的最后一个错误。

返回值

从活跃主机线程调用的最后一个 MC 返回错误码。

返回同一个主机线程中任意运行时调用返回的最后一个错误，然后将保存的错误重置为 mcSuccess。

另请参阅

mcGetErrorString, mcGetLastError, mcPeakAtLastError

2.8.3.4 mcError_t mcPeekAtLastError (void)

返回 MXMACA 运行时 API 调用返回的最后一个错误。

返回值

从活跃主机线程调用的最后一个 MC 返回错误码。

返回同一个主机线程中任意运行时调用产生的最后一个错误。请注意，此调用不会像 mcGetLastError() 那样将错误重置为 mcSuccess。

另请参阅

mcGetErrorString, mcGetLastError, mcPeakAtLastError

2.9 内存分配（Memory Malloc）

2.9.1 模块描述

本章节介绍 MXMACA 运行时 API 的内存分配函数。

2.9.2 函数

- `mcError_t` **`mcMalloc`** `(void **ptr, size_t sizeBytes)`

  分配设备上的内存。

- `mcError_t` **`mcFree`** `(void *ptr)`

  释放设备或主机上的内存。

- `mcError_t` **`mcMallocHost`** `(void **ptr, size_t size, unsigned int flags __dparm(mcMallocHostDefault))`

  分配设备可访问的、页面锁定的主机内存。

- `mcError_t` **`mcFreeHost`** `(void *ptr)`

  释放主机上的内存。

- `mcError_t` **`mcMallocPitch`** `(void **ptr, size_t *pitch, size_t width, size_t height)`

分配设备上的 pitched 内存。

- `mcError_t mcMalloc3D (struct mcPitchedPtr *pitchedDevPtr, struct mcExtent extent)`

  分配设备上的逻辑三维内存对象设。分配设备上最小宽*高*深字节的线性内存，并返回 mcPitchedPtr，其中 ptr 是指向已分配内存的指针。此函数可以填充分配，以确保满足硬件对齐要求。pitchedDevPtr 的 pitch 字段中返回的间距是分配宽度，以字节为单位。

- `mcError_t mcHostRegister (void *hostPtr, size_t sizeBytes, unsigned int flags __dparm(mcHostRegisterDefault))`

  将主机内存注册为页面锁定内存，以便可以从当前设备访问。

- `mcError_t mcHostUnregister (void *hostPtr)`

  注销已用 mcHostRegister 注册的内存范围。

- `mcError_t mcHostGetDevicePointer (void **devPtr, void *hostPtr, unsigned int flags __dparm(0))`

  获取通过 mcHostMalloc 分配或由 mcHostRegister 注册的内存的设备指针。

- `mcError_t mcHostGetFlags (unsigned int *flagsPtr, void *hostPtr)`

  传回用于分配由 mcMallocHost 分配或由 mcHostRegister 注册的固页主机内存的标志。

- `mcError_t mcExtMallocWithFlags (void **ptr, size_t sizeBytes, unsigned int flags)`

  分配默认加速器上的内存。

2.9.3 函数介绍

2.9.3.1 mcError_t mcExtMallocWithFlags (void ** *ptr*, size_t *sizeBytes*, unsigned int *flags*)

分配默认加速器上的内存。

参数

out	*ptr*	指向已分配内存的指针
in	*size*	请求的内存大小
in	*flags*	内存分配的类型

如果大小为 0，则不分配内存，*ptr 返回 nullptr，且此函数返回 mcSuccess。

返回值

mcSuccess, #mcErrorOutOfMemory, mcErrorInvalidValue (bad context, null *ptr)

另请参阅

mcMallocPitch, mcFree, mcMallocArray, mcFreeArray, mcMalloc3D, mcMalloc3DArray, mcFreeHost, mcMallocHost

2.9.3.2 mcError_t mcFree (void * *ptr*)

释放设备或主机上的内存。

参数

in	*ptr*	指向待释放的内存的指针

返回值

mcSuccess, mcErrorInvalidValue (pointer is invalid)

释放 `ptr` 指向的内存空间，该内存空间必须由之前对 mcMalloc()、mcMallocPitch()、mcMallocHost() 等的调用返回。另外，如果之前已经调用过 mcFree(`ptr`)，则会返回错误。如果 `ptr` 为 0，则不执行任何操作。mcFree()失败时，返回 mcErrorInvalidValue。

2.9.3.3 mcError_t mcFreeHost (void * *ptr*)

释放主机上的内存。

参数

in	*ptr*	指向待释放的内存的指针

返回值

mcSuccess, mcErrorInvalidValue (pointer is invalid)

释放 `ptr` 指向的内存空间，该内存空间必须由之前对 mcMallocHost()的调用返回。另外，如果之前已经调用过 mcFreeHost(ptr)，则会返回错误。如果 `ptr` 为 0，则不执行任何操作。mcFreeHost()失败时，返回 mcErrorInvalidValue。

2.9.3.4 mcError_t mcHostGetDevicePointer (void ** *devPtr*, void * *hostPtr*, unsigned int flags *__dparm*0)

获取通过 mcHostMalloc 分配或由 mcHostRegister 注册的内存的设备指针。

参数

out	*devPtr*	指向内存的设备指针

in	*hostPtr*	指向内存的主机指针
in	*flags*	扩展的标志，目前默认为 0

返回值

mcSuccess, mcErrorInvalidValue

对于设备属性 mcDevAttrCanUseHostPointerForRegisteredMem 具有非零值的设备，也可以使用主机指针 `devPtr` 从设备访问内存。**mcHostGetDevicePointer()**返回的设备指针可能与原始主机指针 `devPtr` 匹配或不匹配，取决于应用程序可见的设备。

如果应用程序可见的所有设备的设备属性都为非零值，则 **mcHostGetDevicePointer()**返回的设备指针将与原始指针 `devPtr` 匹配。

如果应用程序可见的任意设备的设备属性为零值，则 **mcHostGetDevicePointer()**返回的设备指针将与原始主机指针 `devPtr` 不匹配，但只要启用了统一虚拟寻址（UVA, Unified Virtual Addressing），它将适用于所有设备。在这样的系统中，使用设备属性具有非零值的设备上的任一指针来访问存储器是有效的。然而，请注意，这样的设备应该仅使用两个指针中的一个来访问存储器，而不是同时使用这两个指针。

2.9.3.5 mcError_t mcHostGetFlags (unsigned int * *flagsPtr*, void * *hostPtr*)

传回用于分配由 mcMallocHost 分配或由 mcHostRegister 注册的固页主机内存的标志。

参数

in	*hostPtr*	指向内存的主机指针
out	*flagsPtr*	主机内存的标志

返回值

mcSuccess, mcErrorInvalidValue

如果输入指针不在由 **mcMallocHost()** 分配或由 **mcHostRegister()** 注册的地址范围内，则 **mcHostGetFlags()**将失败。

说明

- 此函数还可能返回先前异步启动的错误码。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMallocHost, mcHostRegister

2.9.3.6 mcError_t mcHostRegister (void * *hostPtr*, size_t *sizeBytes*, unsigned int flags __*dparm*mcHostRegisterDefault)

将主机内存注册为页面锁定内存，以便可以从当前设备访问。

参数

in	*hostPtr*	指向内存的主机指针
in	*sizeBytes*	将要注册的内存字节数
in	*flags*	扩展的标志，目前默认为 0

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

页面锁定由 `hostPtr` 和 `sizeBytes` 指定的内存范围，并将其映射到由 `flags` 指定的设备。这个内存范围也被添加到与 mcMalllocHost()相同的跟踪机制中，以自动加速对 mcMemcpy*()等函数的调用。由于可以从设备直接访问此内存，因此可以使用比未注册的分页内存高得多的带宽来读取或写入此内存。页面锁定过多的内存可能会降低系统性能，因为这会减少系统可用于分页的内存容量。因此，最好谨慎地使用此函数来注册用于主机和设备之间数据交换的交换区（staging area）。只有设备属性 mcDevAttrHostRegisterSupported 具有非零值的设备才支持 mcHostRegister。

`flags` 参数允许指定影响分配的不同选项，如下所示。

- mcHostRegisterDefault：内存是映射的，且可移植的。
- mcHostRegisterPortable：所有上下文都认为内存是已注册的。
- mcHostRegisterMapped：将分配映射到当前设备的地址空间中。可以通过 mcHostGetDevicePointer 获取设备指针。

对于设备属性 mcDevAttrCanUseHostPointerForRegisteredMem 具有非零值的设备，也可以使用主机指针 `ptr` 从设备访问内存。**mcHostGetDevicePointer()**返回的设备指针可能与原始主机指针 `ptr` 匹配或不匹配，取决于应用程序可见的设备。如果应用程序可见的所有设备的设备属性都为非零值，则 **mcHostGetDevicePointer()**返回的设备指针将与原始指针 `ptr` 匹配。如果应用程序可见的任意设备的设备属性为零值，则 **mcHostGetDevicePointer()**返回的设备指针将与原始主机指针 `ptr` 不匹配，但只要启用了 UVA，它将适用于所有设备。在这样的系统中，使用设备属性具有非零值的设备上的任一指针来访问存储器是有效的。然而，请注意，这样的设备应该仅使用两个指针中的一个来访问存储器，而不是同时使用这两个指针。

由此函数进行页面锁定的内存必须使用 mcHostUnregister()进行注销。

2.9.3.7 mcError_t mcHostUnregister (void * *hostPtr*)

注销已用 mcHostRegister 注册的内存范围。

参数

in	*hostPtr*	指向要注销的内存的主机指针

返回值

mcSuccess, mcErrorInvalidValue

取消映射其基地址由 hostPtr 指定的内存范围，并再次使其分页。

基地址必须与指定给 mcHostRegister()的基地址相同。

2.9.3.8 mcError_t mcMalloc (void ** *ptr*, size_t *sizeBytes*)

分配设备上的内存。

参数

out	*ptr*	指向已分配设备内存的指针
in	*sizeBytes*	请求的分配大小，以字节为单位

返回值

mcSuccess, mcErrorMemoryAllocation (bad context, not enough memory), mcErrorInvalidValue (null *ptr)

在设备上分配 `sizeBytes` 字节的线性内存，并在`*ptr` 中返回一个指向已分配内存的指针。已分配内存对于任何类型的变量都是适当对齐的。如果内存不足，mcMalloc() 会在失败时返回 mcErrorMemoryAllocation。

2.9.3.9 mcError_t mcMalloc3D (struct mcPitchedPtr * *pitchedDevPtr*, struct mcExtent *extent*)

分配设备上的逻辑三维内存对象设。分配设备上最小宽*高*深字节的线性内存，并返回 mcPitchedPtr，其中 ptr 是指向已分配内存的指针。此函数可以填充分配，以确保满足硬件对齐要求。pitchedDevPtr 的 pitch 字段中返回的间距是分配宽度，以字节为单位。

参数

out	*pitchedDevPtr*	指向已分配 pitched 设备内存的指针
in	*extent*	请求的分配大小，以字节为单位

返回值

错误码

2.9.3.10 mcError_t mcMallocHost (void ** *ptr*, size_t *size*, unsigned int flags __*dparm*mcMallocHostDefault)

分配设备可访问的、页面锁定的主机内存。

参数

out	*ptr*	指向已分配主机固页内存的指针
in	*size*	请求的内存大小
in	*flags*	已分配内存的请求属性

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

分配页面锁定且设备可访问的 `size` 字节的主机内存。驱动跟踪此函数分配的虚拟内存范围，并自动加速对 mcMemcpy*()等函数的调用。由于可以从设备直接访问此内存，因此可以使用比 malloc()等函数获得的分页内存高得多的带宽来读取或写入此内存。使用 **mcMallocHost()**分配过多的内存可能会降低系统性能，因为它会减少系统可用于分页的内存容量。因此，最好谨慎地使用此函数来分配用于主机和设备之间数据交换的交换区。

`flags` 参数允许指定影响分配的不同选项，如下所示。

- mcMallocHostDefault：内存是映射的，且可移植的。
- mcMallocHostPortable：所有上下文都认为内存是注册的。
- mcMallocHostMapped：将分配映射到当前设备的地址空间中。可以通过 mcHostGetDevicePointer 获取设备指针。

为了使 mcMallocHostMapped 标志生效，MC 上下文必须支持 mcDeviceMapHost 标志，该标志可以用 **mcGetDeviceFlags()**来检查。mcDeviceMapHost 标志是为通过运行时 API 创建的上下文隐式设置的。

2.9.3.11 mcError_t mcMallocPitch (void ** *ptr*, size_t * *pitch*, size_t *width*, size_t *height*)

分配设备上的 pitched 内存。

参数

out	*ptr*	指向已分配设备内存的指针
out	*pitch*	分配的间距，以字节为单位
in	*width*	请求的 pitched 分配宽度，以字节为单位
in	*height*	请求的 pitched 分配高度

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

在设备上分配至少 `width*height` 字节的线性内存，并在`*devPtr` 中返回指向已分配内存的指针。该函数可以填充分配，以确保当地址从一行更新到另一行时，任何给定行中的对应指针将继续满足用于合并的对齐要求。**mcMallocPitch()**在`*pitch` 中返回的间距是分配宽度，以字节为单位。`pitch` 的预期用途是作为分配的单独参数，用于计算二维数组内的地址。给定类型 `T` 数组元素的行和列，地址计算为：

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

对于二维数组的分配，建议程序员考虑使用 **mcMallocPitch()**执行间距分配。由于硬件中的间距对齐限制，如果应用程序将在设备内存的不同区域之间执行二维内存复制，则情况尤其如此。

说明

- 此函数还可能返回先前异步启动的错误码。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMalloc, mcFree, mcMallocArray, mcFreeArray, mcMallocHost, mcFreeHost, mcMalloc3D, mcMalloc3DArray

2.10 内存复制

2.10.1 模块描述

本章节介绍 MXMACA 运行时 API 的内存复制函数。

2.10.2 函数

- `mcError_t` **`mcMemcpy`** `(void *dst, const void *src, size_t count, mcMemcpyKind kind)`

  将数据从 src 复制到 dst。src 和 dst 不能重叠。

- `mcError_t` **`mcMemcpyAsync`** `(void *dst, const void *src, size_t count, mcMemcpyKind kind, mcStream_t stream __dparm(0))`

  将数据从 src 复制到 dst。src 和 dst 不能重叠。

- `mcError_t` **`mcMemcpyWithStream`** `(void *dst, const void *src, size_t sizeBytes, mcMemcpyKind kind, mcStream_t stream)`

  使用 Stream 异步地将数据从 src 复制到 dst。

- `mcError_t` **`mcMemcpyPeer`** `(void *dst, int dstDevice, const void *src, int srcDevice, size_t sizeBytes)`

  将内存从一个设备复制到另一个设备上。

- `mcError_t` **`mcMemcpyPeerAsync`** `(void *dst, int dstDevice, const void *src, int srcDevice, size_t sizeBytes, mcStream_t stream)`

  在两个设备之间异步复制内存。

- `mcError_t` **`mcMemcpyParam2D`** `(const` **`mc_Memcpy2D`** `*pCopy)`

  复制二维数组的内存。

- `mcError_t` **`mcMemcpyParam2DAsync`** `(const` **`mc_Memcpy2D`** `*pCopy, mcStream_t stream __dparm(0))`

  复制二维数组的内存。

- `mcError_t` **`mcMemcpy2D`** `(void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, mcMemcpyKind kind)`

  在主机和设备之间复制数据。

- `mcError_t` **`mcMemcpy2DAsync`** `(void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, mcMemcpyKind kind, mcStream_t stream)`

  将数据从 src 复制到 dst。src 和 dst 不能重叠。

- `mcError_t` **`mcGetSymbolAddress`** `(void **devPtr, const void *symbol)`

  查找与 MXMACA 符号关联的地址。

- `mcError_t` **`mcGetSymbolSize`** `(size_t *size, const void *symbol)`

  查找与 MXMACA 符号关联的对象的大小。

- `mcError_t` **`mcMemcpyFromSymbol`** `(void *dst, const void *symbol, size_t sizeBytes, size_t offset __dparm(0), mcMemcpyKind kind __dparm(mcMemcpyDefault))`

复制设备上给定符号的数据。

- mcError_t **mcMemcpyFromSymbolAsync** (void *dst, const void *symbol, size_t sizeBytes, size_t offset __dparm(0), mcMemcpyKind kind __dparm(mcMemcpyDefault), mcStream_t stream __dparm(nullptr))

  异步复制设备上给定符号的数据。

- mcError_t **mcMemcpyToSymbol** (const void *symbol, const void *src, size_t sizeBytes, size_t offset __dparm(0), mcMemcpyKind kind __dparm(mcMemcpyDefault))

  将数据复制到设备上的给定符号。

- mcError_t **mcMemcpyToSymbolAsync** (const void *symbol, const void *src, size_t sizeBytes, size_t offset __dparm(0), mcMemcpyKind kind __dparm(mcMemcpyDefault), mcStream_t stream __dparm(nullptr))

  将数据异步复制到设备上的给定符号。

- mcError_t **mcMemcpy2DUnaligned** (const mc_Memcpy2D *pCopy)
- mcError_t **mcMemcpy3D** (const struct mcMemcpy3DParms *pCopy)

  复制二维数组的内存。此函数根据指针值推断传输类型：主机到主机、主机到设备、设备到设备、设备到主机。

- mcError_t **mcMemcpy3DAsync** (const struct **mcMemcpy3DParms** *pCopy, mcStream_t hStream)

  异步复制二维数组的内存。此函数根据指针值推断传输类型：主机到主机、主机到设备、设备到设备、设备到主机。

- mcError_t **mcMemcpy3DPeer** (const struct **mcMemcpy3DParms** *pCopy)

  在上下文之间异步复制内存。

- mcError_t **mcMemcpy3DPeerAsync** (const struct **mcMemcpy3DParms** *pCopy, mcStream_t hStream)

  在上下文之间复制内存。

- mcError_t **mcMemcpyDtoD** (mcDeviceptr_t dstDevice, mcDeviceptr_t srcDevice, size_t ByteCount)

  在设备之间复制内存。

- mcError_t **mcMemcpyDtoDAsync** (mcDeviceptr_t dstDevice, mcDeviceptr_t srcDevice, size_t ByteCount, mcStream_t hStream)

  在设备之间异步复制内存。

- mcError_t **mcMemcpyDtoH** (void *dstHost, mcDeviceptr_t srcDevice, size_t ByteCount)

  将内存从设备复制到主机。

- `mcError_t` **`mcMemcpyDtoHAsync`** `(void *dstHost, mcDeviceptr_t srcDevice, size_t ByteCount, mcStream_t hStream)`

  将内存从设备异步复制到主机。

- `mcError_t` **`mcMemcpyHtoD`** `(mcDeviceptr_t dstDevice, const void *srcHost, size_t ByteCount)`

  将内存从主机复制到设备。

- `mcError_t` **`mcMemcpyHtoDAsync`** `(mcDeviceptr_t dstDevice, const void *srcHost, size_t ByteCount, mcStream_t hStream)`

  将内存从主机异步复制到设备。

2.10.3 函数介绍

2.10.3.1 mcError_t mcGetSymbolAddress (void ** *devPtr*, const void * *symbol*)

查找与 MXMACA 符号关联的地址。

参数

in	*symbol*	设备符号指针
out	*devPtr*	返回的与 symbol 关联的设备指针

返回值

mcSuccess, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice

在`*devPtr` 中返回设备上符号 `symbol` 的地址。`symbol` 是驻留在全局或常量内存空间中的变量。如果找不到 `symbol`，或者 `symbol` 没有在全局或常量内存空间中声明，则`*devPtr` 保持不变，并返回错误 mcErrorInvalidSymbol。

说明

- 此函数还可能返回先前异步启动的错误码。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcGetSymbolSize, mcModuleGetGlobal

2.10.3.2 mcError_t mcGetSymbolSize (size_t * *size*, const void * *symbol*)

查找与 MXMACA 符号关联的对象的大小。

参数

in	*symbol*	设备符号指针
out	*size*	与 symbol 关联的对象的大小

返回值

mcSuccess, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice

在`*size` 中返回符号 `symbol` 的大小。`symbol` 是驻留在全局或常量内存空间中的变量。如果找不到 `symbol`，或者 `symbol` 没有在全局或常量内存空间中声明，则`*size` 保持不变，并返回错误 mcErrorInvalidSymbol。

说明

- 此函数还可能返回先前异步启动的错误码。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcGetSymbolAddress, mcModuleGetGlobal

2.10.3.3 mcError_t mcMemcpy (void * *dst*, const void * *src*, size_t *count*, mcMemcpyKind *kind*)

将数据从 src 复制到 dst。src 和 dst 不能重叠。

参数

in	*dst*	目标内存地址
in	*src*	源内存地址
in	*count*	要复制的大小，以字节为单位
in	*kind*	传输的类型

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

将 count 字节从 src 指向的内存区域复制到 dst 指向的内存区域，其中 kind 指定复制的方向，并且必须是 mcMemcpyHostToHost、mcMemcpyHostToDevice、mcMemcpyDeviceToHost、mcMemcpyDeviceToDevice 或 mcMemcpyDefault 其中之一。建议传递 mcMemcpyDefault，在这种情况下，传输类型是从指针值推断出来的。但是，只有支持 UVA 的系统才允许使用 mcMemcpyDefault。

2.10.3.4 mcError_t mcMemcpy2D (void * *dst*, size_t *dpitch*, const void * *src*, size_t *spitch*, size_t *width*, size_t *height*, mcMemcpyKind *kind*)

在主机和设备之间复制数据。

参数

in	*dst*	目标内存地址
in	*dpitch*	目标内存的 pitch
in	*src*	源内存地址
in	*spitch*	源存的 pitch
in	*width*	矩阵传输的宽度（列），以字节为单位
in	*height*	矩阵传输的高度（行）
in	*kind*	传输的类型

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidPitchValue, mcErrorInvalidDevicePointer, mcErrorInvalidMemcpyDirection

另请参阅

mcMemcpy, mcMemcpyToArray, mcMemcpy2DToArray, mcMemcpyFromArray, **mcMemcpyToSymbol**, **mcMemcpyAsync**

2.10.3.5 mcError_t mcMemcpy2DAsync (void * *dst*, size_t *dpitch*, const void * *src*, size_t *spitch*, size_t *width*, size_t *height*, mcMemcpyKind *kind*, mcStream_t *stream*)

将数据从 src 复制到 dst。src 和 dst 不能重叠。

参数

in	*dst*	目标内存地址

in	*dpitch*	目标内存的 pitch
in	*src*	源内存地址
in	*spitch*	源存的 pitch
in	*width*	矩阵传输的宽度（列），以字节为单位
in	*height*	矩阵传输的高度（行）
in	*kind*	传输的类型
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation, mcErrorUnknown

将一个矩阵（每 `width` 字节的 `height` 行）从 `src` 指向的内存区域复制到 `dst` 指向的内存区域，其中 kind 指定复制的方向，并且必须是 mcMemcpyHostToHost、mcMemcpyHostToDevice、mcMemcpyDeviceToHost、mcMemcpyDeviceToDevice 或 mcMemcpyDefault 其中之一。建议传递 mcMemcpyDefault，在这种情况下，传输类型是从指针值推断出来的。然而，只有支持 UVA 的系统才允许使用 mcMemcpyDefault。`dpitch` 和 `spitch` 是 `dst` 和 `src` 指向的二维数组的内存宽度（以字节为单位），包括添加到每行末尾的填充。内存区域可能不重叠。`width` 不得超过 `dpitch` 或 `spitch`。

使用与复制方向不匹配的 `dst` 和 `src` 指针调用 **mcMemcpy2DAsync()**会导致未定义的行为。如果 `dpitch` 或 `spitch` 大于允许的最大值，**mcMemcpy2DAsync()**会返回错误。

mcMemcpy2DAsync()相对于主机是异步的，因此调用可能在复制完成之前返回。可以选择性地通过传递非零 `stream` 参数将复制与流关联。如果 `kind` 是 mcMemcpyHostToDevice 或 mcMemcpyDeviceToHost，并且 `stream` 不为零，则该副本可能与其他流中的操作重叠。mcMemcpy2DAsync()不支持 mcMemcpyHostToDevice 和 mcMemcpyDeviceToHost 副本之间重叠。

2.10.3.6 mcError_t mcMemcpy2DUnaligned (const mc_Memcpy2D * *pCopy*)

复制二维数组的内存。

参数

pCopy[in]	内存复制的参数

返回值

mcSuccess, mcErrorInvalidValue

2.10.3.7 mcError_t mcMemcpy3D (const struct mcMemcpy3DParms * *pCopy*)

复制二维数组的内存。此函数根据指针值推断传输类型：主机到主机、主机到设备、设备到设备、设备到主机。

参数

in	*pCopy*	内存复制的参数

2.10.3.8 mcError_t mcMemcpy3DAsync (const struct mcMemcpy3DParms * *pCopy*, mcStream_t *hStream*)

异步复制二维数组的内存。此函数根据指针值推断传输类型：主机到主机、主机到设备、设备到设备、设备到主机。

参数

in	*pCopy*	内存复制的参数
in	*hStream*	流标识符

2.10.3.9 mcError_t mcMemcpy3DPeer (const struct mcMemcpy3DParms * *pCopy*)

在上下文之间异步复制内存。

参数

in	*pCopy*	内存复制的参数

2.10.3.10 mcError_t mcMemcpy3DPeerAsync (const struct mcMemcpy3DParms * *pCopy*, mcStream_t *hStream*)

在上下文之间复制内存。

参数

in	*pCopy*	内存复制的参数
in	*hStream*	流标识符

2.10.3.11 mcError_t mcMemcpyAsync (void * *dst*, const void * *src*, size_t *count*, mcMemcpyKind *kind*, mcStream_t stream *__dparm*0)

将数据从 src 复制到 dst。src 和 dst 不能重叠。

参数

in	*dst*	目标内存地址
in	*src*	源内存地址
in	*sizeBytes*	要复制的大小，以字节为单位
in	*kind*	传输的类型
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

将 `count` 字节从 `src` 指向的内存区域复制到 `dst` 指向的内存区域，其中 `kind` 指定复制的方向，并且必须是 mcMemcpyHostToHost、mcMemcpyHostToDevice、mcMemcpyDeviceToHost、mcMemcpyDeviceToDevice 或 mcMemcpyDefault 其中之一。建议传递 mcMemcpyDefault，在这种情况下，传输类型是从指针值推断出来的。但是，只有支持 UVA 的系统才允许使用 mcMemcpyDefault。

mcMemcpyAsync()相对于主机是异步的，因此调用可能在复制完成之前返回。可以选择性地通过传递非零 `stream` 参数将复制与流关联。如果 `kind` 是 mcMemcpyHostToDevice 或 mcMemcpyDeviceToHost，并且 `stream` 不为零，则该副本可能与其他流中的操作重叠。**mcMemcpyAsync()**不支持 mcMemcpyHostToDevice 和 mcMemcpyDeviceToHost 副本之间重叠。

如果未固定主机或目标，则将同步执行内存复制。为了获得最佳性能，请使用 mcMallocHost 来分配异步传输的主机内存。

2.10.3.12 mcError_t mcMemcpyDtoD (mcDeviceptr_t *dstDevice*, mcDeviceptr_t *srcDevice*, size_t *ByteCount*)

在设备之间复制内存。

参数

in	*dstDevice*	目标设备指针
in	*srcDevice*	源设备指针

in	*ByteCount*	内存复制的大小，以字节为单位

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

从设备内存复制到设备内存。`dstDevice` 和 `srcDevice` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D16, mcMemsetD2D32, mcMemsetD8, mcMemsetD16, mcMemsetD32, mcMemcpy, mcMemcpyToSymbol, mcMemcpyFromSymbol

2.10.3.13 mcError_t mcMemcpyDtoDAsync (mcDeviceptr_t *dstDevice*, mcDeviceptr_t *srcDevice*, size_t *ByteCount*, mcStream_t *hStream*)

在设备之间异步复制内存。

参数

in	*dstDevice*	目标设备指针
in	*srcDevice*	源设备指针
in	*ByteCount*	内存复制的大小，以字节为单位
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue, mcErrorInvalidResourceHandle

从设备内存复制到设备内存。`dstDevice` 和 `srcDevice` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemcpyAsync, mcMemcpyToSymbolAsync, mcMemcpyFromSymbolAsync

2.10.3.14 mcError_t mcMemcpyDtoH (void * *dstHost*, mcDeviceptr_t *srcDevice*, size_t *ByteCount*)

在主机和设备之间复制数据。

参数

in	*dstHost*	目标主机指针
in	*srcDevice*	源设备指针
in	*ByteCount*	内存复制的大小，以字节为单位

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

从设备内存复制到主机内存。`dstHost` 和 `srcDevice` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 请求的内存区域必须完全注册到 MXMACA，或者在主机分页传输时，根本不注册。不支持跨越已注册和未注册到 MXMACA 的分配的内存区域，并且会返回 mcErrorInvalidValue。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D16, mcMemsetD2D32, mcMemsetD8, mcMemsetD16, mcMemsetD32, mcMemcpy, mcMemcpyFromSymbol

2.10.3.15 mcError_t mcMemcpyDtoHAsync (void * *dstHost*, mcDeviceptr_t *srcDevice*, size_t *ByteCount*, mcStream_t *hStream*)

在主机和设备之间异步复制数据。

参数

in	*dstHost*	目标主机指针
in	*srcDevice*	源设备指针
in	*ByteCount*	内存复制的大小，以字节为单位
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue, mcErrorInvalidResourceHandle

从设备内存复制到主机内存。`dstHost` 和 `srcDevice` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 此函数使用标准默认流语义。
- 请求的内存区域必须完全注册到 MXMACA，或者在主机分页传输时，根本不注册。不支持跨越已注册和未注册到 MXMACA 的分配的内存区域，并且会返回 mcErrorInvalidValue。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8,

mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemcpyAsync, mcMemcpyToSymbolAsync, mcMemcpyFromSymbolAsync

2.10.3.16 mcError_t mcMemcpyFromSymbol (void * *dst*, const void * *symbol*, size_t *sizeBytes*, size_t offset *__dparm*0, mcMemcpyKind kind *__dparm*mcMemcpyDefault)

从设备上的给定符号指针复制数据。

参数

in	*dst*	目标内存地址
in	*symbol*	设备符号地址
in	*sizeBytes*	内存复制的大小，以字节为单位
in	*offset*	从符号地址开始的偏移量（offset），以字节为单位
in	*kind*	内存复制的方向

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice, mcErrorInvalidMemcpyDirection

将 `sizeBytes` 字节从符号 `symbol` 开头的 `offset` 字节指向的内存区域复制到 `dst` 指向的内存区域。内存域可能不重叠。`symbol` 是驻留在全局或常量内存空间中的变量。建议将 mcMemcpyDefault 作为 `kind` 的传递，因为 MXMACA 支持 UVA。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpy2D, mcMemcpy2DToArray, mcMemcpyToSymbol, mcMemcpyAsync, mcMemcpy2DAsync, mcMemcpyToSymbolAsync, mcMemcpyFromSymbolAsync, mcMemcpyDtoH, mcMemcpyDtoD

2.10.3.17 mcError_t mcMemcpyFromSymbolAsync (void * *dst*, const void * *symbol*, size_t *sizeBytes*, size_t offset __*dparm*0, mcMemcpyKind kind __*dparm*mcMemcpyDefault, mcStream_t stream __*dparm*nullptr)

异步复制设备上给定符号的数据。

参数

in	*dst*	目标内存地址
in	*symbol*	设备符号地址
in	*sizeBytes*	内存复制的大小，以字节为单位
in	*offset*	从符号地址开始的偏移量（offset），以字节为单位
in	*kind*	内存复制的方向
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice, mcErrorInvalidMemcpyDirection

将 `sizeBytes` 字节从符号 `symbol` 开头的 `offset` 字节指向的内存区域复制到 `dst` 指向的内存区域。内存域可能不重叠。`symbol` 是驻留在全局或常量内存空间中的变量。建议将 mcMemcpyDefault 作为 `kind` 的传递，因为 MXMACA 支持 UVA。

mcMemcpyAsync()相对于主机是异步的，因此调用可能在复制完成之前返回。可以选择性地通过传递非零 stream 参数将复制与流关联。如果流不为零，则复制可能与其他流中的操作重叠。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 此函数使用标准默认流语义。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpy2D, mcMemcpy2DToArray, mcMemcpyToSymbol, mcMemcpyFromSymbol, mcMemcpyAsync, mcMemcpy2DAsync, mcMemcpyToSymbolAsync, mcMemcpyDtoHAsync,

mcMemcpyDtoDAsync

2.10.3.18 mcError_t mcMemcpyHtoD (mcDeviceptr_t *dstDevice*, const void * *srcHost*, size_t *ByteCount*)

将内存从主机复制到设备。

参数

in	*dstDevice*	目标设备指针
in	*srcHost*	源主机指针
in	*ByteCount*	内存复制的大小，以字节为单位

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

从主机内存复制到设备内存。`dstDevice` 和 `srcHost` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 请求的内存区域必须完全注册到 MXMACA，或者在主机分页传输时，根本不注册。不支持跨越已注册和未注册到 MXMACA 的分配的内存区域，并且会返回 mcErrorInvalidValue。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D16, mcMemsetD2D32, mcMemsetD8, mcMemsetD16, mcMemsetD32, mcMemcpy, mcMemcpyToSymbol

2.10.3.19 mcError_t mcMemcpyHtoDAsync (mcDeviceptr_t *dstDevice*, const void * *srcHost*, size_t *ByteCount*, mcStream_t *hStream*)

将内存从主机异步复制到设备。

参数

in	*dstDevice*	目标设备指针
in	*srcHost*	源主机指针
in	*ByteCount*	内存复制的大小，以字节为单位
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue, mcErrorInvalidResourceHandle

从主机内存复制到设备内存。`dstDevice` 和 `srcHost` 分别是目标和源的基址指针。`ByteCount` 指定要复制的字节数。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 此函数使用标准默认流语义。
- 请求的内存区域必须完全注册到 MXMACA，或者在主机分页传输时，根本不注册。不支持跨越已注册和未注册到 MXMACA 的分配的内存区域，并且会返回 mcErrorInvalidValue。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemcpyAsync, mcMemcpyToSymbolAsync

2.10.3.20 mcError_t mcMemcpyParam2D (const mc_Memcpy2D * *pCopy*)

复制二维数组的内存。

参数

in	*pCopy*	内存复制的参数

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidPitchValue, mcErrorInvalidDevicePointer, mcErrorInvalidMemcpyDirection

另请参阅

mcMemcpy, **mcMemcpy2D**, mcMemcpyToArray, mcMemcpy2DToArray, mcMemcpyFromArray, **mcMemcpyToSymbol**, **mcMemcpyAsync**

2.10.3.21 mcError_t mcMemcpyParam2DAsync (const mc_Memcpy2D * *pCopy*, mcStream_t stream *__dparm*0)

复制二维数组的内存。

参数

in	*pCopy*	内存复制的参数
in	*stream*	要使用的流

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidPitchValue, mcErrorInvalidDevicePointer, mcErrorInvalidMemcpyDirection

另请参阅

mcMemcpy, **mcMemcpy2D**, mcMemcpyToArray, mcMemcpy2DToArray, mcMemcpyFromArray, **mcMemcpyToSymbol**, **mcMemcpyAsync**

2.10.3.22 mcError_t mcMemcpyPeer (void * *dst*, int *dstDevice*, const void * *src*, int *srcDevice*, size_t *sizeBytes*)

Copy memory from one device to memory on another device.

参数

out	*dst*	目标设备指针
in	*dstDevice*	目标设备
in	*src*	源设备指针
in	*srcDevice*	源设备

in	*sizeBytes*	内存复制的大小，以字节为单位

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidDevice

将内存从一个设备复制到另一个设备。`dst` 是目标内存的基址设备指针，`dstDevice` 是目标设备。`src` 是源内存的基址设备指针，`srcDevice` 是源设备。`sizeBytes` 指定要复制的字节数。请注意，此函数相对于主机是异步的，但对当前设备、`srcDevice` 和 `dstDevice` 中所有挂起和将来的异步工作进行序列化（使用 mcMemcpyPeerAsync 来避免这种同步）。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。
- 根据 mcStreamAddCallback 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpyAsync, mcMemcpyPeerAsync

2.10.3.23 mcError_t mcMemcpyPeerAsync (void * *dst*, int *dstDevice*, const void * *src*, int *srcDevice*, size_t *sizeBytes*, mcStream_t *stream*)

在两个设备之间异步复制内存。

参数

out	*dst*	目标设备指针
in	*dstDevice*	目标设备
in	*src*	源设备指针
in	*srcDevice*	源设备
in	*sizeBytes*	内存复制的大小，以字节为单位
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidDevice

将内存从一个设备复制到另一个设备。`dst` 是目标内存的基址设备指针，`dstDevice` 是目标设备。`src`

是源内存的基址设备指针，srcDevice 是源设备。`sizeBytes` 指定要复制的字节数。请注意，此函数相对于主机是异步的，所有工作都在其他设备上进行。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出异步行为。
- 此函数使用标准默认流语义。
- 根据 mcStreamAddCallback 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpyPeer, mcMemcpyAsync, mcMemcpyPeerAsync

2.10.3.24 mcError_t mcMemcpyToSymbol (const void * *symbol*, const void * *src*, size_t *sizeBytes*, size_t offset __*dparm*0, mcMemcpyKind kind __*dparm*mcMemcpyDefault)

将数据复制到设备上的给定符号。

参数

in	*symbol*	设备符号地址
in	*src*	源内存地址
in	*sizeBytes*	内存复制的大小，以字节为单位
in	*offset*	从符号地址开始的偏移量（offset），以字节为单位
in	*kind*	内存复制的方向

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice, mcErrorInvalidMemcpyDirection

将 `sizeBytes` 字节从 `src` 指向的内存区域复制到符号 `symbol` 开头的 `offset` 字节指向的内存区域。内存域可能不重叠。`symbol` 是驻留在全局或常量内存空间中的变量。建议将 mcMemcpyDefault 作为 `kind` 的传递，因为 MXMACA 支持 UVA。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。

- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpy2D, mcMemcpy2DToArray, mcMemcpyFromSymbol, mcMemcpyAsync, mcMemcpy2DAsync, mcMemcpyToSymbolAsync, mcMemcpyFromSymbolAsync, mcMemcpyHtoD, mcMemcpyDtoD

2.10.3.25 mcError_t mcMemcpyToSymbolAsync (const void * *symbol*, const void * *src*, size_t *sizeBytes*, size_t offset __*dparm*0, mcMemcpyKind kind __*dparm*mcMemcpyDefault, mcStream_t stream __*dparm*nullptr)

将数据异步复制到设备上的给定符号。

参数

in	*symbol*	设备符号地址
in	*src*	源内存地址
in	*sizeBytes*	内存复制的大小，以字节为单位
in	*offset*	从符号地址开始的偏移量（offset），以字节为单位
in	*kind*	内存复制的方向
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidSymbol, mcErrorNoKernelImageForDevice, mcErrorInvalidMemcpyDirection

将 `sizeBytes` 字节从 `src` 指向的内存区域复制到符号 `symbol` 开头的 `offset` 字节指向的内存区域。内存域可能不重叠。`symbol` 是驻留在全局或常量内存空间中的变量。建议将 mcMemcpyDefault 作为 `kind` 的传递，因为 MXMACA 支持 UVA。

mcMemcpyToSymbolAsync()相对于主机是异步的，因此调用可能在复制完成之前返回。可以选择性地通过传递非零 stream 参数将复制与流关联。如果流不为零，则复制可能与其他流中的操作重叠。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数在大多数用例中表现出同步行为。

- 此函数使用标准默认流语义。
- 根据 **mcStreamAddCallback** 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemcpy, mcMemcpy2D, mcMemcpy2DToArray, mcMemcpyToSymbol, mcMemcpyFromSymbol, mcMemcpyAsync, mcMemcpy2DAsync, mcMemcpyFromSymbolAsync, mcMemcpyHtoDAsync, mcMemcpyDtoDAsync

2.10.3.26 mcError_t mcMemcpyWithStream (void * *dst*, const void * *src*, size_t *sizeBytes*, mcMemcpyKind *kind*, mcStream_t *stream*)

使用 Stream 异步地将数据从 src 复制到 dst。

说明

如果未固定主机或目标，则将同步执行内存复制。为了获得最佳性能，请使用 mcMallocHost 来分配异步传输的主机内存。

在 HCC 上，mcMemcpyWithStream 不支持重叠的 H2D 和 D2H 副本。对于 mcMemcpy，始终由与指定流关联的设备执行复制。

对于多 GPU 或端对端配置，建议使用附加在 src 数据物理位于其中的设备上的流。对于最优的端对端复制，复制设备必须能够访问 src 和 dst 指针（通过调用 mcDeviceEnablePeerAccess，将复制代理作为当前设备，将 src/dest 作为 peerDevice 参数）。如果不这样做，mcMemcpy 仍然可以工作，但会使用主机上的暂存缓冲区执行复制。

参数

out	*dst*	正复制过来的数据
in	*src*	正复制出去的数据
in	*sizeBytes*	数据大小，以字节为单位
in	*kind*	传输的类型
in	*stream*	要使用的流

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree, mcErrorUnknown

另请参阅

mcMemcpy, mcMemcpy2D, mcMemcpyToSymbol, mcMemcpyFromSymbol, mcMemcpyToSymbolAsync, mcMemcpyFromSymbolAsync

2.11 内存设置

2.11.1 模块描述

本章节介绍 MXMACA 运行时 API 的内存设置函数。

2.11.2 函数

- `mcError_t` **`mcMemset`** `(void *dst, int value, size_t sizeBytes)`

  将设备内存初始化或设置为某个值。

- `mcError_t` **`mcMemsetAsync`** `(void *dst, int value, size_t sizeBytes, mcStream_t stream)`

  将设备内存初始化或设置为某个值。

- `mcError_t` **`mcMemset2D`** `(void *dst, size_t pitch, int value, size_t width, size_t height)`

  用常数值填充 dst 指向的内存区域。

- `mcError_t` **`mcMemset2DAsync`** `(void *dst, size_t pitch, int value, size_t width, size_t height, mcStream_t stream)`

  用常数值异步填充 dst 指向的内存区域。

- `mcError_t` **`mcMemset3D`** `(struct` **`mcPitchedPtr`** `pitchedDevPtr, int value, struct` **`mcExtent`** `extent)`

  用常量值同步填充 pitchedDevPtr 指向的内存区域。

- `mcError_t` **`mcMemset3DAsync`** `(struct` **`mcPitchedPtr`** `pitchedDevPtr, int value, struct` **`mcExtent`** `extent, mcStream_t stream)`

  用常量值异步填充 pitchedDevPtr 指向的内存区域。

- `mcError_t` **`mcMemsetD8`** `(mcDeviceptr_t dstDevice, unsigned char mValue, size_t mSize)`

  初始化设备内存。

- `mcError_t` **`mcMemsetD8Async`** `(mcDeviceptr_t dstDevice, unsigned char mValue, size_t mSize, mcStream_t hStream)`

  异步初始化设备内存。

- mcError_t **mcMemsetD16** (mcDeviceptr_t dstDevice, unsigned short mValue, size_t mSize)

  初始化设备内存。

- mcError_t **mcMemsetD16Async** (mcDeviceptr_t dstDevice, unsigned short mValue, size_t mSize, mcStream_t hStream)

  异步设置设备内存。

- mcError_t **mcMemsetD32** (mcDeviceptr_t dstDevice, unsigned int mValue, size_t mSize)

  初始化设备内存。

- mcError_t **mcMemsetD32Async** (mcDeviceptr_t dstDevice, unsigned int mValue, size_t mSize, mcStream_t hStream)

  异步初始化设备内存。

- mcError_t **mcMemsetD2D8** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned char mValue, size_t mWidth, size_t mHeight)

  初始化设备内存。

- mcError_t **mcMemsetD2D8Async** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned char mValue, size_t mWidth, size_t mHeight, mcStream_t hStream)

  异步初始化设备内存。

- mcError_t **mcMemsetD2D16** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned short mValue, size_t mWidth, size_t mHeight)

  初始化设备内存。

- mcError_t **mcMemsetD2D16Async** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned short mValue, size_t mWidth, size_t mHeight, mcStream_t hStream)

  异步初始化设备内存。

- mcError_t **mcMemsetD2D32** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned int mValue, size_t mWidth, size_t mHeight)

  初始化设备内存。

- mcError_t **mcMemsetD2D32Async** (mcDeviceptr_t dstDevice, size_t dstPitch, unsigned int mValue, size_t mWidth, size_t mHeight, mcStream_t hStream)

  异步初始化设备内存。

2.11.3 函数介绍

2.11.3.1 mcError_t mcMemset (void * *dst*, int *value*, size_t *sizeBytes*)

将设备内存初始化或设置为某个值。

参数

in	*dst*	指向设备内存的指针
in	*value*	为指定内存的每个字节设置的值
in	*sizeBytes*	要设置的大小，以字节为单位

返回值

mcSuccess, mcErrorInvalidValue, mcErrorMemoryAllocation

用常量字节值 `value` 填充 `dst` 指向的设备内存区域的第一个 `sizeBytes` 字节。

2.11.3.2 mcError_t mcMemset2D (void * *dst*, size_t *pitch*, int *value*, size_t *width*, size_t *height*)

用常数值填充 dst 指向的内存区域。

参数

out	*dst*	指向设备内存的指针
in	*pitch*	数据大小，以字节为单位
in	*value*	要设置的常数值
in	*width*	要设置的宽度
in	*height*	要设置的高度

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree

2.11.3.3 mcError_t mcMemset2DAsync (void * *dst*, size_t *pitch*, int *value*, size_t *width*, size_t *height*, mcStream_t *stream*)

用常数值异步填充 dst 指向的内存区域。

参数

in	*dst*	指向设备内存的指针
in	*pitch*	数据大小，以字节为单位
in	*value*	要设置的常数值
in	*width*	要设置的宽度

in	*height*	要设置的高度
in	*stream*	要设置的流句柄

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree

2.11.3.4 mcError_t mcMemset3D (struct mcPitchedPtr *pitchedDevPtr*, int *value*, struct mcExtent *extent*)

用常量值同步填充 pitchedDevPtr 指向的内存区域。

参数

in	*pitchedDevPtr*	要设置的内存区域指针
in	*value*	要设置的常数值
in	*extent*	要设置的拓展域

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree

2.11.3.5 mcError_t mcMemset3DAsync (struct mcPitchedPtr *pitchedDevPtr*, int *value*, struct mcExtent *extent*, mcStream_t *stream*)

用常量值异步填充 pitchedDevPtr 指向的内存区域。

参数

in	*pitchedDevPtr*	要设置的内存区域指针
in	*value*	要设置的常数值
in	*extent*	要设置的拓展域
in	*stream*	要设置的流句柄

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree

2.11.3.6 mcError_t mcMemsetAsync (void * *dst*, int *value*, size_t *sizeBytes*, mcStream_t *stream*)

将设备内存初始化或设置为某个值。

参数

out	*dst*	指向设备内存的指针
in	*value*	为指定内存的每个字节设置的值
in	*sizeBytes*	要设置的大小，以字节为单位
in	*stream*	流标识符

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorMemoryFree

用常量字节值 `value` 填充 `dst` 指向的设备内存区域的第一个 `sizeBytes` 字节。mcMemsetAsync()相对于主机是异步的，因此调用可能在 memset 完成之前返回。可以选择性地通过传递非零 stream 参数将操作与流关联。如果流不为零，则操作可能与其他流中的操作重叠。

2.11.3.7 mcError_t mcMemsetD16 (mcDeviceptr_t *dstDevice*, unsigned short *mValue*, size_t *mSize*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*mValue*	要设置的值
in	*mSize*	元素的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `nSize` 16 位值的内存范围设置为指定值 `mValue`。`dstDevice` 指针必须是 2 个字节对齐的。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset

2.11.3.8 mcError_t mcMemsetD16Async (mcDeviceptr_t *dstDevice*, unsigned short *mValue*, size_t *mSize*, mcStream_t *hStream*)

异步设置设备内存。

参数

in	*dstDevice*	目标设备指针
in	*mValue*	要设置的值
in	*mSize*	元素的数量
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `mSize` 16 位值的内存范围设置为指定值 `mValue`。`dstDevice` 指针必须是 2 个字节对齐的。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD32, mcMemsetD32Async, mcMemsetAsync

2.11.3.9 mcError_t mcMemsetD2D16 (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned short *mValue*, size_t *mWidth*, size_t *mHeight*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `mWidth` 16 位值的内存范围设置为指定值 `mValue`。`Height` 指定要设置的行数，`dstPitch` 指定每行之间的字节数。`dstDevice` 指针和 `dstPitch` 偏移量必须是 2 个字节对齐的。当间距是由 **mcMallocPitch()**传回的时，此函数执行得最快。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2D

2.11.3.10 mcError_t mcMemsetD2D16Async (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned short *mValue*, size_t *mWidth*, size_t *mHeight*, mcStream_t *hStream*)

异步初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `mWidth` 16 位值的内存范围设置为指定值 `mValue`。`Height` 指定要设置的行数，`dstPitch` 指定每行之间的字节数。`dstDevice` 指针和 `dstPitch` 偏移量必须是 2 个字节对齐的。当间距是由 **mcMallocPitch()**传回的时，此函数执行得最快。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2DAsync

2.11.3.11 mcError_t mcMemsetD2D32 (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned int *mValue*, size_t *mWidth*, size_t *mHeight*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针，必须为 4 个字节对齐的

in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `mWidth` 32 位值的二维内存范围设置为指定值 `mValue`。`Height` 指定要设置的行数，`dstPitch` 指定每行之间的字节数。`dstDevice` 指针和 `dstPitch` 偏移量必须是 4 个字节对齐的。当间距是由 **mcMallocPitch()**传回的时，此函数执行得最快。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2D

2.11.3.12 mcError_t mcMemsetD2D32Async (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned int *mValue*, size_t *mWidth*, size_t *mHeight*, mcStream_t *hStream*)

异步初始化设备内存。

参数

in	*dstDevice*	目标设备指针，必须为 4 个字节对齐的
in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量

in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 mWidth 32 位值的二维内存范围设置为指定值 mValue。Height 指定要设置的行数，dstPitch 指定每行之间的字节数。dstDevice 指针和 dstPitch 偏移量必须是 4 个字节对齐的。当间距是由 **mcMallocPitch()**传回的时，此函数执行得最快。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2DAsync

2.11.3.13 mcError_t mcMemsetD2D8 (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned char *mValue*, size_t *mWidth*, size_t *mHeight*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized,

mcErrorInvalidValue

将 mWidth 8 位值的二维内存范围设置为指定值 mValue。Height 指定要设置的行数，dstPitch 指定每行之间的字节数。当间距是由 mcMallocPitch()传回的时，此函数执行得最快。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2D

2.11.3.14 mcError_t mcMemsetD2D8Async (mcDeviceptr_t *dstDevice*, size_t *dstPitch*, unsigned char *mValue*, size_t *mWidth*, size_t *mHeight*, mcStream_t *hStream*)

异步初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*dstPitch*	指定每行之间的字节数，如果 mHeight 为 1，则未使用
in	*mValue*	要设置的值
in	*mWidth*	行的宽度
in	*mHeight*	行的数量
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 mWidth 8 位值的二维内存范围设置为指定值 mValue。Height 指定要设置的行数，dstPitch 指定每行之间的字节数。当间距是由 mcMallocPitch()传回的时，此函数执行得最快。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset2DAsync

2.11.3.15 mcError_t mcMemsetD32 (mcDeviceptr_t *dstDevice*, unsigned int *mValue*, size_t *mSize*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针，必须为 4 个字节对齐的
in	*mValue*	要设置的值
in	*mSize*	元素的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `nSize` 32 位值的内存范围设置为指定值 `mValue`。`dstDevice` 指针必须是 4 个字节对齐的。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32Async, mcMemset

2.11.3.16 mcError_t mcMemsetD32Async (mcDeviceptr_t *dstDevice*, unsigned int *mValue*, size_t *mSize*, mcStream_t *hStream*)

异步初始化设备内存。

参数

in	*dstDevice*	目标设备指针，必须为 4 个字节对齐的
in	*mValue*	要设置的值
in	*mSize*	元素的数量
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `mSize` 32 位值的内存范围设置为指定值 `mValue`。`dstDevice` 指针必须是 4 个字节对齐的。

说明

- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetAsync

2.11.3.17 mcError_t mcMemsetD8 (mcDeviceptr_t *dstDevice*, unsigned char *mValue*, size_t *mSize*)

初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*mValue*	要设置的值

in	*mSize*	元素的数量

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `nSize` 8 位值的内存范围设置为指定值 `mValue`。

说明

此函数还可能返回先前异步启动的错误码。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8Async, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemset

2.11.3.18 mcError_t mcMemsetD8Async (mcDeviceptr_t *dstDevice*, unsigned char *mValue*, size_t *mSize*, mcStream_t *hStream*)

异步初始化设备内存。

参数

in	*dstDevice*	目标设备指针
in	*mValue*	要设置的值
in	*mSize*	元素的数量
in	*hStream*	流标识符

返回值

mcSuccess, mcErrorDeinitialized, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorInvalidValue

将 `nSize` 8 位值的内存范围设置为指定值 `mValue`。

说明

- 此函数还可能返回先前异步启动的错误码。

- 此函数使用标准默认流语义。

另请参阅

mcMemcpy2D, mcMemcpy2DAsync, mcMemcpy2DUnaligned, mcMemcpy3D, mcMemcpy3DAsync, mcMemcpyDtoD, mcMemcpyDtoDAsync, mcMemcpyDtoH, mcMemcpyDtoHAsync, mcMemcpyHtoD, mcMemcpyHtoDAsync, mcMemGetAddressRange, mcMemGetInfo, mcMemsetD2D8, mcMemsetD2D8Async, mcMemsetD2D16, mcMemsetD2D16Async, mcMemsetD2D32, mcMemsetD2D32Async, mcMemsetD8, mcMemsetD16, mcMemsetD16Async, mcMemsetD32, mcMemsetD32Async, mcMemsetAsync

2.12 模块管理

2.12.1 详细介绍

本章节介绍 MC 运行时 API 的模块管理函数。

2.12.2 函数

- `mcError_t` **`mcLinkAddData`** `(mcLinkState_t state, mcJitInputType type, void *data, size_t size, const char *name, unsigned int numOptions, mcJitOption *options, void **optionValues)`

  向挂起的链接器调用添加输入。

- `mcError_t` **`mcLinkAddFile`** `(mcLinkState_t state, mcJitInputType type, const char *path, unsigned int numOptions, mcJitOption *options, void **optionValues)`

  向挂起的链接器调用添加文件输入。

- `mcError_t` **`mcLinkComplete`** `(mcLinkState_t state, void **dataOut, size_t *sizeOut)`

  完成挂起的链接器调用。

- `mcError_t` **`mcLinkCreate`** `(unsigned int numOptions, mcJitOption *options, void **optionValues, mcLinkState_t *stateOut)`

  创建挂起的 JIT 链接器调用。

- `mcError_t` **`mcLinkDestroy`** `(mcLinkState_t state)`

  销毁 JIT 链接器调用的状态。

- `mcError_t` **`mcModuleLoad`** `(mcModule_t *module, const char *fname)`

  将代码对象从文件加载到模块中。

- `mcError_t` **`mcModuleUnload`** `(mcModule_t module)`

释放模块。

- `mcError_t` **`mcModuleGetFunction`** `(mcFunction_t *function, mcModule_t module, const char *kname)`

  如果模块中存在 kname 函数，将提取该函数。

- `mcError_t` **`mcGetFuncBySymbol`** `(mcFunction_t *functionPtr, const void *symbolPtr)`

  获取指向与入口函数 `symbolPtr` 匹配的设备入口函数的指针。

- `mcError_t` **`mcFuncGetAttribute`** `(int *value, mcFunction_attribute attrib, mcFunction_t hfunc)`

  找出给定函数的特定属性。

- `mcError_t` **`mcFuncGetModule`** `(mcModule_t *module, mcFunction_t f)`

  返回模块句柄。

- `mcError_t` **`mcModuleLoadData`** `(mcModule_t *module, const void *image)`

  从驻留在主机内存中的代码对象构建模块。镜像（image）是指向该位置的指针。

- `mcError_t` **`mcModuleLoadDataEx`** `(mcModule_t *module, const void *image, unsigned int numOptions, mcJitOption *options, void **optionValues)`

  从驻留在主机内存中的代码对象构建模块。镜像（image）是指向该位置的指针。未使用选项。调用 mcDrvModuleLoadData。

- `mcError_t` **`mcModuleLoadFatBinary`** `(mcModule_t *module, const void *fatbin)`

  从驻留在主机内存中的 fatbin 构建模块。fatbin 是指向那个位置的指针。

- `mcError_t` **`mcModuleGetGlobal`** `(mcDeviceptr_t *dptr, size_t *bytes, mcModule_t hmod, const char *name)`

  从模块返回全局指针。

2.12.3 函数介绍

2.12.3.1 mcError_t mcFuncGetAttribute (int * *value*, mcFunction_attribute *attrib*, mcFunction_t *hfunc*)

找出给定函数的特定属性。

参数

out	*value*	给定函数的特定属性值
in	*attrib*	待查询的函数特定属性
in	*hfunc*	待查询的函数

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidDeviceFunction

在*pi 中返回 hfunc 给定的内核属性 attrib 的整数值。支持的属性如下：

MC_FUNC_ATTRIBUTE_MAX_THREADS_PER_BLOCK: 每个块的最大线程数，超过该线程数，函数的启动将失败。此属性根据用户设定的 launch_bounds 返回 maxThreadsPerBlock 的值；若未设定，则默认返回 512，以优化当前设备性能。但需注意，硬件支持的最大值为 1024。

MC_FUNC_ATTRIBUTE_SHARED_SIZE_BYTES: 此函数所需的每个块静态分配的共享内存的大小，以字节为单位。不包括用户在运行时请求的动态分配的共享内存。

MC_FUNC_ATTRIBUTE_CONST_SIZE_BYTES: 此函数所需的用户分配的常量内存的大小，以字节为单位。

MC_FUNC_ATTRIBUTE_LOCAL_SIZE_BYTES: 此函数每个线程使用的本地内存的大小，以字节为单位)。

MC_FUNC_ATTRIBUTE_NUM_REGS: 此函数每个线程使用的寄存器数。

MC_FUNC_ATTRIBUTE_BINARY_VERSION: 为其编译函数的二进制架构版本。此值为：二进制主版本*10+二进制次版本，因此二进制 1.3 函数将返回值 13。请注意，对于没有正确编码的二进制架构版本的遗留输出，将返回值 10。

MC_FUNC_CACHE_MODE_CA: 用于指示是否已使用用户指定的选项“-Xptxas --dlcm=ca”集编译函数的属性。

MC_FUNC_ATTRIBUTE_MAX_DYNAMIC_SHARED_SIZE_BYTES: 动态分配的共享内存的最大大小，以字节为单位。

MC_FUNC_ATTRIBUTE_PREFERRED_SHARED_MEMORY_CARVEOUT: 首选共享内存，L1 缓存拆分率，占总共享内存的百分比。

2.12.3.2 mcError_t mcFuncGetModule *(mcModule_t * module, mcFunction_t f)*

返回模块句柄。

参数

out	*module*	返回的模块句柄
in	*f*	用于检索模块的函数

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInvalidResourceHandle, mcErrorSymbolNotFound

在*module 中返回函数 hfunc 所在的模块的句柄。模块的生命周期与加载它的上下文的生命周期相对应，或者直到显式卸载模块为止。MC 运行时管理加载到主*context 中的自己的模块。如果此 API 返回的句柄引用 MC 运行时加载的模块，则对该模块调用 **mcModuleUnload()**将导致未定义的行为。

2.12.3.3 mcError_t mcGetFuncBySymbol (mcFunction_t * *functionPtr*, const void * *symbolPtr*)

获取指向与入口函数 `symbolPtr` 匹配的设备入口函数的指针。

参数

functionPtr	返回设备入口函数
symbolPtr	指向要搜索的设备入口函数的指针

返回值

#mcSuccess，mcErrorInvalidValue

在 `functionPtr` 中返回符号 `symbolPtr` 对应的设备入口函数。

2.12.3.4 mcError_t mcLinkAddData (mcLinkState_t *state*, mcJitInputType *type*, void * *data*, size_t *size*, const char * *name*, unsigned int *numOptions*, mcJitOption * *options*, void ** *optionValues*)

向挂起的链接器调用添加输入。

参数

out	*state*	挂起的链接器操作
in	*type*	输入数据的类型
in	*data*	挂起的链接器操作
in	*size*	输入数据的类型
in	*name*	挂起的链接器操作
in	*numOptions*	选项的大小
in	*options*	要创建的选项仅适用于此输入

in	*optionValues*	选项值数组，每个值强制转换为 void*

返回值

mcSuccess, mcErrorInvalidHandle, mcErrorInvalidValue, #mcErrorInvalidImage, #mcErrorOutOfMemory, #mcErrorNoBinaryForGpu

数据的所有权由调用者保留。此调用返回后，不会保留对任何输入的引用。

2.12.3.5 mcError_t mcLinkAddFile (mcLinkState_t *state*, mcJitInputType *type*, const char * *path*, unsigned int *numOptions*, mcJitOption * *options*, void ** *optionValues*)

向挂起的链接器调用添加文件输入。

参数

out	*state*	挂起的链接器操作
in	*type*	输入数据的类型
in	*path*	输入文件的路径
in	*numOptions*	选项的大小
in	*options*	要创建的选项仅适用于此输入
in	*optionValues*	选项值数组，每个值强制转换为 void*

返回值

mcSuccess, mcErrorInvalidHandle, mcErrorInvalidValue, #mcErrorInvalidImage, #mcErrorOutOfMemory, #mcErrorNoBinaryForGpu

数据的所有权由调用者保留。此调用返回后，不会保留对任何输入的引用。此方法相当于对文件的内容调用 mcLinkAddData。

2.12.3.6 mcError_t mcLinkComplete (mcLinkState_t *state*, void ** *dataOut*, size_t * *sizeOut*)

完成挂起的链接器调用。

参数

in	*state*	挂起的链接器调用
out	*dataOut*	输入数据的类型

out	*sizeOut*	输入文件的路径

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorOutOfMemory

完成挂起的链接器操作，并返回链接设备代码的数据镜像，可与 mcModuleLoadData 一起使用。数据由 state 所有，因此应在通过 mcLinkDestroy 销毁状态之前加载数据。此调用不销毁状态。

2.12.3.7 mcError_t mcLinkCreate (unsigned int *numOptions*, mcJitOption * *options*, void ** *optionValues*, mcLinkState_t * *stateOut*)

创建挂起的 JIT 链接器调用。

参数

in	*numOptions*	选项数组的大小
in	*options*	链接器和编译器选项的数组
in	*optionValues*	选项值数组，每个值强制转换为 void*
out	*stateOut*	成功后，它将包含一个 mcLinkState_t 来指定并完成此操作

返回值

mcSuccess, mcErrorInvalidValue, #mcErrorOutOfMemory, mcErrorJitCompilerNotFound

如果调用成功，则调用者拥有返回的 mcLinkState_t，该 mcLinkState_t 最终应使用 mcLinkDestroy 销毁。设备代码机器规格（32 或 64 位）将与调用应用相匹配。

如果使用输出选项，optionValues 必须在 mcLinkState_t 的生命周期内保持有效。此调用返回后，将不保留对输入的其他引用。

2.12.3.8 mcError_t mcLinkDestroy (mcLinkState_t *state*)

销毁 JIT 链接器调用的状态。

参数

in	*state*	链接器调用的状态对象

返回值

mcSuccess, mcErrorInvalidValue

2.12.3.9 mcError_t mcModuleGetFunction (mcFunction_t * *function*, mcModule_t *module*, const char * *kname*)

如果模块中存在 kname 函数，将提取该函数。

参数

out	*function*	返回的函数句柄
in	*module*	从中检索函数的模块
in	*kname*	要检索的函数的名称

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotFound, mcErrorInvalidDevice

在 *function 中返回模块中名称为 name 的函数的句柄。如果不存在该名称的函数，**mcModuleGetFunction()**返回 mcErrorNotFound。

2.12.3.10 mcError_t mcModuleGetGlobal (mcDeviceptr_t * *dptr*, size_t * *bytes*, mcModule_t *hmod*, const char * *name*)

从模块返回全局指针。

参数

out	*dptr*	返回的全局设备指针
in	*bytes*	返回的全局大小，以字节为单位
in	*hmod*	从中检索全局的模块
in	*name*	要检索的全局的名称

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotFound, mcErrorInvalidDevice

在*iadptr 和*字 bytes 中返回位于模块 hmod 中、名称为 name 的全局的基址指针和大小。如果不存在该名称的变量，**mcModuleGetFunction()**返回 mcErrorNotFound。参数 dptr 和 bytes 都是可选的。如果其中一个为 NULL，则会忽略它。

2.12.3.11 mcError_t mcModuleLoad (mcModule_t * *module*, const char * *fname*)

将代码对象从文件加载到模块中。

参数

in	*fname*	要加载的模块的文件名
out	*module*	返回的模块

返回值

mcSuccess, mcErrorInvalidValue, mcErrorDuplicateLoadModule, mcErrorFileNotFound, mcErrorInvalidKernelFile, #mcErrorNoBinaryForGpu

获取文件名 fname 并将相应的模块模 module 加载到当前上下文中。MC 驱动 API 不试图延迟分配模块所需的资源；如果无法分配模块所需的函数和数据（常量和全局）的内存，则 mcModuleLoad 将失败。该文件应该是 mxcc 输出的 out 文件或 mcfb 文件。

2.12.3.12 mcError_t mcModuleLoadData (mcModule_t * *module*, const void * *image*)

从驻留在主机内存中的代码对象构建模块。镜像（image）是指向该位置的指针。

参数

out	*module*	返回的模块
in	*image*	要加载的模块数据

返回值

mcSuccess, mcErrorInvalidValue, mcErrorAlreadyMapped, mcErrorDuplicateLoadModule, mcErrorFileNotFound, mcErrorInvalidKernelFile, #mcErrorNoBinaryForGpu

获取指针 image 并将相应的模块模 module 加载到当前上下文中。指针可以通过映射 out 或 mcfb 文件、将 out 或 mcfb 文件作为以'\0'结尾的文本字符串传递、或者将 out 或 mcfb 对象合并到可执行资源中并使用诸如 Windows FindResource()之类的操作系统调用来获得指针。

2.12.3.13 mcError_t mcModuleLoadDataEx (mcModule_t * *module*, const void * *image*, unsigned int *numOptions*, mcJitOption * *options*, void ** *optionValues*)

从驻留在主机内存中的代码对象构建模块。图像是指向该位置的指针。未使用选项。调用 mcDrvModuleLoadData。

参数

out	*module*	返回的模块
in	*image*	要加载的模块数据
in	*numOptions*	选项的尺寸
in	*options*	JIT
in	*option*	JIT 的值

返回值

mcSuccess, mcErrorInvalidValue, mcErrorAlreadyMapped, mcErrorDuplicateLoadModule, mcErrorFileNotFound, mcErrorInvalidKernelFile, #mcErrorNoBinaryForGpu

获取指针 image 并将相应的模块模 module 加载到当前上下文中。指针可以通过映射 out 或 mcfb 文件、将 out 或 mcfb 文件作为以'\0'结尾的文本字符串传递、或者将 out 或 mcfb 对象合并到可执行资源中并使用诸如 Windows FindResource()之类的操作系统调用来获得指针。选项作为数组通过 options 传递，任何相应的参数都以 optionValues 形式传递。使用 numOptions 提供选项总数。使用 optionValues 返回输出。

2.12.3.14 mcError_t mcModuleLoadFatBinary (mcModule_t * *module*, const void * *fatbin*)

从驻留在主机内存中的 fatbin 构建模块。fatbin 是指向那个位置的指针。

参数

out	*module*	返回的模块
in	*fatbin*	要加载的胖二进制（fat binary）

返回值

mcSuccess, mcErrorInvalidValue, mcErrorAlreadyMapped, mcErrorDuplicateLoadModule, mcErrorFileNotFound, mcErrorInvalidKernelFile, #mcErrorNoBinaryForGpu

获取指针 fatbin 并将相应的模块模 module 加载到当前上下文中。指针表示一个胖二进制对象，它是不同输出文件的集合，所有文件都表示相同的设备代码，但针对不同的架构进行了编译和优化。

2.12.3.15 mcError_t mcModuleUnload (mcModule_t *module*)

释放模块。

参数

in	*module*	要卸载的模块

返回值

mcSuccess, mcErrorNotFound

从当前上下文中卸载模块。

说明

模块被释放，与之相关的代码对象被销毁。

2.13 虚拟内存管理

2.13.1 模块描述

本章节介绍 MXMACA 运行时 API 的虚拟内存管理函数。

2.13.2 函数

- `mcError_t` **`mcMemAddressReserve`** `(mcDeviceptr_t *ptr, size_t size, size_t alignment, mcDeviceptr_t addr, unsigned long long flags)`

  分配虚拟地址范围预留。

- `mcError_t` **`mcMemAddressFree`** `(mcDeviceptr_t ptr, size_t size)`

  释放虚拟地址范围。

- `mcError_t` **`mcMemCreate`** `(mcMemGenericAllocationHandle *handle, size_t size, const` **`mcMemAllocationProp`** `*prop, unsigned long long flags)`

  创建一个 MXMACA 内存句柄，表示由给定属性描述的给定大小的内存分配。

- `mcError_t` **`mcMemRelease`** `(mcMemGenericAllocationHandle handle)`

  释放一个内存句柄，表示先前通过 mcMemCreate 分配的内存分配。

- `mcError_t` **`mcMemMap`** `(mcDeviceptr_t ptr, size_t size, size_t offset, mcMemGenericAllocationHandle handle, unsigned long long flags)`

  将分配句柄映射到预留的虚拟地址范围。

- `mcError_t` **`mcMemUnmap`** `(mcDeviceptr_t ptr, size_t size)`

  取消映射给定地址范围的后备内存。

- `mcError_t` **`mcMemSetAccess`** `(mcDeviceptr_t ptr, size_t size, const struct` **`mcMemAccessDesc`** `*desc, size_t count)`

为给定虚拟地址范围的 desc 中指定的每个位置设置访问标志。

- `mcError_t` **`mcMemGetAccess`** `(unsigned long long *flags, const struct` **`mcMemLocation`** `*location, mcDeviceptr_t ptr)`

  获取为给定位置和 ptr 设置的访问标志。

- `mcError_t` **`mcMemGetAllocationGranularity`** `(size_t *granularity, const` **`mcMemAllocationProp`** `*prop, mcMemAllocationGranularity_flags option)`

  计算最小粒度或推荐粒度。

- `mcError_t` **`mcMemGetAllocationPropertiesFromHandle`** `(mcMemAllocationProp *prop, mcMemGenericAllocationHandle handle)`

  检索定义此句柄属性的属性结构的内容。

- `mcError_t` **`mcMemRetainAllocationHandle`** `(mcMemGenericAllocationHandle *handle, void *addr)`

  给定一个地址，返回后备内存分配的分配句柄。

- `mcError_t` **`mcMemImportFromShareableHandle`** `(mcMemGenericAllocationHandle *handle, void *shareableHandle, enum mcMemAllocationHandleType shHandleType)`

  从请求的可共享句柄类型导入分配。

- `mcError_t` **`mcMemExportToShareableHandle`** `(void *shareableHandle, mcMemGenericAllocationHandle handle, enum mcMemAllocationHandleType handleType,` unsigned long long flags)

  将分配导出到请求的可共享句柄类型。

2.13.3 函数介绍

2.13.3.1 mcError_t mcMemAddressFree (mcDeviceptr_t *ptr*, size_t *size*)

释放虚拟地址范围。

参数

in	*ptr*	要释放的虚拟地址范围的起始地址
in	*size*	要释放的预留虚拟地址范围的大小

2.13.3.2 mcError_t mcMemAddressReserve (mcDeviceptr_t * *ptr*, size_t *size*, size_t *alignment*, mcDeviceptr_t *addr*, unsigned long long *flags*)

分配虚拟地址范围预留。

参数

out	*ptr*	指向已分配虚拟地址范围起始位置的结果指针
in	*size*	请求的预留虚拟地址范围的大小
in	*alignment*	请求的预留虚拟地址范围的对齐
in	*addr*	请求的固定起始地址范围
in	*flags*	当前未使用，目前必须为 0

2.13.3.3 mcError_t mcMemCreate (mcMemGenericAllocationHandle * *handle*, size_t *size*, const mcMemAllocationProp * *prop*, unsigned long long *flags*)

创建一个 MXMACA 内存句柄，表示由给定属性描述的给定大小的内存分配。

参数

out	*handle*	返回的句柄的值。此分配的所有操作都将使用此句柄执行。
in	*size*	请求的分配的大小
in	*prop*	要创建的分配的属性
in	*flags*	供将来使用的标志，目前必须为 0

2.13.3.4 mcError_t mcMemExportToShareableHandle (void * *shareableHandle*, mcMemGenericAllocationHandle *handle*, enum mcMemAllocationHandleType *handleType*, unsigned long long *flags*)

将分配导出到请求的可共享句柄类型。

参数

out	*shareableHandle*	指向存储请求句柄类型的位置的指针
in	*handle*	用于内存分配的 MXMACA 句柄

in	*handleType*	请求的可共享句柄的类型，定义可共享句柄输出参数的类型和大小
in	*flags*	预留，目前必须为 0

2.13.3.5 mcError_t mcMemGetAccess (unsigned long long * *flags*, const struct mcMemLocation * *location*, mcDeviceptr_t *ptr*)

获取为给定位置和 ptr 设置的访问标志。

参数

out	*flags*	为此位置设置的标志
in	*location*	检查标志的位置
in	*ptr*	检查访问标志的地址

2.13.3.6 mcError_t mcMemGetAllocationGranularity (size_t * *granularity*, const mcMemAllocationProp * *prop*, mcMemAllocationGranularity_flags *option*)

计算最小粒度或推荐粒度。

参数

out	*granularity*	返回的粒度
in	*prop*	要确定其粒度的属性
in	*option*	确定要返回的粒度

2.13.3.7 mcError_t mcMemGetAllocationPropertiesFromHandle (mcMemAllocationProp * *prop*, mcMemGenericAllocationHandle *handle*)

检索定义此句柄属性的属性结构的内容。

参数

out	*prop*	指向将保存有此句柄相关信息的属性结构的指针
in	*handle*	要对其执行查询的句柄

2.13.3.8 mcError_t mcMemImportFromShareableHandle (mcMemGenericAllocationHandle * *handle*, void * *shareableHandle*, enum mcMemAllocationHandleType *shHandleType*)

从请求的可共享句柄类型导入分配。

参数

in	*handle*	用于内存分配的 MXMACA 内存句柄
out	*shareableHandle*	表示要导入的内存分配的可共享句柄
in	*shHandleType*	导出句柄 mcMemAllocationHandleType 的句柄类型

2.13.3.9 mcError_t mcMemMap (mcDeviceptr_t *ptr*, size_t *size*, size_t *offset*, mcMemGenericAllocationHandle *handle*, unsigned long long *flags*)

将分配句柄映射到预留的虚拟地址范围。

参数

out	*ptr*	要映射内存的地址
in	*size*	内存映射的大小
in	*offset*	要从中开始映射的句柄表示的内存的偏移，当前必须为 0
in	*handle*	可共享内存的句柄
in	*flags*	供将来使用的标志，目前必须为 0

2.13.3.10 mcError_t mcMemRelease (mcMemGenericAllocationHandle *handle*)

释放一个内存句柄，表示先前通过 mcMemCreate 分配的内存分配。

参数

out	*handle*	之前由 mcMemCreate 返回的句柄的值

2.13.3.11 mcError_t mcMemRetainAllocationHandle (mcMemGenericAllocationHandle * *handle*, void * *addr*)

给定一个地址，返回后备内存分配的分配句柄。

参数

out	*handle*	用于后备内存分配的 MXMACA 内存句柄
in	*addr*	要查询的、先前已映射的内存地址

2.13.3.12 mcError_t mcMemSetAccess (mcDeviceptr_t *ptr*, size_t *size*, const struct mcMemAccessDesc * *desc*, size_t *count*)

为给定虚拟地址范围的 desc 中指定的每个位置设置访问标志。

参数

in	*ptr*	虚拟地址范围的起始地址
in	*size*	虚拟地址范围的长度
in	*desc*	**mcMemAccessDesc** 数组的指针，用于描述如何更改每个指定位置的映射
in	*count*	描述中的 mcMemAccessDesc 数量

2.13.3.13 mcError_t mcMemUnmap (mcDeviceptr_t *ptr*, size_t *size*)

取消映射给定地址范围的后备内存。

参数

in	*ptr*	要取消映射的虚拟地址范围的起始地址
in	*size*	要取消映射的虚拟地址范围的大小

2.14 流序内存分配器

2.14.1 模块描述

本章节介绍 MXMACA 运行时 API 的流序内存分配器函数。

2.14.2 函数

- `mcError_t` **`mcFreeAsync`** `(void *devPtr, mcStream_t hStream)`

  使用流序语义释放内存。

- `mcError_t` **`mcMallocAsync`** `(void **devPtr, size_t size, mcStream_t hStream)`

  使用流序语义分配内存。

- mcError_t **mcMallocFromPoolAsync** (void **ptr, size_t size, mcMemPool_t memPool, mcStream_t stream)

  从具有流序语义的指定池中分配内存。

- mcError_t **mcMemAllocAsync** (mcDeviceptr_t *dptr, size_t bytesize, mcStream_t hStream)

  使用流序语义分配内存。

- mcError_t **mcMemAllocFromPoolAsync** (mcDeviceptr_t *dptr, size_t bytesize, mcMemPool_t pool, mcStream_t hStream)

  从具有流序语义的指定池中分配内存。

- mcError_t **mcMemFreeAsync** (mcDeviceptr_t dptr, mcStream_t hStream)

  使用流序语义释放内存。

- mcError_t **mcMemPoolCreate** (mcMemPool_t *memPool, const struct **mcMemPoolProps** *poolProps)

  创建内存池。

- mcError_t **mcMemPoolDestroy** (mcMemPool_t memPool)

  销毁指定的内存池。

- mcError_t **mcMemPoolExportPointer** (struct mcMemPoolPtrExportData *exportData, void *ptr)

  导出数据以在进程之间共享内存池分配。

- mcError_t **mcMemPoolExportToShareableHandle** (void *sharedHandle, mcMemPool_t memPool, enum mcMemAllocationHandleType handleType, unsigned int flags)

  将内存池导出为请求的句柄类型。

- mcError_t **mcMemPoolGetAccess** (enum mcMemAccessFlags *flags, mcMemPool_t memPool, struct **mcMemLocation** *location)

  从设备返回内存池的可访问性。

- mcError_t **mcMemPoolGetAttribute** (mcMemPool_t memPool, enum mcMemPoolAttr attr, void *value)

  获取内存池的属性。

- mcError_t **mcMemPoolImportFromShareableHandle** (mcMemPool_t *memPool, void *sharedHandle, enum mcMemAllocationHandleType handleType, unsigned int flags)

  从共享句柄导入内存池。

- mcError_t **mcMemPoolImportPointer** (void **ptr, mcMemPool_t memPool, struct **mcMemPoolPtrExportData** *exportData)

  从另一个进程导入内存池分配。

- `mcError_t` **`mcMemPoolSetAccess`** `(mcMemPool_t memPool, const struct` **`mcMemAccessDesc`** `*descList, size_t count)`

  控制设备之间池的可见性。

- `mcError_t` **`mcMemPoolSetAttribute`** `(mcMemPool_t memPool, enum mcMemPoolAttr attr, void *value)`

  设置内存池的属性。

- `mcError_t` **`mcMemPoolTrimTo`** `(mcMemPool_t memPool, size_t minBytesToKeep)`

  尝试将内存释放回操作系统。

2.14.3 函数介绍

2.14.3.1 mcError_t mcFreeAsync (void * *devPtr*, mcStream_t *hStream*)

使用流序语义释放内存。

参数

in	*devPtr*	设备指针
in	*hStream*	建立流排序承诺的流

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotSupported

在 `hStream` 中插入一个空闲操作。流执行到空闲操作后，不得访问分配。此 API 返回后，从 GPU 上启动的任意后续工作访问内存或查询其指针属性将导致未定义的行为。

说明

- 在流捕获过程中，此函数会创建一个空闲节点，因此必须传递图形分配的地址。
- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。
- 根据 mcStreamAddCallback 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemFreeAsync, mcMallocAsync

2.14.3.2 mcError_t mcMallocAsync (void ** *devPtr*, size_t *size*, mcStream_t *hStream*)

使用流序语义分配内存。

参数

in	*devPtr*	返回的设备指针
in	*size*	要分配的字节数
in	*hStream*	建立流排序契约的流和要从中分配的内存池

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotSupported

在 `hStream` 中入一个分配操作。指向已分配内存的指针会立即在`*devPtr` 中返回。在分配操作完成之前，不得访问分配。分配来自与流设备关联的内存池。

说明

- 设备的默认内存池包含该设备的设备内存。
- 基本流排序允许将来提交到同一个流中的工作使用分配。可使用流查询、流同步和 MXMACA 事件确保在单独流中提交的工作运行之前完成分配操作。
- 在流捕获过程中，此函数会创建分一个分配节点。在这种情况下，分配由图形而不是内存池所有。使用内存池的属性设置节点的创建参数。
- 此函数还可能返回先前异步启动的错误码。
- 此函数使用标准默认流语义。
- 根据 mcStreamAddCallback 的指定，不能从回调中调用 MXMACA 函数。在这种情况下，mcErrorNotPermitted 可以作为诊断返回，但不能保证返回。

另请参阅

mcMemAllocAsync, mcMallocAsync(C++ API), mcMallocFromPoolAsync, mcFreeAsync, mcDeviceSetMemPool, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolSetAccess, mcMemPoolSetAttribute, mcMemPoolGetAttribute

2.14.3.3 mcError_t mcMallocFromPoolAsync (void ** *ptr*, size_t *size*, mcMemPool_t *memPool*, mcStream_t *stream*)

从具有流序语义的指定池中分配内存。

参数

in	*ptr*	返回的设备指针
in	*size*	要分配的字节数
in	*memPool*	要从中分配的内存池
in	*stream*	建立流顺序承诺的流

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotSupported

在 `stream` 中入一个分配操作。指向已分配内存的指针会立即在`*ptr` 中返回。在分配操作完成之前，不得访问分配。分配来自指定的内存池。

基本流排序允许将来提交到同一个流中的工作使用分配。可使用流查询、流同步和 MXMACA 事件确保在单独流中提交的工作运行之前完成分配操作。

说明

- 指定的内存池可能来自不同于指定 `stream` 的设备。
- 在流捕获过程中，此函数会创建分一个分配节点。在这种情况下，分配由图形而不是内存池所有。使用内存池的属性设置节点的创建参数。

另请参阅

mcMemAllocFromPoolAsync, mcMallocAsync(C++ API), mcMallocAsync, mcFreeAsync, mcDeviceGetDefaultMemPool, mcMemPoolCreate, mcMemPoolSetAccess, mcMemPoolSetAttribute

2.14.3.4 mcError_t mcMemAllocAsync (mcDeviceptr_t * *dptr*, size_t *bytesize*, mcStream_t *hStream*)

使用流序语义分配内存。

参数

in	*dptr*	返回的设备指针
in	*bytesize*	要分配的字节数
in	*hStream*	建立流排序契约的流和要从中分配的内存池

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorNotSupported, mcErrorMemoryAllocation

在 hStream 中入一个分配操作。指向已分配内存的指针会立即在*dptr 中返回。在分配操作完成之前，不得访问分配。分配来自流设备的当前内存池。

说明

- 设备的默认内存池包含该设备的设备内存。
- 基本流排序允许将来提交到同一个流中的工作使用分配。可使用流查询、流同步和 MXMACA 事件确保在单独流中提交的工作运行之前完成分配操作。
- 在流捕获过程中，此函数会创建分一个分配节点。在这种情况下，分配由图形而不是内存池所有。使用内存池的属性设置节点的创建参数。

另请参阅

mcMemAllocFromPoolAsync, mcMemFreeAsync, mcDeviceSetMemPool, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate, mcMemPoolSetAccess, mcMemPoolSetAttribute

2.14.3.5 mcError_t mcMemAllocFromPoolAsync (mcDeviceptr_t * *dptr*, size_t *bytesize*, mcMemPool_t *pool*, mcStream_t *hStream*)

从具有流序语义的指定池中分配内存。

参数

in	*dptr*	返回的设备指针
in	*bytesize*	要分配的字节数
in	*pool*	要从中分配的内存池
in	*hStream*	建立流顺序承诺的流

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorNotSupported, mcErrorMemoryAllocation

在 hStream 中入一个分配操作。指向已分配内存的指针会立即在*dptr 中返回。在分配操作完成之前，不得访问分配。分配来自指定的内存池。

基本流排序允许将来提交到同一个流中的工作使用分配。可使用流查询、流同步和 MXMACA 事件确保在单独流中提交的工作运行之前完成分配操作。

说明

- 指定的内存池可能来自不同于指定 hStream 的设备。

- 在流捕获过程中，此函数会创建分一个分配节点。在这种情况下，分配由图形而不是内存池所有。使用内存池的属性设置节点的创建参数。

另请参阅

mcMemAllocAsync, mcMemFreeAsync, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate, mcMemPoolSetAccess, mcMemPoolSetAttribute

2.14.3.6 mcError_t mcMemFreeAsync (mcDeviceptr_t *dptr*, mcStream_t *hStream*)

使用流序语义释放内存。

参数

in	*dptr*	要释放的内存指针
in	*hStream*	建立流排序契约的流

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInitializationError, mcErrorDeviceUninitialized, mcErrorNotSupported

在 `hStream` 中插入一个空闲操作。流执行到空闲操作后，不得访问分配。此 API 返回后，从 GPU 上启动的任意后续工作访问内存或查询其指针属性将导致未定义的行为。

说明

在流捕获过程中，此函数会创建一个空闲节点，因此必须传递图形分配的地址。

2.14.3.7 mcError_t mcMemPoolCreate (mcMemPool_t * *memPool*, const struct mcMemPoolProps * *poolProps*)

创建内存池。

参数

out	*memPool*	内存池句柄
in	*poolProps*	内存池的属性

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotSupported

创建一个 MXMACA 内存池，并在 `memPool` 中返回句柄。`poolProps` 确定池的属性，例如后备设备和

IPC 功能。

默认情况下，可以从分配内存池的设备访问其内存。

说明

指定 mcMemHandleTypeNone 会创建一个不支持 IPC 的内存池。

另请参阅

mcMemPoolCreate, mcDeviceSetMemPool, mcMallocFromPoolAsync, mcMemPoolExportToShareableHandle, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool

2.14.3.8 mcError_t mcMemPoolDestroy (mcMemPool_t *memPool*)

销毁指定的内存池。

参数

in	*memPool*	内存池句柄

返回值

mcSuccess, mcErrorInvalidValue

如果在调用 **mcMemPoolDestroy** 时，从该池获得的指针尚未释放，或者该池有尚未完成的释放操作，则该函数将立即返回，并且一旦没有未完成的分配，与该池关联的资源将自动释放。

销毁设备的当前内存池会将该设备的默认内存池设置为该设备的当前内存池。

说明

无法销毁设备的默认内存池。

另请参阅

mcMemPoolDestroy, mcFreeAsync, mcDeviceSetMemPool, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate

2.14.3.9 mcError_t mcMemPoolExportPointer (struct mcMemPoolPtrExportData * *exportData*, void * *ptr*)

导出数据以在进程之间共享内存池分配。

参数

out	*exportData*	返回的导出数据

in	*ptr*	指向正在导出的内存的指针

返回值

mcSuccess, mcErrorInvalidValue

从已共享内存池构造为了共享特定分配的 `exportData`。接收方进程可以使用 **mcMemPoolImportPointer** API 导入分配。数据不是句柄，可以通过任意 IPC 机制共享。

另请参阅

mcMemPoolExportPointer, mcMemPoolExportToShareableHandle, mcMemPoolImportFromShareableHandle, mcMemPoolImportPointer

2.14.3.10 mcError_t mcMemPoolExportToShareableHandle (void * *sharedHandle*, mcMemPool_t *memPool*, enum mcMemAllocationHandleType *handleType*, unsigned int *flags*)

将内存池导出为请求的句柄类型。

参数

out	*sharedHandle*	指向存储请求句柄的位置的指针
in	*memPool*	要导出的内存池
in	*handleType*	要创建的句柄的类型
in	*flags*	必须为 0

返回值

mcSuccess, mcErrorInvalidValue

给定一个支持 IPC 的内存池，创建一个操作系统句柄，以便与另一个进程共享该池。接收方进程可以使用 **mcMemPoolImportFromShareableHandle** 将可共享句柄转换为内存池。然后可以与 **mcMemPoolExportPointer** 和 **mcMemPoolImportPointer** API 共享各个指针。可共享句柄是什么以及如何传输的实现由请求的句柄类型定义。

说明

要创建一个支持 IPC 的内存池，请使用 mcDrvMemAllocationHandleType_t 而不是 mcMemHandleTypeNone 创建内存池。

另请参阅

mcMemPoolExportToShareableHandle, mcMemPoolImportFromShareableHandle, mcMemPoolExportPointer, mcMemPoolImportPointer

2.14.3.11 mcError_t mcMemPoolGetAccess (enum mcMemAccessFlags * *flags*, mcMemPool_t *memPool*, struct mcMemLocation * *location*)

从设备返回内存池的可访问性。

参数

out	*flags*	指定位置内存池的可访问性
in	*memPool*	正在查询的内存池
in	*location*	访问内存的位置。

从指定位置返回池内存的可访问性

另请参阅

mcMemPoolGetAccess, mcMemPoolSetAccess

2.14.3.12 mcError_t mcMemPoolGetAttribute (mcMemPool_t *memPool*, enum mcMemPoolAttr *attr*, void * *value*)

获取内存池的属性。

参数

in	*memPool*	要获取属性的内存池
in	*attr*	要获取的属性
in	*value*	检索值

返回值

mcSuccess, mcErrorInvalidValue

另请参阅

mcMemPoolGetAttribute, mcMallocAsync, mcFreeAsync, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate

2.14.3.13 mcError_t mcMemPoolImportFromShareableHandle (mcMemPool_t * *memPool*, void * *sharedHandle*, enum mcMemAllocationHandleType *handleType*, unsigned int *flags*)

imports a memory pool from a shared handle.

参数

out	*memPool*	返回的内存池
in	*sharedHandle*	要打开的内存池的操作系统句柄
in	*handleType*	要导入的句柄类型
in	*flags*	必须为 0

返回值

mcSuccess, mcErrorInvalidValue

可以使用 **mcMemPoolImportPointer** 从导入的池中导入特定的分配。

说明

导入的内存池不支持创建新的分配。因此，导入的内存池可能不会在 **mcDeviceSetMemPool** 或 **mcMallocFromPoolAsync** 调用中使用。

另请参阅

mcMemPoolImportFromShareableHandle, mcMemPoolExportToShareableHandle, mcMemPoolExportPointer, mcMemPoolImportPointer

2.14.3.14 mcError_t mcMemPoolImportPointer (void ** *ptr*, mcMemPool_t *memPool*, struct mcMemPoolPtrExportData * *exportData*)

从另一个进程导入内存池分配。

参数

out	*ptr*	指向导入内存的指针
in	*memPool*	要从中导入的池
in	*exportData*	指定要导入的内存的数据

返回值

mcSuccess, mcErrorInvalidValue, mcErrorInitializationError, mcErrorMemoryAllocation

在 `ptr` 中返回指向导入内存的指针。在导出进程中完成分配操作之前，不得访问导入的内存。导入的内存必须先从所有导入进程中释放，然后才能在导出进程中释放。可以使用 `mcFree` 或 `mcFreeAsync` 释放指针。如果使用 `mcFreeAsync`，则必须先在导入进程中完成释放，然后才能在导出进程中释放。

说明

mcFreeAsync 操作在其流中完成之前，可以在导出进程中使用 `mcFreeAsync` API，只要导出进程中的 **mcFreeAsync** 指定流依赖于导入进程的 **mcFreeAsync** 即可。

另请参阅

mcMemPoolImportPointer, mcMemPoolExportToShareableHandle, mcMemPoolImportFromShareableHandle, mcMemPoolExportPointer

2.14.3.15 mcError_t mcMemPoolSetAccess (mcMemPool_t *memPool*, const struct mcMemAccessDesc * *descList*, size_t *count*)

控制设备之间池的可见性。

参数

in	*memPool*	正在修改的内存池
in	*descList*	访问描述符数组。每个描述符为单个 GPU 指示启用访问
in	*count*	映射数组中的描述符数

返回值

mcSuccess, mcErrorInvalidValue

另请参阅

mcMemPoolSetAccess, mcMemPoolGetAccess, mcMallocAsync, mcFreeAsync

2.14.3.16 mcError_t mcMemPoolSetAttribute (mcMemPool_t *memPool*, enum mcMemPoolAttr *attr*, void * *value*)

设置内存池的属性。

参数

in	*memPool*	要修改的内存池
in	*attr*	要修改的属性
in	*value*	指向要赋予的值的指针

返回值

mcSuccess, mcErrorInvalidValue

另请参阅

mcMemPoolSetAttribute, mcMallocAsync, mcFreeAsync, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate

2.14.3.17 mcError_t mcMemPoolTrimTo (mcMemPool_t *memPool*, size_t *minBytesToKeep*)

尝试将内存释放回操作系统。

参数

in	*memPool*	要修剪的内存池
in	*minBytesToKeep*	如果内存池预留的字节数少于 minBytesToKeep，则 TrimTo 操作为 no-op。否则，要保证内存池在操作后至少预留的 minBytesToKeep 字节

返回值

mcSuccess, mcErrorInvalidValue, mcErrorNotSupported

将内存释放回操作系统，直到池中包含的预留字节数少于 minBytesToKeep，或者分配器无法安全释放更多内存。分配器无法释放支持未完成异步分配的操作系统分配。操作系统分配的粒度可能与用户分配的粒度不同。

说明

- 尚未释放的分配算作未完成。
- 已异步释放但尚未在主机上观察到其完成的分配（例如，通过同步）可以算作未完成的分配。

另请参阅

mcMemPoolTrimTo, mcMallocAsync, mcFreeAsync, mcDeviceGetDefaultMemPool, mcDeviceGetMemPool, mcMemPoolCreate

2.15 流内存操作

2.15.1 模块描述

本章节介绍 MXMACA 运行时 API 的流内存操作函数。

2.15.2 函数

- `mcError_t` **`mcStreamWaitValue32`** `(mcStream_t stream, void *ptr, uint32_t value, unsigned int flags)`
- `mcError_t` **`mcStreamWaitValue64`** `(mcStream_t stream, void *ptr, uint64_t value, unsigned int flags)`
- `mcError_t` **`mcStreamWriteValue32`** `(mcStream_t stream, void *ptr, uint32_t value, unsigned int flags)`
- `mcError_t` **`mcStreamWriteValue64`** `(mcStream_t stream, void *ptr, uint64_t value, unsigned int flags)`

3. 数据结构和数据字段

参见 MXMACA 软件包中的头文件，MXMACA 的安装参见《曦云®系列通用计算 GPU 快速上手指南》。

4. 附录

4.1 术语/缩略语

术语/缩略语	全名	描述
IPC	Interprocess Communication	进程间通信
mc		MXMACA 运行时 API 函数的前缀
MXMACA	MetaX Advanced Compute Architecture	沐曦推出的 GPU 软件栈，包含了沐曦 GPU 的底层驱动、编译器、数学库及整套软件工具套件
PCIe	Peripheral Component Interconnect-Express	高速串行计算机扩展总线
SIMD	Single Instruction Multiple Data	单指令、多数据
SIMT	Single Instruction Multiple Thread	单指令、多线程
SVM	Shared Virtual Memory	共享虚拟内存

声明