HIP API

Generated by Doxygen 1.8.20

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Module Documentation

4.1 Global enum and defines

Classes

- struct dim3
- struct hipLaunchParams_t
- struct hipExternalMemoryHandleDesc_st
- struct hipExternalMemoryBufferDesc_st
- struct hipExternalSemaphoreHandleDesc_st
- struct hipExternalSemaphoreSignalParams_st
- struct hipExternalSemaphoreWaitParams_st

Macros

- #define **__HIP_NODISCARD**
- #define hipStreamDefault 0x00

  Flags that can be used with hipStreamCreateWithFlags.
- #define hipStreamNonBlocking 0x01
- #define hipEventDefault 0x0

  Flags that can be used with hipEventCreateWithFlags:
- #define hipEventBlockingSync 0x1

  Waiting will yield CPU. Power-friendly and usage-friendly but may increase latency.
- #define hipEventDisableTiming 0x2

  Disable event's capability to record timing information. May improve performance.
- #define hipEventInterprocess 0x4

  Event can support IPC.
- #define **hipEventReleaseToDevice** 0x40000000
- #define hipEventReleaseToSystem 0x80000000
- #define hipHostMallocDefault 0x0

  Flags that can be used with hipHostMalloc.
- #define hipHostMallocPortable 0x1

  Memory is considered allocated by all contexts.
- #define hipHostMallocMapped 0x2

  can be obtained with hipHostGetDevicePointer.
- #define **hipHostMallocWriteCombined** 0x4

- #define hipHostMallocNumaUser 0x20000000

  Host memory allocation will follow numa policy set by user.
- #define hipHostMallocCoherent 0x40000000

  allocation.
- #define hipHostMallocNonCoherent 0x80000000

  allocation.
- #define hipMemAttachGlobal 0x01

  Memory can be accessed by any stream on any device.
- #define hipMemAttachHost 0x02

  Memory cannot be accessed by any stream on any device.
- #define hipMemAttachSingle 0x04

  the associated device
- #define **hipDeviceMallocDefault** 0x0
- #define hipDeviceMallocFinegrained 0x1

  Memory is allocated in fine grained region of device.
- #define hipMallocSignalMemory 0x2
- #define hipHostRegisterDefault 0x0

  Flags that can be used with hipHostRegister.
- #define hipHostRegisterPortable 0x1

  Memory is considered registered by all contexts.
- #define hipHostRegisterMapped 0x2

  can be obtained with hipHostGetDevicePointer.
- #define hipHostRegisterIoMemory 0x4

  Not supported.
- #define hipExtHostRegisterCoarseGrained 0x8

  Coarse Grained host memory lock.
- #define hipDeviceScheduleAuto 0x0

  Automatically select between Spin and Yield.
- #define hipDeviceScheduleSpin 0x1

  may consume more power.
- #define hipDeviceScheduleYield 0x2

  power and is friendlier to other threads in the system.
- #define **hipDeviceScheduleBlockingSync** 0x4
- #define **hipDeviceScheduleMask** 0x7
- #define **hipDeviceMapHost** 0x8
- #define **hipDeviceLmemResizeToMax** 0x16
- #define hipArrayDefault 0x00

  Default HIP array allocation flag.
- #define **hipArrayLayered** 0x01
- #define **hipArraySurfaceLoadStore** 0x02
- #define **hipArrayCubemap** 0x04
- #define **hipArrayTextureGather** 0x08
- #define **hipOccupancyDefault** 0x00
- #define **hipCooperativeLaunchMultiDeviceNoPreSync** 0x01
- #define **hipCooperativeLaunchMultiDeviceNoPostSync** 0x02
- #define **hipCpuDeviceId** ((int)-1)
- #define **hipInvalidDeviceId** ((int)-2)
- #define hipExtAnyOrderLaunch 0x01

  AnyOrderLaunch of kernels.
- #define **hipStreamWaitValueGte** 0x0
- #define **hipStreamWaitValueEq** 0x1
- #define **hipStreamWaitValueAnd** 0x2
- #define **hipStreamWaitValueNor** 0x3
- #define hipStreamPerThread ((hipStream_t)2)

  Implicit stream per application thread.

Typedefs

- typedef enum __HIP_NODISCARD hipError_t **hipError_t**
- typedef enum hipDeviceAttribute_t **hipDeviceAttribute_t**
- typedef enum hipMemoryAdvise **hipMemoryAdvise**
- typedef enum hipMemRangeCoherencyMode **hipMemRangeCoherencyMode**
- typedef enum hipMemRangeAttribute **hipMemRangeAttribute**
- typedef enum hipJitOption **hipJitOption**
- typedef enum hipFuncAttribute hipFuncAttribute
- typedef enum hipFuncCache_t hipFuncCache_t
- typedef enum hipSharedMemConfig hipSharedMemConfig
- typedef struct dim3 dim3
- typedef struct hipLaunchParams_t **hipLaunchParams**
- typedef enum hipExternalMemoryHandleType_enum **hipExternalMemoryHandleType**
- typedef struct hipExternalMemoryHandleDesc_st **hipExternalMemoryHandleDesc**
- typedef struct hipExternalMemoryBufferDesc_st **hipExternalMemoryBufferDesc**
- typedef void ∗ **hipExternalMemory_t**
- typedef enum hipExternalSemaphoreHandleType_enum **hipExternalSemaphoreHandleType**
- typedef struct hipExternalSemaphoreHandleDesc_st **hipExternalSemaphoreHandleDesc**
- typedef void ∗ **hipExternalSemaphore_t**
- typedef struct hipExternalSemaphoreSignalParams_st **hipExternalSemaphoreSignalParams**
- typedef struct hipExternalSemaphoreWaitParams_st hipExternalSemaphoreWaitParams
- typedef enum hipGLDeviceList **hipGLDeviceList**
- typedef enum hipGraphicsRegisterFlags **hipGraphicsRegisterFlags**
- typedef struct _hipGraphicsResource **hipGraphicsResource**
- typedef hipGraphicsResource ∗ **hipGraphicsResource_t**

Enumerations

- enum hipDeviceAttribute_t {
  hipDeviceAttributeMaxThreadsPerBlock, hipDeviceAttributeMaxBlockDimX, hipDeviceAttributeMaxBlockDimY, hipDeviceAttributeMaxBlockDimZ,
  hipDeviceAttributeMaxGridDimX, hipDeviceAttributeMaxGridDimY, hipDeviceAttributeMaxGridDimZ, hipDeviceAttributeMaxSharedMemoryPerBlock,
  hipDeviceAttributeTotalConstantMemory, hipDeviceAttributeWarpSize, hipDeviceAttributeMaxRegistersPerBlock, hipDeviceAttributeClockRate,
  hipDeviceAttributeMemoryClockRate, hipDeviceAttributeMemoryBusWidth, hipDeviceAttributeMultiprocessorCount, hipDeviceAttributeComputeMode,
  hipDeviceAttributeL2CacheSize, hipDeviceAttributeMaxThreadsPerMultiProcessor, hipDeviceAttributeComputeCapabilityMajor
  hipDeviceAttributeComputeCapabilityMinor,
  hipDeviceAttributeConcurrentKernels, hipDeviceAttributePciBusId, hipDeviceAttributePciDeviceId, hipDeviceAttributeMaxShare
  hipDeviceAttributeIsMultiGpuBoard, hipDeviceAttributeIntegrated, hipDeviceAttributeCooperativeLaunch, hipDeviceAttributeCooperativeMultiDeviceLaunch,
  hipDeviceAttributeMaxTexture1DWidth, hipDeviceAttributeMaxTexture2DWidth, hipDeviceAttributeMaxTexture2DHeight, hipDeviceAttributeMaxTexture3DWidth,
  hipDeviceAttributeMaxTexture3DHeight, hipDeviceAttributeMaxTexture3DDepth, hipDeviceAttributeHdpMemFlushCntl, hipDeviceAttributeHdpRegFlushCntl,
  hipDeviceAttributeMaxPitch, hipDeviceAttributeTextureAlignment, hipDeviceAttributeTexturePitchAlignment, hipDeviceAttributeKernelExecTimeout,
  hipDeviceAttributeCanMapHostMemory, hipDeviceAttributeEccEnabled, hipDeviceAttributeCooperativeMultiDeviceUnmatched
  hipDeviceAttributeCooperativeMultiDeviceUnmatchedGridDim,
  hipDeviceAttributeCooperativeMultiDeviceUnmatchedBlockDim, hipDeviceAttributeCooperativeMultiDeviceUnmatchedSharedM
  hipDeviceAttributeAsicRevision, hipDeviceAttributeManagedMemory,
  hipDeviceAttributeDirectManagedMemAccessFromHost, hipDeviceAttributeConcurrentManagedAccess,
  hipDeviceAttributePageableMemoryAccess, hipDeviceAttributePageableMemoryAccessUsesHostPageTables,
  hipDeviceAttributeCanUseStreamWaitValue }

- enum **hipComputeMode** { **hipComputeModeDefault** = 0, **hipComputeModeExclusive** = 1, **hipComputeModeProhibited** = 2, **hipComputeModeExclusiveProcess** = 3 }
- enum hipMemoryAdvise {
  hipMemAdviseSetReadMostly = 1, hipMemAdviseUnsetReadMostly = 2, hipMemAdviseSetPreferredLocation = 3, hipMemAdviseUnsetPreferredLocation = 4,
  hipMemAdviseSetAccessedBy = 5, hipMemAdviseUnsetAccessedBy = 6, hipMemAdviseSetCoarseGrain = 100, hipMemAdviseUnsetCoarseGrain = 101,
  hipMemAdviseSetReadMostly, **hipMemAdviseUnsetReadMostly**, **hipMemAdviseSetPreferredLocation**, **hipMemAdviseUnsetPreferredLocation**,
  hipMemAdviseSetAccessedBy, **hipMemAdviseUnsetAccessedBy**, **hipMemAdviseSetReadMostly**, **hipMemAdviseUnsetReadMostly**,
  hipMemAdviseSetPreferredLocation, **hipMemAdviseUnsetPreferredLocation**, **hipMemAdviseSetAccessedBy**, **hipMemAdviseUnsetAccessedBy** }
- enum hipMemRangeCoherencyMode { hipMemRangeCoherencyModeFineGrain = 0, hipMemRangeCoherencyModeCoarseGr
  = 1, hipMemRangeCoherencyModeIndeterminate = 2 }
- enum hipMemRangeAttribute {
  hipMemRangeAttributeReadMostly = 1, hipMemRangeAttributePreferredLocation = 2, hipMemRangeAttributeAccessedBy = 3, hipMemRangeAttributeLastPrefetchLocation = 4,
  hipMemRangeAttributeCoherencyMode = 100 }
- enum **hipJitOption** {
  hipJitOptionMaxRegisters = 0, **hipJitOptionThreadsPerBlock**, **hipJitOptionWallTime**, **hipJitOptionInfoLogBuffer**,
  hipJitOptionInfoLogBufferSizeBytes, **hipJitOptionErrorLogBuffer**, **hipJitOptionErrorLogBufferSizeBytes**, **hipJitOptionOptimizationLevel**,
  hipJitOptionTargetFromContext, **hipJitOptionTarget**, **hipJitOptionFallbackStrategy**, **hipJitOptionGenerateDebugInfo**,
  hipJitOptionLogVerbose, **hipJitOptionGenerateLineInfo**, **hipJitOptionCacheMode**, **hipJitOptionSm3xOpt**,
  hipJitOptionFastCompile, **hipJitOptionNumOptions** }
- enum hipFuncAttribute { **hipFuncAttributeMaxDynamicSharedMemorySize** = 8, **hipFuncAttributePreferredSharedMemoryCarveout** = 9, **hipFuncAttributeMax** }
- enum hipFuncCache_t { hipFuncCachePreferNone, hipFuncCachePreferShared, hipFuncCachePreferL1, hipFuncCachePreferEqual }
- enum hipSharedMemConfig { hipSharedMemBankSizeDefault, hipSharedMemBankSizeFourByte, hipSharedMemBankSizeEightByte }
- enum **hipExternalMemoryHandleType_enum** {
  hipExternalMemoryHandleTypeOpaqueFd = 1, **hipExternalMemoryHandleTypeOpaqueWin32** = 2, **hipExternalMemoryHandleTypeOpaqueWin32Kmt** = 3, **hipExternalMemoryHandleTypeD3D12Heap** = 4,
  hipExternalMemoryHandleTypeD3D12Resource = 5, **hipExternalMemoryHandleTypeD3D11Resource** = 6, **hipExternalMemoryHandleTypeD3D11ResourceKmt** = 7 }
- enum **hipExternalSemaphoreHandleType_enum** { **hipExternalSemaphoreHandleTypeOpaqueFd** = 1, **hipExternalSemaphoreHandleTypeOpaqueWin32** = 2, **hipExternalSemaphoreHandleTypeOpaqueWin32Kmt** = 3, **hipExternalSemaphoreHandleTypeD3D12Fence** = 4 }
- enum hipGLDeviceList { hipGLDeviceListAll = 1, hipGLDeviceListCurrentFrame = 2, hipGLDeviceListNextFrame = 3 }
- enum hipGraphicsRegisterFlags {
  hipGraphicsRegisterFlagsNone = 0, hipGraphicsRegisterFlagsReadOnly = 1, hipGraphicsRegisterFlagsWriteDiscard, hipGraphicsRegisterFlagsSurfaceLoadStore = 4,
  hipGraphicsRegisterFlagsTextureGather }

4.1.1 Detailed Description

4.1.2 Macro Definition Documentation

4.1.2.1 hipDeviceScheduleSpin

```
#define hipDeviceScheduleSpin  0x1
```

may consume more power.

Dedicate a CPU core to spin-wait. Provides lowest latency, but burns a CPU core and

4.1.2.2 hipDeviceScheduleYield

```
#define hipDeviceScheduleYield  0x2
```

power and is friendlier to other threads in the system.

Yield the CPU to the operating system when waiting. May increase latency, but lowers

4.1.2.3 hipEventDefault

```
#define hipEventDefault 0x0
```

Flags that can be used with hipEventCreateWithFlags:

Default flags

4.1.2.4 hipEventInterprocess

```
#define hipEventInterprocess 0x4
```

Event can support IPC.

Warning

- not supported in HIP.

4.1.2.5 hipEventReleaseToSystem

```
#define hipEventReleaseToSystem  0x80000000
```

< Use a device-scope release when recording this event. This flag is useful to obtain more precise timings of commands between events. The flag is a no-op on CUDA platforms.

4.1.2.6 hipHostMallocCoherent

```
#define hipHostMallocCoherent  0x40000000
```

allocation.

Allocate coherent memory. Overrides HIP_COHERENT_HOST_ALLOC for specific

4.1.2.7 hipHostMallocDefault

```
#define hipHostMallocDefault 0x0
```

Flags that can be used with hipHostMalloc.

< Use a system-scope release when recording this event. This flag is useful to make non-coherent host memory visible to the host. The flag is a no-op on CUDA platforms.

4.1.2.8 hipHostMallocMapped

```
#define hipHostMallocMapped  0x2
```

can be obtained with hipHostGetDevicePointer.

Map the allocation into the address space for the current device. The device pointer

4.1.2.9 hipHostMallocNonCoherent

```
#define hipHostMallocNonCoherent  0x80000000
```

allocation.

Allocate non-coherent memory. Overrides HIP_COHERENT_HOST_ALLOC for specific

4.1.2.10 hipHostRegisterDefault

```
#define hipHostRegisterDefault 0x0
```

Flags that can be used with hipHostRegister.

Memory is Mapped and Portable

4.1.2.11 hipHostRegisterMapped

```
#define hipHostRegisterMapped  0x2
```

can be obtained with hipHostGetDevicePointer.

Map the allocation into the address space for the current device. The device pointer

4.1.2.12 hipMallocSignalMemory

```
#define hipMallocSignalMemory 0x2
```

Memory represents a HSA signal.

4.1.2.13 hipMemAttachSingle

```
#define hipMemAttachSingle 0x04
```

the associated device

Memory can only be accessed by a single stream on

4.1.2.14 hipStreamDefault

```
#define hipStreamDefault  0x00
```

Flags that can be used with hipStreamCreateWithFlags.

Default stream creation flags. These are used with hipStreamCreate().

4.1.2.15 hipStreamNonBlocking

```
#define hipStreamNonBlocking 0x01
```

Stream does not implicitly synchronize with null stream

4.1.3 Typedef Documentation

4.1.3.1 dim3

```
typedef struct dim3 dim3
```

Struct for data in 3D

4.1.3.2 hipExternalSemaphoreWaitParams

```
typedef struct hipExternalSemaphoreWaitParams_st hipExternalSemaphoreWaitParams
```

External semaphore wait parameters, compatible with driver type

4.1.3.3 hipFuncAttribute

```
typedef enum hipFuncAttribute hipFuncAttribute
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

4.1.3.4 hipFuncCache_t

```
typedef enum hipFuncCache_t hipFuncCache_t
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

4.1.3.5 hipSharedMemConfig

```
typedef enum hipSharedMemConfig hipSharedMemConfig
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

4.1.4 Enumeration Type Documentation

4.1.4.1 hipDeviceAttribute_t

```
enum hipDeviceAttribute_t
```

Enumerator

hipDeviceAttributeMaxThreadsPerBlock	Maximum number of threads per block.
hipDeviceAttributeMaxBlockDimX	Maximum x-dimension of a block.
hipDeviceAttributeMaxBlockDimY	Maximum y-dimension of a block.
hipDeviceAttributeMaxBlockDimZ	Maximum z-dimension of a block.
hipDeviceAttributeMaxGridDimX	Maximum x-dimension of a grid.
hipDeviceAttributeMaxGridDimY	Maximum y-dimension of a grid.
hipDeviceAttributeMaxGridDimZ	Maximum z-dimension of a grid.
hipDeviceAttributeMaxSharedMemoryPerBlock	Maximum shared memory available per block in bytes.
hipDeviceAttributeTotalConstantMemory	Constant memory size in bytes.
hipDeviceAttributeWarpSize	Warp size in threads.
hipDeviceAttributeMaxRegistersPerBlock	Maximum number of 32-bit registers available to a thread block. This number is shared by all thread blocks simultaneously resident on a multiprocessor.
hipDeviceAttributeClockRate	Peak clock frequency in kilohertz.
hipDeviceAttributeMemoryClockRate	Peak memory clock frequency in kilohertz.
hipDeviceAttributeMemoryBusWidth	Global memory bus width in bits.
hipDeviceAttributeMultiprocessorCount	Number of multiprocessors on the device.
hipDeviceAttributeComputeMode	Compute mode that device is currently in.
hipDeviceAttributeL2CacheSize	Size of L2 cache in bytes. 0 if the device doesn't have L2 cache.

Enumerator

hipDeviceAttributeMaxThreadsPerMultiProcessor	Maximum resident threads per multiprocessor.
hipDeviceAttributeComputeCapabilityMajor	Major compute capability version number.
hipDeviceAttributeComputeCapabilityMinor	Minor compute capability version number.
hipDeviceAttributeConcurrentKernels	Device can possibly execute multiple kernels concurrently.
hipDeviceAttributePciBusId	PCI Bus ID.
hipDeviceAttributePciDeviceId	PCI Device ID.
hipDeviceAttributeMaxSharedMemoryPer↩ Multiprocessor	Maximum Shared Memory Per Multiprocessor.
hipDeviceAttributeIsMultiGpuBoard	Multiple GPU devices.
hipDeviceAttributeIntegrated	iGPU
hipDeviceAttributeCooperativeLaunch	Support cooperative launch.
hipDeviceAttributeCooperativeMultiDeviceLaunch	Support cooperative launch on multiple devices.
hipDeviceAttributeMaxTexture1DWidth	Maximum number of elements in 1D images.
hipDeviceAttributeMaxTexture2DWidth	Maximum dimension width of 2D images in image elements.
hipDeviceAttributeMaxTexture2DHeight	Maximum dimension height of 2D images in image elements.
hipDeviceAttributeMaxTexture3DWidth	Maximum dimension width of 3D images in image elements.
hipDeviceAttributeMaxTexture3DHeight	Maximum dimensions height of 3D images in image elements.
hipDeviceAttributeMaxTexture3DDepth	Maximum dimensions depth of 3D images in image elements.
hipDeviceAttributeHdpMemFlushCntl	Address of the HDP_MEM_COHERENCY_FLUSH_CNTL register.
hipDeviceAttributeHdpRegFlushCntl	Address of the HDP_REG_COHERENCY_FLUSH_CNTL register.
hipDeviceAttributeMaxPitch	Maximum pitch in bytes allowed by memory copies.
hipDeviceAttributeTextureAlignment	Alignment requirement for textures.
hipDeviceAttributeTexturePitchAlignment	Pitch alignment requirement for 2D texture references bound to pitched memory;.
hipDeviceAttributeKernelExecTimeout	Run time limit for kernels executed on the device.
hipDeviceAttributeCanMapHostMemory	Device can map host memory into device address space.
hipDeviceAttributeEccEnabled	Device has ECC support enabled.
hipDeviceAttributeCooperativeMultiDevice↩ UnmatchedFunc	Supports cooperative launch on multiple devices with unmatched functions
hipDeviceAttributeCooperativeMultiDevice↩ UnmatchedGridDim	Supports cooperative launch on multiple devices with unmatched grid dimensions
hipDeviceAttributeCooperativeMultiDevice↩ UnmatchedBlockDim	Supports cooperative launch on multiple devices with unmatched block dimensions
hipDeviceAttributeCooperativeMultiDevice↩ UnmatchedSharedMem	Supports cooperative launch on multiple devices with unmatched shared memories
hipDeviceAttributeAsicRevision	Revision of the GPU in this device.
hipDeviceAttributeManagedMemory	Device supports allocating managed memory on this system.
hipDeviceAttributeDirectManagedMemAccessFrom↩ Host	Host can directly access managed memory on the device without migration
hipDeviceAttributeConcurrentManagedAccess	Device can coherently access managed memory concurrently with the CPU

Enumerator

hipDeviceAttributePageableMemoryAccess	Device supports coherently accessing pageable memory without calling hipHostRegister on it
hipDeviceAttributePageableMemoryAccessUses↩ HostPageTables	Device accesses pageable memory via the host's page tables
hipDeviceAttributeCanUseStreamWaitValue	'1' if Device supports hipStreamWaitValue32() and hipStreamWaitValue64() , '0' otherwise.

4.1.4.2 hipFuncAttribute

```
enum hipFuncAttribute
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

4.1.4.3 hipFuncCache_t

```
enum hipFuncCache_t
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

Enumerator

hipFuncCachePreferNone	no preference for shared memory or L1 (default)
hipFuncCachePreferShared	prefer larger shared memory and smaller L1 cache
hipFuncCachePreferL1	prefer larger L1 cache and smaller shared memory
hipFuncCachePreferEqual	prefer equal size L1 cache and shared memory

4.1.4.4 hipGLDeviceList

```
enum hipGLDeviceList
```

Enumerator

hipGLDeviceListAll	All hip devices used by current OpenGL context.
hipGLDeviceListCurrentFrame	frame Hip devices used by current OpenGL context in current
hipGLDeviceListNextFrame	frame. Hip devices used by current OpenGL context in next

4.1.4.5 hipGraphicsRegisterFlags

```
enum hipGraphicsRegisterFlags
```

Enumerator

hipGraphicsRegisterFlagsReadOnly	HIP will not write to this registered resource.
hipGraphicsRegisterFlagsWriteDiscard	HIP will only write and will not read from this registered resource.
hipGraphicsRegisterFlagsSurfaceLoadStore	HIP will bind this resource to a surface.
hipGraphicsRegisterFlagsTextureGather	HIP will perform texture gather operations on this registered resource.

4.1.4.6 hipMemoryAdvise

```
enum hipMemoryAdvise
```

Enumerator

hipMemAdviseSetReadMostly	Data will mostly be read and only occassionally be written to
hipMemAdviseUnsetReadMostly	Undo the effect of hipMemAdviseSetReadMostly.
hipMemAdviseSetPreferredLocation	Set the preferred location for the data as the specified device
hipMemAdviseUnsetPreferredLocation	Clear the preferred location for the data.
hipMemAdviseSetAccessedBy	Data will be accessed by the specified device, so prevent page faults as much as possible
hipMemAdviseUnsetAccessedBy	Let HIP to decide on the page faulting policy for the specified device
hipMemAdviseSetCoarseGrain	The default memory model is fine-grain. That allows coherent operations between host and device, while executing kernels. The coarse-grain can be used for data that only needs to be coherent at dispatch boundaries for better performance
hipMemAdviseUnsetCoarseGrain	Restores cache coherency policy back to fine-grain.

4.1.4.7 hipMemRangeAttribute

```
enum hipMemRangeAttribute
```

Enumerator

hipMemRangeAttributeReadMostly	Whether the range will mostly be read and only occassionally be written to
hipMemRangeAttributePreferredLocation	The preferred location of the range.

Enumerator

hipMemRangeAttributeAccessedBy	Memory range has hipMemAdviseSetAccessedBy set for the specified device
hipMemRangeAttributeLastPrefetchLocation	prefetched The last location to where the range was
hipMemRangeAttributeCoherencyMode	Returns coherency mode hipMemRangeCoherencyMode for the range

4.1.4.8 hipMemRangeCoherencyMode

```
enum hipMemRangeCoherencyMode
```

Enumerator

hipMemRangeCoherencyModeFineGrain	Updates to memory with this attribute can be done coherently from all devices
hipMemRangeCoherencyModeCoarseGrain	Writes to memory with this attribute can be performed by a single device at a time
hipMemRangeCoherencyModeIndeterminate	Memory region queried contains subregions with both hipMemRangeCoherencyModeFineGrain and hipMemRangeCoherencyModeCoarseGrain attributes

4.1.4.9 hipSharedMemConfig

```
enum hipSharedMemConfig
```

Warning

On AMD devices and some Nvidia devices, these hints and controls are ignored.

Enumerator

hipSharedMemBankSizeDefault	The compiler selects a device-specific value for the banking.
hipSharedMemBankSizeFourByte	Shared mem is banked at 4-bytes intervals and performs best when adjacent threads access data 4 bytes apart.
hipSharedMemBankSizeEightByte	Shared mem is banked at 8-byte intervals and performs best when adjacent threads access data 4 bytes apart.

4.2 HIP API

Modules

- Initialization and Version
- Device Management
- Execution Control
- Error Handling
- Stream Management
- Event Management
- Memory Management
- PeerToPeer Device Memory Access
- Context Management
- Module Management
- Occupancy
- Profiler Control[Deprecated]
- Launch API to support the triple-chevron syntax
- Texture Management
- Runtime Compilation
- Callback Activity APIs
- Graph Management
- Interop

4.2.1 Detailed Description

Defines the HIP API. See the individual sections for more information.

4.3 Initialization and Version

Functions

- hipError_t hipInit (unsigned int flags)

  Explicitly initializes the HIP runtime.
- hipError_t hipDriverGetVersion (int ∗driverVersion)

  Returns the approximate HIP driver version.
- hipError_t hipRuntimeGetVersion (int ∗runtimeVersion)

  Returns the approximate HIP Runtime version.
- hipError_t hipDeviceGet (hipDevice_t ∗device, int ordinal)

  Returns a handle to a compute device.
- hipError_t hipDeviceComputeCapability (int ∗major, int ∗minor, hipDevice_t device)

  Returns the compute capability of the device.
- hipError_t hipDeviceGetName (char ∗name, int len, hipDevice_t device)

  Returns an identifer string for the device.
- hipError_t hipDeviceGetP2PAttribute (int ∗value, hipDeviceP2PAttr attr, int srcDevice, int dstDevice)

  Returns a value for attr of link between two devices.
- hipError_t hipDeviceGetPCIBusId (char ∗pciBusId, int len, int device)

  Returns a PCI Bus Id string for the device, overloaded to take int device ID.
- hipError_t hipDeviceGetByPCIBusId (int ∗device, const char ∗pciBusId)

  Returns a handle to a compute device.
- hipError_t hipDeviceTotalMem (size_t ∗bytes, hipDevice_t device)

  Returns the total amount of memory on the device.

4.3.1 Detailed Description

This section describes the initializtion and version functions of HIP runtime API.

4.3.2 Function Documentation

4.3.2.1 hipDeviceComputeCapability()

```
hipError_t hipDeviceComputeCapability (
            int ∗ major,
            int ∗ minor,
            hipDevice_t device )
```

Returns the compute capability of the device.

Parameters

out	*major*	
out	*minor*	
in	*device*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.2 hipDeviceGet()

```
hipError_t hipDeviceGet (
            hipDevice_t * device,
            int ordinal )
```

Returns a handle to a compute device.

Parameters

out	*device*	
in	*ordinal*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.3 hipDeviceGetByPCIBusId()

```
hipError_t hipDeviceGetByPCIBusId (
            int * device,
            const char * pciBusId )
```

Returns a handle to a compute device.

Parameters

out	*device*	handle
in	*PCI*	Bus ID

Returns

#hipSuccess, #hipErrorInavlidDevice, #hipErrorInvalidValue

4.3.2.4 hipDeviceGetName()

```
hipError_t hipDeviceGetName (
            char * name,
            int len,
            hipDevice_t device )
```

Returns an identifer string for the device.

Parameters

out	*name*	
in	*len*	
in	*device*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.5 hipDeviceGetP2PAttribute()

```
hipError_t hipDeviceGetP2PAttribute (
            int * value,
            hipDeviceP2PAttr attr,
            int srcDevice,
            int dstDevice )
```

Returns a value for attr of link between two devices.

Parameters

out	*value*	
in	*attr*	
in	*srcDevice*	
in	*dstDevice*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.6 hipDeviceGetPCIBusId()

```
hipError_t hipDeviceGetPCIBusId (
            char * pciBusId,
            int len,
            int device )
```

Returns a PCI Bus Id string for the device, overloaded to take int device ID.

Parameters

out	*pci↩ BusId*	
in	*len*	
in	*device*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.7 hipDeviceTotalMem()

```
hipError_t hipDeviceTotalMem (
            size_t * bytes,
            hipDevice_t device )
```

Returns the total amount of memory on the device.

Parameters

out	*bytes*	
in	*device*	

Returns

#hipSuccess, #hipErrorInavlidDevice

4.3.2.8 hipDriverGetVersion()

```
hipError_t hipDriverGetVersion (
            int * driverVersion )
```

Returns the approximate HIP driver version.

Parameters

out	*driverVersion*	

Returns

#hipSuccess, #hipErrorInavlidValue

Warning

The HIP feature set does not correspond to an exact CUDA SDK driver revision. This function always set ∗driverVersion to 4 as an approximation though HIP supports some features which were introduced in later CUDA SDK revisions. HIP apps code should not rely on the driver revision number here and should use arch feature flags to test device capabilities or conditional compilation.

See also

hipRuntimeGetVersion

4.3.2.9 hipInit()

```
hipError_t hipInit (
            unsigned int flags )
```

Explicitly initializes the HIP runtime.

Most HIP APIs implicitly initialize the HIP runtime. This API provides control over the timing of the initialization.

4.3.2.10 hipRuntimeGetVersion()

```
hipError_t hipRuntimeGetVersion (
            int * runtimeVersion )
```

Returns the approximate HIP Runtime version.

Parameters

out	*runtimeVersion*	

Returns

#hipSuccess, #hipErrorInavlidValue

Warning

The version definition of HIP runtime is different from CUDA. On AMD platform, the function returns HIP runtime version, while on NVIDIA platform, it returns CUDA runtime version. And there is no mapping/correlation between HIP version and CUDA version.

See also

hipDriverGetVersion

4.4 Device Management

Functions

- hipError_t hipDeviceSynchronize (void)

  Waits on all active streams on current device.
- hipError_t hipDeviceReset (void)

  The state of current device is discarded and updated to a fresh state.
- hipError_t hipSetDevice (int deviceId)

  Set default device to be used for subsequent hip API calls from this thread.
- hipError_t hipGetDevice (int ∗deviceId)

  Return the default device id for the calling host thread.
- hipError_t hipGetDeviceCount (int ∗count)

  Return number of compute-capable devices.
- hipError_t hipDeviceGetAttribute (int ∗pi, hipDeviceAttribute_t attr, int deviceId)

  Query for a specific device attribute.
- hipError_t hipGetDeviceProperties (hipDeviceProp_t ∗prop, int deviceId)

  Returns device properties.
- hipError_t hipDeviceSetCacheConfig (hipFuncCache_t cacheConfig)

  Set L1/Shared cache partition.
- hipError_t hipDeviceGetCacheConfig (hipFuncCache_t ∗cacheConfig)

  Set Cache configuration for a specific function.
- hipError_t hipDeviceGetLimit (size_t ∗pValue, enum hipLimit_t limit)

  Get Resource limits of current device.
- hipError_t hipDeviceGetSharedMemConfig (hipSharedMemConfig ∗pConfig)

  Returns bank width of shared memory for current device.
- hipError_t hipGetDeviceFlags (unsigned int ∗flags)

  Gets the flags set for current device.
- hipError_t hipDeviceSetSharedMemConfig (hipSharedMemConfig config)

  The bank width of shared memory on current device is set.
- hipError_t hipSetDeviceFlags (unsigned flags)

  The current device behavior is changed according the flags passed.
- hipError_t hipChooseDevice (int ∗device, const hipDeviceProp_t ∗prop)

  Device which matches hipDeviceProp_t is returned.
- hipError_t hipExtGetLinkTypeAndHopCount (int device1, int device2, uint32_t ∗linktype, uint32_t ∗hopcount)

  Returns the link type and hop count between two devices.
- hipError_t hipIpcGetMemHandle (hipIpcMemHandle_t ∗handle, void ∗devPtr)

  Gets an interprocess memory handle for an existing device memory allocation.
- hipError_t hipIpcOpenMemHandle (void ∗∗devPtr, hipIpcMemHandle_t handle, unsigned int flags)

  Opens an interprocess memory handle exported from another process and returns a device pointer usable in the local process.
- hipError_t hipIpcCloseMemHandle (void ∗devPtr)

  Close memory mapped with hipIpcOpenMemHandle.
- hipError_t hipIpcGetEventHandle (hipIpcEventHandle_t ∗handle, hipEvent_t event)

  Gets an opaque interprocess handle for an event.
- hipError_t hipIpcOpenEventHandle (hipEvent_t ∗event, hipIpcEventHandle_t handle)

  Opens an interprocess event handles.

4.4.1 Detailed Description

This section describes the device management functions of HIP runtime API.

4.4.2 Function Documentation

4.4.2.1 hipChooseDevice()

```
hipError_t hipChooseDevice (
            int * device,
            const hipDeviceProp_t * prop )
```

Device which matches hipDeviceProp_t is returned.

Parameters

out	*device*	ID
in	*device*	properties pointer

Returns

#hipSuccess, #hipErrorInvalidValue

4.4.2.2 hipDeviceGetAttribute()

```
hipError_t hipDeviceGetAttribute (
            int * pi,
            hipDeviceAttribute_t attr,
            int deviceId )
```

Query for a specific device attribute.

Parameters

out	*pi*	pointer to value to return
in	*attr*	attribute to query
in	*device↵ Id*	which device to query for information

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

4.4.2.3 hipDeviceGetCacheConfig()

```
hipError_t hipDeviceGetCacheConfig (
            hipFuncCache_t * cacheConfig )
```

Set Cache configuration for a specific function.

Parameters

in	*cacheConfig*	

Returns

#hipSuccess, #hipErrorNotInitialized Note: AMD devices and some Nvidia GPUS do not support reconfigurable cache. This hint is ignored on those architectures.

4.4.2.4 hipDeviceGetLimit()

```
hipError_t hipDeviceGetLimit (
            size_t * pValue,
            enum hipLimit_t limit )
```

Get Resource limits of current device.

Parameters

out	*pValue*	
in	*limit*	

Returns

#hipSuccess, #hipErrorUnsupportedLimit, #hipErrorInvalidValue Note: Currently, only hipLimitMallocHeap↵
Size is available

4.4.2.5 hipDeviceGetSharedMemConfig()

```
hipError_t hipDeviceGetSharedMemConfig (
            hipSharedMemConfig * pConfig )
```

Returns bank width of shared memory for current device.

Parameters

out	*pConfig*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

Note: AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

4.4.2.6 hipDeviceReset()

```
hipError_t hipDeviceReset (
            void  )
```

The state of current device is discarded and updated to a fresh state.

Calling this function deletes all streams created, memory allocated, kernels running, events created. Make sure that no other thread is using the device or streams, memory, kernels, events associated with the current device.

Returns

#hipSuccess

See also

hipDeviceSynchronize

4.4.2.7 hipDeviceSetCacheConfig()

```
hipError_t hipDeviceSetCacheConfig (
            hipFuncCache_t cacheConfig )
```

Set L1/Shared cache partition.

Parameters

in	*cacheConfig*	

Returns

#hipSuccess, #hipErrorNotInitialized Note: AMD devices and some Nvidia GPUS do not support reconfigurable cache. This hint is ignored on those architectures.

4.4.2.8 hipDeviceSetSharedMemConfig()

```
hipError_t hipDeviceSetSharedMemConfig (
            hipSharedMemConfig config )
```

The bank width of shared memory on current device is set.

Parameters

in	*config*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

Note: AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

4.4.2.9 hipDeviceSynchronize()

```
hipError_t hipDeviceSynchronize (
            void   )
```

Waits on all active streams on current device.

When this command is invoked, the host thread gets blocked until all the commands associated with streams associated with the device. HIP does not support multiple blocking modes (yet!).

Returns

#hipSuccess

See also

hipSetDevice, hipDeviceReset

4.4.2.10 hipExtGetLinkTypeAndHopCount()

```
hipError_t hipExtGetLinkTypeAndHopCount (
            int device1,
            int device2,
            uint32_t * linktype,
            uint32_t * hopcount )
```

Returns the link type and hop count between two devices.

Parameters

in	*device1*	Ordinal for device1
in	*device2*	Ordinal for device2
out	*linktype*	Returns the link type (See hsa_amd_link_info_type_t) between the two devices
out	*hopcount*	Returns the hop count between the two devices

Queries and returns the HSA link type and the hop count between the two specified devices.

Returns

#hipSuccess, #hipInvalidDevice, #hipErrorRuntimeOther

4.4.2.11 hipGetDevice()

```
hipError_t hipGetDevice (
            int * deviceId )
```

Return the default device id for the calling host thread.

Parameters

out	*device*	*device is written with the default device

HIP maintains an default device for each thread using thread-local-storage. This device is used implicitly for HIP runtime APIs called by this thread. hipGetDevice returns in `* device` the default device for the calling host thread.

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

hipSetDevice, hipGetDevicesizeBytes

4.4.2.12 hipGetDeviceCount()

```
hipError_t hipGetDeviceCount (
            int * count )
```

Return number of compute-capable devices.

Parameters

[output]	count Returns number of compute-capable devices.

Returns

#hipSuccess, #hipErrorNoDevice

Returns in `*count` the number of devices that have ability to run compute commands. If there are no such devices, then hipGetDeviceCount will return #hipErrorNoDevice. If 1 or more devices can be found, then hipGetDeviceCount returns #hipSuccess.

4.4.2.13 hipGetDeviceFlags()

```
hipError_t hipGetDeviceFlags (
            unsigned int * flags )
```

Gets the flags set for current device.

Parameters

out	*flags*	

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

4.4.2.14 hipGetDeviceProperties()

```
hipError_t hipGetDeviceProperties (
            hipDeviceProp_t * prop,
            int deviceId )
```

Returns device properties.

Parameters

out	*prop*	written with device properties
in	*device←↩ Id*	which device to query for information

Returns

#hipSuccess, #hipErrorInvalidDevice

Populates hipGetDeviceProperties with information for the specified device.

4.4.2.15 hipIpcCloseMemHandle()

```
hipError_t hipIpcCloseMemHandle (
            void * devPtr )
```

Close memory mapped with hipIpcOpenMemHandle.

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

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

Parameters

devPtr	- Device pointer returned by hipIpcOpenMemHandle

Returns

hipSuccess, hipErrorMapFailed, hipErrorInvalidHandle,

4.4.2.16 hipIpcGetEventHandle()

```
hipError_t hipIpcGetEventHandle (
            hipIpcEventHandle_t * handle,
            hipEvent_t event )
```

Gets an opaque interprocess handle for an event.

This opaque handle may be copied into other processes and opened with cudaIpcOpenEventHandle. Then cuda↩
EventRecord, cudaEventSynchronize, cudaStreamWaitEvent and cudaEventQuery may be used in either process. Operations on the imported event after the exported event has been freed with hipEventDestroy will result in undefined behavior.

Parameters

out	*handle*	Pointer to cudaIpcEventHandle to return the opaque event handle
in	*event*	Event allocated with cudaEventInterprocess and cudaEventDisableTiming flags

Returns

#hipSuccess, #hipErrorInvalidConfiguration, #hipErrorInvalidValue

4.4.2.17 hipIpcGetMemHandle()

```
hipError_t hipIpcGetMemHandle (
            hipIpcMemHandle_t * handle,
            void * devPtr )
```

Gets an interprocess memory handle for an existing device memory allocation.

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

If a region of memory is freed with hipFree and a subsequent call to hipMalloc returns memory with the same device address, hipIpcGetMemHandle will return a unique handle for the new memory.

Parameters

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

Returns

hipSuccess, hipErrorInvalidHandle, hipErrorOutOfMemory, hipErrorMapFailed,

4.4.2.18 hipIpcOpenEventHandle()

```
hipError_t hipIpcOpenEventHandle (
            hipEvent_t * event,
            hipIpcEventHandle_t handle )
```

Opens an interprocess event handles.

Opens an interprocess event handle exported from another process with cudaIpcGetEventHandle. The returned hipEvent_t behaves like a locally created event with the hipEventDisableTiming flag specified. This event need be freed with hipEventDestroy. Operations on the imported event after the exported event has been freed with hip↩ EventDestroy will result in undefined behavior. If the function is called within the same process where handle is returned by hipIpcGetEventHandle, it will return hipErrorInvalidContext.

Parameters

out	*event*	Pointer to hipEvent_t to return the event
in	*handle*	The opaque interprocess handle to open

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidContext

4.4.2.19 hipIpcOpenMemHandle()

```
hipError_t hipIpcOpenMemHandle (
            void ** devPtr,
            hipIpcMemHandle_t handle,
            unsigned int flags )
```

Opens an interprocess memory handle exported from another process and returns a device pointer usable in the local process.

Maps memory exported from another process with hipIpcGetMemHandle into the current device address space. For contexts on different devices hipIpcOpenMemHandle can attempt to enable peer access between the devices as if the user called hipDeviceEnablePeerAccess. This behavior is controlled by the hipIpcMemLazyEnablePeerAccess flag. hipDeviceCanAccessPeer can determine if a mapping is possible.

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

Memory returned from hipIpcOpenMemHandle must be freed with hipIpcCloseMemHandle.

Calling hipFree on an exported memory region before calling hipIpcCloseMemHandle in the importing context will result in undefined behavior.

Parameters

devPtr	- Returned device pointer
handle	- hipIpcMemHandle to open
flags	- Flags for this operation. Must be specified as hipIpcMemLazyEnablePeerAccess

Returns

hipSuccess, hipErrorMapFailed, hipErrorInvalidHandle, hipErrorTooManyPeers

Note

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

4.4.2.20 hipSetDevice()

```
hipError_t hipSetDevice (
            int deviceId )
```

Set default device to be used for subsequent hip API calls from this thread.

Parameters

in	*device↵ Id*	Valid device in range 0...hipGetDeviceCount().

Sets `device` as the default device for the calling host thread. Valid device id's are 0... (hipGetDeviceCount()-1).

Many HIP APIs implicitly use the "default device" :

- Any device memory subsequently allocated from this host thread (using hipMalloc) will be allocated on device.
- Any streams or events created from this host thread will be associated with device.
- Any kernels launched from this host thread (using hipLaunchKernel) will be executed on device (unless a specific stream is specified, in which case the device associated with that stream will be used).

This function may be called from any host thread. Multiple host threads may use the same device. This function does no synchronization with the previous or new device, and has very little runtime overhead. Applications can use hipSetDevice to quickly switch the default device before making a HIP runtime call which uses the default device.

The default device is stored in thread-local-storage for each thread. Thread-pool implementations may inherit the default device of the previous thread. A good practice is to always call hipSetDevice at the start of HIP coding sequency to establish a known standard device.

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorDeviceAlreadyInUse

See also

hipGetDevice, hipGetDeviceCount

4.4.2.21 hipSetDeviceFlags()

```
hipError_t hipSetDeviceFlags (
            unsigned flags )
```

The current device behavior is changed according the flags passed.

Parameters

in	*flags*	The schedule flags impact how HIP waits for the completion of a command running on a device. hipDeviceScheduleSpin : HIP runtime will actively spin in the thread which submitted the work until the command completes. This offers the lowest latency, but will consume a CPU core and may increase power. hipDeviceScheduleYield : The HIP runtime will yield the CPU to system so that other tasks can use it. This may increase latency to detect the completion but will consume less power and is friendlier to other tasks in the system. hipDeviceScheduleBlockingSync : On ROCm platform, this is a synonym for hipDeviceScheduleYield. hipDeviceScheduleAuto : Use a hueristic to select between Spin and Yield modes. If the number of HIP contexts is greater than the number of logical processors in the system, use Spin scheduling. Else use Yield scheduling.

hipDeviceMapHost : Allow mapping host memory. On ROCM, this is always allowed and the flag is ignored. hip↩
DeviceLmemResizeToMax :

Warning

ROCm silently ignores this flag.

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorSetOnActiveProcess

4.5 Execution Control

Functions

- hipError_t hipFuncSetAttribute (const void ∗func, hipFuncAttribute attr, int value)

  Set attribute for a specific function.
- hipError_t hipFuncSetCacheConfig (const void ∗func, hipFuncCache_t config)

  Set Cache configuration for a specific function.
- hipError_t hipFuncSetSharedMemConfig (const void ∗func, hipSharedMemConfig config)

  Set shared memory configuation for a specific function.

4.5.1 Detailed Description

This section describes the execution control functions of HIP runtime API.

4.5.2 Function Documentation

4.5.2.1 hipFuncSetAttribute()

```
hipError_t hipFuncSetAttribute (
             const void * func,
             hipFuncAttribute attr,
             int value )
```

Set attribute for a specific function.

Parameters

in	*func;*	
in	*attr;*	
in	*value;*	

Returns

#hipSuccess, #hipErrorInvalidDeviceFunction, #hipErrorInvalidValue

Note: AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

4.5.2.2 hipFuncSetCacheConfig()

```
hipError_t hipFuncSetCacheConfig (
             const void * func,
             hipFuncCache_t config )
```

Set Cache configuration for a specific function.

Parameters

in	*config;*	

Returns

#hipSuccess, #hipErrorNotInitialized Note: AMD devices and some Nvidia GPUS do not support reconfigurable cache. This hint is ignored on those architectures.

4.5.2.3 hipFuncSetSharedMemConfig()

```
hipError_t hipFuncSetSharedMemConfig (
            const void * func,
            hipSharedMemConfig config )
```

Set shared memory configuation for a specific function.

Parameters

in	*func*	
in	*config*	

Returns

#hipSuccess, #hipErrorInvalidDeviceFunction, #hipErrorInvalidValue

Note: AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

4.6 Error Handling

Functions

- hipError_t hipGetLastError (void)

  Return last error returned by any HIP runtime API call and resets the stored error code to #hipSuccess.
- hipError_t hipPeekAtLastError (void)

  Return last error returned by any HIP runtime API call.
- const char ∗ hipGetErrorName (hipError_t hip_error)

  Return name of the specified error code in text form.
- const char ∗ hipGetErrorString (hipError_t hipError)

  Return handy text string message to explain the error which occurred.

4.6.1 Detailed Description

This section describes the error handling functions of HIP runtime API.

4.6.2 Function Documentation

4.6.2.1 hipGetErrorName()

```
const char* hipGetErrorName (
             hipError_t hip_error )
```

Return name of the specified error code in text form.

Parameters

hip_error	Error code to convert to name.

Returns

const char pointer to the NULL-terminated error name

See also

hipGetErrorString, hipGetLastError, hipPeakAtLastError, hipError_t

4.6.2.2 hipGetErrorString()

```
const char* hipGetErrorString (
             hipError_t hipError )
```

Return handy text string message to explain the error which occurred.

Parameters

hipError	Error code to convert to string.

Returns

const char pointer to the NULL-terminated error string

Warning

: on HCC, this function returns the name of the error (same as hipGetErrorName)

See also

hipGetErrorName, hipGetLastError, hipPeakAtLastError, hipError_t

4.6.2.3 hipGetLastError()

```
hipError_t hipGetLastError (
            void )
```

Return last error returned by any HIP runtime API call and resets the stored error code to #hipSuccess.

Returns

return code from last HIP called from the active host thread

Returns the last error that has been returned by any of the runtime calls in the same host thread, and then resets the saved error to #hipSuccess.

See also

hipGetErrorString, hipGetLastError, hipPeakAtLastError, hipError_t

4.6.2.4 hipPeekAtLastError()

```
hipError_t hipPeekAtLastError (
            void )
```

Return last error returned by any HIP runtime API call.

Returns

#hipSuccess

Returns the last error that has been returned by any of the runtime calls in the same host thread. Unlike hipGet↩ LastError, this function does not reset the saved error code.

See also

hipGetErrorString, hipGetLastError, hipPeakAtLastError, hipError_t

4.7 Stream Management

Typedefs

- typedef void(∗ hipStreamCallback_t) (hipStream_t stream, hipError_t status, void ∗userData)

Functions

- hipError_t hipStreamCreate (hipStream_t ∗stream)

  Create an asynchronous stream.
- hipError_t hipStreamCreateWithFlags (hipStream_t ∗stream, unsigned int flags)

  Create an asynchronous stream.
- hipError_t hipStreamCreateWithPriority (hipStream_t ∗stream, unsigned int flags, int priority)

  Create an asynchronous stream with the specified priority.
- hipError_t hipDeviceGetStreamPriorityRange (int ∗leastPriority, int ∗greatestPriority)

  Returns numerical values that correspond to the least and greatest stream priority.
- hipError_t hipStreamDestroy (hipStream_t stream)

  Destroys the specified stream.
- hipError_t hipStreamQuery (hipStream_t stream)

  Return #hipSuccess if all of the operations in the specified `stream` have completed, or #hipErrorNotReady if not.
- hipError_t hipStreamSynchronize (hipStream_t stream)

  Wait for all commands in stream to complete.
- hipError_t hipStreamWaitEvent (hipStream_t stream, hipEvent_t event, unsigned int flags)

  Make the specified compute stream wait for an event.
- hipError_t hipStreamGetFlags (hipStream_t stream, unsigned int ∗flags)

  Return flags associated with this stream.
- hipError_t hipStreamGetPriority (hipStream_t stream, int ∗priority)

  Query the priority of a stream.
- hipError_t hipExtStreamCreateWithCUMask (hipStream_t ∗stream, uint32_t cuMaskSize, const uint32_↩ t ∗cuMask)

  Create an asynchronous stream with the specified CU mask.
- hipError_t hipExtStreamGetCUMask (hipStream_t stream, uint32_t cuMaskSize, uint32_t ∗cuMask)

  Get CU mask associated with an asynchronous stream.
- hipError_t hipStreamAddCallback (hipStream_t stream, hipStreamCallback_t callback, void ∗userData, unsigned int flags)

  Adds a callback to be called on the host after all currently enqueued items in the stream have completed. For each hipStreamAddCallback call, a callback will be executed exactly once. The callback will block later work in the stream until it is finished.
- hipError_t hipStreamWaitValue32 (hipStream_t stream, void ∗ptr, uint32_t value, unsigned int flags, uint32_t mask __dparm(0xFFFFFFFF))

  Enqueues a wait command to the stream.[BETA].
- hipError_t hipStreamWaitValue64 (hipStream_t stream, void ∗ptr, uint64_t value, unsigned int flags, uint64_t mask __dparm(0xFFFFFFFFFFFFFFFF))

  Enqueues a wait command to the stream.[BETA].
- hipError_t hipStreamWriteValue32 (hipStream_t stream, void ∗ptr, uint32_t value, unsigned int flags)

  Enqueues a write command to the stream.[BETA].
- hipError_t hipStreamWriteValue64 (hipStream_t stream, void ∗ptr, uint64_t value, unsigned int flags)

  Enqueues a write command to the stream.[BETA].

4.7.1 Detailed Description

This section describes the stream management functions of HIP runtime API. The following Stream APIs are not (yet) supported in HIP:

- hipStreamAttachMemAsync is a nop

This section describes Stream Memory Wait and Write functions of HIP runtime API.

4.7.2 Typedef Documentation

4.7.2.1 hipStreamCallback_t

```
typedef void(* hipStreamCallback_t) (hipStream_t stream, hipError_t status, void *userData)
```

Stream CallBack struct

4.7.3 Function Documentation

4.7.3.1 hipDeviceGetStreamPriorityRange()

```
hipError_t hipDeviceGetStreamPriorityRange (
            int * leastPriority,
            int * greatestPriority )
```

Returns numerical values that correspond to the least and greatest stream priority.

Parameters

in,out	*leastPriority*	pointer in which value corresponding to least priority is returned.
in,out	*greatestPriority*	pointer in which value corresponding to greatest priority is returned.

Returns in ∗leastPriority and ∗greatestPriority the numerical values that correspond to the least and greatest stream priority respectively. Stream priorities follow a convention where lower numbers imply greater priorities. The range of meaningful stream priorities is given by [∗greatestPriority, ∗leastPriority]. If the user attempts to create a stream with a priority value that is outside the the meaningful range as specified by this API, the priority is automatically clamped to within the valid range.

4.7.3.2 hipExtStreamCreateWithCUMask()

```
hipError_t hipExtStreamCreateWithCUMask (
            hipStream_t * stream,
            uint32_t cuMaskSize,
            const uint32_t * cuMask )
```

Create an asynchronous stream with the specified CU mask.

Parameters

in,out	*stream*	Pointer to new stream
in	*cuMaskSize*	Size of CU mask bit array passed in.
in	*cuMask*	Bit-vector representing the CU mask. Each active bit represents using one CU. The first 32 bits represent the first 32 CUs, and so on. If its size is greater than physical CU number (i.e., multiProcessorCount member of hipDeviceProp_t), the extra elements are ignored. It is user's responsibility to make sure the input is meaningful.

Returns

#hipSuccess, #hipErrorInvalidHandle, #hipErrorInvalidValue

Create a new asynchronous stream with the specified CU mask. `stream` returns an opaque handle that can be used to reference the newly created stream in subsequent hipStream∗ commands. The stream is allocated on the heap and will remain allocated even if the handle goes out-of-scope. To release the memory used by the stream, application must call hipStreamDestroy.

See also

hipStreamCreate, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy

4.7.3.3 hipExtStreamGetCUMask()

```
hipError_t hipExtStreamGetCUMask (
            hipStream_t stream,
            uint32_t cuMaskSize,
            uint32_t * cuMask )
```

Get CU mask associated with an asynchronous stream.

Parameters

in	*stream*	stream to be queried
in	*cuMaskSize*	number of the block of memories (uint32_t ∗) allocated by user
out	*cuMask*	Pointer to a pre-allocated block of memories (uint32_t ∗) in which the stream's CU mask is returned. The CU mask is returned in a chunck of 32 bits where each active bit represents one active CU

Returns

#hipSuccess, #hipErrorInvalidHandle, #hipErrorInvalidValue

See also

hipStreamCreate, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy

4.7.3.4 hipStreamAddCallback()

```
hipError_t hipStreamAddCallback (
            hipStream_t stream,
            hipStreamCallback_t callback,
            void * userData,
            unsigned int flags )
```

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

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidHandle, #hipErrorNotSupported

See also

hipStreamCreate, hipStreamCreateWithFlags, hipStreamQuery, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy, hipStreamCreateWithPriority

4.7.3.5 hipStreamCreate()

```
hipError_t hipStreamCreate (
            hipStream_t * stream )
```

Create an asynchronous stream.

Parameters

in,out	*stream*	Valid pointer to hipStream_t. This function writes the memory with the newly created stream.

Returns

#hipSuccess, #hipErrorInvalidValue

Create a new asynchronous stream. `stream` returns an opaque handle that can be used to reference the newly created stream in subsequent hipStream* commands. The stream is allocated on the heap and will remain allocated even if the handle goes out-of-scope. To release the memory used by the stream, applicaiton must call hipStream↩ Destroy.

Returns

#hipSuccess, #hipErrorInvalidValue

See also

hipStreamCreateWithFlags, hipStreamCreateWithPriority, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy

4.7.3.6 hipStreamCreateWithFlags()

```
hipError_t hipStreamCreateWithFlags (
            hipStream_t * stream,
            unsigned int flags )
```

Create an asynchronous stream.

Parameters

in,out	*stream*	Pointer to new stream
in	*flags*	to control stream creation.

Returns

#hipSuccess, #hipErrorInvalidValue

Create a new asynchronous stream. `stream` returns an opaque handle that can be used to reference the newly created stream in subsequent hipStream* commands. The stream is allocated on the heap and will remain allocated even if the handle goes out-of-scope. To release the memory used by the stream, applicaiton must call hipStream↩ Destroy. Flags controls behavior of the stream. See hipStreamDefault, hipStreamNonBlocking.

See also

hipStreamCreate, hipStreamCreateWithPriority, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy

4.7.3.7 hipStreamCreateWithPriority()

```
hipError_t hipStreamCreateWithPriority (
            hipStream_t * stream,
            unsigned int flags,
            int priority )
```

Create an asynchronous stream with the specified priority.

Parameters

in,out	*stream*	Pointer to new stream
in	*flags*	to control stream creation.
in	*priority*	of the stream. Lower numbers represent higher priorities.

Returns

#hipSuccess, #hipErrorInvalidValue

Create a new asynchronous stream with the specified priority. `stream` returns an opaque handle that can be used to reference the newly created stream in subsequent hipStream∗ commands. The stream is allocated on the heap and will remain allocated even if the handle goes out-of-scope. To release the memory used by the stream, applicaiton must call hipStreamDestroy. Flags controls behavior of the stream. See hipStreamDefault, hipStreamNonBlocking.

See also

hipStreamCreate, hipStreamSynchronize, hipStreamWaitEvent, hipStreamDestroy

4.7.3.8 hipStreamDestroy()

```
hipError_t hipStreamDestroy (
            hipStream_t stream )
```

Destroys the specified stream.

Parameters

in,out	*stream*	Valid pointer to hipStream_t. This function writes the memory with the newly created stream.

Returns

#hipSuccess #hipErrorInvalidHandle

Destroys the specified stream.
If commands are still executing on the specified stream, some may complete execution before the queue is deleted.
The queue may be destroyed while some commands are still inflight, or may wait for all commands queued to the stream before destroying it.

See also

hipStreamCreate, hipStreamCreateWithFlags, hipStreamCreateWithPriority, hipStreamQuery, hipStreamWaitEvent, hipStreamSynchronize

4.7.3.9 hipStreamGetFlags()

```
hipError_t hipStreamGetFlags (
            hipStream_t stream,
            unsigned int * flags )
```

Return flags associated with this stream.

Parameters

`in`	*stream*	stream to be queried
`in,out`	*flags*	Pointer to an unsigned integer in which the stream's flags are returned

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidHandle

#hipSuccess #hipErrorInvalidValue #hipErrorInvalidHandle

Return flags associated with this stream in `*flags`.

See also

hipStreamCreateWithFlags

4.7.3.10 hipStreamGetPriority()

```
hipError_t hipStreamGetPriority (
            hipStream_t stream,
            int * priority )
```

Query the priority of a stream.

Parameters

`in`	*stream*	stream to be queried
`in,out`	*priority*	Pointer to an unsigned integer in which the stream's priority is returned

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidHandle

#hipSuccess #hipErrorInvalidValue #hipErrorInvalidHandle

Query the priority of a stream. The priority is returned in in priority.

See also

hipStreamCreateWithFlags

4.7.3.11 hipStreamQuery()

```
hipError_t hipStreamQuery (
            hipStream_t stream )
```

Return #hipSuccess if all of the operations in the specified `stream` have completed, or #hipErrorNotReady if not.

Parameters

`in`	*stream*	stream to query

Returns

#hipSuccess, #hipErrorNotReady, #hipErrorInvalidHandle

This is thread-safe and returns a snapshot of the current state of the queue. However, if other host threads are sending work to the stream, the status may change immediately after the function is called. It is typically used for debug.

See also

hipStreamCreate, hipStreamCreateWithFlags, hipStreamCreateWithPriority, hipStreamWaitEvent, hipStreamSynchronize, hipStreamDestroy

4.7.3.12 hipStreamSynchronize()

```
hipError_t hipStreamSynchronize (
             hipStream_t stream )
```

Wait for all commands in stream to complete.

Parameters

in	*stream*	stream identifier.

Returns

#hipSuccess, #hipErrorInvalidHandle

This command is host-synchronous : the host will block until the specified stream is empty.
This command follows standard null-stream semantics. Specifically, specifying the null stream will cause the command to wait for other streams on the same device to complete all pending operations.
This command honors the hipDeviceLaunchBlocking flag, which controls whether the wait is active or blocking.

See also

hipStreamCreate, hipStreamCreateWithFlags, hipStreamCreateWithPriority, hipStreamWaitEvent, hipStreamDestroy

4.7.3.13 hipStreamWaitEvent()

```
hipError_t hipStreamWaitEvent (
             hipStream_t stream,
             hipEvent_t event,
             unsigned int flags )
```

Make the specified compute stream wait for an event.

Parameters

in	*stream*	stream to make wait.
in	*event*	event to wait on
in	*flags*	control operation [must be 0]

Returns

#hipSuccess, #hipErrorInvalidHandle

This function inserts a wait operation into the specified stream. All future work submitted to `stream` will wait until `event` reports completion before beginning execution.
This function only waits for commands in the current stream to complete. Notably,, this function does not impliciy wait for commands in the default stream to complete, even if the specified stream is created with hipStreamNonBlocking = 0.

See also

hipStreamCreate, hipStreamCreateWithFlags, hipStreamCreateWithPriority, hipStreamSynchronize, hipStreamDestroy

4.7.3.14 hipStreamWaitValue32()

```
hipError_t hipStreamWaitValue32 (
            hipStream_t stream,
            void * ptr,
            uint32_t value,
            unsigned int flags,
            uint32_t mask  __dparm0xFFFFFFFF )
```

Enqueues a wait command to the stream.[BETA].

Parameters

in	*stream*	- Stream identifier
in	*ptr*	- Pointer to memory object allocated using 'hipMallocSignalMemory' flag
in	*value*	- Value to be used in compare operation
in	*flags*	- Defines the compare operation, supported values are hipStreamWaitValueGte hipStreamWaitValueEq, hipStreamWaitValueAnd and hipStreamWaitValueNor
in	*mask*	- Mask to be applied on value at memory before it is compared with value, default value is set to enable every bit

Returns

#hipSuccess, #hipErrorInvalidValue

Enqueues a wait command to the stream, all operations enqueued on this stream after this, will not execute until the defined wait condition is true.
hipStreamWaitValueGte: waits until ∗ptr&mask >= value hipStreamWaitValueEq : waits until ∗ptr&mask == value hipStreamWaitValueAnd: waits until ((∗ptr&mask) & value) != 0 hipStreamWaitValueNor: waits until ∼((∗ptr&mask) | (value&mask)) != 0

Note

when using 'hipStreamWaitValueNor', mask is applied on both 'value' and '∗ptr'.

Support for hipStreamWaitValue32 can be queried using 'hipDeviceGetAttribute()' and 'hipDeviceAttribute↩ CanUseStreamWaitValue' flag.

Warning

This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

See also

hipExtMallocWithFlags, hipFree, hipStreamWaitValue64, hipStreamWriteValue64, hipStreamWriteValue32, hipDeviceGetAttribute

4.7.3.15 hipStreamWaitValue64()

```
hipError_t hipStreamWaitValue64 (
            hipStream_t stream,
            void * ptr,
            uint64_t value,
            unsigned int flags,
            uint64_t mask  __dparm0xFFFFFFFFFFFFFFFF )
```

Enqueues a wait command to the stream.[BETA].

Parameters

in	*stream*	- Stream identifier
in	*ptr*	- Pointer to memory object allocated using 'hipMallocSignalMemory' flag
in	*value*	- Value to be used in compare operation
in	*flags*	- Defines the compare operation, supported values are hipStreamWaitValueGte hipStreamWaitValueEq, hipStreamWaitValueAnd and hipStreamWaitValueNor.
in	*mask*	- Mask to be applied on value at memory before it is compared with value default value is set to enable every bit

Returns

#hipSuccess, #hipErrorInvalidValue

Enqueues a wait command to the stream, all operations enqueued on this stream after this, will not execute until the defined wait condition is true.
hipStreamWaitValueGte: waits until ∗ptr&mask >= value hipStreamWaitValueEq : waits until ∗ptr&mask == value hipStreamWaitValueAnd: waits until ((∗ptr&mask) & value) != 0 hipStreamWaitValueNor: waits until ∼((∗ptr&mask) | (value&mask)) != 0

Note

when using 'hipStreamWaitValueNor', mask is applied on both 'value' and '∗ptr'.

Support for hipStreamWaitValue64 can be queried using 'hipDeviceGetAttribute()' and 'hipDeviceAttribute←CanUseStreamWaitValue' flag.

Warning

This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

See also

hipExtMallocWithFlags, hipFree, hipStreamWaitValue32, hipStreamWriteValue64, hipStreamWriteValue32, hipDeviceGetAttribute

4.7.3.16 hipStreamWriteValue32()

```
hipError_t hipStreamWriteValue32 (
            hipStream_t stream,
            void * ptr,
            uint32_t value,
            unsigned int flags )
```

Enqueues a write command to the stream.[BETA].

Parameters

in	*stream*	- Stream identifier
in	*ptr*	- Pointer to a GPU accessible memory object
in	*value*	- Value to be written
in	*flags*	- reserved, ignored for now, will be used in future releases

Returns

#hipSuccess, #hipErrorInvalidValue

Enqueues a write command to the stream, write operation is performed after all earlier commands on this stream have completed the execution.

Warning

This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

See also

hipExtMallocWithFlags, hipFree, hipStreamWriteValue32, hipStreamWaitValue32, hipStreamWaitValue64

4.7.3.17 hipStreamWriteValue64()

```
hipError_t hipStreamWriteValue64 (
             hipStream_t stream,
             void * ptr,
             uint64_t value,
             unsigned int flags )
```

Enqueues a write command to the stream.[BETA].

Parameters

in	*stream*	- Stream identifier
in	*ptr*	- Pointer to a GPU accessible memory object
in	*value*	- Value to be written
in	*flags*	- reserved, ignored for now, will be used in future releases

Returns

#hipSuccess, #hipErrorInvalidValue

Enqueues a write command to the stream, write operation is performed after all earlier commands on this stream have completed the execution.

Warning

This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

See also

hipExtMallocWithFlags, hipFree, hipStreamWriteValue32, hipStreamWaitValue32, hipStreamWaitValue64

4.8 Event Management

Functions

- hipError_t hipEventCreateWithFlags (hipEvent_t ∗event, unsigned flags)

  Create an event with the specified flags.
- hipError_t hipEventCreate (hipEvent_t ∗event)
- hipError_t hipEventRecord (hipEvent_t event, hipStream_t stream)

  Record an event in the specified stream.
- hipError_t hipEventDestroy (hipEvent_t event)

  Destroy the specified event.
- hipError_t hipEventSynchronize (hipEvent_t event)

  Wait for an event to complete.
- hipError_t hipEventElapsedTime (float ∗ms, hipEvent_t start, hipEvent_t stop)

  Return the elapsed time between two events.
- hipError_t hipEventQuery (hipEvent_t event)

  Query event status.

4.8.1 Detailed Description

This section describes the event management functions of HIP runtime API.

4.8.2 Function Documentation

4.8.2.1 hipEventCreate()

```
hipError_t hipEventCreate (
            hipEvent_t * event )
```

Create an event

Parameters

in,out	*event*	Returns the newly created event.

Returns

#hipSuccess, #hipErrorNotInitialized, #hipErrorInvalidValue, #hipErrorLaunchFailure, #hipErrorOutOfMemory

See also

hipEventCreateWithFlags, hipEventRecord, hipEventQuery, hipEventSynchronize, hipEventDestroy, hipEventElapsedTime

4.8.2.2 hipEventCreateWithFlags()

```
hipError_t hipEventCreateWithFlags (
            hipEvent_t * event,
            unsigned flags )
```

Create an event with the specified flags.

Parameters

in,out	*event*	Returns the newly created event.

Parameters

in	*flags*	Flags to control event behavior. Valid values are hipEventDefault, hipEventBlockingSync, hipEventDisableTiming, hipEventInterprocess hipEventDefault : Default flag. The event will use active synchronization and will support timing. Blocking synchronization provides lowest possible latency at the expense of dedicating a CPU to poll on the event. hipEventBlockingSync : The event will use blocking synchronization : if hipEventSynchronize is called on this event, the thread will block until the event completes. This can increase latency for the synchroniation but can result in lower power and more resources for other CPU threads. hipEventDisableTiming : Disable recording of timing information. Events created with this flag would not record profiling data and provide best performance if used for synchronization.

Warning

On AMD platform, hipEventInterprocess support is under development. Use of this flag will return an error.

Returns

#hipSuccess, #hipErrorNotInitialized, #hipErrorInvalidValue, #hipErrorLaunchFailure, #hipErrorOutOfMemory

See also

hipEventCreate, hipEventSynchronize, hipEventDestroy, hipEventElapsedTime

4.8.2.3 hipEventDestroy()

```
hipError_t hipEventDestroy (
            hipEvent_t event )
```

Destroy the specified event.

Parameters

in	*event*	Event to destroy.

Returns

#hipSuccess, #hipErrorNotInitialized, #hipErrorInvalidValue, #hipErrorLaunchFailure

Releases memory associated with the event. If the event is recording but has not completed recording when hipEventDestroy() is called, the function will return immediately and the completion_future resources will be released later, when the hipDevice is synchronized.

See also

hipEventCreate, hipEventCreateWithFlags, hipEventQuery, hipEventSynchronize, hipEventRecord, hipEventElapsedTime

Returns

#hipSuccess

4.8.2.4 hipEventElapsedTime()

```
hipError_t hipEventElapsedTime (
            float * ms,
            hipEvent_t start,
            hipEvent_t stop )
```

Return the elapsed time between two events.

Parameters

out	*ms*	: Return time between start and stop in ms.
in	*start*	: Start event.
in	*stop*	: Stop event.

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotReady, #hipErrorInvalidHandle, #hipErrorNotInitialized, #hipErrorLaunchFailure

Computes the elapsed time between two events. Time is computed in ms, with a resolution of approximately 1 us.
Events which are recorded in a NULL stream will block until all commands on all other streams complete execution, and then record the timestamp.
Events which are recorded in a non-NULL stream will record their timestamp when they reach the head of the specified stream, after all previous commands in that stream have completed executing. Thus the time that the event recorded may be significantly after the host calls hipEventRecord().
If hipEventRecord() has not been called on either event, then #hipErrorInvalidHandle is returned. If hipEventRecord() has been called on both events, but the timestamp has not yet been recorded on one or both events (that is, hipEventQuery() would return #hipErrorNotReady on at least one of the events), then #hip↩ErrorNotReady is returned.
Note, for HIP Events used in kernel dispatch using hipExtLaunchKernelGGL/hipExtLaunchKernel, events passed in hipExtLaunchKernelGGL/hipExtLaunchKernel are not explicitly recorded and should only be used to get elapsed time for that specific launch. In case events are used across multiple dispatches, for example, start and stop events from different hipExtLaunchKernelGGL/ hipExtLaunchKernel calls, they will be treated as invalid unrecorded events, HIP will throw error "hipErrorInvalidHandle" from hipEventElapsedTime.

See also

hipEventCreate, hipEventCreateWithFlags, hipEventQuery, hipEventDestroy, hipEventRecord, hipEventSynchronize

4.8.2.5 hipEventQuery()

```
hipError_t hipEventQuery (
            hipEvent_t event )
```

Query event status.

Parameters

in	*event*	Event to query.

Returns

#hipSuccess, #hipErrorNotReady, #hipErrorInvalidHandle, #hipErrorInvalidValue, #hipErrorNotInitialized, #hipErrorLaunchFailure

Query the status of the specified event. This function will return #hipErrorNotReady if all commands in the appropriate stream (specified to hipEventRecord()) have completed. If that work has not completed, or if hipEventRecord() was not called on the event, then #hipSuccess is returned.

See also

hipEventCreate, hipEventCreateWithFlags, hipEventRecord, hipEventDestroy, hipEventSynchronize, hipEventElapsedTime

4.8.2.6 hipEventRecord()

```
hipError_t hipEventRecord (
             hipEvent_t event,
             hipStream_t stream )
```

Record an event in the specified stream.

Parameters

in	*event*	event to record.
in	*stream*	stream in which to record event.

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized, #hipErrorInvalidHandle, #hipErrorLaunchFailure

hipEventQuery() or hipEventSynchronize() must be used to determine when the event transitions from "recording" (after hipEventRecord() is called) to "recorded" (when timestamps are set, if requested).
Events which are recorded in a non-NULL stream will transition to from recording to "recorded" state when they reach the head of the specified stream, after all previous commands in that stream have completed executing.
If hipEventRecord() has been previously called on this event, then this call will overwrite any existing state in event.
If this function is called on an event that is currently being recorded, results are undefined

- either outstanding recording may save state into the event, and the order is not guaranteed.

See also

hipEventCreate, hipEventCreateWithFlags, hipEventQuery, hipEventSynchronize, hipEventDestroy, hipEventElapsedTime

4.8.2.7 hipEventSynchronize()

```
hipError_t hipEventSynchronize (
             hipEvent_t event )
```

Wait for an event to complete.
This function will block until the event is ready, waiting for all previous work in the stream specified when event was recorded with hipEventRecord().
If hipEventRecord() has not been called on `event`, this function returns immediately.
TODO-hip- This function needs to support hipEventBlockingSync parameter.

Parameters

in	*event*	Event on which to wait.

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized, #hipErrorInvalidHandle, #hipErrorLaunchFailure

See also

hipEventCreate, hipEventCreateWithFlags, hipEventQuery, hipEventDestroy, hipEventRecord, hipEventElapsedTime

4.9 Memory Management

Functions

- hipError_t hipPointerGetAttributes (hipPointerAttribute_t ∗attributes, const void ∗ptr)

  Return attributes for the specified pointer.
- hipError_t hipImportExternalSemaphore (hipExternalSemaphore_t ∗extSem_out, const hipExternalSemaphoreHandleDesc ∗semHandleDesc)

  Imports an external semaphore.
- hipError_t hipSignalExternalSemaphoresAsync (const hipExternalSemaphore_t ∗extSemArray, const hipExternalSemaphoreSignalParams ∗paramsArray, unsigned int numExtSems, hipStream_t stream)

  Signals a set of external semaphore objects.
- hipError_t hipWaitExternalSemaphoresAsync (const hipExternalSemaphore_t ∗extSemArray, const hipExternalSemaphoreWaitParams ∗paramsArray, unsigned int numExtSems, hipStream_t stream)

  Waits on a set of external semaphore objects.
- hipError_t hipDestroyExternalSemaphore (hipExternalSemaphore_t extSem)

  Destroys an external semaphore object and releases any references to the underlying resource. Any outstanding signals or waits must have completed before the semaphore is destroyed.
- hipError_t hipImportExternalMemory (hipExternalMemory_t ∗extMem_out, const hipExternalMemoryHandleDesc ∗memHandleDesc)

  Imports an external memory object.
- hipError_t hipExternalMemoryGetMappedBuffer (void ∗∗devPtr, hipExternalMemory_t extMem, const hipExternalMemoryBufferDesc ∗bufferDesc)

  Maps a buffer onto an imported memory object.
- hipError_t hipDestroyExternalMemory (hipExternalMemory_t extMem)

  Destroys an external memory object.
- hipError_t hipMalloc (void ∗∗ptr, size_t size)

  Allocate memory on the default accelerator.
- hipError_t hipExtMallocWithFlags (void ∗∗ptr, size_t sizeBytes, unsigned int flags)

  Allocate memory on the default accelerator.
- hipError_t hipMallocHost (void ∗∗ptr, size_t size)

  Allocate pinned host memory [Deprecated].
- hipError_t hipMemAllocHost (void ∗∗ptr, size_t size)

  Allocate pinned host memory [Deprecated].
- hipError_t hipHostMalloc (void ∗∗ptr, size_t size, unsigned int flags)

  Allocate device accessible page locked host memory.
- hipError_t hipMallocManaged (void ∗∗dev_ptr, size_t size, unsigned int flags __dparm(hipMemAttachGlobal))

  Allocates memory that will be automatically managed by HIP.
- hipError_t hipMemPrefetchAsync (const void ∗dev_ptr, size_t count, int device, hipStream_t stream __↩
dparm(0))

  Prefetches memory to the specified destination device using HIP.
- hipError_t hipMemAdvise (const void ∗dev_ptr, size_t count, hipMemoryAdvise advice, int device)

  Advise about the usage of a given memory range to HIP.
- hipError_t hipMemRangeGetAttribute (void ∗data, size_t data_size, hipMemRangeAttribute attribute, const void ∗dev_ptr, size_t count)

  Query an attribute of a given memory range in HIP.
- hipError_t hipMemRangeGetAttributes (void ∗∗data, size_t ∗data_sizes, hipMemRangeAttribute ∗attributes, size_t num_attributes, const void ∗dev_ptr, size_t count)

  Query attributes of a given memory range in HIP.
- hipError_t hipStreamAttachMemAsync (hipStream_t stream, void ∗dev_ptr, size_t length __dparm(0), unsigned int flags __dparm(hipMemAttachSingle))

  Attach memory to a stream asynchronously in HIP.
- hipError_t hipHostAlloc (void ∗∗ptr, size_t size, unsigned int flags)

Allocate device accessible page locked host memory [Deprecated].

- hipError_t hipHostGetDevicePointer (void ∗∗devPtr, void ∗hstPtr, unsigned int flags)

  Get Device pointer from Host Pointer allocated through hipHostMalloc.
- hipError_t hipHostGetFlags (unsigned int ∗flagsPtr, void ∗hostPtr)

  Return flags associated with host pointer.
- hipError_t hipHostRegister (void ∗hostPtr, size_t sizeBytes, unsigned int flags)

  Register host memory so it can be accessed from the current device.
- hipError_t hipHostUnregister (void ∗hostPtr)

  Un-register host pointer.
- hipError_t hipMallocPitch (void ∗∗ptr, size_t ∗pitch, size_t width, size_t height)
- hipError_t hipMemAllocPitch (hipDeviceptr_t ∗dptr, size_t ∗pitch, size_t widthInBytes, size_t height, unsigned int elementSizeBytes)
- hipError_t hipFree (void ∗ptr)

  Free memory allocated by the hcc hip memory allocation API. This API performs an implicit hipDeviceSynchronize() call. If pointer is NULL, the hip runtime is initialized and hipSuccess is returned.
- hipError_t hipFreeHost (void ∗ptr)

  Free memory allocated by the hcc hip host memory allocation API. [Deprecated].
- hipError_t hipHostFree (void ∗ptr)

  Free memory allocated by the hcc hip host memory allocation API This API performs an implicit hipDeviceSynchronize() call. If pointer is NULL, the hip runtime is initialized and hipSuccess is returned.
- hipError_t hipMemcpy (void ∗dst, const void ∗src, size_t sizeBytes, hipMemcpyKind kind)

  Copy data from src to dst.
- hipError_t **hipMemcpyWithStream** (void ∗dst, const void ∗src, size_t sizeBytes, hipMemcpyKind kind, hip↩ Stream_t stream)
- hipError_t hipMemcpyHtoD (hipDeviceptr_t dst, void ∗src, size_t sizeBytes)

  Copy data from Host to Device.
- hipError_t hipMemcpyDtoH (void ∗dst, hipDeviceptr_t src, size_t sizeBytes)

  Copy data from Device to Host.
- hipError_t hipMemcpyDtoD (hipDeviceptr_t dst, hipDeviceptr_t src, size_t sizeBytes)

  Copy data from Device to Device.
- hipError_t hipMemcpyHtoDAsync (hipDeviceptr_t dst, void ∗src, size_t sizeBytes, hipStream_t stream)

  Copy data from Host to Device asynchronously.
- hipError_t hipMemcpyDtoHAsync (void ∗dst, hipDeviceptr_t src, size_t sizeBytes, hipStream_t stream)

  Copy data from Device to Host asynchronously.
- hipError_t hipMemcpyDtoDAsync (hipDeviceptr_t dst, hipDeviceptr_t src, size_t sizeBytes, hipStream_↩ t stream)

  Copy data from Device to Device asynchronously.
- hipError_t hipModuleGetGlobal (hipDeviceptr_t ∗dptr, size_t ∗bytes, hipModule_t hmod, const char ∗name)

  Returns a global pointer from a module. Returns in ∗dptr and ∗bytes the pointer and size of the global of name name located in module hmod. If no variable of that name exists, it returns hipErrorNotFound. Both parameters dptr and bytes are optional. If one of them is NULL, it is ignored and hipSuccess is returned.
- hipError_t **hipGetSymbolAddress** (void ∗∗devPtr, const void ∗symbol)
- hipError_t **hipGetSymbolSize** (size_t ∗size, const void ∗symbol)
- hipError_t **hipMemcpyToSymbol** (const void ∗symbol, const void ∗src, size_t sizeBytes, size_t offset __↩ dparm(0), hipMemcpyKind kind __dparm(hipMemcpyHostToDevice))
- hipError_t **hipMemcpyToSymbolAsync** (const void ∗symbol, const void ∗src, size_t sizeBytes, size_t offset, hipMemcpyKind kind, hipStream_t stream __dparm(0))
- hipError_t **hipMemcpyFromSymbol** (void ∗dst, const void ∗symbol, size_t sizeBytes, size_t offset __↩ dparm(0), hipMemcpyKind kind __dparm(hipMemcpyDeviceToHost))
- hipError_t **hipMemcpyFromSymbolAsync** (void ∗dst, const void ∗symbol, size_t sizeBytes, size_t offset, hipMemcpyKind kind, hipStream_t stream __dparm(0))
- hipError_t hipMemcpyAsync (void ∗dst, const void ∗src, size_t sizeBytes, hipMemcpyKind kind, hipStream_t stream __dparm(0))

Copy data from src to dst asynchronously.

- hipError_t hipMemset (void ∗dst, int value, size_t sizeBytes)

  Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value.
- hipError_t hipMemsetD8 (hipDeviceptr_t dest, unsigned char value, size_t count)

  Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value.
- hipError_t hipMemsetD8Async (hipDeviceptr_t dest, unsigned char value, size_t count, hipStream_t stream __dparm(0))

  Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value.
- hipError_t hipMemsetD16 (hipDeviceptr_t dest, unsigned short value, size_t count)

  Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant short value value.
- hipError_t hipMemsetD16Async (hipDeviceptr_t dest, unsigned short value, size_t count, hipStream_t stream __dparm(0))

  Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant short value value.
- hipError_t hipMemsetD32 (hipDeviceptr_t dest, int value, size_t count)

  Fills the memory area pointed to by dest with the constant integer value for specified number of times.
- hipError_t hipMemsetAsync (void ∗dst, int value, size_t sizeBytes, hipStream_t stream __dparm(0))

  Fills the first sizeBytes bytes of the memory area pointed to by dev with the constant byte value value.
- hipError_t hipMemsetD32Async (hipDeviceptr_t dst, int value, size_t count, hipStream_t stream __dparm(0))

  Fills the memory area pointed to by dev with the constant integer value for specified number of times.
- hipError_t hipMemset2D (void ∗dst, size_t pitch, int value, size_t width, size_t height)

  Fills the memory area pointed to by dst with the constant value.
- hipError_t hipMemset2DAsync (void ∗dst, size_t pitch, int value, size_t width, size_t height, hipStream_t stream __dparm(0))

  Fills asynchronously the memory area pointed to by dst with the constant value.
- hipError_t hipMemset3D (hipPitchedPtr pitchedDevPtr, int value, hipExtent extent)

  Fills synchronously the memory area pointed to by pitchedDevPtr with the constant value.
- hipError_t hipMemset3DAsync (hipPitchedPtr pitchedDevPtr, int value, hipExtent extent, hipStream_t stream __dparm(0))

  Fills asynchronously the memory area pointed to by pitchedDevPtr with the constant value.
- hipError_t hipMemGetInfo (size_t ∗free, size_t ∗total)

  Query memory info. Return snapshot of free memory, and total allocatable memory on the device.
- hipError_t **hipMemPtrGetInfo** (void ∗ptr, size_t ∗size)
- hipError_t hipMallocArray (hipArray ∗∗array, const hipChannelFormatDesc ∗desc, size_t width, size_t height __dparm(0), unsigned int flags __dparm(hipArrayDefault))

  Allocate an array on the device.
- hipError_t **hipArrayCreate** (hipArray ∗∗pHandle, const HIP_ARRAY_DESCRIPTOR ∗pAllocateArray)
- hipError_t **hipArrayDestroy** (hipArray ∗array)
- hipError_t **hipArray3DCreate** (hipArray ∗∗array, const HIP_ARRAY3D_DESCRIPTOR ∗pAllocateArray)
- hipError_t **hipMalloc3D** (hipPitchedPtr ∗pitchedDevPtr, hipExtent extent)
- hipError_t hipFreeArray (hipArray ∗array)

  Frees an array on the device.
- hipError_t hipFreeMipmappedArray (hipMipmappedArray_t mipmappedArray)

  Frees a mipmapped array on the device.
- hipError_t hipMalloc3DArray (hipArray ∗∗array, const struct hipChannelFormatDesc ∗desc, struct hipExtent extent, unsigned int flags)

  Allocate an array on the device.
- hipError_t hipMallocMipmappedArray (hipMipmappedArray_t ∗mipmappedArray, const struct hipChannelFormatDesc ∗desc, struct hipExtent extent, unsigned int numLevels, unsigned int flags __dparm(0))

  Allocate a mipmapped array on the device.
- hipError_t hipGetMipmappedArrayLevel (hipArray_t ∗levelArray, hipMipmappedArray_const_t mipmapped↩ Array, unsigned int level)

  Gets a mipmap level of a HIP mipmapped array.

- hipError_t hipMemcpy2D (void ∗dst, size_t dpitch, const void ∗src, size_t spitch, size_t width, size_t height, hipMemcpyKind kind)

  Copies data between host and device.
- hipError_t hipMemcpyParam2D (const hip_Memcpy2D ∗pCopy)

  Copies memory for 2D arrays.
- hipError_t hipMemcpyParam2DAsync (const hip_Memcpy2D ∗pCopy, hipStream_t stream __dparm(0))

  Copies memory for 2D arrays.
- hipError_t hipMemcpy2DAsync (void ∗dst, size_t dpitch, const void ∗src, size_t spitch, size_t width, size_t height, hipMemcpyKind kind, hipStream_t stream __dparm(0))

  Copies data between host and device.
- hipError_t hipMemcpy2DToArray (hipArray ∗dst, size_t wOffset, size_t hOffset, const void ∗src, size_t spitch, size_t width, size_t height, hipMemcpyKind kind)

  Copies data between host and device.
- hipError_t hipMemcpy2DToArrayAsync (hipArray ∗dst, size_t wOffset, size_t hOffset, const void ∗src, size_t spitch, size_t width, size_t height, hipMemcpyKind kind, hipStream_t stream __dparm(0))

  Copies data between host and device.
- hipError_t hipMemcpyToArray (hipArray ∗dst, size_t wOffset, size_t hOffset, const void ∗src, size_t count, hipMemcpyKind kind)

  Copies data between host and device.
- hipError_t hipMemcpyFromArray (void ∗dst, hipArray_const_t srcArray, size_t wOffset, size_t hOffset, size_t count, hipMemcpyKind kind)

  Copies data between host and device.
- hipError_t hipMemcpy2DFromArray (void ∗dst, size_t dpitch, hipArray_const_t src, size_t wOffset, size_t h↩ Offset, size_t width, size_t height, hipMemcpyKind kind)

  Copies data between host and device.
- hipError_t hipMemcpy2DFromArrayAsync (void ∗dst, size_t dpitch, hipArray_const_t src, size_t wOffset, size_t hOffset, size_t width, size_t height, hipMemcpyKind kind, hipStream_t stream __dparm(0))

  Copies data between host and device asynchronously.
- hipError_t hipMemcpyAtoH (void ∗dst, hipArray ∗srcArray, size_t srcOffset, size_t count)

  Copies data between host and device.
- hipError_t hipMemcpyHtoA (hipArray ∗dstArray, size_t dstOffset, const void ∗srcHost, size_t count)

  Copies data between host and device.
- hipError_t hipMemcpy3D (const struct hipMemcpy3DParms ∗p)

  Copies data between host and device.
- hipError_t hipMemcpy3DAsync (const struct hipMemcpy3DParms ∗p, hipStream_t stream __dparm(0))

  Copies data between host and device asynchronously.
- hipError_t hipDrvMemcpy3D (const HIP_MEMCPY3D ∗pCopy)

  Copies data between host and device.
- hipError_t hipDrvMemcpy3DAsync (const HIP_MEMCPY3D ∗pCopy, hipStream_t stream)

  Copies data between host and device asynchronously.

4.9.1 Detailed Description

This section describes the memory management functions of HIP runtime API. The following CUDA APIs are not currently supported:

- cudaMalloc3D
- cudaMalloc3DArray
- TODO - more 2D, 3D, array APIs here.

This section describes the managed memory management functions of HIP runtime API.

4.9.2 Function Documentation

4.9.2.1 hipDestroyExternalMemory()

```
hipError_t hipDestroyExternalMemory (
            hipExternalMemory_t extMem )
```

Destroys an external memory object.

Parameters

in	*extMem*	External memory object to be destroyed

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.2 hipDestroyExternalSemaphore()

```
hipError_t hipDestroyExternalSemaphore (
            hipExternalSemaphore_t extSem )
```

Destroys an external semaphore object and releases any references to the underlying resource. Any outstanding signals or waits must have completed before the semaphore is destroyed.

Parameters

in	*extSem*	handle to an external memory object

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.3 hipDrvMemcpy3D()

```
hipError_t hipDrvMemcpy3D (
            const HIP_MEMCPY3D * pCopy )
```

Copies data between host and device.

Parameters

in	*pCopy*	3D memory copy parameters

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩
InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.4 hipDrvMemcpy3DAsync()

```
hipError_t hipDrvMemcpy3DAsync (
            const HIP_MEMCPY3D * pCopy,
            hipStream_t stream )
```

Copies data between host and device asynchronously.

Parameters

in	*pCopy*	3D memory copy parameters
in	*stream*	Stream to use

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.5 hipExternalMemoryGetMappedBuffer()

```
hipError_t hipExternalMemoryGetMappedBuffer (
            void ** devPtr,
            hipExternalMemory_t extMem,
            const hipExternalMemoryBufferDesc * bufferDesc )
```

Maps a buffer onto an imported memory object.

Parameters

out	*devPtr*	Returned device pointer to buffer
in	*extMem*	Handle to external memory object
in	*bufferDesc*	Buffer descriptor

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.6 hipExtMallocWithFlags()

```
hipError_t hipExtMallocWithFlags (
            void ** ptr,
            size_t sizeBytes,
            unsigned int flags )
```

Allocate memory on the default accelerator.

Parameters

out	*ptr*	Pointer to the allocated memory
in	*size*	Requested memory size
in	*flags*	Type of memory allocation

If size is 0, no memory is allocated, *ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory, #hipErrorInvalidValue (bad context, null *ptr)

See also

hipMallocPitch, hipFree, hipMallocArray, hipFreeArray, hipMalloc3D, hipMalloc3DArray, hipHostFree, hipHostMalloc

4.9.2.7 hipFree()

```
hipError_t hipFree (
            void * ptr )
```

Free memory allocated by the hcc hip memory allocation API. This API performs an implicit hipDeviceSynchronize() call. If pointer is NULL, the hip runtime is initialized and hipSuccess is returned.

Parameters

in	*ptr*	Pointer to memory to be freed

Returns

#hipSuccess

#hipErrorInvalidDevicePointer (if pointer is invalid, including host pointers allocated with hipHostMalloc)

See also

hipMalloc, hipMallocPitch, hipMallocArray, hipFreeArray, hipHostFree, hipMalloc3D, hipMalloc3DArray, hipHostMalloc

4.9.2.8 hipFreeArray()

```
hipError_t hipFreeArray (
            hipArray * array )
```

Frees an array on the device.

Parameters

in	*array*	Pointer to array to free

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

See also

hipMalloc, hipMallocPitch, hipFree, hipMallocArray, hipHostMalloc, hipHostFree

4.9.2.9 hipFreeHost()

```
static hipError_t hipFreeHost (
            void * ptr )  [inline]
```

Free memory allocated by the hcc hip host memory allocation API. [Deprecated].

Parameters

in	*ptr*	Pointer to memory to be freed

Returns

#hipSuccess, #hipErrorInvalidValue (if pointer is invalid, including device pointers allocated with hipMalloc)

4.9.2.10 hipFreeMipmappedArray()

```
hipError_t hipFreeMipmappedArray (
            hipMipmappedArray_t mipmappedArray )
```

Frees a mipmapped array on the device.

Parameters

in	*mipmappedArray*	- Pointer to mipmapped array to free

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.11 hipGetMipmappedArrayLevel()

```
hipError_t hipGetMipmappedArrayLevel (
            hipArray_t * levelArray,
            hipMipmappedArray_const_t mipmappedArray,
            unsigned int level )
```

Gets a mipmap level of a HIP mipmapped array.

Parameters

out	*levelArray*	- Returned mipmap level HIP array
in	*mipmappedArray*	- HIP mipmapped array
in	*level*	- Mipmap level

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.12 hipHostAlloc()

```
static hipError_t hipHostAlloc (
              void ** ptr,
              size_t size,
              unsigned int flags )  [inline]
```

Allocate device accessible page locked host memory [Deprecated].

Parameters

out	*ptr*	Pointer to the allocated host pinned memory
in	*size*	Requested memory size
in	*flags*	Type of host memory allocation

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory

4.9.2.13 hipHostFree()

```
hipError_t hipHostFree (
              void * ptr )
```

Free memory allocated by the hcc hip host memory allocation API This API performs an implicit hipDeviceSynchronize() call. If pointer is NULL, the hip runtime is initialized and hipSuccess is returned.

Parameters

in	*ptr*	Pointer to memory to be freed

Returns

#hipSuccess, #hipErrorInvalidValue (if pointer is invalid, including device pointers allocated with hipMalloc)

See also

hipMalloc, hipMallocPitch, hipFree, hipMallocArray, hipFreeArray, hipMalloc3D, hipMalloc3DArray, hipHostMalloc

4.9.2.14 hipHostGetDevicePointer()

```
hipError_t hipHostGetDevicePointer (
              void ** devPtr,
              void * hstPtr,
              unsigned int flags )
```

Get Device pointer from Host Pointer allocated through hipHostMalloc.

Parameters

out	*dstPtr*	Device Pointer mapped to passed host pointer

Parameters

in	*hstPtr*	Host Pointer allocated through hipHostMalloc
in	*flags*	Flags to be passed for extension

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorOutOfMemory

See also

hipSetDeviceFlags, hipHostMalloc

4.9.2.15 hipHostGetFlags()

```
hipError_t hipHostGetFlags (
            unsigned int * flagsPtr,
            void * hostPtr )
```

Return flags associated with host pointer.

Parameters

out	*flagsPtr*	Memory location to store flags
in	*hostPtr*	Host Pointer allocated through hipHostMalloc

Returns

#hipSuccess, #hipErrorInvalidValue

See also

hipHostMalloc

4.9.2.16 hipHostMalloc()

```
hipError_t hipHostMalloc (
            void ** ptr,
            size_t size,
            unsigned int flags )
```

Allocate device accessible page locked host memory.

Parameters

out	*ptr*	Pointer to the allocated host pinned memory
in	*size*	Requested memory size
in	*flags*	Type of host memory allocation

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory

See also

hipSetDeviceFlags, hipHostFree

4.9.2.17 hipHostRegister()

```
hipError_t hipHostRegister (
            void * hostPtr,
            size_t sizeBytes,
            unsigned int flags )
```

Register host memory so it can be accessed from the current device.

Parameters

out	*hostPtr*	Pointer to host memory to be registered.
in	*sizeBytes*	size of the host memory
in	*flags.*	See below.

Flags:

- hipHostRegisterDefault Memory is Mapped and Portable
- hipHostRegisterPortable Memory is considered registered by all contexts. HIP only supports one context so this is always assumed true.
- hipHostRegisterMapped Map the allocation into the address space for the current device. The device pointer can be obtained with hipHostGetDevicePointer.

After registering the memory, use hipHostGetDevicePointer to obtain the mapped device pointer. On many systems, the mapped device pointer will have a different value than the mapped host pointer. Applications must use the device pointer in device code, and the host pointer in device code.
On some systems, registered memory is pinned. On some systems, registered memory may not be actually be pinned but uses OS or hardware facilities to all GPU access to the host memory.
Developers are strongly encouraged to register memory blocks which are aligned to the host cache-line size. (typically 64-bytes but can be obtains from the CPUID instruction).
If registering non-aligned pointers, the application must take care when register pointers from the same cache line on different devices. HIP's coarse-grained synchronization model does not guarantee correct results if different devices write to different parts of the same cache block - typically one of the writes will "win" and overwrite data from the other registered memory region.

Returns

#hipSuccess, #hipErrorOutOfMemory

See also

hipHostUnregister, hipHostGetFlags, hipHostGetDevicePointer

4.9.2.18 hipHostUnregister()

```
hipError_t hipHostUnregister (
            void * hostPtr )
```

Un-register host pointer.

Parameters

in	*hostPtr*	Host pointer previously registered with hipHostRegister

Returns

Error code

See also

hipHostRegister

4.9.2.19 hipImportExternalMemory()

```
hipError_t hipImportExternalMemory (
            hipExternalMemory_t * extMem_out,
            const hipExternalMemoryHandleDesc * memHandleDesc )
```

Imports an external memory object.

Parameters

out	*extMem_out*	Returned handle to an external memory object
in	*memHandleDesc*	Memory import handle descriptor

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.20 hipImportExternalSemaphore()

```
hipError_t hipImportExternalSemaphore (
            hipExternalSemaphore_t * extSem_out,
            const hipExternalSemaphoreHandleDesc * semHandleDesc )
```

Imports an external semaphore.

Parameters

out	*extSem_out*	External semaphores to be waited on
in	*semHandleDesc*	Semaphore import handle descriptor

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.21 hipMalloc()

```
hipError_t hipMalloc (
            void ** ptr,
            size_t size )
```

Allocate memory on the default accelerator.

Parameters

out	*ptr*	Pointer to the allocated memory
in	*size*	Requested memory size

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory, #hipErrorInvalidValue (bad context, null ∗ptr)

See also

hipMallocPitch, hipFree, hipMallocArray, hipFreeArray, hipMalloc3D, hipMalloc3DArray, hipHostFree, hipHostMalloc

4.9.2.22 hipMalloc3DArray()

```
hipError_t hipMalloc3DArray (
            hipArray ** array,
            const struct hipChannelFormatDesc * desc,
            struct hipExtent extent,
            unsigned int flags )
```

Allocate an array on the device.

Parameters

out	*array*	Pointer to allocated array in device memory
in	*desc*	Requested channel format
in	*extent*	Requested array allocation width, height and depth
in	*flags*	Requested properties of allocated array

Returns

#hipSuccess, #hipErrorOutOfMemory

See also

hipMalloc, hipMallocPitch, hipFree, hipFreeArray, hipHostMalloc, hipHostFree

4.9.2.23 hipMallocArray()

```
hipError_t hipMallocArray (
            hipArray ** array,
            const hipChannelFormatDesc * desc,
            size_t width,
            size_t height  __dparm0,
            unsigned int flags  __dparmhipArrayDefault )
```

Allocate an array on the device.

Parameters

out	*array*	Pointer to allocated array in device memory
in	*desc*	Requested channel format
in	*width*	Requested array allocation width

Parameters

in	*height*	Requested array allocation height
in	*flags*	Requested properties of allocated array

Returns

#hipSuccess, #hipErrorOutOfMemory

See also

hipMalloc, hipMallocPitch, hipFree, hipFreeArray, hipHostMalloc, hipHostFree

4.9.2.24 hipMallocHost()

```
static hipError_t hipMallocHost (
              void ** ptr,
              size_t size )  [inline]
```

Allocate pinned host memory [Deprecated].

Parameters

out	*ptr*	Pointer to the allocated host pinned memory
in	*size*	Requested memory size

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory

4.9.2.25 hipMallocManaged()

```
hipError_t hipMallocManaged (
              void ** dev_ptr,
              size_t size,
              unsigned int flags  __dparmhipMemAttachGlobal )
```

Allocates memory that will be automatically managed by HIP.

Parameters

out	*dev_ptr*	- pointer to allocated device memory
in	*size*	- requested allocation size in bytes
in	*flags*	- must be either hipMemAttachGlobal or hipMemAttachHost (defaults to hipMemAttachGlobal)

Returns

#hipSuccess, #hipErrorMemoryAllocation, #hipErrorNotSupported, #hipErrorInvalidValue

4.9.2.26 hipMallocMipmappedArray()

```
hipError_t hipMallocMipmappedArray (
              hipMipmappedArray_t * mipmappedArray,
              const struct hipChannelFormatDesc * desc,
              struct hipExtent extent,
              unsigned int numLevels,
              unsigned int flags  __dparm0 )
```

Allocate a mipmapped array on the device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryAllocation

4.9.2.27 hipMallocPitch()

```
hipError_t hipMallocPitch (
              void ** ptr,
              size_t * pitch,
              size_t width,
              size_t height )
```

Allocates at least width (in bytes) ∗ height bytes of linear memory Padding may occur to ensure alighnment requirements are met for the given row The change in width size due to padding will be returned in ∗pitch. Currently the alignment is set to 128 bytes

Parameters

out	*ptr*	Pointer to the allocated device memory
out	*pitch*	Pitch for allocation (in bytes)
in	*width*	Requested pitched allocation width (in bytes)
in	*height*	Requested pitched allocation height

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

Error code

See also

hipMalloc, hipFree, hipMallocArray, hipFreeArray, hipHostFree, hipMalloc3D, hipMalloc3DArray, hipHostMalloc

4.9.2.28 hipMemAdvise()

```
hipError_t hipMemAdvise (
              const void * dev_ptr,
              size_t count,
              hipMemoryAdvise advice,
              int device )
```

Advise about the usage of a given memory range to HIP.

Parameters

in	*dev_ptr*	pointer to memory to set the advice for
in	*count*	size in bytes of the memory range
in	*advice*	advice to be applied for the specified memory range
in	*device*	device to apply the advice for

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.29 hipMemAllocHost()

```
static hipError_t hipMemAllocHost (
            void ** ptr,
            size_t size )  [inline]
```

Allocate pinned host memory [Deprecated].

Parameters

out	*ptr*	Pointer to the allocated host pinned memory
in	*size*	Requested memory size

If size is 0, no memory is allocated, ∗ptr returns nullptr, and hipSuccess is returned.

Returns

#hipSuccess, #hipErrorOutOfMemory

4.9.2.30 hipMemAllocPitch()

```
hipError_t hipMemAllocPitch (
            hipDeviceptr_t * dptr,
            size_t * pitch,
            size_t widthInBytes,
            size_t height,
            unsigned int elementSizeBytes )
```

Allocates at least width (in bytes) ∗ height bytes of linear memory Padding may occur to ensure alighnment requirements are met for the given row The change in width size due to padding will be returned in ∗pitch. Currently the alignment is set to 128 bytes

Parameters

out	*dptr*	Pointer to the allocated device memory
out	*pitch*	Pitch for allocation (in bytes)
in	*width*	Requested pitched allocation width (in bytes)
in	*height*	Requested pitched allocation height

If size is 0, no memory is allocated, *ptr returns nullptr, and hipSuccess is returned. The intended usage of pitch is as a separate parameter of the allocation, used to compute addresses within the 2D array. Given the row and column of an array element of type T, the address is computed as: T* pElement = (T∗)((char∗)BaseAddress + Row ∗ Pitch) + Column;

Returns

Error code

See also

hipMalloc, hipFree, hipMallocArray, hipFreeArray, hipHostFree, hipMalloc3D, hipMalloc3DArray, hipHostMalloc

4.9.2.31 hipMemcpy()

```
hipError_t hipMemcpy (
            void * dst,
            const void * src,
            size_t sizeBytes,
            hipMemcpyKind kind )
```

Copy data from src to dst.

It supports memory from host to device, device to host, device to device and host to host The src and dst must not overlap.

For hipMemcpy, the copy is always performed by the current device (set by hipSetDevice). For multi-gpu or peer-to-peer configurations, it is recommended to set the current device to the device where the src data is physically located. For optimal peer-to-peer copies, the copy device must be able to access the src and dst pointers (by calling hipDeviceEnablePeerAccess with copy agent as the current device and src/dest as the peerDevice argument. if this is not done, the hipMemcpy will still work, but will perform the copy using a staging buffer on the host. Calling hipMemcpy with dst and src pointers that do not match the hipMemcpyKind results in undefined behavior.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes
in	*copyType*	Memory copy type

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree, #hipErrorUnknowni

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem↵
HostGetDevicePointer

4.9.2.32 hipMemcpy2D()

```
hipError_t hipMemcpy2D (
            void * dst,
            size_t dpitch,
            const void * src,
            size_t spitch,
            size_t width,
            size_t height,
            hipMemcpyKind kind )
```

Copies data between host and device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpyToArray, hipMemcpy2DToArray, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.33 hipMemcpy2DAsync()

```
hipError_t hipMemcpy2DAsync (
            void * dst,
            size_t dpitch,
            const void * src,
            size_t spitch,
            size_t width,
            size_t height,
            hipMemcpyKind kind,
            hipStream_t stream  __dparm0 )
```

Copies data between host and device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpyToArray, hipMemcpy2DToArray, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.34 hipMemcpy2DFromArray()

```
hipError_t hipMemcpy2DFromArray (
             void * dst,
             size_t dpitch,
             hipArray_const_t src,
             size_t wOffset,
             size_t hOffset,
             size_t width,
             size_t height,
             hipMemcpyKind kind )
```

Copies data between host and device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↵ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.35 hipMemcpy2DFromArrayAsync()

```
hipError_t hipMemcpy2DFromArrayAsync (
             void * dst,
             size_t dpitch,
             hipArray_const_t src,
             size_t wOffset,
             size_t hOffset,
             size_t width,
             size_t height,
             hipMemcpyKind kind,
             hipStream_t stream  __dparm0 )
```

Copies data between host and device asynchronously.

Parameters

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

Parameters

in	*height*	Height of matrix transfer (rows)
in	*kind*	Type of transfer
in	*stream*	Accelerator view which the copy is being enqueued

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.36 hipMemcpy2DToArray()

```
hipError_t hipMemcpy2DToArray (
            hipArray * dst,
            size_t wOffset,
            size_t hOffset,
            const void * src,
            size_t spitch,
            size_t width,
            size_t height,
            hipMemcpyKind kind )
```

Copies data between host and device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpyToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.37 hipMemcpy2DToArrayAsync()

```
hipError_t hipMemcpy2DToArrayAsync (
            hipArray * dst,
            size_t wOffset,
            size_t hOffset,
            const void * src,
            size_t spitch,
            size_t width,
            size_t height,
            hipMemcpyKind kind,
            hipStream_t stream  __dparm0 )
```

Copies data between host and device.

Parameters

in	*dst*	Destination memory address
in	*wOffset*	Destination starting X offset
in	*hOffset*	Destination starting Y offset
in	*src*	Source memory address
in	*spitch*	Pitch of source memory
in	*width*	Width of matrix transfer (columns in bytes)
in	*height*	Height of matrix transfer (rows)
in	*kind*	Type of transfer
in	*stream*	Accelerator view which the copy is being enqueued

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpyToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.38 hipMemcpy3D()

```
hipError_t hipMemcpy3D (
            const struct hipMemcpy3DParms * p )
```

Copies data between host and device.

Parameters

in	*p*	3D memory copy parameters

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.39 hipMemcpy3DAsync()

```
hipError_t hipMemcpy3DAsync (
            const struct hipMemcpy3DParms * p,
            hipStream_t stream  __dparm0 )
```

Copies data between host and device asynchronously.

Parameters

in	*p*	3D memory copy parameters
in	*stream*	Stream to use

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError←InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.40 hipMemcpyAsync()

```
hipError_t hipMemcpyAsync (
            void * dst,
            const void * src,
            size_t sizeBytes,
            hipMemcpyKind kind,
            hipStream_t stream  __dparm0 )
```

Copy data from src to dst asynchronously.

Warning

If host or dest are not pinned, the memory copy will be performed synchronously. For best performance, use hipHostMalloc to allocate host memory that is transferred asynchronously.

on HCC hipMemcpyAsync does not support overlapped H2D and D2H copies. For hipMemcpy, the copy is always performed by the device associated with the specified stream.

For multi-gpu or peer-to-peer configurations, it is recommended to use a stream which is a attached to the device where the src data is physically located. For optimal peer-to-peer copies, the copy device must be able to access the src and dst pointers (by calling hipDeviceEnablePeerAccess with copy agent as the current device and src/dest as the peerDevice argument. if this is not done, the hipMemcpy will still work, but will perform the copy using a staging buffer on the host.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes
in	*accelerator_view*	Accelerator view which the copy is being enqueued

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree, #hipErrorUnknown

See also

hipMemcpy, hipMemcpy2D, hipMemcpyToArray, hipMemcpy2DToArray, hipMemcpyFromArray, hipMemcpy2DFromArray, hipMemcpyArrayToArray, hipMemcpy2DArrayToArray, hipMemcpyToSymbol, hipMemcpyFromSymbol, hipMemcpy2DAsync, hipMemcpyToArrayAsync, hipMemcpy2DToArrayAsync, hipMemcpyFromArrayAsync, hipMemcpy2DFromArrayAsync, hipMemcpyToSymbolAsync, hipMemcpyFromSymbolAsync

4.9.2.41 hipMemcpyAtoH()

```
hipError_t hipMemcpyAtoH (
            void * dst,
            hipArray * srcArray,
            size_t srcOffset,
            size_t count )
```

Copies data between host and device.

Parameters

in	*dst*	Destination memory address
in	*srcArray*	Source array
in	*srcoffset*	Offset in bytes of source array
in	*count*	Size of memory copy in bytes

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.42 hipMemcpyDtoD()

```
hipError_t hipMemcpyDtoD (
            hipDeviceptr_t dst,
            hipDeviceptr_t src,
            size_t sizeBytes )
```

Copy data from Device to Device.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem↩ HostGetDevicePointer

4.9.2.43 hipMemcpyDtoDAsync()

```
hipError_t hipMemcpyDtoDAsync (
            hipDeviceptr_t dst,
            hipDeviceptr_t src,
            size_t sizeBytes,
            hipStream_t stream )
```

Copy data from Device to Device asynchronously.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem←HostGetDevicePointer

4.9.2.44 hipMemcpyDtoH()

```
hipError_t hipMemcpyDtoH (
            void * dst,
            hipDeviceptr_t src,
            size_t sizeBytes )
```

Copy data from Device to Host.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem←HostGetDevicePointer

4.9.2.45 hipMemcpyDtoHAsync()

```
hipError_t hipMemcpyDtoHAsync (
            void * dst,
            hipDeviceptr_t src,
            size_t sizeBytes,
            hipStream_t stream )
```

Copy data from Device to Host asynchronously.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem←HostGetDevicePointer

4.9.2.46 hipMemcpyFromArray()

```
hipError_t hipMemcpyFromArray (
            void * dst,
            hipArray_const_t srcArray,
            size_t wOffset,
            size_t hOffset,
            size_t count,
            hipMemcpyKind kind )
```

Copies data between host and device.

Parameters

in	*dst*	Destination memory address
in	*srcArray*	Source memory address
in	*woffset*	Source starting X offset
in	*hOffset*	Source starting Y offset
in	*count*	Size in bytes to copy
in	*kind*	Type of transfer

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError←InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.47 hipMemcpyHtoA()

```
hipError_t hipMemcpyHtoA (
            hipArray * dstArray,
            size_t dstOffset,
            const void * srcHost,
            size_t count )
```

Copies data between host and device.

Parameters

in	*dstArray*	Destination memory address
in	*dstOffset*	Offset in bytes of destination array
in	*srcHost*	Source host pointer
in	*count*	Size of memory copy in bytes

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.48 hipMemcpyHtoD()

```
hipError_t hipMemcpyHtoD (
            hipDeviceptr_t dst,
            void * src,
            size_t sizeBytes )
```

Copy data from Host to Device.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem↩ HostGetDevicePointer

4.9.2.49 hipMemcpyHtoDAsync()

```
hipError_t hipMemcpyHtoDAsync (
            hipDeviceptr_t dst,
            void * src,
            size_t sizeBytes,
            hipStream_t stream )
```

Copy data from Host to Device asynchronously.

Parameters

out	*dst*	Data being copy to
in	*src*	Data being copy from
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorDeInitialized, #hipErrorNotInitialized, #hipErrorInvalidContext, #hipErrorInvalidValue

See also

hipArrayCreate, hipArrayDestroy, hipArrayGetDescriptor, hipMemAlloc, hipMemAllocHost, hipMemAllocPitch, hipMemcpy2D, hipMemcpy2DAsync, hipMemcpy2DUnaligned, hipMemcpyAtoA, hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD, hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAsync, hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo, hipMemHostAlloc, hipMem↩ HostGetDevicePointer

4.9.2.50 hipMemcpyParam2D()

```
hipError_t hipMemcpyParam2D (
            const hip_Memcpy2D * pCopy )
```

Copies memory for 2D arrays.

Parameters

in	*pCopy*	Parameters for the memory copy

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩ InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2D, hipMemcpyToArray, hipMemcpy2DToArray, hipMemcpyFromArray, hipMemcpy↩ ToSymbol, hipMemcpyAsync

4.9.2.51 hipMemcpyParam2DAsync()

```
hipError_t hipMemcpyParam2DAsync (
            const hip_Memcpy2D * pCopy,
            hipStream_t stream  __dparm0 )
```

Copies memory for 2D arrays.

Parameters

in	*pCopy*	Parameters for the memory copy
in	*stream*	Stream to use

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩
InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2D, hipMemcpyToArray, hipMemcpy2DToArray, hipMemcpyFromArray, hipMemcpy↩
ToSymbol, hipMemcpyAsync

4.9.2.52 hipMemcpyToArray()

```
hipError_t hipMemcpyToArray (
            hipArray * dst,
            size_t wOffset,
            size_t hOffset,
            const void * src,
            size_t count,
            hipMemcpyKind kind )
```

Copies data between host and device.

Parameters

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

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidPitchValue, #hipErrorInvalidDevicePointer, #hipError↩
InvalidMemcpyDirection

See also

hipMemcpy, hipMemcpy2DToArray, hipMemcpy2D, hipMemcpyFromArray, hipMemcpyToSymbol, hipMemcpyAsync

4.9.2.53 hipMemGetInfo()

```
hipError_t hipMemGetInfo (
            size_t * free,
            size_t * total )
```

Query memory info. Return snapshot of free memory, and total allocatable memory on the device.
Returns in *free a snapshot of the current free memory.

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

Warning

On HCC, the free memory only accounts for memory allocated by this process and may be optimistic.

4.9.2.54 hipMemPrefetchAsync()

```
hipError_t hipMemPrefetchAsync (
            const void * dev_ptr,
            size_t count,
            int device,
            hipStream_t stream __dparm0 )
```
Prefetches memory to the specified destination device using HIP.

Parameters

in	*dev_ptr*	pointer to be prefetched
in	*count*	size in bytes for prefetching
in	*device*	destination device to prefetch to
in	*stream*	stream to enqueue prefetch operation

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.55 hipMemRangeGetAttribute()

```
hipError_t hipMemRangeGetAttribute (
            void * data,
            size_t data_size,
            hipMemRangeAttribute attribute,
            const void * dev_ptr,
            size_t count )
```
Query an attribute of a given memory range in HIP.

Parameters

in,out	*data*	a pointer to a memory location where the result of each attribute query will be written to
in	*data_size*	the size of data
in	*attribute*	the attribute to query
in	*dev_ptr*	start of the range to query
in	*count*	size of the range to query

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.56 hipMemRangeGetAttributes()

```
hipError_t hipMemRangeGetAttributes (
            void ** data,
            size_t * data_sizes,
            hipMemRangeAttribute * attributes,
            size_t num_attributes,
            const void * dev_ptr,
            size_t count )
```
Query attributes of a given memory range in HIP.

Parameters

in,out	*data*	a two-dimensional array containing pointers to memory locations where the result of each attribute query will be written to
in	*data_sizes*	an array, containing the sizes of each result
in	*attributes*	the attribute to query
in	*num_attributes*	an array of attributes to query (numAttributes and the number of attributes in this array should match)
in	*dev_ptr*	start of the range to query
in	*count*	size of the range to query

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.57 hipMemset()

```
hipError_t hipMemset (
            void * dst,
            int value,
            size_t sizeBytes )
```

Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value.

Parameters

out	*dst*	Data being filled
in	*constant*	value to be set
in	*sizeBytes*	Data size in bytes

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.58 hipMemset2D()

```
hipError_t hipMemset2D (
            void * dst,
            size_t pitch,
            int value,
            size_t width,
            size_t height )
```

Fills the memory area pointed to by dst with the constant value.

Parameters

out	*dst*	Pointer to device memory
in	*pitch*	- data size in bytes
in	*value*	- constant value to be set
in	*width*	
in	*height*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.59 hipMemset2DAsync()

```
hipError_t hipMemset2DAsync (
            void * dst,
            size_t pitch,
            int value,
            size_t width,
            size_t height,
            hipStream_t stream  __dparm0 )
```

Fills asynchronously the memory area pointed to by dst with the constant value.

Parameters

in	*dst*	Pointer to device memory
in	*pitch*	- data size in bytes
in	*value*	- constant value to be set
in	*width*	
in	*height*	
in	*stream*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.60 hipMemset3D()

```
hipError_t hipMemset3D (
            hipPitchedPtr pitchedDevPtr,
            int value,
            hipExtent extent )
```

Fills synchronously the memory area pointed to by pitchedDevPtr with the constant value.

Parameters

in	*pitchedDevPtr*	
in	*value*	- constant value to be set
in	*extent*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.61 hipMemset3DAsync()

```
hipError_t hipMemset3DAsync (
            hipPitchedPtr pitchedDevPtr,
            int value,
            hipExtent extent,
            hipStream_t stream  __dparm0 )
```

Fills asynchronously the memory area pointed to by pitchedDevPtr with the constant value.

Parameters

in	*pitchedDevPtr*	
in	*value*	- constant value to be set
in	*extent*	
in	*stream*	

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.62 hipMemsetAsync()

```
hipError_t hipMemsetAsync (
            void * dst,
            int value,
            size_t sizeBytes,
            hipStream_t stream  __dparm0 )
```

Fills the first sizeBytes bytes of the memory area pointed to by dev with the constant byte value value.
hipMemsetAsync() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero stream argument. If stream is non-zero, the operation may overlap with operations in other streams.

Parameters

out	*dst*	Pointer to device memory
in	*value*	- Value to set for each byte of specified memory
in	*sizeBytes*	- Size in bytes to set
in	*stream*	- Stream identifier

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.63 hipMemsetD16()

```
hipError_t hipMemsetD16 (
            hipDeviceptr_t dest,
            unsigned short value,
            size_t count )
```

Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant short value value.

Parameters

out	*dst*	Data ptr to be filled
in	*constant*	value to be set
in	*number*	of values to be set

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.64 hipMemsetD16Async()

```
hipError_t hipMemsetD16Async (
            hipDeviceptr_t dest,
            unsigned short value,
            size_t count,
            hipStream_t stream  __dparm0 )
```

Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant short value value.
hipMemsetD16Async() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero stream argument. If stream is non-zero, the operation may overlap with operations in other streams.

Parameters

out	*dst*	Data ptr to be filled
in	*constant*	value to be set
in	*number*	of values to be set
in	*stream*	- Stream identifier

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.65 hipMemsetD32()

```
hipError_t hipMemsetD32 (
            hipDeviceptr_t dest,
            int value,
            size_t count )
```

Fills the memory area pointed to by dest with the constant integer value for specified number of times.

Parameters

out	*dst*	Data being filled
in	*constant*	value to be set
in	*number*	of values to be set

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.66 hipMemsetD32Async()

```
hipError_t hipMemsetD32Async (
            hipDeviceptr_t dst,
            int value,
            size_t count,
            hipStream_t stream  __dparm0 )
```

Fills the memory area pointed to by dev with the constant integer value for specified number of times.
hipMemsetD32Async() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero stream argument. If stream is non-zero, the operation may overlap with operations in other streams.

Parameters

out	*dst*	Pointer to device memory
in	*value*	- Value to set for each byte of specified memory
in	*count*	- number of values to be set
in	*stream*	- Stream identifier

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorMemoryFree

4.9.2.67 hipMemsetD8()

```
hipError_t hipMemsetD8 (
            hipDeviceptr_t dest,
            unsigned char value,
            size_t count )
```

Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value.

Parameters

out	*dst*	Data ptr to be filled
in	*constant*	value to be set
in	*number*	of values to be set

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.68 hipMemsetD8Async()

```
hipError_t hipMemsetD8Async (
            hipDeviceptr_t dest,
            unsigned char value,
            size_t count,
            hipStream_t stream  __dparm0 )
```

Fills the first sizeBytes bytes of the memory area pointed to by dest with the constant byte value value. hipMemsetD8Async() is asynchronous with respect to the host, so the call may return before the memset is complete. The operation can optionally be associated to a stream by passing a non-zero stream argument. If stream is non-zero, the operation may overlap with operations in other streams.

Parameters

out	*dst*	Data ptr to be filled
in	*constant*	value to be set
in	*number*	of values to be set
in	*stream*	- Stream identifier

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotInitialized

4.9.2.69 hipModuleGetGlobal()

```
hipError_t hipModuleGetGlobal (
            hipDeviceptr_t * dptr,
            size_t * bytes,
            hipModule_t hmod,
            const char * name )
```

Returns a global pointer from a module. Returns in ∗dptr and ∗bytes the pointer and size of the global of name name located in module hmod. If no variable of that name exists, it returns hipErrorNotFound. Both parameters dptr and bytes are optional. If one of them is NULL, it is ignored and hipSuccess is returned.

Parameters

out	*dptr*	Returned global device pointer
out	*bytes*	Returned global size in bytes
in	*hmod*	Module to retrieve global from
in	*name*	Name of global to retrieve

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorNotFound, #hipErrorInvalidContext

4.9.2.70 hipPointerGetAttributes()

```
hipError_t hipPointerGetAttributes (
            hipPointerAttribute_t * attributes,
            const void * ptr )
```

Return attributes for the specified pointer.

Parameters

out	*attributes*	for the specified pointer
in	*pointer*	to get attributes for

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

hipGetDeviceCount, hipGetDevice, hipSetDevice, hipChooseDevice

4.9.2.71 hipSignalExternalSemaphoresAsync()

```
hipError_t hipSignalExternalSemaphoresAsync (
            const hipExternalSemaphore_t * extSemArray,
            const hipExternalSemaphoreSignalParams * paramsArray,
            unsigned int numExtSems,
            hipStream_t stream )
```

Signals a set of external semaphore objects.

Parameters

in	*extSem_out*	External semaphores to be waited on

Parameters

in	*paramsArray*	Array of semaphore parameters
in	*numExtSems*	Number of semaphores to wait on
in	*stream*	Stream to enqueue the wait operations in

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.9.2.72 hipStreamAttachMemAsync()

```
hipError_t hipStreamAttachMemAsync (
            hipStream_t stream,
            void * dev_ptr,
            size_t length  __dparm0,
            unsigned int flags  __dparmhipMemAttachSingle )
```

Attach memory to a stream asynchronously in HIP.

Parameters

in	*stream*	- stream in which to enqueue the attach operation
in	*dev_ptr*	- pointer to memory (must be a pointer to managed memory or to a valid host-accessible region of system-allocated memory)
in	*length*	- length of memory (defaults to zero)
in	*flags*	- must be one of hipMemAttachGlobal, hipMemAttachHost or hipMemAttachSingle (defaults to hipMemAttachSingle)

Returns

#hipSuccess, #hipErrorInvalidValue

4.9.2.73 hipWaitExternalSemaphoresAsync()

```
hipError_t hipWaitExternalSemaphoresAsync (
            const hipExternalSemaphore_t * extSemArray,
            const hipExternalSemaphoreWaitParams * paramsArray,
            unsigned int numExtSems,
            hipStream_t stream )
```

Waits on a set of external semaphore objects.

Parameters

in	*extSem_out*	External semaphores to be waited on
in	*paramsArray*	Array of semaphore parameters
in	*numExtSems*	Number of semaphores to wait on
in	*stream*	Stream to enqueue the wait operations in

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue

See also

4.10 PeerToPeer Device Memory Access

Macros

- #define **USE_PEER_NON_UNIFIED** 1

Functions

- hipError_t hipDeviceCanAccessPeer (int ∗canAccessPeer, int deviceId, int peerDeviceId)

  Determine if a device can access a peer's memory.
- hipError_t hipDeviceEnablePeerAccess (int peerDeviceId, unsigned int flags)

  Enable direct access from current device's virtual address space to memory allocations physically located on a peer device.
- hipError_t hipDeviceDisablePeerAccess (int peerDeviceId)

  Disable direct access from current device's virtual address space to memory allocations physically located on a peer device.
- hipError_t hipMemGetAddressRange (hipDeviceptr_t ∗pbase, size_t ∗psize, hipDeviceptr_t dptr)

  Get information on memory allocations.
- hipError_t hipMemcpyPeer (void ∗dst, int dstDeviceId, const void ∗src, int srcDeviceId, size_t sizeBytes)

  Copies memory from one device to memory on another device.
- hipError_t hipMemcpyPeerAsync (void ∗dst, int dstDeviceId, const void ∗src, int srcDevice, size_t sizeBytes, hipStream_t stream __dparm(0))

  Copies memory from one device to memory on another device.

4.10.1 Detailed Description

Warning

PeerToPeer support is experimental. This section describes the PeerToPeer device memory access functions of HIP runtime API.

4.10.2 Function Documentation

4.10.2.1 hipDeviceCanAccessPeer()

```
hipError_t hipDeviceCanAccessPeer (
            int * canAccessPeer,
            int deviceId,
            int peerDeviceId )
```

Determine if a device can access a peer's memory.

Parameters

out	*canAccessPeer*	Returns the peer access capability (0 or 1)
in	*device*	- device from where memory may be accessed.
in	*peerDevice*	- device where memory is physically located

Returns "1" in `canAccessPeer` if the specified `device` is capable of directly accessing memory physically located on peerDevice , or "0" if not.

Returns "0" in `canAccessPeer` if deviceId == peerDeviceId, and both are valid devices : a device is not a peer of itself.

Returns

#hipSuccess,

#hipErrorInvalidDevice if deviceId or peerDeviceId are not valid devices

4.10.2.2 hipDeviceDisablePeerAccess()

```
hipError_t hipDeviceDisablePeerAccess (
            int peerDeviceId )
```

Disable direct access from current device's virtual address space to memory allocations physically located on a peer device.

Returns hipErrorPeerAccessNotEnabled if direct access to memory on peerDevice has not yet been enabled from the current device.

Parameters

in	*peer↩ DeviceId*	

Returns

#hipSuccess, #hipErrorPeerAccessNotEnabled

4.10.2.3 hipDeviceEnablePeerAccess()

```
hipError_t hipDeviceEnablePeerAccess (
            int peerDeviceId,
            unsigned int flags )
```

Enable direct access from current device's virtual address space to memory allocations physically located on a peer device.

Memory which already allocated on peer device will be mapped into the address space of the current device. In addition, all future memory allocations on peerDeviceId will be mapped into the address space of the current device when the memory is allocated. The peer memory remains accessible from the current device until a call to hip↩ DeviceDisablePeerAccess or hipDeviceReset.

Parameters

in	*peer↩ DeviceId*	
in	*flags*	Returns #hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue,

Returns

#hipErrorPeerAccessAlreadyEnabled if peer access is already enabled for this device.

4.10.2.4 hipMemcpyPeer()

```
hipError_t hipMemcpyPeer (
            void * dst,
            int dstDeviceId,
            const void * src,
            int srcDeviceId,
            size_t sizeBytes )
```

Copies memory from one device to memory on another device.

Parameters

out	*dst*	- Destination device pointer.

Parameters

in	*dst↩ DeviceId*	- Destination device
in	*src*	- Source device pointer
in	*src↩ DeviceId*	- Source device
in	*sizeBytes*	- Size of memory copy in bytes

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidDevice

4.10.2.5 hipMemcpyPeerAsync()

```
hipError_t hipMemcpyPeerAsync (
            void * dst,
            int dstDeviceId,
            const void * src,
            int srcDevice,
            size_t sizeBytes,
            hipStream_t stream  __dparm0 )
```

Copies memory from one device to memory on another device.

Parameters

out	*dst*	- Destination device pointer.
in	*dstDevice*	- Destination device
in	*src*	- Source device pointer
in	*srcDevice*	- Source device
in	*sizeBytes*	- Size of memory copy in bytes
in	*stream*	- Stream identifier

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidDevice

4.10.2.6 hipMemGetAddressRange()

```
hipError_t hipMemGetAddressRange (
            hipDeviceptr_t * pbase,
            size_t * psize,
            hipDeviceptr_t dptr )
```

Get information on memory allocations.

Parameters

out	*pbase*	- BAse pointer address
out	*psize*	- Size of allocation
in	*dptr-*	Device Pointer

Returns

#hipSuccess, #hipErrorInvalidDevicePointer

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.11 Context Management

Modules

- Context Management [Deprecated]

Functions

- hipError_t hipDevicePrimaryCtxGetState (hipDevice_t dev, unsigned int ∗flags, int ∗active)

    Get the state of the primary context.
- hipError_t hipDevicePrimaryCtxRelease (hipDevice_t dev)

    Release the primary context on the GPU.
- hipError_t hipDevicePrimaryCtxRetain (hipCtx_t ∗pctx, hipDevice_t dev)

    Retain the primary context on the GPU.
- hipError_t hipDevicePrimaryCtxReset (hipDevice_t dev)

    Resets the primary context on the GPU.
- hipError_t hipDevicePrimaryCtxSetFlags (hipDevice_t dev, unsigned int flags)

    Set flags for the primary context.

4.11.1 Detailed Description

This section describes the context management functions of HIP runtime API.

4.11.2 Function Documentation

4.11.2.1 hipDevicePrimaryCtxGetState()

```
hipError_t hipDevicePrimaryCtxGetState (
              hipDevice_t dev,
              unsigned int * flags,
              int * active )
```

Get the state of the primary context.

Parameters

in	*Device*	to get primary context flags for
out	*Pointer*	to store flags
out	*Pointer*	to store context state; 0 = inactive, 1 = active

Returns

#hipSuccess

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.11.2.2 hipDevicePrimaryCtxRelease()

```
hipError_t hipDevicePrimaryCtxRelease (
              hipDevice_t dev )
```

Release the primary context on the GPU.

Parameters

`in`	*Device*	which primary context is released

Returns

#hipSuccess

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

Warning

This function return #hipSuccess though doesn't release the primaryCtx by design on HIP/HCC path.

4.11.2.3 hipDevicePrimaryCtxReset()

```
hipError_t hipDevicePrimaryCtxReset (
            hipDevice_t dev )
```

Resets the primary context on the GPU.

Parameters

`in`	*Device*	which primary context is reset

Returns

#hipSuccess

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.11.2.4 hipDevicePrimaryCtxRetain()

```
hipError_t hipDevicePrimaryCtxRetain (
            hipCtx_t * pctx,
            hipDevice_t dev )
```

Retain the primary context on the GPU.

Parameters

`out`	*Returned*	context handle of the new context
`in`	*Device*	which primary context is released

Returns

#hipSuccess

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.11.2.5 hipDevicePrimaryCtxSetFlags()

```
hipError_t hipDevicePrimaryCtxSetFlags (
            hipDevice_t dev,
            unsigned int flags )
```

Set flags for the primary context.

Parameters

in	*Device*	for which the primary context flags are set
in	*New*	flags for the device

Returns

#hipSuccess, #hipErrorContextAlreadyInUse

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.12 Module Management

Functions

- HIP_PUBLIC_API hipError_t hipExtModuleLaunchKernel (hipFunction_t f, uint32_t globalWorkSizeX, uint32_t globalWorkSizeY, uint32_t globalWorkSizeZ, uint32_t localWorkSizeX, uint32_t localWorkSizeY, uint32_t localWorkSizeZ, size_t sharedMemBytes, hipStream_t hStream, void ∗∗kernelParams, void ∗∗extra, hipEvent_t startEvent=nullptr, hipEvent_t stopEvent=nullptr, uint32_t flags=0)

  launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra
- HIP_PUBLIC_API hipError_t **hipHccModuleLaunchKernel** (hipFunction_t f, uint32_t globalWorkSizeX, uint32_t globalWorkSizeY, uint32_t globalWorkSizeZ, uint32_t localWorkSizeX, uint32_t localWork←SizeY, uint32_t localWorkSizeZ, size_t sharedMemBytes, hipStream_t hStream, void ∗∗kernelParams, void ∗∗extra, hipEvent_t startEvent=nullptr, hipEvent_t stopEvent=nullptr) __attribute__((deprecated("use hipExtModuleLaunchKernel instead")))
- hipError_t hipModuleLoad (hipModule_t ∗module, const char ∗fname)

  Loads code object from file into a hipModule_t.
- hipError_t hipModuleUnload (hipModule_t module)

  Frees the module.
- hipError_t hipModuleGetFunction (hipFunction_t ∗function, hipModule_t module, const char ∗kname)

  Function with kname will be extracted if present in module.
- hipError_t hipFuncGetAttributes (struct hipFuncAttributes ∗attr, const void ∗func)

  Find out attributes for a given function.
- hipError_t hipFuncGetAttribute (int ∗value, hipFunction_attribute attrib, hipFunction_t hfunc)

  Find out a specific attribute for a given function.
- hipError_t hipModuleGetTexRef (textureReference ∗∗texRef, hipModule_t hmod, const char ∗name)

  returns the handle of the texture reference with the name from the module.
- hipError_t hipModuleLoadData (hipModule_t ∗module, const void ∗image)

  builds module from code object which resides in host memory. Image is pointer to that location.
- hipError_t hipModuleLoadDataEx (hipModule_t ∗module, const void ∗image, unsigned int numOptions, hip←JitOption ∗options, void ∗∗optionValues)

  builds module from code object which resides in host memory. Image is pointer to that location. Options are not used. hipModuleLoadData is called.
- hipError_t hipModuleLaunchKernel (hipFunction_t f, unsigned int gridDimX, unsigned int gridDimY, unsigned int gridDimZ, unsigned int blockDimX, unsigned int blockDimY, unsigned int blockDimZ, unsigned int shared←MemBytes, hipStream_t stream, void ∗∗kernelParams, void ∗∗extra)

  launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra
- hipError_t hipLaunchCooperativeKernel (const void ∗f, dim3 gridDim, dim3 blockDimX, void ∗∗kernelParams, unsigned int sharedMemBytes, hipStream_t stream)

  launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra, where thread blocks can cooperate and synchronize as they execute
- hipError_t hipLaunchCooperativeKernelMultiDevice (hipLaunchParams ∗launchParamsList, int numDevices, unsigned int flags)

  Launches kernels on multiple devices where thread blocks can cooperate and synchronize as they execute.
- hipError_t hipExtLaunchMultiKernelMultiDevice (hipLaunchParams ∗launchParamsList, int numDevices, unsigned int flags)

  Launches kernels on multiple devices and guarantees all specified kernels are dispatched on respective streams before enqueuing any other work on the specified streams from any other threads.

4.12.1 Detailed Description

This section describes the module management functions of HIP runtime API.

4.12.2 Function Documentation

4.12.2.1 hipExtLaunchMultiKernelMultiDevice()

```
hipError_t hipExtLaunchMultiKernelMultiDevice (
             hipLaunchParams * launchParamsList,
             int numDevices,
             unsigned int flags )
```

Launches kernels on multiple devices and guarantees all specified kernels are dispatched on respective streams before enqueuing any other work on the specified streams from any other threads.

Parameters

in	*hipLaunchParams*	List of launch parameters, one per device.
in	*numDevices*	Size of the launchParamsList array.
in	*flags*	Flags to control launch behavior.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

4.12.2.2 hipExtModuleLaunchKernel()

```
HIP_PUBLIC_API hipError_t hipExtModuleLaunchKernel (
             hipFunction_t f,
             uint32_t globalWorkSizeX,
             uint32_t globalWorkSizeY,
             uint32_t globalWorkSizeZ,
             uint32_t localWorkSizeX,
             uint32_t localWorkSizeY,
             uint32_t localWorkSizeZ,
             size_t sharedMemBytes,
             hipStream_t hStream,
             void ** kernelParams,
             void ** extra,
             hipEvent_t startEvent = nullptr,
             hipEvent_t stopEvent = nullptr,
             uint32_t flags = 0 )
```

launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra

Parameters

	[in[	f Kernel to launch.
in	*gridDimX*	X grid dimension specified in work-items
in	*gridDimY*	Y grid dimension specified in work-items
in	*gridDimZ*	Z grid dimension specified in work-items
in	*blockDimX*	X block dimensions specified in work-items
in	*blockDimY*	Y grid dimension specified in work-items
in	*blockDimZ*	Z grid dimension specified in work-items
in	*sharedMemBytes*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations
in	*stream*	Stream where the kernel should be dispatched. May be 0, in which case th default stream is used with associated synchronization rules.

Parameters

in	*kernelParams*	
in	*extra*	Pointer to kernel arguments. These are passed directly to the kernel and must be in the memory layout and alignment expected by the kernel.
in	*startEvent*	If non-null, specified event will be updated to track the start time of the kernel launch. The event must be created before calling this API.
in	*stopEvent*	If non-null, specified event will be updated to track the stop time of the kernel launch. The event must be created before calling this API.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

Warning

kernellParams argument is not yet implemented in HIP. Please use extra instead. Please refer to hip_↩ porting_driver_api.md for sample usage. HIP/ROCm actually updates the start event when the associated kernel completes. Currently, timing between startEvent and stopEvent does not include the time it takes to perform a system scope release / cache flush - only the time it takes to issues writes to cache.

4.12.2.3 hipFuncGetAttribute()

```
hipError_t hipFuncGetAttribute (
            int * value,
            hipFunction_attribute attrib,
            hipFunction_t hfunc )
```

Find out a specific attribute for a given function.

Parameters

out	*value*	
in	*attrib*	
in	*hfunc*	

Returns

hipSuccess, hipErrorInvalidValue, hipErrorInvalidDeviceFunction

4.12.2.4 hipFuncGetAttributes()

```
hipError_t hipFuncGetAttributes (
            struct hipFuncAttributes * attr,
            const void * func )
```

Find out attributes for a given function.

Parameters

out	*attr*	
in	*func*	

Returns

hipSuccess, hipErrorInvalidValue, hipErrorInvalidDeviceFunction

4.12.2.5 hipLaunchCooperativeKernel()

```
hipError_t hipLaunchCooperativeKernel (
            const void * f,
            dim3 gridDim,
            dim3 blockDimX,
            void ** kernelParams,
            unsigned int sharedMemBytes,
            hipStream_t stream )
```

launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra, where thread blocks can cooperate and synchronize as they execute

Parameters

in	*f*	Kernel to launch.
in	*gridDim*	Grid dimensions specified as multiple of blockDim.
in	*blockDim*	Block dimensions specified in work-items
in	*kernelParams*	A list of kernel arguments
in	*sharedMemBytes*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
in	*stream*	Stream where the kernel should be dispatched. May be 0, in which case th default stream is used with associated synchronization rules.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue, hipErrorCooperativeLaunchToo↩ Large

4.12.2.6 hipLaunchCooperativeKernelMultiDevice()

```
hipError_t hipLaunchCooperativeKernelMultiDevice (
            hipLaunchParams * launchParamsList,
            int numDevices,
            unsigned int flags )
```

Launches kernels on multiple devices where thread blocks can cooperate and synchronize as they execute.

Parameters

in	*launchParamsList*	List of launch parameters, one per device.
in	*numDevices*	Size of the launchParamsList array.
in	*flags*	Flags to control launch behavior.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue, hipErrorCooperativeLaunchToo←
Large

4.12.2.7 hipModuleGetFunction()

```
hipError_t hipModuleGetFunction (
            hipFunction_t * function,
            hipModule_t module,
            const char * kname )
```

Function with kname will be extracted if present in module.

Parameters

in	*module*	
in	*kname*	
out	*function*	

Returns

hipSuccess, hipErrorInvalidValue, hipErrorInvalidContext, hipErrorNotInitialized, hipErrorNotFound,

4.12.2.8 hipModuleGetTexRef()

```
hipError_t hipModuleGetTexRef (
            textureReference ** texRef,
            hipModule_t hmod,
            const char * name )
```

returns the handle of the texture reference with the name from the module.

Parameters

in	*hmod*	
in	*name*	
out	*texRef*	

Returns

hipSuccess, hipErrorNotInitialized, hipErrorNotFound, hipErrorInvalidValue

4.12.2.9 hipModuleLaunchKernel()

```
hipError_t hipModuleLaunchKernel (
            hipFunction_t f,
            unsigned int gridDimX,
            unsigned int gridDimY,
            unsigned int gridDimZ,
            unsigned int blockDimX,
            unsigned int blockDimY,
            unsigned int blockDimZ,
            unsigned int sharedMemBytes,
            hipStream_t stream,
            void ** kernelParams,
            void ** extra )
```

launches kernel f with launch parameters and shared memory on stream with arguments passed to kernelparams or extra

Parameters

in	*f*	Kernel to launch.
in	*gridDimX*	X grid dimension specified as multiple of blockDimX.
in	*gridDimY*	Y grid dimension specified as multiple of blockDimY.
in	*gridDimZ*	Z grid dimension specified as multiple of blockDimZ.
in	*blockDimX*	X block dimensions specified in work-items
in	*blockDimY*	Y grid dimension specified in work-items
in	*blockDimZ*	Z grid dimension specified in work-items
in	*sharedMemBytes*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
in	*stream*	Stream where the kernel should be dispatched. May be 0, in which case th default stream is used with associated synchronization rules.
in	*kernelParams*	
in	*extra*	Pointer to kernel arguments. These are passed directly to the kernel and must be in the memory layout and alignment expected by the kernel.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

Warning

kernellParams argument is not yet implemented in HIP. Please use extra instead. Please refer to hip_porting←_driver_api.md for sample usage.

4.12.2.10 hipModuleLoad()

```
hipError_t hipModuleLoad (
            hipModule_t * module,
            const char * fname )
```

Loads code object from file into a hipModule_t.

Parameters

in	*fname*	
out	*module*	

Returns

hipSuccess, hipErrorInvalidValue, hipErrorInvalidContext, hipErrorFileNotFound, hipErrorOutOfMemory, hip←ErrorSharedObjectInitFailed, hipErrorNotInitialized

4.12.2.11 hipModuleLoadData()

```
hipError_t hipModuleLoadData (
            hipModule_t * module,
            const void * image )
```

builds module from code object which resides in host memory. Image is pointer to that location.

Parameters

`in`	*image*	
`out`	*module*	

Returns

hipSuccess, hipErrorNotInitialized, hipErrorOutOfMemory, hipErrorNotInitialized

4.12.2.12 hipModuleLoadDataEx()

```
hipError_t hipModuleLoadDataEx (
            hipModule_t * module,
            const void * image,
            unsigned int numOptions,
            hipJitOption * options,
            void ** optionValues )
```

builds module from code object which resides in host memory. Image is pointer to that location. Options are not used. hipModuleLoadData is called.

Parameters

`in`	*image*	
`out`	*module*	
`in`	*number*	of options
`in`	*options*	for JIT
`in`	*option*	values for JIT

Returns

hipSuccess, hipErrorNotInitialized, hipErrorOutOfMemory, hipErrorNotInitialized

4.12.2.13 hipModuleUnload()

```
hipError_t hipModuleUnload (
            hipModule_t module )
```

Frees the module.

Parameters

`in`	*module*	

Returns

hipSuccess, hipInvalidValue module is freed and the code objects associated with it are destroyed

4.13 Occupancy

Functions

- hipError_t hipModuleOccupancyMaxPotentialBlockSize (int ∗gridSize, int ∗blockSize, hipFunction_t f, size_t dynSharedMemPerBlk, int blockSizeLimit)

  determine the grid and block sizes to achieves maximum occupancy for a kernel
- hipError_t hipModuleOccupancyMaxPotentialBlockSizeWithFlags (int ∗gridSize, int ∗blockSize, hip←Function_t f, size_t dynSharedMemPerBlk, int blockSizeLimit, unsigned int flags)

  determine the grid and block sizes to achieves maximum occupancy for a kernel
- hipError_t hipModuleOccupancyMaxActiveBlocksPerMultiprocessor (int ∗numBlocks, hipFunction_t f, int blockSize, size_t dynSharedMemPerBlk)

  Returns occupancy for a device function.
- hipError_t hipModuleOccupancyMaxActiveBlocksPerMultiprocessorWithFlags (int ∗numBlocks, hip←Function_t f, int blockSize, size_t dynSharedMemPerBlk, unsigned int flags)

  Returns occupancy for a device function.
- hipError_t hipOccupancyMaxActiveBlocksPerMultiprocessor (int ∗numBlocks, const void ∗f, int blockSize, size_t dynSharedMemPerBlk)

  Returns occupancy for a device function.
- hipError_t hipOccupancyMaxActiveBlocksPerMultiprocessorWithFlags (int ∗numBlocks, const void ∗f, int blockSize, size_t dynSharedMemPerBlk, unsigned int flags __dparm(hipOccupancyDefault))

  Returns occupancy for a device function.
- hipError_t hipOccupancyMaxPotentialBlockSize (int ∗gridSize, int ∗blockSize, const void ∗f, size_t dyn←SharedMemPerBlk, int blockSizeLimit)

  determine the grid and block sizes to achieves maximum occupancy for a kernel

4.13.1 Detailed Description

This section describes the occupancy functions of HIP runtime API.

4.13.2 Function Documentation

4.13.2.1 hipModuleOccupancyMaxActiveBlocksPerMultiprocessor()

```
hipError_t hipModuleOccupancyMaxActiveBlocksPerMultiprocessor (
             int * numBlocks,
             hipFunction_t f,
             int blockSize,
             size_t dynSharedMemPerBlk )
```

Returns occupancy for a device function.

Parameters

out	*numBlocks*	Returned occupancy
in	*func*	Kernel function (hipFunction) for which occupancy is calulated
in	*blockSize*	Block size the kernel is intended to be launched with
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block

4.13.2.2 hipModuleOccupancyMaxActiveBlocksPerMultiprocessorWithFlags()

```
hipError_t hipModuleOccupancyMaxActiveBlocksPerMultiprocessorWithFlags (
             int * numBlocks,
             hipFunction_t f,
             int blockSize,
             size_t dynSharedMemPerBlk,
             unsigned int flags )
```

Returns occupancy for a device function.

Parameters

out	*numBlocks*	Returned occupancy
in	*f*	Kernel function(hipFunction_t) for which occupancy is calulated
in	*blockSize*	Block size the kernel is intended to be launched with
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block
in	*flags*	Extra flags for occupancy calculation (only default supported)

4.13.2.3 hipModuleOccupancyMaxPotentialBlockSize()

```
hipError_t hipModuleOccupancyMaxPotentialBlockSize (
            int * gridSize,
            int * blockSize,
            hipFunction_t f,
            size_t dynSharedMemPerBlk,
            int blockSizeLimit )
```

determine the grid and block sizes to achieves maximum occupancy for a kernel

Parameters

out	*gridSize*	minimum grid size for maximum potential occupancy
out	*blockSize*	block size for maximum potential occupancy
in	*f*	kernel function for which occupancy is calulated
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block
in	*blockSizeLimit*	the maximum block size for the kernel, use 0 for no limit

Returns

hipSuccess, hipInvalidDevice, hipErrorInvalidValue

4.13.2.4 hipModuleOccupancyMaxPotentialBlockSizeWithFlags()

```
hipError_t hipModuleOccupancyMaxPotentialBlockSizeWithFlags (
            int * gridSize,
            int * blockSize,
            hipFunction_t f,
            size_t dynSharedMemPerBlk,
            int blockSizeLimit,
            unsigned int flags )
```

determine the grid and block sizes to achieves maximum occupancy for a kernel

Parameters

out	*gridSize*	minimum grid size for maximum potential occupancy
out	*blockSize*	block size for maximum potential occupancy
in	*f*	kernel function for which occupancy is calulated

Parameters

in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block
in	*blockSizeLimit*	the maximum block size for the kernel, use 0 for no limit
in	*flags*	Extra flags for occupancy calculation (only default supported)

Returns

hipSuccess, hipInvalidDevice, hipErrorInvalidValue

4.13.2.5 hipOccupancyMaxActiveBlocksPerMultiprocessor()

```
hipError_t hipOccupancyMaxActiveBlocksPerMultiprocessor (
            int * numBlocks,
            const void * f,
            int blockSize,
            size_t dynSharedMemPerBlk )
```

Returns occupancy for a device function.

Parameters

out	*numBlocks*	Returned occupancy
in	*func*	Kernel function for which occupancy is calulated
in	*blockSize*	Block size the kernel is intended to be launched with
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block

4.13.2.6 hipOccupancyMaxActiveBlocksPerMultiprocessorWithFlags()

```
hipError_t hipOccupancyMaxActiveBlocksPerMultiprocessorWithFlags (
            int * numBlocks,
            const void * f,
            int blockSize,
            size_t dynSharedMemPerBlk,
            unsigned int flags  __dparmhipOccupancyDefault )
```

Returns occupancy for a device function.

Parameters

out	*numBlocks*	Returned occupancy
in	*f*	Kernel function for which occupancy is calulated
in	*blockSize*	Block size the kernel is intended to be launched with
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block
in	*flags*	Extra flags for occupancy calculation (currently ignored)

4.13.2.7 hipOccupancyMaxPotentialBlockSize()

```
hipError_t hipOccupancyMaxPotentialBlockSize (
            int * gridSize,
            int * blockSize,
            const void * f,
            size_t dynSharedMemPerBlk,
            int blockSizeLimit )
```

determine the grid and block sizes to achieves maximum occupancy for a kernel

Parameters

out	*gridSize*	minimum grid size for maximum potential occupancy
out	*blockSize*	block size for maximum potential occupancy
in	*f*	kernel function for which occupancy is calulated
in	*dynSharedMemPerBlk*	dynamic shared memory usage (in bytes) intended for each block
in	*blockSizeLimit*	the maximum block size for the kernel, use 0 for no limit

Returns

hipSuccess, hipInvalidDevice, hipErrorInvalidValue

4.14 Profiler Control[Deprecated]

Functions

- hipError_t hipProfilerStart ()

  Start recording of profiling information When using this API, start the profiler with profiling disabled. (–startdisabled)
- hipError_t hipProfilerStop ()

  Stop recording of profiling information. When using this API, start the profiler with profiling disabled. (–startdisabled)

4.14.1 Detailed Description

This section describes the profiler control functions of HIP runtime API.

Warning

The cudaProfilerInitialize API format for "configFile" is not supported.

4.14.2 Function Documentation

4.14.2.1 hipProfilerStart()

```
hipError_t hipProfilerStart ( )
```

Start recording of profiling information When using this API, start the profiler with profiling disabled. (–startdisabled)

Warning

: hipProfilerStart API is under development.

4.14.2.2 hipProfilerStop()

```
hipError_t hipProfilerStop ( )
```

Stop recording of profiling information. When using this API, start the profiler with profiling disabled. (–startdisabled)

Warning

: hipProfilerStop API is under development.

4.15 Launch API to support the triple-chevron syntax

Functions

- hipError_t hipConfigureCall (dim3 gridDim, dim3 blockDim, size_t sharedMem __dparm(0), hipStream_↩ t stream __dparm(0))

    Configure a kernel launch.
- hipError_t hipSetupArgument (const void ∗arg, size_t size, size_t offset)

    Set a kernel argument.
- hipError_t hipLaunchByPtr (const void ∗func)

    Launch a kernel.
- hipError_t __hipPushCallConfiguration (dim3 gridDim, dim3 blockDim, size_t sharedMem __dparm(0), hip↩ Stream_t stream __dparm(0))

    Push configuration of a kernel launch.
- hipError_t __hipPopCallConfiguration (dim3 ∗gridDim, dim3 ∗blockDim, size_t ∗sharedMem, hipStream_↩ t ∗stream)

    Pop configuration of a kernel launch.
- hipError_t hipLaunchKernel (const void ∗function_address, dim3 numBlocks, dim3 dimBlocks, void ∗∗args, size_t sharedMemBytes __dparm(0), hipStream_t stream __dparm(0))

    C compliant kernel launch API.
- hipError_t hipDrvMemcpy2DUnaligned (const hip_Memcpy2D ∗pCopy)
- hipError_t **hipExtLaunchKernel** (const void ∗function_address, dim3 numBlocks, dim3 dimBlocks, void ∗∗args, size_t sharedMemBytes, hipStream_t stream, hipEvent_t startEvent, hipEvent_t stopEvent, int flags)

4.15.1 Detailed Description

This section describes the API to support the triple-chevron syntax.

4.15.2 Function Documentation

4.15.2.1 __hipPopCallConfiguration()

```
hipError_t __hipPopCallConfiguration (
             dim3 * gridDim,
             dim3 * blockDim,
             size_t * sharedMem,
             hipStream_t * stream )
```

Pop configuration of a kernel launch.

Parameters

out	*gridDim*	grid dimension specified as multiple of blockDim.
out	*blockDim*	block dimensions specified in work-items
out	*sharedMem*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
out	*stream*	Stream where the kernel should be dispatched. May be 0, in which case the default stream is used with associated synchronization rules.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

4.15.2.2 __hipPushCallConfiguration()

```
hipError_t __hipPushCallConfiguration (
            dim3 gridDim,
            dim3 blockDim,
            size_t sharedMem  __dparm0,
            hipStream_t stream  __dparm0 )
```

Push configuration of a kernel launch.

Parameters

in	*gridDim*	grid dimension specified as multiple of blockDim.
in	*blockDim*	block dimensions specified in work-items
in	*sharedMem*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
in	*stream*	Stream where the kernel should be dispatched. May be 0, in which case the default stream is used with associated synchronization rules.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

4.15.2.3 hipConfigureCall()

```
hipError_t hipConfigureCall (
            dim3 gridDim,
            dim3 blockDim,
            size_t sharedMem  __dparm0,
            hipStream_t stream  __dparm0 )
```

Configure a kernel launch.

Parameters

in	*gridDim*	grid dimension specified as multiple of blockDim.
in	*blockDim*	block dimensions specified in work-items
in	*sharedMem*	Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
in	*stream*	Stream where the kernel should be dispatched. May be 0, in which case the default stream is used with associated synchronization rules.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

4.15.2.4 hipDrvMemcpy2DUnaligned()

```
hipError_t hipDrvMemcpy2DUnaligned (
            const hip_Memcpy2D * pCopy )
```

Copies memory for 2D arrays.

Parameters

pCopy	- Parameters for the memory copy

Returns

#hipSuccess, #hipErrorInvalidValue

4.15.2.5 hipLaunchByPtr()

```
hipError_t hipLaunchByPtr (
            const void * func )
```

Launch a kernel.

Parameters

in	*func*	Kernel to launch.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

4.15.2.6 hipLaunchKernel()

```
hipError_t hipLaunchKernel (
            const void * function_address,
            dim3 numBlocks,
            dim3 dimBlocks,
            void ** args,
            size_t sharedMemBytes  __dparm0,
            hipStream_t stream  __dparm0 )
```

C compliant kernel launch API.

Parameters

in	*function_address*	- kernel stub function pointer.
in	*numBlocks*	- number of blocks
in	*dimBlocks*	- dimension of a block
in	*args*	- kernel arguments
in	*sharedMemBytes*	- Amount of dynamic shared memory to allocate for this kernel. The HIP-Clang compiler provides support for extern shared declarations.
in	*stream*	- Stream where the kernel should be dispatched. May be 0, in which case th default stream is used with associated synchronization rules.

Returns

#hipSuccess, #hipErrorInvalidValue, hipInvalidDevice

4.15.2.7 hipSetupArgument()

```
hipError_t hipSetupArgument (
            const void * arg,
            size_t size,
            size_t offset )
```

Set a kernel argument.

Returns

hipSuccess, hipInvalidDevice, hipErrorNotInitialized, hipErrorInvalidValue

Parameters

in	*arg*	Pointer the argument in host memory.
in	*size*	Size of the argument.
in	*offset*	Offset of the argument on the argument stack.

4.16 Texture Management

Modules

- Texture Management [Deprecated]
- Texture Management [Not supported]

Functions

- hipError_t **hipBindTextureToMipmappedArray** (const textureReference ∗tex, hipMipmappedArray_const_t mipmappedArray, const hipChannelFormatDesc ∗desc)
- hipError_t **hipGetTextureReference** (const textureReference ∗∗texref, const void ∗symbol)
- hipError_t **hipCreateTextureObject** (hipTextureObject_t ∗pTexObject, const hipResourceDesc ∗pResDesc, const hipTextureDesc ∗pTexDesc, const struct hipResourceViewDesc ∗pResViewDesc)
- hipError_t **hipDestroyTextureObject** (hipTextureObject_t textureObject)
- hipError_t **hipGetChannelDesc** (hipChannelFormatDesc ∗desc, hipArray_const_t array)
- hipError_t **hipGetTextureObjectResourceDesc** (hipResourceDesc ∗pResDesc, hipTextureObject_↩ t textureObject)
- hipError_t **hipGetTextureObjectResourceViewDesc** (struct hipResourceViewDesc ∗pResViewDesc, hip↩ TextureObject_t textureObject)
- hipError_t **hipGetTextureObjectTextureDesc** (hipTextureDesc ∗pTexDesc, hipTextureObject_t texture↩ Object)
- hipError_t **hipTexRefSetAddressMode** (textureReference ∗texRef, int dim, enum hipTextureAddressMode am)
- hipError_t **hipTexRefSetArray** (textureReference ∗tex, hipArray_const_t array, unsigned int flags)
- hipError_t **hipTexRefSetFilterMode** (textureReference ∗texRef, enum hipTextureFilterMode fm)
- hipError_t **hipTexRefSetFlags** (textureReference ∗texRef, unsigned int Flags)
- hipError_t **hipTexRefSetFormat** (textureReference ∗texRef, hipArray_Format fmt, int NumPacked↩ Components)
- hipError_t **hipTexObjectCreate** (hipTextureObject_t ∗pTexObject, const HIP_RESOURCE_DESC ∗pRes↩ Desc, const HIP_TEXTURE_DESC ∗pTexDesc, const HIP_RESOURCE_VIEW_DESC ∗pResViewDesc)
- hipError_t **hipTexObjectDestroy** (hipTextureObject_t texObject)
- hipError_t **hipTexObjectGetResourceDesc** (HIP_RESOURCE_DESC ∗pResDesc, hipTextureObject_↩ t texObject)
- hipError_t **hipTexObjectGetResourceViewDesc** (HIP_RESOURCE_VIEW_DESC ∗pResViewDesc, hip↩ TextureObject_t texObject)
- hipError_t **hipTexObjectGetTextureDesc** (HIP_TEXTURE_DESC ∗pTexDesc, hipTextureObject_t tex↩ Object)

4.16.1 Detailed Description

This section describes the texture management functions of HIP runtime API.

4.17 Runtime Compilation

Typedefs

- typedef enum hiprtcResult **hiprtcResult**
- typedef struct _hiprtcProgram ∗ **hiprtcProgram**

Enumerations

- enum **hiprtcResult** {
HIPRTC_SUCCESS = 0, **HIPRTC_ERROR_OUT_OF_MEMORY** = 1, **HIPRTC_ERROR_PROGRAM_CREATION_FAILURE** = 2, **HIPRTC_ERROR_INVALID_INPUT** = 3,
HIPRTC_ERROR_INVALID_PROGRAM = 4, **HIPRTC_ERROR_INVALID_OPTION** = 5, **HIPRTC_ERROR_COMPILATION** = 6, **HIPRTC_ERROR_BUILTIN_OPERATION_FAILURE** = 7,
HIPRTC_ERROR_NO_NAME_EXPRESSIONS_AFTER_COMPILATION = 8, **HIPRTC_ERROR_NO_LOWERED_NAMES_BEFORE_COMPILATION** = 9, **HIPRTC_ERROR_NAME_EXPRESSION_NOT_VALID** = 10, **HIPRTC_ERROR_INTERNAL_ERROR** = 11,
HIPRTC_SUCCESS = 0, **HIPRTC_ERROR_OUT_OF_MEMORY** = 1, **HIPRTC_ERROR_PROGRAM_CREATION_FAILURE** = 2, **HIPRTC_ERROR_INVALID_INPUT** = 3,
HIPRTC_ERROR_INVALID_PROGRAM = 4, **HIPRTC_ERROR_INVALID_OPTION** = 5, **HIPRTC_ERROR_COMPILATION** = 6, **HIPRTC_ERROR_BUILTIN_OPERATION_FAILURE** = 7,
HIPRTC_ERROR_NO_NAME_EXPRESSIONS_AFTER_COMPILATION = 8, **HIPRTC_ERROR_NO_LOWERED_NAMES_BEFORE_COMPILATION** = 9, **HIPRTC_ERROR_NAME_EXPRESSION_NOT_VALID** = 10, **HIPRTC_ERROR_INTERNAL_ERROR** = 11,
HIPRTC_SUCCESS = 0, **HIPRTC_ERROR_OUT_OF_MEMORY** = 1, **HIPRTC_ERROR_PROGRAM_CREATION_FAILURE** = 2, **HIPRTC_ERROR_INVALID_INPUT** = 3,
HIPRTC_ERROR_INVALID_PROGRAM = 4, **HIPRTC_ERROR_INVALID_OPTION** = 5, **HIPRTC_ERROR_COMPILATION** = 6, **HIPRTC_ERROR_BUILTIN_OPERATION_FAILURE** = 7,
HIPRTC_ERROR_NO_NAME_EXPRESSIONS_AFTER_COMPILATION = 8, **HIPRTC_ERROR_NO_LOWERED_NAMES_BEFORE_COMPILATION** = 9, **HIPRTC_ERROR_NAME_EXPRESSION_NOT_VALID** = 10, **HIPRTC_ERROR_INTERNAL_ERROR** = 11 }

Functions

- const char ∗ hiprtcGetErrorString (hiprtcResult result)

  Returns text string message to explain the error which occurred.
- hiprtcResult hiprtcVersion (int ∗major, int ∗minor)

  Sets the parameters as major and minor version.
- hiprtcResult hiprtcAddNameExpression (hiprtcProgram prog, const char ∗name_expression)

  Adds the given name exprssion to the runtime compilation program.
- hiprtcResult hiprtcCompileProgram (hiprtcProgram prog, int numOptions, const char ∗∗options)

  Compiles the given runtime compilation program.
- hiprtcResult hiprtcCreateProgram (hiprtcProgram ∗prog, const char ∗src, const char ∗name, int numHeaders, const char ∗∗headers, const char ∗∗includeNames)

  Creates an instance of hiprtcProgram with the given input parameters, and sets the output hiprtcProgram prog with it.
- hiprtcResult hiprtcDestroyProgram (hiprtcProgram ∗prog)

  Destroys an instance of given hiprtcProgram.
- hiprtcResult hiprtcGetLoweredName (hiprtcProgram prog, const char ∗name_expression, const char ∗∗lowered_name)

  Gets the lowered (mangled) name from an instance of hiprtcProgram with the given input parameters, and sets the output lowered_name with it.
- hiprtcResult hiprtcGetProgramLog (hiprtcProgram prog, char ∗log)

  Gets the log generated by the runtime compilation program instance.
- hiprtcResult hiprtcGetProgramLogSize (hiprtcProgram prog, size_t ∗logSizeRet)

  Gets the size of log generated by the runtime compilation program instance.
- hiprtcResult hiprtcGetCode (hiprtcProgram prog, char ∗code)

Gets the pointer of compilation binary by the runtime compilation program instance.

- hiprtcResult hiprtcGetCodeSize (hiprtcProgram prog, size_t ∗codeSizeRet)

  Gets the size of compilation binary by the runtime compilation program instance.

4.17.1 Detailed Description

This section describes the runtime compilation functions of HIP runtime API.

4.17.2 Function Documentation

4.17.2.1 hiprtcAddNameExpression()

```
hiprtcResult hiprtcAddNameExpression (
            hiprtcProgram prog,
            const char * name_expression )
```

Adds the given name exprssion to the runtime compilation program.

Parameters

in	*prog*	runtime compilation program instance.
in	*name_expression*	const char pointer to the name expression.

Returns

HIPRTC_SUCCESS

If const char pointer is NULL, it will return HIPRTC_ERROR_INVALID_INPUT.

See also

hiprtcResult

4.17.2.2 hiprtcCompileProgram()

```
hiprtcResult hiprtcCompileProgram (
            hiprtcProgram prog,
            int numOptions,
            const char ** options )
```

Compiles the given runtime compilation program.

Parameters

in	*prog*	runtime compilation program instance.
in	*numOptions*	number of compiler options.
in	*options*	compiler options as const array of strins.

Returns

HIPRTC_SUCCESS

If the compiler failed to build the runtime compilation program, it will return HIPRTC_ERROR_COMPILATION.

See also

hiprtcResult

4.17.2.3 hiprtcCreateProgram()

```
hiprtcResult hiprtcCreateProgram (
            hiprtcProgram * prog,
            const char * src,
            const char * name,
            int numHeaders,
            const char ** headers,
            const char ** includeNames )
```

Creates an instance of hiprtcProgram with the given input parameters, and sets the output hiprtcProgram prog with it.

Parameters

in,out	*prog*	runtime compilation program instance.
in	*src*	const char pointer to the program source.
in	*name*	const char pointer to the program name.
in	*numHeaders*	number of headers.
in	*headers*	array of strings pointing to headers.
in	*includeNames*	array of strings pointing to names included in program source.

Returns

HIPRTC_SUCCESS

Any invalide input parameter, it will return HIPRTC_ERROR_INVALID_INPUT or HIPRTC_ERROR_INVALID_P↩ ROGRAM.
If failed to create the program, it will return HIPRTC_ERROR_PROGRAM_CREATION_FAILURE.

See also

hiprtcResult

4.17.2.4 hiprtcDestroyProgram()

```
hiprtcResult hiprtcDestroyProgram (
            hiprtcProgram * prog )
```

Destroys an instance of given hiprtcProgram.

Parameters

in	*prog*	runtime compilation program instance.

Returns

HIPRTC_SUCCESS

If prog is NULL, it will return HIPRTC_ERROR_INVALID_INPUT.

See also

hiprtcResult

4.17.2.5 hiprtcGetCode()

```
hiprtcResult hiprtcGetCode (
            hiprtcProgram prog,
            char * code )
```

Gets the pointer of compilation binary by the runtime compilation program instance.

Parameters

in	*prog*	runtime compilation program instance.
out	*code*	char pointer to binary.

Returns

HIPRTC_SUCCESS

See also

hiprtcResult

4.17.2.6 hiprtcGetCodeSize()

```
hiprtcResult hiprtcGetCodeSize (
             hiprtcProgram prog,
             size_t * codeSizeRet )
```

Gets the size of compilation binary by the runtime compilation program instance.

Parameters

in	*prog*	runtime compilation program instance.
out	*code*	the size of binary.

Returns

HIPRTC_SUCCESS

See also

hiprtcResult

4.17.2.7 hiprtcGetErrorString()

```
const char* hiprtcGetErrorString (
             hiprtcResult result )
```

Returns text string message to explain the error which occurred.

Parameters

in	*result*	code to convert to string.

Returns

const char pointer to the NULL-terminated error string

Warning

In HIP, this function returns the name of the error, if the hiprtc result is defined, it will return "Invalid HIPRTC error code"

See also

hiprtcResult

4.17.2.8 hiprtcGetLoweredName()

```
hiprtcResult hiprtcGetLoweredName (
            hiprtcProgram prog,
            const char * name_expression,
            const char ** lowered_name )
```

Gets the lowered (mangled) name from an instance of hiprtcProgram with the given input parameters, and sets the output lowered_name with it.

Parameters

in	*prog*	runtime compilation program instance.
in	*name_expression*	const char pointer to the name expression.
in,out	*lowered_name*	const char array to the lowered (mangled) name.

Returns

HIPRTC_SUCCESS

If any invalide nullptr input parameters, it will return HIPRTC_ERROR_INVALID_INPUT
If name_expression is not found, it will return HIPRTC_ERROR_NAME_EXPRESSION_NOT_VALID
If failed to get lowered_name from the program, it will return HIPRTC_ERROR_COMPILATION.

See also

hiprtcResult

4.17.2.9 hiprtcGetProgramLog()

```
hiprtcResult hiprtcGetProgramLog (
            hiprtcProgram prog,
            char * log )
```

Gets the log generated by the runtime compilation program instance.

Parameters

in	*prog*	runtime compilation program instance.
out	*log*	memory pointer to the generated log.

Returns

HIPRTC_SUCCESS

See also

hiprtcResult

4.17.2.10 hiprtcGetProgramLogSize()

```
hiprtcResult hiprtcGetProgramLogSize (
            hiprtcProgram prog,
            size_t * logSizeRet )
```

Gets the size of log generated by the runtime compilation program instance.

Parameters

`in`	*prog*	runtime compilation program instance.
`out`	*logSizeRet*	size of generated log.

Returns

HIPRTC_SUCCESS

See also

hiprtcResult

4.17.2.11 hiprtcVersion()

```
hiprtcResult hiprtcVersion (
            int * major,
            int * minor )
```

Sets the parameters as major and minor version.

Parameters

`out`	*major*	HIP Runtime Compilation major version.
`out`	*minor*	HIP Runtime Compilation minor version.

4.18 Callback Activity APIs

Functions

- hipError_t **hipRegisterApiCallback** (uint32_t id, void ∗fun, void ∗arg)
- hipError_t **hipRemoveApiCallback** (uint32_t id)
- hipError_t **hipRegisterActivityCallback** (uint32_t id, void ∗fun, void ∗arg)
- hipError_t **hipRemoveActivityCallback** (uint32_t id)
- const char ∗ **hipApiName** (uint32_t id)
- const char ∗ **hipKernelNameRef** (const hipFunction_t f)
- const char ∗ **hipKernelNameRefByPtr** (const void ∗hostFunction, hipStream_t stream)
- int **hipGetStreamDeviceId** (hipStream_t stream)

4.18.1 Detailed Description

This section describes the callback/Activity of HIP runtime API.

4.19 Graph Management

Classes

- struct hipHostNodeParams
- struct hipKernelNodeParams
- struct hipMemsetParams

Typedefs

- typedef struct ihipGraph ∗ hipGraph_t
- typedef struct hipGraphNode ∗ hipGraphNode_t
- typedef struct hipGraphExec ∗ hipGraphExec_t
- typedef enum hipGraphNodeType **hipGraphNodeType**
- typedef void(∗ **hipHostFn_t**) (void ∗userData)
- typedef struct hipHostNodeParams **hipHostNodeParams**
- typedef struct hipKernelNodeParams **hipKernelNodeParams**
- typedef struct hipMemsetParams **hipMemsetParams**
- typedef enum hipGraphExecUpdateResult **hipGraphExecUpdateResult**
- typedef enum hipStreamCaptureMode **hipStreamCaptureMode**
- typedef enum hipStreamCaptureStatus **hipStreamCaptureStatus**

Enumerations

- enum hipGraphNodeType {
  hipGraphNodeTypeKernel = 1, hipGraphNodeTypeMemcpy = 2, hipGraphNodeTypeMemset = 3, hipGraphNodeTypeHost = 4,
  hipGraphNodeTypeGraph = 5, hipGraphNodeTypeEmpty = 6, hipGraphNodeTypeWaitEvent = 7, hipGraphNodeTypeEventRecord = 8,
  hipGraphNodeTypeMemcpy1D = 9, hipGraphNodeTypeMemcpyFromSymbol = 10, hipGraphNodeTypeMemcpyToSymbol = 11, **hipGraphNodeTypeCount** }
- enum hipGraphExecUpdateResult {
  hipGraphExecUpdateSuccess = 0x0, hipGraphExecUpdateError = 0x1, hipGraphExecUpdateErrorTopologyChanged = 0x2, hipGraphExecUpdateErrorNodeTypeChanged = 0x3,
  hipGraphExecUpdateErrorFunctionChanged, hipGraphExecUpdateErrorParametersChanged, hipGraphExecUpdateErrorNotSu
  hipGraphExecUpdateErrorUnsupportedFunctionChange = 0x7 }
- enum **hipStreamCaptureMode** { **hipStreamCaptureModeGlobal** = 0, **hipStreamCaptureModeThread**↩ **Local**, **hipStreamCaptureModeRelaxed** }
- enum hipStreamCaptureStatus { hipStreamCaptureStatusNone = 0, hipStreamCaptureStatusActive, hipStreamCaptureStatusInvalidated }

Functions

- hipError_t hipStreamBeginCapture (hipStream_t stream, hipStreamCaptureMode mode)

  Begins graph capture on a stream.
- hipError_t hipStreamEndCapture (hipStream_t stream, hipGraph_t ∗pGraph)

  Ends capture on a stream, returning the captured graph.
- hipError_t hipGraphCreate (hipGraph_t ∗pGraph, unsigned int flags)

  Creates a graph.
- hipError_t hipGraphDestroy (hipGraph_t graph)

  Destroys a graph.
- hipError_t hipGraphExecDestroy (hipGraphExec_t pGraphExec)

  Destroys an executable graph.
- hipError_t hipGraphInstantiate (hipGraphExec_t ∗pGraphExec, hipGraph_t graph, hipGraphNode_t ∗p↩ ErrorNode, char ∗pLogBuffer, size_t bufferSize)

  Creates an executable graph from a graph.

- hipError_t hipGraphLaunch (hipGraphExec_t graphExec, hipStream_t stream)

  launches an executable graph in a stream
- hipError_t hipGraphAddKernelNode (hipGraphNode_t ∗pGraphNode, hipGraph_t graph, const hipGraphNode_t ∗pDependencies, size_t numDependencies, const hipKernelNodeParams ∗pNodeParams)

  Creates a kernel execution node and adds it to a graph.
- hipError_t hipGraphAddMemcpyNode (hipGraphNode_t ∗pGraphNode, hipGraph_t graph, const hipGraphNode_t ∗pDependencies, size_t numDependencies, const hipMemcpy3DParms ∗pCopyParams)

  Creates a memcpy node and adds it to a graph.
- hipError_t hipGraphAddMemcpyNode1D (hipGraphNode_t ∗pGraphNode, hipGraph_t graph, const hipGraphNode_t ∗pDependencies, size_t numDependencies, void ∗dst, const void ∗src, size_t count, hipMemcpyKind kind)

  Creates a 1D memcpy node and adds it to a graph.
- hipError_t hipGraphAddMemsetNode (hipGraphNode_t ∗pGraphNode, hipGraph_t graph, const hipGraphNode_t ∗pDependencies, size_t numDependencies, const hipMemsetParams ∗pMemsetParams)

  Creates a memset node and adds it to a graph.
- hipError_t hipGraphGetNodes (hipGraph_t graph, hipGraphNode_t ∗nodes, size_t ∗numNodes)

  Returns graph nodes.
- hipError_t hipGraphGetRootNodes (hipGraph_t graph, hipGraphNode_t ∗pRootNodes, size_t ∗pNumRoot↩ Nodes)

  Returns graph's root nodes.
- hipError_t hipGraphKernelNodeGetParams (hipGraphNode_t node, hipKernelNodeParams ∗pNodeParams)

  Gets kernel node's parameters.
- hipError_t hipGraphKernelNodeSetParams (hipGraphNode_t node, const hipKernelNodeParams ∗pNode↩ Params)

  Sets a kernel node's parameters.
- hipError_t hipGraphMemcpyNodeGetParams (hipGraphNode_t node, hipMemcpy3DParms ∗pNodeParams)

  Gets a memcpy node's parameters.
- hipError_t hipGraphMemcpyNodeSetParams (hipGraphNode_t node, const hipMemcpy3DParms ∗pNode↩ Params)

  Sets a memcpy node's parameters.
- hipError_t hipGraphMemsetNodeGetParams (hipGraphNode_t node, hipMemsetParams ∗pNodeParams)

  Gets a memset node's parameters.
- hipError_t hipGraphMemsetNodeSetParams (hipGraphNode_t node, const hipMemsetParams ∗pNode↩ Params)

  Sets a memset node's parameters.
- hipError_t hipGraphExecKernelNodeSetParams (hipGraphExec_t hGraphExec, hipGraphNode_t node, const hipKernelNodeParams ∗pNodeParams)

  Sets the parameters for a kernel node in the given graphExec.
- hipError_t hipGraphAddDependencies (hipGraph_t graph, const hipGraphNode_t ∗from, const hipGraphNode_t ∗to, size_t numDependencies)

  Adds dependency edges to a graph.
- hipError_t hipGraphAddEmptyNode (hipGraphNode_t ∗pGraphNode, hipGraph_t graph, const hipGraphNode_t ∗pDependencies, size_t numDependencies)

  Creates an empty node and adds it to a graph.

4.19.1 Detailed Description

This section describes the graph management types & functions of HIP runtime API.

4.19.2 Typedef Documentation

4.19.2.1 hipGraph_t

```
typedef struct ihipGraph* hipGraph_t
```

An opaque value that represents a hip graph

4.19.2.2 hipGraphExec_t

```
typedef struct hipGraphExec* hipGraphExec_t
```

An opaque value that represents a hip graph Exec

4.19.2.3 hipGraphNode_t

```
typedef struct hipGraphNode* hipGraphNode_t
```

An opaque value that represents a hip graph node

4.19.3 Enumeration Type Documentation

4.19.3.1 hipGraphExecUpdateResult

```
enum hipGraphExecUpdateResult
```

Enumerator

hipGraphExecUpdateSuccess	The update succeeded.
hipGraphExecUpdateError	The update failed for an unexpected reason which is described in the return value of the function
hipGraphExecUpdateErrorTopologyChanged	The update failed because the topology changed.
hipGraphExecUpdateErrorNodeTypeChanged	The update failed because a node type changed.
hipGraphExecUpdateErrorFunctionChanged	The update failed because the function of a kernel node changed.
hipGraphExecUpdateErrorParametersChanged	The update failed because the parameters changed in a way that is not supported.
hipGraphExecUpdateErrorNotSupported	The update failed because something about the node is not supported.

4.19.3.2 hipGraphNodeType

```
enum hipGraphNodeType
```

Enumerator

hipGraphNodeTypeKernel	GPU kernel node.
hipGraphNodeTypeMemcpy	Memcpy 3D node.
hipGraphNodeTypeMemset	Memset 1D node.
hipGraphNodeTypeHost	Host (executable) node.
hipGraphNodeTypeGraph	Node which executes an embedded graph.
hipGraphNodeTypeEmpty	Empty (no-op) node.
hipGraphNodeTypeWaitEvent	External event wait node.
hipGraphNodeTypeEventRecord	External event record node.
hipGraphNodeTypeMemcpy1D	Memcpy 1D node.
hipGraphNodeTypeMemcpyFromSymbol	MemcpyFromSymbol node.
hipGraphNodeTypeMemcpyToSymbol	MemcpyToSymbol node.

4.19.3.3 hipStreamCaptureStatus

```
enum hipStreamCaptureStatus
```

Enumerator

hipStreamCaptureStatusNone	Stream is not capturing.
hipStreamCaptureStatusActive	Stream is actively capturing.
hipStreamCaptureStatusInvalidated	Stream is part of a capture sequence that has been invalidated, but not terminated

4.19.4 Function Documentation

4.19.4.1 hipGraphAddDependencies()

```
hipError_t hipGraphAddDependencies (
            hipGraph_t graph,
            const hipGraphNode_t * from,
            const hipGraphNode_t * to,
            size_t numDependencies )
```

Adds dependency edges to a graph.

Parameters

in	*graph*	- instance of the graph to add dependencies.
in	*from*	- pointer to the graph nodes with dependenties to add from.
in	*to*	- pointer to the graph nodes to add dependenties to.
in	*numDependencies*	- the number of dependencies to add.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.2 hipGraphAddEmptyNode()

```
hipError_t hipGraphAddEmptyNode (
            hipGraphNode_t * pGraphNode,
            hipGraph_t graph,
            const hipGraphNode_t * pDependencies,
            size_t numDependencies )
```

Creates an empty node and adds it to a graph.

Parameters

out	*pGraphNode*	- pointer to the graph node to create and add to the graph.
in,out	*graph*	- instane of the graph the node is add to.

Parameters

in	*pDependencies*	- const pointer to the node dependenties.
in	*numDependencies*	- the number of dependencies.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.3 hipGraphAddKernelNode()

```
hipError_t hipGraphAddKernelNode (
            hipGraphNode_t * pGraphNode,
            hipGraph_t graph,
            const hipGraphNode_t * pDependencies,
            size_t numDependencies,
            const hipKernelNodeParams * pNodeParams )
```

Creates a kernel execution node and adds it to a graph.

Parameters

out	*pGraphNode*	- pointer to graph node to create.
in,out	*graph*	- instance of graph to add the created node.
in	*pDependencies*	- pointer to the dependencies on the kernel execution node.
in	*numDependencies*	- the number of the dependencies.
in	*pNodeParams*	- pointer to the parameters to the kernel execution node on the GPU.

Returns

#hipSuccess, #hipErrorInvalidValue, #hipErrorInvalidDeviceFunction

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.4 hipGraphAddMemcpyNode()

```
hipError_t hipGraphAddMemcpyNode (
            hipGraphNode_t * pGraphNode,
            hipGraph_t graph,
            const hipGraphNode_t * pDependencies,
            size_t numDependencies,
            const hipMemcpy3DParms * pCopyParams )
```

Creates a memcpy node and adds it to a graph.

Parameters

out	*pGraphNode*	- pointer to graph node to create.

Parameters

in,out	*graph*	- instance of graph to add the created node.
in	*pDependencies*	- const pointer to the dependencies on the kernel execution node.
in	*numDependencies*	- the number of the dependencies.
in	*pCopyParams*	- const pointer to the parameters for the memory copy.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.5 hipGraphAddMemcpyNode1D()

```
hipError_t hipGraphAddMemcpyNode1D (
            hipGraphNode_t * pGraphNode,
            hipGraph_t graph,
            const hipGraphNode_t * pDependencies,
            size_t numDependencies,
            void * dst,
            const void * src,
            size_t count,
            hipMemcpyKind kind )
```

Creates a 1D memcpy node and adds it to a graph.

Parameters

out	*pGraphNode*	- pointer to graph node to create.
in,out	*graph*	- instance of the graph to add the created node.
in	*pDependencies*	- const pointer to the dependencies on the kernel execution node.
in	*numDependencies*	- the number of the dependencies.
in	*dst*	- pointer to memory address to the destination.
in	*src*	- pointer to memory address to the source.
in	*count*	- the size of the memory to copy.
in	*kind*	- the type of memory copy.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.6 hipGraphAddMemsetNode()

```
hipError_t hipGraphAddMemsetNode (
            hipGraphNode_t * pGraphNode,
            hipGraph_t graph,
            const hipGraphNode_t * pDependencies,
            size_t numDependencies,
            const hipMemsetParams * pMemsetParams )
```

Creates a memset node and adds it to a graph.

Parameters

out	*pGraphNode*	- pointer to the graph node to create.
in,out	*graph*	- instance of the graph to add the created node.
in	*pDependencies*	- const pointer to the dependencies on the kernel execution node.
in	*numDependencies*	- the number of the dependencies.
in	*pMemsetParams*	- const pointer to the parameters for the memory set.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.7 hipGraphCreate()

```
hipError_t hipGraphCreate (
            hipGraph_t * pGraph,
            unsigned int flags )
```

Creates a graph.

Parameters

out	*pGraph*	- pointer to graph to create.
in	*flags*	- flags for graph creation, must be 0.

Returns

#hipSuccess.

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.8 hipGraphDestroy()

```
hipError_t hipGraphDestroy (
            hipGraph_t graph )
```

Destroys a graph.

Parameters

in	*graph*	- instance of graph to destroy.

Returns

#hipSuccess.

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.9 hipGraphExecDestroy()

```
hipError_t hipGraphExecDestroy (
            hipGraphExec_t pGraphExec )
```

Destroys an executable graph.

Parameters

in	*pGraphExec*	- instance of executable graph to destry.

Returns

#hipSuccess.

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.10 hipGraphExecKernelNodeSetParams()

```
hipError_t hipGraphExecKernelNodeSetParams (
            hipGraphExec_t hGraphExec,
            hipGraphNode_t node,
            const hipKernelNodeParams * pNodeParams )
```

Sets the parameters for a kernel node in the given graphExec.

Parameters

in	*hGraphExec*	- instance of the executable graph with the node.
in	*node*	- instance of the node to set parameters to.
in	*pNodeParams*	- const pointer to the kernel node parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.11 hipGraphGetNodes()

```
hipError_t hipGraphGetNodes (
            hipGraph_t graph,
            hipGraphNode_t * nodes,
            size_t * numNodes )
```

Returns graph nodes.

Parameters

in	*graph*	- instance of graph to get the nodes.
out	*nodes*	- pointer to the graph nodes.
out	*numNodes*	- the number of graph nodes.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.12 hipGraphGetRootNodes()

```
hipError_t hipGraphGetRootNodes (
            hipGraph_t graph,
            hipGraphNode_t * pRootNodes,
            size_t * pNumRootNodes )
```

Returns graph's root nodes.

Parameters

in	*graph*	- instance of the graph to get the nodes.
out	*pRootNodes*	- pointer to the graph's root nodes.
out	*pNumRootNodes*	- the number of graph's root nodes.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.13 hipGraphInstantiate()

```
hipError_t hipGraphInstantiate (
            hipGraphExec_t * pGraphExec,
            hipGraph_t graph,
            hipGraphNode_t * pErrorNode,
            char * pLogBuffer,
            size_t bufferSize )
```

Creates an executable graph from a graph.

Parameters

out	*pGraphExec*	- pointer to instantiated executable graph to create.
in	*graph*	- instance of graph to instantiate.
out	*pErrorNode*	- pointer to error node in case error occured in graph instantiation, it could modify the correponding node.
out	*pLogBuffer*	- pointer to log buffer.
in	*bufferSize*	- the size of log buffer.

Returns

#hipSuccess, #hipErrorOutOfMemory.

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.14 hipGraphKernelNodeGetParams()

```
hipError_t hipGraphKernelNodeGetParams (
            hipGraphNode_t node,
            hipKernelNodeParams * pNodeParams )
```

Gets kernel node's parameters.

Parameters

in	*node*	- instance of the node to get parameters from.
out	*pNodeParams*	- pointer to the parameters

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.15 hipGraphKernelNodeSetParams()

```
hipError_t hipGraphKernelNodeSetParams (
            hipGraphNode_t node,
            const hipKernelNodeParams * pNodeParams )
```

Sets a kernel node's parameters.

Parameters

in	*node*	- instance of the node to set parameters to.
in	*pNodeParams*	- const pointer to the parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.16 hipGraphLaunch()

```
hipError_t hipGraphLaunch (
            hipGraphExec_t graphExec,
            hipStream_t stream )
```

launches an executable graph in a stream

Parameters

in	*graphExec*	- instance of executable graph to launch.
in	*stream*	- instance of stream in which to launch executable graph.

Returns

#hipSuccess, #hipErrorOutOfMemory, #hipErrorInvalidHandle, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.17 hipGraphMemcpyNodeGetParams()

```
hipError_t hipGraphMemcpyNodeGetParams (
            hipGraphNode_t node,
            hipMemcpy3DParms * pNodeParams )
```

Gets a memcpy node's parameters.

Parameters

in	*node*	- instance of the node to get parameters from.
out	*pNodeParams*	- pointer to the parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.18 hipGraphMemcpyNodeSetParams()

```
hipError_t hipGraphMemcpyNodeSetParams (
            hipGraphNode_t node,
            const hipMemcpy3DParms * pNodeParams )
```

Sets a memcpy node's parameters.

Parameters

in	*node*	- instance of the node to set parameters to.
in	*pNodeParams*	- const pointer to the parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.19 hipGraphMemsetNodeGetParams()

```
hipError_t hipGraphMemsetNodeGetParams (
            hipGraphNode_t node,
            hipMemsetParams * pNodeParams )
```

Gets a memset node's parameters.

Parameters

in	*node*	- instane of the node to get parameters from.
out	*pNodeParams*	- pointer to the parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.20 hipGraphMemsetNodeSetParams()

```
hipError_t hipGraphMemsetNodeSetParams (
            hipGraphNode_t node,
            const hipMemsetParams * pNodeParams )
```

Sets a memset node's parameters.

Parameters

in	*node*	- instance of the node to set parameters to.
in	*pNodeParams*	- pointer to the parameters.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.21 hipStreamBeginCapture()

```
hipError_t hipStreamBeginCapture (
            hipStream_t stream,
            hipStreamCaptureMode mode )
```

Begins graph capture on a stream.

Parameters

in	*stream*	- Stream to initiate capture.
in	*mode*	- Controls the interaction of this capture sequence with other API calls that are not safe.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.19.4.22 hipStreamEndCapture()

```
hipError_t hipStreamEndCapture (
            hipStream_t stream,
            hipGraph_t * pGraph )
```

Ends capture on a stream, returning the captured graph.

Parameters

in	*stream*	- Stream to end capture.
out	*pGraph*	- returns the graph captured.

Returns

#hipSuccess, #hipErrorInvalidValue

Warning

: This API is marked as beta, meaning, while this is feature complete, it is still open to changes and may have outstanding issues.

4.20 Interop

Typedefs

- typedef unsigned int **GLuint**

Functions

- hipError_t **hipGLGetDevices** (unsigned int ∗pHipDeviceCount, int ∗pHipDevices, unsigned int hipDevice↵ Count, hipGLDeviceList deviceList)
- hipError_t **hipGraphicsGLRegisterBuffer** (hipGraphicsResource ∗∗resource, GLuint buffer, unsigned int flags)
- hipError_t **hipGraphicsMapResources** (int count, hipGraphicsResource_t ∗resources, hipStream_t stream __dparm(0))
- hipError_t **hipGraphicsResourceGetMappedPointer** (void ∗∗devPtr, size_t ∗size, hipGraphicsResource_t resource)
- hipError_t **hipGraphicsUnmapResources** (int count, hipGraphicsResource_t ∗resources, hipStream_↵ t stream __dparm(0))
- hipError_t **hipGraphicsUnregisterResource** (hipGraphicsResource_t resource)

4.20.1 Detailed Description

This section describes Stream Memory Wait and Write functions of HIP runtime API.

4.21 Context Management [Deprecated]

Functions

- hipError_t hipCtxCreate (hipCtx_t ∗ctx, unsigned int flags, hipDevice_t device)

  Create a context and set it as current/ default context.
- hipError_t hipCtxDestroy (hipCtx_t ctx)

  Destroy a HIP context.
- hipError_t hipCtxPopCurrent (hipCtx_t ∗ctx)

  Pop the current/default context and return the popped context.
- hipError_t hipCtxPushCurrent (hipCtx_t ctx)

  Push the context to be set as current/ default context.
- hipError_t hipCtxSetCurrent (hipCtx_t ctx)

  Set the passed context as current/default.
- hipError_t hipCtxGetCurrent (hipCtx_t ∗ctx)

  Get the handle of the current/ default context.
- hipError_t hipCtxGetDevice (hipDevice_t ∗device)

  Get the handle of the device associated with current/default context.
- hipError_t hipCtxGetApiVersion (hipCtx_t ctx, int ∗apiVersion)

  Returns the approximate HIP api version.
- hipError_t hipCtxGetCacheConfig (hipFuncCache_t ∗cacheConfig)

  Set Cache configuration for a specific function.
- hipError_t hipCtxSetCacheConfig (hipFuncCache_t cacheConfig)

  Set L1/Shared cache partition.
- hipError_t hipCtxSetSharedMemConfig (hipSharedMemConfig config)

  Set Shared memory bank configuration.
- hipError_t hipCtxGetSharedMemConfig (hipSharedMemConfig ∗pConfig)

  Get Shared memory bank configuration.
- hipError_t hipCtxSynchronize (void)

  Blocks until the default context has completed all preceding requested tasks.
- hipError_t hipCtxGetFlags (unsigned int ∗flags)

  Return flags used for creating default context.
- hipError_t hipCtxEnablePeerAccess (hipCtx_t peerCtx, unsigned int flags)

  Enables direct access to memory allocations in a peer context.
- hipError_t hipCtxDisablePeerAccess (hipCtx_t peerCtx)

  Disable direct access from current context's virtual address space to memory allocations physically located on a peer context.Disables direct access to memory allocations in a peer context and unregisters any registered allocations.

4.21.1 Detailed Description

This section describes the deprecated context management functions of HIP runtime API.

4.21.2 Function Documentation

4.21.2.1 hipCtxCreate()

```
hipError_t hipCtxCreate (
            hipCtx_t ∗ ctx,
            unsigned int flags,
            hipDevice_t device )
```

Create a context and set it as current/ default context.

Parameters

out	*ctx*	
in	*flags*	
in	*associated*	device handle

Returns

#hipSuccess

See also

hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.2 hipCtxDestroy()

```
hipError_t hipCtxDestroy (
            hipCtx_t ctx )
```

Destroy a HIP context.

Parameters

in	*ctx*	Context to destroy

Returns

#hipSuccess, #hipErrorInvalidValue

See also

hipCtxCreate, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent,hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize , hipCtxGetDevice

4.21.2.3 hipCtxDisablePeerAccess()

```
hipError_t hipCtxDisablePeerAccess (
            hipCtx_t peerCtx )
```

Disable direct access from current context's virtual address space to memory allocations physically located on a peer context.Disables direct access to memory allocations in a peer context and unregisters any registered allocations.

Returns hipErrorPeerAccessNotEnabled if direct access to memory on peerDevice has not yet been enabled from the current device.

Parameters

in	*peerCtx*	

Returns

#hipSuccess, #hipErrorPeerAccessNotEnabled

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

Warning

PeerToPeer support is experimental.

4.21.2.4 hipCtxEnablePeerAccess()

```
hipError_t hipCtxEnablePeerAccess (
            hipCtx_t peerCtx,
            unsigned int flags )
```

Enables direct access to memory allocations in a peer context.

Memory which already allocated on peer device will be mapped into the address space of the current device. In addition, all future memory allocations on peerDeviceId will be mapped into the address space of the current device when the memory is allocated. The peer memory remains accessible from the current device until a call to hip↩DeviceDisablePeerAccess or hipDeviceReset.

Parameters

in	*peerCtx*	
in	*flags*	

Returns

#hipSuccess, #hipErrorInvalidDevice, #hipErrorInvalidValue, #hipErrorPeerAccessAlreadyEnabled

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

Warning

PeerToPeer support is experimental.

4.21.2.5 hipCtxGetApiVersion()

```
hipError_t hipCtxGetApiVersion (
            hipCtx_t ctx,
            int * apiVersion )
```

Returns the approximate HIP api version.

Parameters

in	*ctx*	Context to check
out	*apiVersion*	

Returns

#hipSuccess

Warning

The HIP feature set does not correspond to an exact CUDA SDK api revision. This function always set ∗api↩ Version to 4 as an approximation though HIP supports some features which were introduced in later CUDA SDK revisions. HIP apps code should not rely on the api revision number here and should use arch feature flags to test device capabilities or conditional compilation.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetDevice, hipCtxGetFlags, hipCtxPopCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.6 hipCtxGetCacheConfig()

```
hipError_t hipCtxGetCacheConfig (
            hipFuncCache_t ∗ cacheConfig )
```

Set Cache configuration for a specific function.

Parameters

out	*cacheConfiguration*	

Returns

#hipSuccess

Warning

AMD devices and some Nvidia GPUS do not support reconfigurable cache. This hint is ignored on those architectures.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.7 hipCtxGetCurrent()

```
hipError_t hipCtxGetCurrent (
            hipCtx_t ∗ ctx )
```

Get the handle of the current/ default context.

Parameters

out	*ctx*	

Returns

#hipSuccess, #hipErrorInvalidContext

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetDevice, hipCtxGetFlags, hipCtxPopCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.8 hipCtxGetDevice()

```
hipError_t hipCtxGetDevice (
            hipDevice_t * device )
```

Get the handle of the device associated with current/default context.

Parameters

out	*device*	

Returns

#hipSuccess, #hipErrorInvalidContext

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize

4.21.2.9 hipCtxGetFlags()

```
hipError_t hipCtxGetFlags (
            unsigned int * flags )
```

Return flags used for creating default context.

Parameters

out	*flags*	

Returns

#hipSuccess

See also

hipCtxCreate, hipCtxDestroy, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.10 hipCtxGetSharedMemConfig()

```
hipError_t hipCtxGetSharedMemConfig (
            hipSharedMemConfig * pConfig )
```

Get Shared memory bank configuration.

Parameters

out	*sharedMemoryConfiguration*	

Returns

#hipSuccess

Warning

AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.11 hipCtxPopCurrent()

```
hipError_t hipCtxPopCurrent (
            hipCtx_t * ctx )
```

Pop the current/default context and return the popped context.

Parameters

out	*ctx*	

Returns

#hipSuccess, #hipErrorInvalidContext

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxSetCurrent, hipCtxGetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.12 hipCtxPushCurrent()

```
hipError_t hipCtxPushCurrent (
            hipCtx_t ctx )
```

Push the context to be set as current/ default context.

Parameters

in	*ctx*	

Returns

#hipSuccess, #hipErrorInvalidContext

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize , hipCtxGetDevice

4.21.2.13 hipCtxSetCacheConfig()

```
hipError_t hipCtxSetCacheConfig (
            hipFuncCache_t cacheConfig )
```

Set L1/Shared cache partition.

Parameters

in	*cacheConfiguration*	

Returns

#hipSuccess

Warning

AMD devices and some Nvidia GPUS do not support reconfigurable cache. This hint is ignored on those architectures.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.14 hipCtxSetCurrent()

```
hipError_t hipCtxSetCurrent (
            hipCtx_t ctx )
```

Set the passed context as current/default.

Parameters

in	*ctx*	

Returns

#hipSuccess, #hipErrorInvalidContext

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize , hipCtxGetDevice

4.21.2.15 hipCtxSetSharedMemConfig()

```
hipError_t hipCtxSetSharedMemConfig (
            hipSharedMemConfig config )
```

Set Shared memory bank configuration.

Parameters

in	*sharedMemoryConfiguration*	

Returns

#hipSuccess

Warning

AMD devices and some Nvidia GPUS do not support shared cache banking, and the hint is ignored on those architectures.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxSynchronize, hipCtxGetDevice

4.21.2.16 hipCtxSynchronize()

```
hipError_t hipCtxSynchronize (
            void  )
```

Blocks until the default context has completed all preceding requested tasks.

Returns

#hipSuccess

Warning

This function waits for all streams on the default context to complete execution, and then returns.

See also

hipCtxCreate, hipCtxDestroy, hipCtxGetFlags, hipCtxPopCurrent, hipCtxGetCurrent, hipCtxSetCurrent, hipCtxPushCurrent, hipCtxSetCacheConfig, hipCtxGetDevice

4.22 Texture Management [Deprecated]

Functions

- hipError_t **hipBindTexture** (size_t ∗offset, const textureReference ∗tex, const void ∗devPtr, const hipChannelFormatDesc ∗desc, size_t size __dparm(UINT_MAX))
- hipError_t **hipBindTexture2D** (size_t ∗offset, const textureReference ∗tex, const void ∗devPtr, const hipChannelFormatDesc ∗desc, size_t width, size_t height, size_t pitch)
- hipError_t **hipBindTextureToArray** (const textureReference ∗tex, hipArray_const_t array, const hipChannelFormatDesc ∗desc)
- hipError_t **hipGetTextureAlignmentOffset** (size_t ∗offset, const textureReference ∗texref)
- hipError_t **hipUnbindTexture** (const textureReference ∗tex)
- hipError_t **hipTexRefGetAddress** (hipDeviceptr_t ∗dev_ptr, const textureReference ∗texRef)
- hipError_t **hipTexRefGetAddressMode** (enum hipTextureAddressMode ∗pam, const textureReference ∗texRef, int dim)
- hipError_t **hipTexRefGetFilterMode** (enum hipTextureFilterMode ∗pfm, const textureReference ∗texRef)
- hipError_t **hipTexRefGetFlags** (unsigned int ∗pFlags, const textureReference ∗texRef)
- hipError_t **hipTexRefGetFormat** (hipArray_Format ∗pFormat, int ∗pNumChannels, const textureReference ∗texRef)
- hipError_t **hipTexRefGetMaxAnisotropy** (int ∗pmaxAnsio, const textureReference ∗texRef)
- hipError_t **hipTexRefGetMipmapFilterMode** (enum hipTextureFilterMode ∗pfm, const textureReference ∗texRef)
- hipError_t **hipTexRefGetMipmapLevelBias** (float ∗pbias, const textureReference ∗texRef)
- hipError_t **hipTexRefGetMipmapLevelClamp** (float ∗pminMipmapLevelClamp, float ∗pmaxMipmapLevel←↩ Clamp, const textureReference ∗texRef)
- hipError_t **hipTexRefGetMipMappedArray** (hipMipmappedArray_t ∗pArray, const textureReference ∗tex←↩ Ref)
- hipError_t **hipTexRefSetAddress** (size_t ∗ByteOffset, textureReference ∗texRef, hipDeviceptr_t dptr, size←↩ _t bytes)
- hipError_t **hipTexRefSetAddress2D** (textureReference ∗texRef, const HIP_ARRAY_DESCRIPTOR ∗desc, hipDeviceptr_t dptr, size_t Pitch)
- hipError_t **hipTexRefSetMaxAnisotropy** (textureReference ∗texRef, unsigned int maxAniso)

4.22.1 Detailed Description

This section describes the deprecated texture management functions of HIP runtime API.

4.23 Texture Management [Not supported]

Functions

- hipError_t **hipTexRefSetBorderColor** (textureReference ∗texRef, float ∗pBorderColor)
- hipError_t **hipTexRefSetMipmapFilterMode** (textureReference ∗texRef, enum hipTextureFilterMode fm)
- hipError_t **hipTexRefSetMipmapLevelBias** (textureReference ∗texRef, float bias)
- hipError_t **hipTexRefSetMipmapLevelClamp** (textureReference ∗texRef, float minMipMapLevelClamp, float maxMipMapLevelClamp)
- hipError_t **hipTexRefSetMipmappedArray** (textureReference ∗texRef, struct hipMipmappedArray ∗mipmappedArray, unsigned int Flags)
- hipError_t **hipMipmappedArrayCreate** (hipMipmappedArray_t ∗pHandle, HIP_ARRAY3D_DESCRIPTOR ∗pMipmappedArrayDesc, unsigned int numMipmapLevels)
- hipError_t **hipMipmappedArrayDestroy** (hipMipmappedArray_t hMipmappedArray)
- hipError_t **hipMipmappedArrayGetLevel** (hipArray_t ∗pLevelArray, hipMipmappedArray_t hMipMapped←Array, unsigned int level)

4.23.1 Detailed Description

This section describes the texture management functions currently unsupported in HIP runtime.

Chapter 5

Class Documentation

5.1 __half2_raw Struct Reference

Public Attributes

- unsigned short **x**
- unsigned short **y**

5.2 __half_raw Struct Reference

Public Attributes

- unsigned short **x**

5.3 __hip_enable_if< __B, __T > Struct Template Reference

5.4 __hip_enable_if< true, __T > Struct Template Reference

Public Types

- typedef __T **type**
- typedef __T **type**

5.5 char1 Union Reference

Public Attributes

- char **data**

5.6 char16 Union Reference

Public Attributes

- char **data** [16]

5.7 char2 Union Reference

Public Attributes

- char **data** [2]

5.8 char3 Union Reference

Public Attributes

- char4 **data**

5.9 char4 Union Reference

Public Attributes

- char **data** [4]

5.10 char8 Union Reference

Public Attributes

- char **data** [8]

5.11 dim3 Struct Reference

Public Attributes

- uint32_t x

  x
- uint32_t y

  y
- uint32_t z

  z

5.11.1 Detailed Description

Struct for data in 3D

5.12 double1 Union Reference

Public Attributes

- double **data**

5.13 double16 Union Reference

Public Attributes

- double **data** [16]

5.14 double2 Union Reference

Public Attributes

- double **data** [2]

5.15 double3 Union Reference

Public Attributes

- double4 **data**

5.16 double4 Union Reference

Public Attributes

- double **data** [4]

5.17 double8 Union Reference

Public Attributes

- double **data** [8]

5.18 float1 Union Reference

Public Attributes

- float **data**

5.19 float16 Union Reference

Public Attributes

- float **data** [16]

5.20 float2 Union Reference

Public Attributes

- float **data** [2]

5.21 float3 Union Reference

Public Attributes

- float4 **data**

5.22 float4 Union Reference

Public Attributes

- float **data** [4]

5.23 float8 Union Reference

Public Attributes

- float **data** [8]

5.24 gl_dim3 Struct Reference

Public Member Functions

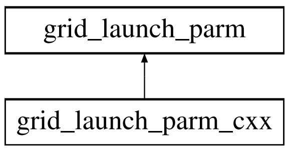
- **gl_dim3** (uint32_t _x=1, uint32_t _y=1, uint32_t _z=1)
- **gl_dim3** (uint32_t _x=1, uint32_t _y=1, uint32_t _z=1)

Public Attributes

- int **x**
- int **y**
- int **z**

5.25 grid_launch_parm Struct Reference

Inheritance diagram for grid_launch_parm:



Public Attributes

- gl_dim3 grid_dim

  Grid dimensions.
- gl_dim3 group_dim

  Group dimensions.
- unsigned int dynamic_group_mem_bytes
- enum gl_barrier_bit barrier_bit
- unsigned int launch_fence
- hc::accelerator_view ∗ av
- hc::completion_future ∗ cf

5.25.1 Member Data Documentation

5.25.1.1 av

```
hc::accelerator_view * grid_launch_parm::av
```

Pointer to the accelerator_view where the kernel should execute. If NULL, the default view on the default accelerator is used.

5.25.1.2 barrier_bit

```
enum gl_barrier_bit grid_launch_parm::barrier_bit
```

Control setting of barrier bit on per-packet basis: See gl_barrier_bit description.
Placeholder, is not used to control packet dispatch yet

5.25.1.3 cf

```
hc::completion_future * grid_launch_parm::cf
```

Pointer to the completion_future used to track the status of the command. If NULL, the command does not write status. In this case, synchronization can be enforced with queue-level waits or waiting on younger commands.

5.25.1.4 dynamic_group_mem_bytes

`unsigned int grid_launch_parm::dynamic_group_mem_bytes`

Amount of dynamic group memory to use with the kernel launch. This memory is in addition to the amount used statically in the kernel.

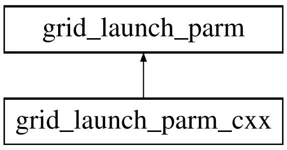
5.25.1.5 launch_fence

`unsigned int grid_launch_parm::launch_fence`

Value of packet fences to apply to launch. The correspond to the value of bits 9:14 in the AQL packet, see HSA_↩ PACKET_HEADER_ACQUIRE_FENCE_SCOPE and hsa_fence_scope_t.

5.26 grid_launch_parm_cxx Class Reference

Inheritance diagram for grid_launch_parm_cxx:



Public Member Functions

- **__attribute__** ((annotate("serialize"))) void __cxxamp_serialize(Kalmar
- **__attribute__** ((annotate("serialize"))) void __cxxamp_serialize(Kalmar

Additional Inherited Members

5.27 hip_api_data_s Struct Reference

Public Attributes

- uint64_t **correlation_id**
- uint32_t **phase**
-

union {
 struct {
  dim3 ∗ **gridDim**
  dim3 **gridDim__val**
  dim3 ∗ **blockDim**
  dim3 **blockDim__val**
  size_t ∗ **sharedMem**
  size_t **sharedMem__val**
  hipStream_t ∗ **stream**
  hipStream_t **stream__val**
 } **__hipPopCallConfiguration**
 struct {
  dim3 **gridDim**
  dim3 **blockDim**
  size_t **sharedMem**
  hipStream_t **stream**
 } **__hipPushCallConfiguration**
 struct {
  hipArray ∗∗ **array**
  hipArray ∗ **array__val**
  const HIP_ARRAY3D_DESCRIPTOR ∗ **pAllocateArray**

```
   HIP_ARRAY3D_DESCRIPTOR pAllocateArray__val
} hipArray3DCreate
struct {
   hipArray ∗∗ pHandle
   hipArray ∗ pHandle__val
   const HIP_ARRAY_DESCRIPTOR ∗ pAllocateArray
   HIP_ARRAY_DESCRIPTOR pAllocateArray__val
} hipArrayCreate
struct {
   hipArray ∗ array
   hipArray array__val
} hipArrayDestroy
struct {
   int ∗ device
   int device__val
   const hipDeviceProp_t ∗ prop
   hipDeviceProp_t prop__val
} hipChooseDevice
struct {
   dim3 gridDim
   dim3 blockDim
   size_t sharedMem
   hipStream_t stream
} hipConfigureCall
struct {
   hipSurfaceObject_t ∗ pSurfObject
   hipSurfaceObject_t pSurfObject__val
   const hipResourceDesc ∗ pResDesc
   hipResourceDesc pResDesc__val
} hipCreateSurfaceObject
struct {
   hipCtx_t ∗ ctx
   hipCtx_t ctx__val
   unsigned int flags
   hipDevice_t device
} hipCtxCreate
struct {
   hipCtx_t ctx
} hipCtxDestroy
struct {
   hipCtx_t peerCtx
} hipCtxDisablePeerAccess
struct {
   hipCtx_t peerCtx
   unsigned int flags
} hipCtxEnablePeerAccess
struct {
   hipCtx_t ctx
   int ∗ apiVersion
   int apiVersion__val
} hipCtxGetApiVersion
struct {
   hipFuncCache_t ∗ cacheConfig
   hipFuncCache_t cacheConfig__val
} hipCtxGetCacheConfig
struct {
   hipCtx_t ∗ ctx
   hipCtx_t ctx__val
```

} **hipCtxGetCurrent**
struct {
 hipDevice_t ∗ **device**
 hipDevice_t **device__val**
} **hipCtxGetDevice**
struct {
 unsigned int ∗ **flags**
 unsigned int **flags__val**
} **hipCtxGetFlags**
struct {
 hipSharedMemConfig ∗ **pConfig**
 hipSharedMemConfig **pConfig__val**
} **hipCtxGetSharedMemConfig**
struct {
 hipCtx_t ∗ **ctx**
 hipCtx_t **ctx__val**
} **hipCtxPopCurrent**
struct {
 hipCtx_t **ctx**
} **hipCtxPushCurrent**
struct {
 hipFuncCache_t **cacheConfig**
} **hipCtxSetCacheConfig**
struct {
 hipCtx_t **ctx**
} **hipCtxSetCurrent**
struct {
 hipSharedMemConfig **config**
} **hipCtxSetSharedMemConfig**
struct {
 hipExternalMemory_t **extMem**
} **hipDestroyExternalMemory**
struct {
 hipExternalSemaphore_t **extSem**
} **hipDestroyExternalSemaphore**
struct {
 hipSurfaceObject_t **surfaceObject**
} **hipDestroySurfaceObject**
struct {
 int ∗ **canAccessPeer**
 int **canAccessPeer__val**
 int **deviceId**
 int **peerDeviceId**
} **hipDeviceCanAccessPeer**
struct {
 int ∗ **major**
 int **major__val**
 int ∗ **minor**
 int **minor__val**
 hipDevice_t **device**
} **hipDeviceComputeCapability**
struct {
 int **peerDeviceId**
} **hipDeviceDisablePeerAccess**
struct {
 int **peerDeviceId**
 unsigned int **flags**
} **hipDeviceEnablePeerAccess**

struct {
  hipDevice_t ∗ **device**
  hipDevice_t **device__val**
  int **ordinal**
} **hipDeviceGet**
struct {
  int ∗ **pi**
  int **pi__val**
  hipDeviceAttribute_t **attr**
  int **deviceId**
} **hipDeviceGetAttribute**
struct {
  int ∗ **device**
  int **device__val**
  const char ∗ **pciBusId**
  char **pciBusId__val**
} **hipDeviceGetByPCIBusId**
struct {
  hipFuncCache_t ∗ **cacheConfig**
  hipFuncCache_t **cacheConfig__val**
} **hipDeviceGetCacheConfig**
struct {
  size_t ∗ **pValue**
  size_t **pValue__val**
  enum hipLimit_t **limit**
} **hipDeviceGetLimit**
struct {
  char ∗ **name**
  char **name__val**
  int **len**
  hipDevice_t **device**
} **hipDeviceGetName**
struct {
  int ∗ **value**
  int **value__val**
  hipDeviceP2PAttr **attr**
  int **srcDevice**
  int **dstDevice**
} **hipDeviceGetP2PAttribute**
struct {
  char ∗ **pciBusId**
  char **pciBusId__val**
  int **len**
  int **device**
} **hipDeviceGetPCIBusId**
struct {
  hipSharedMemConfig ∗ **pConfig**
  hipSharedMemConfig **pConfig__val**
} **hipDeviceGetSharedMemConfig**
struct {
  int ∗ **leastPriority**
  int **leastPriority__val**
  int ∗ **greatestPriority**
  int **greatestPriority__val**
} **hipDeviceGetStreamPriorityRange**
struct {
  hipDevice_t **dev**
  unsigned int ∗ **flags**

unsigned int **flags__val**
int ∗ **active**
int **active__val**
} **hipDevicePrimaryCtxGetState**
struct {
hipDevice_t **dev**
} **hipDevicePrimaryCtxRelease**
struct {
hipDevice_t **dev**
} **hipDevicePrimaryCtxReset**
struct {
hipCtx_t ∗ **pctx**
hipCtx_t **pctx__val**
hipDevice_t **dev**
} **hipDevicePrimaryCtxRetain**
struct {
hipDevice_t **dev**
unsigned int **flags**
} **hipDevicePrimaryCtxSetFlags**
struct {
hipFuncCache_t **cacheConfig**
} **hipDeviceSetCacheConfig**
struct {
hipSharedMemConfig **config**
} **hipDeviceSetSharedMemConfig**
struct {
size_t ∗ **bytes**
size_t **bytes__val**
hipDevice_t **device**
} **hipDeviceTotalMem**
struct {
int ∗ **driverVersion**
int **driverVersion__val**
} **hipDriverGetVersion**
struct {
const hip_Memcpy2D ∗ **pCopy**
hip_Memcpy2D **pCopy__val**
} **hipDrvMemcpy2DUnaligned**
struct {
const HIP_MEMCPY3D ∗ **pCopy**
HIP_MEMCPY3D **pCopy__val**
} **hipDrvMemcpy3D**
struct {
const HIP_MEMCPY3D ∗ **pCopy**
HIP_MEMCPY3D **pCopy__val**
hipStream_t **stream**
} **hipDrvMemcpy3DAsync**
struct {
hipEvent_t ∗ **event**
hipEvent_t **event__val**
} **hipEventCreate**
struct {
hipEvent_t ∗ **event**
hipEvent_t **event__val**
unsigned int **flags**
} **hipEventCreateWithFlags**
struct {
hipEvent_t **event**

} **hipEventDestroy**
struct {
  float ∗ **ms**
  float **ms__val**
  hipEvent_t **start**
  hipEvent_t **stop**
} **hipEventElapsedTime**
struct {
  hipEvent_t **event**
} **hipEventQuery**
struct {
  hipEvent_t **event**
  hipStream_t **stream**
} **hipEventRecord**
struct {
  hipEvent_t **event**
} **hipEventSynchronize**
struct {
  int **device1**
  int **device2**
  unsigned int ∗ **linktype**
  unsigned int **linktype__val**
  unsigned int ∗ **hopcount**
  unsigned int **hopcount__val**
} **hipExtGetLinkTypeAndHopCount**
struct {
  const void ∗ **function_address**
  dim3 **numBlocks**
  dim3 **dimBlocks**
  void ∗∗ **args**
  void ∗ **args__val**
  size_t **sharedMemBytes**
  hipStream_t **stream**
  hipEvent_t **startEvent**
  hipEvent_t **stopEvent**
  int **flags**
} **hipExtLaunchKernel**
struct {
  hipLaunchParams ∗ **launchParamsList**
  hipLaunchParams **launchParamsList__val**
  int **numDevices**
  unsigned int **flags**
} **hipExtLaunchMultiKernelMultiDevice**
struct {
  void ∗∗ **ptr**
  void ∗ **ptr__val**
  size_t **sizeBytes**
  unsigned int **flags**
} **hipExtMallocWithFlags**
struct {
  hipFunction_t **f**
  unsigned int **globalWorkSizeX**
  unsigned int **globalWorkSizeY**
  unsigned int **globalWorkSizeZ**
  unsigned int **localWorkSizeX**
  unsigned int **localWorkSizeY**
  unsigned int **localWorkSizeZ**
  size_t **sharedMemBytes**

hipStream_t **hStream**
void ∗∗ **kernelParams**
void ∗ **kernelParams__val**
void ∗∗ **extra**
void ∗ **extra__val**
hipEvent_t **startEvent**
hipEvent_t **stopEvent**
unsigned int **flags**
} **hipExtModuleLaunchKernel**
struct {
hipStream_t ∗ **stream**
hipStream_t **stream__val**
unsigned int **cuMaskSize**
const unsigned int ∗ **cuMask**
unsigned int **cuMask__val**
} **hipExtStreamCreateWithCUMask**
struct {
hipStream_t **stream**
unsigned int **cuMaskSize**
unsigned int ∗ **cuMask**
unsigned int **cuMask__val**
} **hipExtStreamGetCUMask**
struct {
void ∗∗ **devPtr**
void ∗ **devPtr__val**
hipExternalMemory_t **extMem**
const hipExternalMemoryBufferDesc ∗ **bufferDesc**
hipExternalMemoryBufferDesc **bufferDesc__val**
} **hipExternalMemoryGetMappedBuffer**
struct {
void ∗ **ptr**
} **hipFree**
struct {
hipArray ∗ **array**
hipArray **array__val**
} **hipFreeArray**
struct {
void ∗ **ptr**
} **hipFreeHost**
struct {
hipMipmappedArray_t **mipmappedArray**
} **hipFreeMipmappedArray**
struct {
int ∗ **value**
int **value__val**
hipFunction_attribute **attrib**
hipFunction_t **hfunc**
} **hipFuncGetAttribute**
struct {
hipFuncAttributes ∗ **attr**
hipFuncAttributes **attr__val**
const void ∗ **func**
} **hipFuncGetAttributes**
struct {
const void ∗ **func**
hipFuncAttribute **attr**
int **value**
} **hipFuncSetAttribute**

struct {
  const void ∗ **func**
  hipFuncCache_t **config**
} **hipFuncSetCacheConfig**
struct {
  const void ∗ **func**
  hipSharedMemConfig **config**
} **hipFuncSetSharedMemConfig**
struct {
  unsigned int ∗ **pHipDeviceCount**
  unsigned int **pHipDeviceCount__val**
  int ∗ **pHipDevices**
  int **pHipDevices__val**
  unsigned int **hipDeviceCount**
  hipGLDeviceList **deviceList**
} **hipGLGetDevices**
struct {
  int ∗ **deviceId**
  int **deviceId__val**
} **hipGetDevice**
struct {
  int ∗ **count**
  int **count__val**
} **hipGetDeviceCount**
struct {
  unsigned int ∗ **flags**
  unsigned int **flags__val**
} **hipGetDeviceFlags**
struct {
  hipDeviceProp_t ∗ **props**
  hipDeviceProp_t **props__val**
  hipDevice_t **device**
} **hipGetDeviceProperties**
struct {
  hipArray_t ∗ **levelArray**
  hipArray_t **levelArray__val**
  hipMipmappedArray_const_t **mipmappedArray**
  unsigned int **level**
} **hipGetMipmappedArrayLevel**
struct {
  void ∗∗ **devPtr**
  void ∗ **devPtr__val**
  const void ∗ **symbol**
} **hipGetSymbolAddress**
struct {
  size_t ∗ **size**
  size_t **size__val**
  const void ∗ **symbol**
} **hipGetSymbolSize**
struct {
  hipGraph_t **graph**
  const hipGraphNode_t ∗ **from**
  hipGraphNode_t **from__val**
  const hipGraphNode_t ∗ **to**
  hipGraphNode_t **to__val**
  size_t **numDependencies**
} **hipGraphAddDependencies**
struct {

hipGraphNode_t ∗ **pGraphNode**
hipGraphNode_t **pGraphNode__val**
hipGraph_t **graph**
const hipGraphNode_t ∗ **pDependencies**
hipGraphNode_t **pDependencies__val**
size_t **numDependencies**
} **hipGraphAddEmptyNode**
struct {
hipGraphNode_t ∗ **pGraphNode**
hipGraphNode_t **pGraphNode__val**
hipGraph_t **graph**
const hipGraphNode_t ∗ **pDependencies**
hipGraphNode_t **pDependencies__val**
size_t **numDependencies**
const hipKernelNodeParams ∗ **pNodeParams**
hipKernelNodeParams **pNodeParams__val**
} **hipGraphAddKernelNode**
struct {
hipGraphNode_t ∗ **pGraphNode**
hipGraphNode_t **pGraphNode__val**
hipGraph_t **graph**
const hipGraphNode_t ∗ **pDependencies**
hipGraphNode_t **pDependencies__val**
size_t **numDependencies**
const hipMemcpy3DParms ∗ **pCopyParams**
hipMemcpy3DParms **pCopyParams__val**
} **hipGraphAddMemcpyNode**
struct {
hipGraphNode_t ∗ **pGraphNode**
hipGraphNode_t **pGraphNode__val**
hipGraph_t **graph**
const hipGraphNode_t ∗ **pDependencies**
hipGraphNode_t **pDependencies__val**
size_t **numDependencies**
const hipMemsetParams ∗ **pMemsetParams**
hipMemsetParams **pMemsetParams__val**
} **hipGraphAddMemsetNode**
struct {
hipGraph_t ∗ **pGraph**
hipGraph_t **pGraph__val**
unsigned int **flags**
} **hipGraphCreate**
struct {
hipGraph_t **graph**
} **hipGraphDestroy**
struct {
hipGraphExec_t **pGraphExec**
} **hipGraphExecDestroy**
struct {
hipGraphExec_t **hGraphExec**
hipGraphNode_t **node**
const hipKernelNodeParams ∗ **pNodeParams**
hipKernelNodeParams **pNodeParams__val**
} **hipGraphExecKernelNodeSetParams**
struct {
hipGraph_t **graph**
hipGraphNode_t ∗ **nodes**
hipGraphNode_t **nodes__val**

size_t ∗ **numNodes**
size_t **numNodes__val**
} **hipGraphGetNodes**
struct {
hipGraph_t **graph**
hipGraphNode_t ∗ **pRootNodes**
hipGraphNode_t **pRootNodes__val**
size_t ∗ **pNumRootNodes**
size_t **pNumRootNodes__val**
} **hipGraphGetRootNodes**
struct {
hipGraphExec_t ∗ **pGraphExec**
hipGraphExec_t **pGraphExec__val**
hipGraph_t **graph**
hipGraphNode_t ∗ **pErrorNode**
hipGraphNode_t **pErrorNode__val**
char ∗ **pLogBuffer**
char **pLogBuffer__val**
size_t **bufferSize**
} **hipGraphInstantiate**
struct {
hipGraphNode_t **node**
hipKernelNodeParams ∗ **pNodeParams**
hipKernelNodeParams **pNodeParams__val**
} **hipGraphKernelNodeGetParams**
struct {
hipGraphNode_t **node**
const hipKernelNodeParams ∗ **pNodeParams**
hipKernelNodeParams **pNodeParams__val**
} **hipGraphKernelNodeSetParams**
struct {
hipGraphExec_t **graphExec**
hipStream_t **stream**
} **hipGraphLaunch**
struct {
hipGraphNode_t **node**
hipMemcpy3DParms ∗ **pNodeParams**
hipMemcpy3DParms **pNodeParams__val**
} **hipGraphMemcpyNodeGetParams**
struct {
hipGraphNode_t **node**
const hipMemcpy3DParms ∗ **pNodeParams**
hipMemcpy3DParms **pNodeParams__val**
} **hipGraphMemcpyNodeSetParams**
struct {
hipGraphNode_t **node**
hipMemsetParams ∗ **pNodeParams**
hipMemsetParams **pNodeParams__val**
} **hipGraphMemsetNodeGetParams**
struct {
hipGraphNode_t **node**
const hipMemsetParams ∗ **pNodeParams**
hipMemsetParams **pNodeParams__val**
} **hipGraphMemsetNodeSetParams**
struct {
hipGraphicsResource ∗∗ **resource**
hipGraphicsResource ∗ **resource__val**
GLuint **buffer**

unsigned int **flags**
} **hipGraphicsGLRegisterBuffer**
struct {
int **count**
hipGraphicsResource_t ∗ **resources**
hipGraphicsResource_t **resources__val**
hipStream_t **stream**
} **hipGraphicsMapResources**
struct {
void ∗∗ **devPtr**
void ∗ **devPtr__val**
size_t ∗ **size**
size_t **size__val**
hipGraphicsResource_t **resource**
} **hipGraphicsResourceGetMappedPointer**
struct {
int **count**
hipGraphicsResource_t ∗ **resources**
hipGraphicsResource_t **resources__val**
hipStream_t **stream**
} **hipGraphicsUnmapResources**
struct {
hipGraphicsResource_t **resource**
} **hipGraphicsUnregisterResource**
struct {
hipFunction_t **f**
unsigned int **globalWorkSizeX**
unsigned int **globalWorkSizeY**
unsigned int **globalWorkSizeZ**
unsigned int **blockDimX**
unsigned int **blockDimY**
unsigned int **blockDimZ**
size_t **sharedMemBytes**
hipStream_t **hStream**
void ∗∗ **kernelParams**
void ∗ **kernelParams__val**
void ∗∗ **extra**
void ∗ **extra__val**
hipEvent_t **startEvent**
hipEvent_t **stopEvent**
} **hipHccModuleLaunchKernel**
struct {
void ∗∗ **ptr**
void ∗ **ptr__val**
size_t **size**
unsigned int **flags**
} **hipHostAlloc**
struct {
void ∗ **ptr**
} **hipHostFree**
struct {
void ∗∗ **devPtr**
void ∗ **devPtr__val**
void ∗ **hstPtr**
unsigned int **flags**
} **hipHostGetDevicePointer**
struct {
unsigned int ∗ **flagsPtr**

```
    unsigned int flagsPtr__val
    void * hostPtr
} hipHostGetFlags
struct {
    void ** ptr
    void * ptr__val
    size_t size
    unsigned int flags
} hipHostMalloc
struct {
    void * hostPtr
    size_t sizeBytes
    unsigned int flags
} hipHostRegister
struct {
    void * hostPtr
} hipHostUnregister
struct {
    hipExternalMemory_t * extMem_out
    hipExternalMemory_t extMem_out__val
    const hipExternalMemoryHandleDesc * memHandleDesc
    hipExternalMemoryHandleDesc memHandleDesc__val
} hipImportExternalMemory
struct {
    hipExternalSemaphore_t * extSem_out
    hipExternalSemaphore_t extSem_out__val
    const hipExternalSemaphoreHandleDesc * semHandleDesc
    hipExternalSemaphoreHandleDesc semHandleDesc__val
} hipImportExternalSemaphore
struct {
    unsigned int flags
} hipInit
struct {
    void * devPtr
} hipIpcCloseMemHandle
struct {
    hipIpcEventHandle_t * handle
    hipIpcEventHandle_t handle__val
    hipEvent_t event
} hipIpcGetEventHandle
struct {
    hipIpcMemHandle_t * handle
    hipIpcMemHandle_t handle__val
    void * devPtr
} hipIpcGetMemHandle
struct {
    hipEvent_t * event
    hipEvent_t event__val
    hipIpcEventHandle_t handle
} hipIpcOpenEventHandle
struct {
    void ** devPtr
    void * devPtr__val
    hipIpcMemHandle_t handle
    unsigned int flags
} hipIpcOpenMemHandle
struct {
    const void * hostFunction
```

} **hipLaunchByPtr**
struct {
  const void ∗ **f**
  dim3 **gridDim**
  dim3 **blockDimX**
  void ∗∗ **kernelParams**
  void ∗ **kernelParams__val**
  unsigned int **sharedMemBytes**
  hipStream_t **stream**
} **hipLaunchCooperativeKernel**
struct {
  hipLaunchParams ∗ **launchParamsList**
  hipLaunchParams **launchParamsList__val**
  int **numDevices**
  unsigned int **flags**
} **hipLaunchCooperativeKernelMultiDevice**
struct {
  const void ∗ **function_address**
  dim3 **numBlocks**
  dim3 **dimBlocks**
  void ∗∗ **args**
  void ∗ **args__val**
  size_t **sharedMemBytes**
  hipStream_t **stream**
} **hipLaunchKernel**
struct {
  void ∗∗ **ptr**
  void ∗ **ptr__val**
  size_t **size**
} **hipMalloc**
struct {
  hipPitchedPtr ∗ **pitchedDevPtr**
  hipPitchedPtr **pitchedDevPtr__val**
  hipExtent **extent**
} **hipMalloc3D**
struct {
  hipArray_t ∗ **array**
  hipArray_t **array__val**
  const hipChannelFormatDesc ∗ **desc**
  hipChannelFormatDesc **desc__val**
  hipExtent **extent**
  unsigned int **flags**
} **hipMalloc3DArray**
struct {
  hipArray ∗∗ **array**
  hipArray ∗ **array__val**
  const hipChannelFormatDesc ∗ **desc**
  hipChannelFormatDesc **desc__val**
  size_t **width**
  size_t **height**
  unsigned int **flags**
} **hipMallocArray**
struct {
  void ∗∗ **ptr**
  void ∗ **ptr__val**
  size_t **size**
} **hipMallocHost**
struct {

void ∗∗ **dev_ptr**
void ∗ **dev_ptr__val**
size_t **size**
unsigned int **flags**
} **hipMallocManaged**
struct {
hipMipmappedArray_t ∗ **mipmappedArray**
hipMipmappedArray_t **mipmappedArray__val**
const hipChannelFormatDesc ∗ **desc**
hipChannelFormatDesc **desc__val**
hipExtent **extent**
unsigned int **numLevels**
unsigned int **flags**
} **hipMallocMipmappedArray**
struct {
void ∗∗ **ptr**
void ∗ **ptr__val**
size_t ∗ **pitch**
size_t **pitch__val**
size_t **width**
size_t **height**
} **hipMallocPitch**
struct {
const void ∗ **dev_ptr**
size_t **count**
hipMemoryAdvise **advice**
int **device**
} **hipMemAdvise**
struct {
void ∗∗ **ptr**
void ∗ **ptr__val**
size_t **size**
} **hipMemAllocHost**
struct {
hipDeviceptr_t ∗ **dptr**
hipDeviceptr_t **dptr__val**
size_t ∗ **pitch**
size_t **pitch__val**
size_t **widthInBytes**
size_t **height**
unsigned int **elementSizeBytes**
} **hipMemAllocPitch**
struct {
hipDeviceptr_t ∗ **pbase**
hipDeviceptr_t **pbase__val**
size_t ∗ **psize**
size_t **psize__val**
hipDeviceptr_t **dptr**
} **hipMemGetAddressRange**
struct {
size_t ∗ **free**
size_t **free__val**
size_t ∗ **total**
size_t **total__val**
} **hipMemGetInfo**
struct {
const void ∗ **dev_ptr**
size_t **count**

int **device**
hipStream_t **stream**
} **hipMemPrefetchAsync**
struct {
void ∗ **ptr**
size_t ∗ **size**
size_t **size__val**
} **hipMemPtrGetInfo**
struct {
void ∗ **data**
size_t **data_size**
hipMemRangeAttribute **attribute**
const void ∗ **dev_ptr**
size_t **count**
} **hipMemRangeGetAttribute**
struct {
void ∗∗ **data**
void ∗ **data__val**
size_t ∗ **data_sizes**
size_t **data_sizes__val**
hipMemRangeAttribute ∗ **attributes**
hipMemRangeAttribute **attributes__val**
size_t **num_attributes**
const void ∗ **dev_ptr**
size_t **count**
} **hipMemRangeGetAttributes**
struct {
void ∗ **dst**
const void ∗ **src**
size_t **sizeBytes**
hipMemcpyKind **kind**
} **hipMemcpy**
struct {
void ∗ **dst**
size_t **dpitch**
const void ∗ **src**
size_t **spitch**
size_t **width**
size_t **height**
hipMemcpyKind **kind**
} **hipMemcpy2D**
struct {
void ∗ **dst**
size_t **dpitch**
const void ∗ **src**
size_t **spitch**
size_t **width**
size_t **height**
hipMemcpyKind **kind**
hipStream_t **stream**
} **hipMemcpy2DAsync**
struct {
void ∗ **dst**
size_t **dpitch**
hipArray_const_t **src**
size_t **wOffset**
size_t **hOffset**
size_t **width**

size_t **height**
hipMemcpyKind **kind**
} **hipMemcpy2DFromArray**
struct {
void ∗ **dst**
size_t **dpitch**
hipArray_const_t **src**
size_t **wOffset**
size_t **hOffset**
size_t **width**
size_t **height**
hipMemcpyKind **kind**
hipStream_t **stream**
} **hipMemcpy2DFromArrayAsync**
struct {
hipArray ∗ **dst**
hipArray **dst__val**
size_t **wOffset**
size_t **hOffset**
const void ∗ **src**
size_t **spitch**
size_t **width**
size_t **height**
hipMemcpyKind **kind**
} **hipMemcpy2DToArray**
struct {
hipArray ∗ **dst**
hipArray **dst__val**
size_t **wOffset**
size_t **hOffset**
const void ∗ **src**
size_t **spitch**
size_t **width**
size_t **height**
hipMemcpyKind **kind**
hipStream_t **stream**
} **hipMemcpy2DToArrayAsync**
struct {
const hipMemcpy3DParms ∗ **p**
hipMemcpy3DParms **p__val**
} **hipMemcpy3D**
struct {
const hipMemcpy3DParms ∗ **p**
hipMemcpy3DParms **p__val**
hipStream_t **stream**
} **hipMemcpy3DAsync**
struct {
void ∗ **dst**
const void ∗ **src**
size_t **sizeBytes**
hipMemcpyKind **kind**
hipStream_t **stream**
} **hipMemcpyAsync**
struct {
void ∗ **dst**
hipArray ∗ **srcArray**
hipArray **srcArray__val**
size_t **srcOffset**

```
    size_t count
} hipMemcpyAtoH
struct {
    hipDeviceptr_t dst
    hipDeviceptr_t src
    size_t sizeBytes
} hipMemcpyDtoD
struct {
    hipDeviceptr_t dst
    hipDeviceptr_t src
    size_t sizeBytes
    hipStream_t stream
} hipMemcpyDtoDAsync
struct {
    void * dst
    hipDeviceptr_t src
    size_t sizeBytes
} hipMemcpyDtoH
struct {
    void * dst
    hipDeviceptr_t src
    size_t sizeBytes
    hipStream_t stream
} hipMemcpyDtoHAsync
struct {
    void * dst
    hipArray_const_t srcArray
    size_t wOffset
    size_t hOffset
    size_t count
    hipMemcpyKind kind
} hipMemcpyFromArray
struct {
    void * dst
    const void * symbol
    size_t sizeBytes
    size_t offset
    hipMemcpyKind kind
} hipMemcpyFromSymbol
struct {
    void * dst
    const void * symbol
    size_t sizeBytes
    size_t offset
    hipMemcpyKind kind
    hipStream_t stream
} hipMemcpyFromSymbolAsync
struct {
    hipArray * dstArray
    hipArray dstArray__val
    size_t dstOffset
    const void * srcHost
    size_t count
} hipMemcpyHtoA
struct {
    hipDeviceptr_t dst
    void * src
    size_t sizeBytes
```

} **hipMemcpyHtoD**
struct {
  hipDeviceptr_t **dst**
  void ∗ **src**
  size_t **sizeBytes**
  hipStream_t **stream**
} **hipMemcpyHtoDAsync**
struct {
  const hip_Memcpy2D ∗ **pCopy**
  hip_Memcpy2D **pCopy__val**
} **hipMemcpyParam2D**
struct {
  const hip_Memcpy2D ∗ **pCopy**
  hip_Memcpy2D **pCopy__val**
  hipStream_t **stream**
} **hipMemcpyParam2DAsync**
struct {
  void ∗ **dst**
  int **dstDeviceId**
  const void ∗ **src**
  int **srcDeviceId**
  size_t **sizeBytes**
} **hipMemcpyPeer**
struct {
  void ∗ **dst**
  int **dstDeviceId**
  const void ∗ **src**
  int **srcDevice**
  size_t **sizeBytes**
  hipStream_t **stream**
} **hipMemcpyPeerAsync**
struct {
  hipArray ∗ **dst**
  hipArray **dst__val**
  size_t **wOffset**
  size_t **hOffset**
  const void ∗ **src**
  size_t **count**
  hipMemcpyKind **kind**
} **hipMemcpyToArray**
struct {
  const void ∗ **symbol**
  const void ∗ **src**
  size_t **sizeBytes**
  size_t **offset**
  hipMemcpyKind **kind**
} **hipMemcpyToSymbol**
struct {
  const void ∗ **symbol**
  const void ∗ **src**
  size_t **sizeBytes**
  size_t **offset**
  hipMemcpyKind **kind**
  hipStream_t **stream**
} **hipMemcpyToSymbolAsync**
struct {
  void ∗ **dst**
  const void ∗ **src**

size_t **sizeBytes**
hipMemcpyKind **kind**
hipStream_t **stream**
} **hipMemcpyWithStream**
struct {
void ∗ **dst**
int **value**
size_t **sizeBytes**
} **hipMemset**
struct {
void ∗ **dst**
size_t **pitch**
int **value**
size_t **width**
size_t **height**
} **hipMemset2D**
struct {
void ∗ **dst**
size_t **pitch**
int **value**
size_t **width**
size_t **height**
hipStream_t **stream**
} **hipMemset2DAsync**
struct {
hipPitchedPtr **pitchedDevPtr**
int **value**
hipExtent **extent**
} **hipMemset3D**
struct {
hipPitchedPtr **pitchedDevPtr**
int **value**
hipExtent **extent**
hipStream_t **stream**
} **hipMemset3DAsync**
struct {
void ∗ **dst**
int **value**
size_t **sizeBytes**
hipStream_t **stream**
} **hipMemsetAsync**
struct {
hipDeviceptr_t **dest**
unsigned short **value**
size_t **count**
} **hipMemsetD16**
struct {
hipDeviceptr_t **dest**
unsigned short **value**
size_t **count**
hipStream_t **stream**
} **hipMemsetD16Async**
struct {
hipDeviceptr_t **dest**
int **value**
size_t **count**
} **hipMemsetD32**
struct {

hipDeviceptr_t **dst**
int **value**
size_t **count**
hipStream_t **stream**
} **hipMemsetD32Async**
struct {
hipDeviceptr_t **dest**
unsigned char **value**
size_t **count**
} **hipMemsetD8**
struct {
hipDeviceptr_t **dest**
unsigned char **value**
size_t **count**
hipStream_t **stream**
} **hipMemsetD8Async**
struct {
hipMipmappedArray_t ∗ **pHandle**
hipMipmappedArray_t **pHandle__val**
HIP_ARRAY3D_DESCRIPTOR ∗ **pMipmappedArrayDesc**
HIP_ARRAY3D_DESCRIPTOR **pMipmappedArrayDesc__val**
unsigned int **numMipmapLevels**
} **hipMipmappedArrayCreate**
struct {
hipMipmappedArray_t **hMipmappedArray**
} **hipMipmappedArrayDestroy**
struct {
hipArray_t ∗ **pLevelArray**
hipArray_t **pLevelArray__val**
hipMipmappedArray_t **hMipMappedArray**
unsigned int **level**
} **hipMipmappedArrayGetLevel**
struct {
hipFunction_t ∗ **function**
hipFunction_t **function__val**
hipModule_t **module**
const char ∗ **kname**
char **kname__val**
} **hipModuleGetFunction**
struct {
hipDeviceptr_t ∗ **dptr**
hipDeviceptr_t **dptr__val**
size_t ∗ **bytes**
size_t **bytes__val**
hipModule_t **hmod**
const char ∗ **name**
char **name__val**
} **hipModuleGetGlobal**
struct {
textureReference ∗∗ **texRef**
textureReference ∗ **texRef__val**
hipModule_t **hmod**
const char ∗ **name**
char **name__val**
} **hipModuleGetTexRef**
struct {
hipFunction_t **f**
unsigned int **gridDimX**

unsigned int **gridDimY**
unsigned int **gridDimZ**
unsigned int **blockDimX**
unsigned int **blockDimY**
unsigned int **blockDimZ**
unsigned int **sharedMemBytes**
hipStream_t **stream**
void ∗∗ **kernelParams**
void ∗ **kernelParams__val**
void ∗∗ **extra**
void ∗ **extra__val**
} **hipModuleLaunchKernel**
struct {
hipModule_t ∗ **module**
hipModule_t **module__val**
const char ∗ **fname**
char **fname__val**
} **hipModuleLoad**
struct {
hipModule_t ∗ **module**
hipModule_t **module__val**
const void ∗ **image**
} **hipModuleLoadData**
struct {
hipModule_t ∗ **module**
hipModule_t **module__val**
const void ∗ **image**
unsigned int **numOptions**
hipJitOption ∗ **options**
hipJitOption **options__val**
void ∗∗ **optionsValues**
void ∗ **optionsValues__val**
} **hipModuleLoadDataEx**
struct {
int ∗ **numBlocks**
int **numBlocks__val**
hipFunction_t **f**
int **blockSize**
size_t **dynSharedMemPerBlk**
} **hipModuleOccupancyMaxActiveBlocksPerMultiprocessor**
struct {
int ∗ **numBlocks**
int **numBlocks__val**
hipFunction_t **f**
int **blockSize**
size_t **dynSharedMemPerBlk**
unsigned int **flags**
} **hipModuleOccupancyMaxActiveBlocksPerMultiprocessorWithFlags**
struct {
int ∗ **gridSize**
int **gridSize__val**
int ∗ **blockSize**
int **blockSize__val**
hipFunction_t **f**
size_t **dynSharedMemPerBlk**
int **blockSizeLimit**
} **hipModuleOccupancyMaxPotentialBlockSize**
struct {

int ∗ **gridSize**
int **gridSize__val**
int ∗ **blockSize**
int **blockSize__val**
hipFunction_t **f**
size_t **dynSharedMemPerBlk**
int **blockSizeLimit**
unsigned int **flags**
} **hipModuleOccupancyMaxPotentialBlockSizeWithFlags**
struct {
hipModule_t **module**
} **hipModuleUnload**
struct {
int ∗ **numBlocks**
int **numBlocks__val**
const void ∗ **f**
int **blockSize**
size_t **dynamicSMemSize**
} **hipOccupancyMaxActiveBlocksPerMultiprocessor**
struct {
int ∗ **numBlocks**
int **numBlocks__val**
const void ∗ **f**
int **blockSize**
size_t **dynamicSMemSize**
unsigned int **flags**
} **hipOccupancyMaxActiveBlocksPerMultiprocessorWithFlags**
struct {
int ∗ **gridSize**
int **gridSize__val**
int ∗ **blockSize**
int **blockSize__val**
const void ∗ **f**
size_t **dynSharedMemPerBlk**
int **blockSizeLimit**
} **hipOccupancyMaxPotentialBlockSize**
struct {
hipPointerAttribute_t ∗ **attributes**
hipPointerAttribute_t **attributes__val**
const void ∗ **ptr**
} **hipPointerGetAttributes**
struct {
int ∗ **runtimeVersion**
int **runtimeVersion__val**
} **hipRuntimeGetVersion**
struct {
int **deviceId**
} **hipSetDevice**
struct {
unsigned int **flags**
} **hipSetDeviceFlags**
struct {
const void ∗ **arg**
size_t **size**
size_t **offset**
} **hipSetupArgument**
struct {
const hipExternalSemaphore_t ∗ **extSemArray**

hipExternalSemaphore_t **extSemArray__val**
const hipExternalSemaphoreSignalParams ∗ **paramsArray**
hipExternalSemaphoreSignalParams **paramsArray__val**
unsigned int **numExtSems**
hipStream_t **stream**
} **hipSignalExternalSemaphoresAsync**
struct {
hipStream_t **stream**
hipStreamCallback_t **callback**
void ∗ **userData**
unsigned int **flags**
} **hipStreamAddCallback**
struct {
hipStream_t **stream**
void ∗ **dev_ptr**
size_t **length**
unsigned int **flags**
} **hipStreamAttachMemAsync**
struct {
hipStream_t **stream**
hipStreamCaptureMode **mode**
} **hipStreamBeginCapture**
struct {
hipStream_t ∗ **stream**
hipStream_t **stream__val**
} **hipStreamCreate**
struct {
hipStream_t ∗ **stream**
hipStream_t **stream__val**
unsigned int **flags**
} **hipStreamCreateWithFlags**
struct {
hipStream_t ∗ **stream**
hipStream_t **stream__val**
unsigned int **flags**
int **priority**
} **hipStreamCreateWithPriority**
struct {
hipStream_t **stream**
} **hipStreamDestroy**
struct {
hipStream_t **stream**
hipGraph_t ∗ **pGraph**
hipGraph_t **pGraph__val**
} **hipStreamEndCapture**
struct {
hipStream_t **stream**
unsigned int ∗ **flags**
unsigned int **flags__val**
} **hipStreamGetFlags**
struct {
hipStream_t **stream**
int ∗ **priority**
int **priority__val**
} **hipStreamGetPriority**
struct {
hipStream_t **stream**
} **hipStreamQuery**

struct {
   hipStream_t **stream**
} **hipStreamSynchronize**
struct {
   hipStream_t **stream**
   hipEvent_t **event**
   unsigned int **flags**
} **hipStreamWaitEvent**
struct {
   hipStream_t **stream**
   void ∗ **ptr**
   uint32_t **value**
   unsigned int **flags**
   unsigned int **mask**
} **hipStreamWaitValue32**
struct {
   hipStream_t **stream**
   void ∗ **ptr**
   uint64_t **value**
   unsigned int **flags**
   uint64_t **mask**
} **hipStreamWaitValue64**
struct {
   hipStream_t **stream**
   void ∗ **ptr**
   uint32_t **value**
   unsigned int **flags**
} **hipStreamWriteValue32**
struct {
   hipStream_t **stream**
   void ∗ **ptr**
   uint64_t **value**
   unsigned int **flags**
} **hipStreamWriteValue64**
struct {
   hipDeviceptr_t ∗ **dev_ptr**
   hipDeviceptr_t **dev_ptr__val**
   const textureReference ∗ **texRef**
   textureReference **texRef__val**
} **hipTexRefGetAddress**
struct {
   unsigned int ∗ **pFlags**
   unsigned int **pFlags__val**
   const textureReference ∗ **texRef**
   textureReference **texRef__val**
} **hipTexRefGetFlags**
struct {
   hipArray_Format ∗ **pFormat**
   hipArray_Format **pFormat__val**
   int ∗ **pNumChannels**
   int **pNumChannels__val**
   const textureReference ∗ **texRef**
   textureReference **texRef__val**
} **hipTexRefGetFormat**
struct {
   int ∗ **pmaxAnsio**
   int **pmaxAnsio__val**
   const textureReference ∗ **texRef**

textureReference **texRef__val**
} **hipTexRefGetMaxAnisotropy**
struct {
hipMipmappedArray_t ∗ **pArray**
hipMipmappedArray_t **pArray__val**
const textureReference ∗ **texRef**
textureReference **texRef__val**
} **hipTexRefGetMipMappedArray**
struct {
float ∗ **pbias**
float **pbias__val**
const textureReference ∗ **texRef**
textureReference **texRef__val**
} **hipTexRefGetMipmapLevelBias**
struct {
float ∗ **pminMipmapLevelClamp**
float **pminMipmapLevelClamp__val**
float ∗ **pmaxMipmapLevelClamp**
float **pmaxMipmapLevelClamp__val**
const textureReference ∗ **texRef**
textureReference **texRef__val**
} **hipTexRefGetMipmapLevelClamp**
struct {
size_t ∗ **ByteOffset**
size_t **ByteOffset__val**
textureReference ∗ **texRef**
textureReference **texRef__val**
hipDeviceptr_t **dptr**
size_t **bytes**
} **hipTexRefSetAddress**
struct {
textureReference ∗ **texRef**
textureReference **texRef__val**
const HIP_ARRAY_DESCRIPTOR ∗ **desc**
HIP_ARRAY_DESCRIPTOR **desc__val**
hipDeviceptr_t **dptr**
size_t **Pitch**
} **hipTexRefSetAddress2D**
struct {
textureReference ∗ **texRef**
textureReference **texRef__val**
float ∗ **pBorderColor**
float **pBorderColor__val**
} **hipTexRefSetBorderColor**
struct {
textureReference ∗ **texRef**
textureReference **texRef__val**
hipArray_Format **fmt**
int **NumPackedComponents**
} **hipTexRefSetFormat**
struct {
textureReference ∗ **texRef**
textureReference **texRef__val**
unsigned int **maxAniso**
} **hipTexRefSetMaxAnisotropy**
struct {
textureReference ∗ **texRef**
textureReference **texRef__val**

```
        float minMipMapLevelClamp
        float maxMipMapLevelClamp
    } hipTexRefSetMipmapLevelClamp
    struct {
        textureReference ∗ texRef
        textureReference texRef__val
        hipMipmappedArray ∗ mipmappedArray
        hipMipmappedArray mipmappedArray__val
        unsigned int Flags
    } hipTexRefSetMipmappedArray
    struct {
        const hipExternalSemaphore_t ∗ extSemArray
        hipExternalSemaphore_t extSemArray__val
        const hipExternalSemaphoreWaitParams ∗ paramsArray
        hipExternalSemaphoreWaitParams paramsArray__val
        unsigned int numExtSems
        hipStream_t stream
    } hipWaitExternalSemaphoresAsync
} args
```

5.28 HIP_ARRAY3D_DESCRIPTOR Struct Reference

Public Attributes

- size_t **Width**
- size_t **Height**
- size_t **Depth**
- enum hipArray_Format **Format**
- unsigned int **NumChannels**
- unsigned int **Flags**

5.29 HIP_ARRAY_DESCRIPTOR Struct Reference

Public Attributes

- size_t **Width**
- size_t **Height**
- enum hipArray_Format **Format**
- unsigned int **NumChannels**

5.30 hip_bfloat16 Struct Reference

Struct to represent a 16 bit brain floating point number.

Public Attributes

- uint16_t **data**

5.30.1 Detailed Description

Struct to represent a 16 bit brain floating point number.

5.31 hip_Memcpy2D Struct Reference

Public Attributes

- size_t **srcXInBytes**
- size_t **srcY**
- hipMemoryType **srcMemoryType**
- const void ∗ **srcHost**
- hipDeviceptr_t **srcDevice**
- hipArray ∗ **srcArray**
- size_t **srcPitch**
- size_t **dstXInBytes**
- size_t **dstY**
- hipMemoryType **dstMemoryType**
- void ∗ **dstHost**
- hipDeviceptr_t **dstDevice**
- hipArray ∗ **dstArray**
- size_t **dstPitch**
- size_t **WidthInBytes**
- size_t **Height**

5.32 HIP_MEMCPY3D Struct Reference

Public Attributes

- unsigned int **srcXInBytes**
- unsigned int **srcY**
- unsigned int **srcZ**
- unsigned int **srcLOD**
- hipMemoryType **srcMemoryType**
- const void ∗ **srcHost**
- hipDeviceptr_t **srcDevice**
- hipArray_t **srcArray**
- unsigned int **srcPitch**
- unsigned int **srcHeight**
- unsigned int **dstXInBytes**
- unsigned int **dstY**
- unsigned int **dstZ**
- unsigned int **dstLOD**
- hipMemoryType **dstMemoryType**
- void ∗ **dstHost**
- hipDeviceptr_t **dstDevice**
- hipArray_t **dstArray**
- unsigned int **dstPitch**
- unsigned int **dstHeight**
- unsigned int **WidthInBytes**
- unsigned int **Height**
- unsigned int **Depth**

5.33 HIP_RESOURCE_DESC_st Struct Reference

Public Attributes

- HIPresourcetype resType
-
```
union {
  struct {
    hipArray_t hArray
  } array
  struct {
    hipMipmappedArray_t hMipmappedArray
  } mipmap
  struct {
    hipDeviceptr_t devPtr
    hipArray_Format format
    unsigned int numChannels
    size_t sizeInBytes
  } linear
  struct {
    hipDeviceptr_t devPtr
    hipArray_Format format
    unsigned int numChannels
    size_t width
    size_t height
    size_t pitchInBytes
  } pitch2D
  struct {
    int reserved [32]
  } reserved
} res
```
- unsigned int flags

5.33.1 Member Data Documentation

5.33.1.1 devPtr

`hipDeviceptr_t HIP_RESOURCE_DESC_st::devPtr`

Device pointer

5.33.1.2 flags

`unsigned int HIP_RESOURCE_DESC_st::flags`

Flags (must be zero)

5.33.1.3 format

`hipArray_Format HIP_RESOURCE_DESC_st::format`

Array format

5.33.1.4 hArray

`hipArray_t HIP_RESOURCE_DESC_st::hArray`

HIP array

5.33.1.5 height

`size_t HIP_RESOURCE_DESC_st::height`

Height of the array in elements

5.33.1.6 hMipmappedArray

`hipMipmappedArray_t HIP_RESOURCE_DESC_st::hMipmappedArray`

HIP mipmapped array

5.33.1.7 numChannels

`unsigned int HIP_RESOURCE_DESC_st::numChannels`

Channels per array element

5.33.1.8 pitchInBytes

`size_t HIP_RESOURCE_DESC_st::pitchInBytes`

Pitch between two rows in bytes

5.33.1.9 resType

`HIPresourcetype HIP_RESOURCE_DESC_st::resType`

Resource type

5.33.1.10 sizeInBytes

`size_t HIP_RESOURCE_DESC_st::sizeInBytes`

Size in bytes

5.33.1.11 width

`size_t HIP_RESOURCE_DESC_st::width`

Width of the array in elements

5.34 HIP_RESOURCE_VIEW_DESC_st Struct Reference

Public Attributes

- HIPresourceViewFormat format
- size_t width
- size_t height
- size_t depth
- unsigned int firstMipmapLevel
- unsigned int lastMipmapLevel
- unsigned int firstLayer
- unsigned int lastLayer
- unsigned int **reserved** [16]

5.34.1 Detailed Description

Resource view descriptor

5.34.2 Member Data Documentation

5.34.2.1 depth

`size_t HIP_RESOURCE_VIEW_DESC_st::depth`

Depth of the resource view

5.34.2.2 firstLayer

`unsigned int HIP_RESOURCE_VIEW_DESC_st::firstLayer`

First layer index

5.34.2.3 firstMipmapLevel

`unsigned int HIP_RESOURCE_VIEW_DESC_st::firstMipmapLevel`

First defined mipmap level

5.34.2.4 format

`HIPresourceViewFormat HIP_RESOURCE_VIEW_DESC_st::format`

Resource view format

5.34.2.5 height

`size_t HIP_RESOURCE_VIEW_DESC_st::height`

Height of the resource view

5.34.2.6 lastLayer

`unsigned int HIP_RESOURCE_VIEW_DESC_st::lastLayer`

Last layer index

5.34.2.7 lastMipmapLevel

`unsigned int HIP_RESOURCE_VIEW_DESC_st::lastMipmapLevel`

Last defined mipmap level

5.34.2.8 width

`size_t HIP_RESOURCE_VIEW_DESC_st::width`

Width of the resource view

5.35 HIP_TEXTURE_DESC_st Struct Reference

Public Attributes

- HIPaddress_mode addressMode [3]
- HIPfilter_mode filterMode
- unsigned int flags
- unsigned int maxAnisotropy
- HIPfilter_mode mipmapFilterMode
- float mipmapLevelBias
- float minMipmapLevelClamp
- float maxMipmapLevelClamp
- float borderColor [4]
- int **reserved** [12]

5.35.1 Detailed Description

Texture descriptor

5.35.2 Member Data Documentation

5.35.2.1 addressMode

```
HIPaddress_mode HIP_TEXTURE_DESC_st::addressMode[3]
```

Address modes

5.35.2.2 borderColor

```
float HIP_TEXTURE_DESC_st::borderColor[4]
```

Border Color

5.35.2.3 filterMode

```
HIPfilter_mode HIP_TEXTURE_DESC_st::filterMode
```

Filter mode

5.35.2.4 flags

```
unsigned int HIP_TEXTURE_DESC_st::flags
```

Flags

5.35.2.5 maxAnisotropy

```
unsigned int HIP_TEXTURE_DESC_st::maxAnisotropy
```

Maximum anisotropy ratio

5.35.2.6 maxMipmapLevelClamp

```
float HIP_TEXTURE_DESC_st::maxMipmapLevelClamp
```

Mipmap maximum level clamp

5.35.2.7 minMipmapLevelClamp

```
float HIP_TEXTURE_DESC_st::minMipmapLevelClamp
```

Mipmap minimum level clamp

5.35.2.8 mipmapFilterMode

```
HIPfilter_mode HIP_TEXTURE_DESC_st::mipmapFilterMode
```

Mipmap filter mode

5.35.2.9 mipmapLevelBias

```
float HIP_TEXTURE_DESC_st::mipmapLevelBias
```

Mipmap level bias

5.36 hipArray Struct Reference

Public Attributes

- void ∗ **data**
- struct hipChannelFormatDesc **desc**
- unsigned int **type**
- unsigned int **width**
- unsigned int **height**
- unsigned int **depth**

- enum hipArray_Format **Format**
- unsigned int **NumChannels**
- bool **isDrv**
- unsigned int **textureType**

5.37 hipChannelFormatDesc Struct Reference

Public Attributes

- int **x**
- int **y**
- int **z**
- int **w**
- enum hipChannelFormatKind **f**

5.38 hipDeviceArch_t Struct Reference

Public Attributes

- unsigned hasGlobalInt32Atomics: 1

  32-bit integer atomics for global memory.
- unsigned hasGlobalFloatAtomicExch: 1

  32-bit float atomic exch for global memory.
- unsigned hasSharedInt32Atomics: 1

  32-bit integer atomics for shared memory.
- unsigned hasSharedFloatAtomicExch: 1

  32-bit float atomic exch for shared memory.
- unsigned hasFloatAtomicAdd: 1

  32-bit float atomic add in global and shared memory.
- unsigned hasGlobalInt64Atomics: 1

  64-bit integer atomics for global memory.
- unsigned hasSharedInt64Atomics: 1

  64-bit integer atomics for shared memory.
- unsigned hasDoubles: 1

  Double-precision floating point.
- unsigned hasWarpVote: 1

  Warp vote instructions (__any, __all).
- unsigned hasWarpBallot: 1

  Warp ballot instructions (__ballot).
- unsigned hasWarpShuffle: 1

  Warp shuffle operations. (__shfl_∗).
- unsigned hasFunnelShift: 1

  Funnel two words into one with shift&mask caps.
- unsigned hasThreadFenceSystem: 1

  __threadfence_system.
- unsigned hasSyncThreadsExt: 1

  __syncthreads_count, syncthreads_and, syncthreads_or.
- unsigned hasSurfaceFuncs: 1

  Surface functions.
- unsigned has3dGrid: 1

  Grid and group dims are 3D (rather than 2D).
- unsigned hasDynamicParallelism: 1

  Dynamic parallelism.

5.39 hipDeviceProp_t Struct Reference

Public Attributes

- char name [256]

  Device name.
- size_t totalGlobalMem

  Size of global memory region (in bytes).
- size_t sharedMemPerBlock

  Size of shared memory region (in bytes).
- int regsPerBlock

  Registers per block.
- int warpSize

  Warp size.
- int maxThreadsPerBlock

  Max work items per work group or workgroup max size.
- int maxThreadsDim [3]

  Max number of threads in each dimension (XYZ) of a block.
- int maxGridSize [3]

  Max grid dimensions (XYZ).
- int clockRate

  Max clock frequency of the multiProcessors in khz.
- int memoryClockRate

  Max global memory clock frequency in khz.
- int memoryBusWidth

  Global memory bus width in bits.
- size_t totalConstMem

  Size of shared memory region (in bytes).
- int major
- int minor
- int multiProcessorCount

  Number of multi-processors (compute units).
- int l2CacheSize

  L2 cache size.
- int maxThreadsPerMultiProcessor

  Maximum resident threads per multi-processor.
- int computeMode

  Compute mode.
- int clockInstructionRate
- hipDeviceArch_t arch

  Architectural feature flags. New for HIP.
- int concurrentKernels

  Device can possibly execute multiple kernels concurrently.
- int pciDomainID

  PCI Domain ID.
- int pciBusID

  PCI Bus ID.
- int pciDeviceID

  PCI Device ID.
- size_t maxSharedMemoryPerMultiProcessor

  Maximum Shared Memory Per Multiprocessor.
- int isMultiGpuBoard

    1 if device is on a multi-GPU board, 0 if not.
- int canMapHostMemory

    Check whether HIP can map host memory.
- int gcnArch

    DEPRECATED: use gcnArchName instead.
- char gcnArchName [256]

    AMD GCN Arch Name.
- int integrated

    APU vs dGPU.
- int cooperativeLaunch

    HIP device supports cooperative launch.
- int cooperativeMultiDeviceLaunch

    HIP device supports cooperative launch on multiple devices.
- int maxTexture1DLinear

    Maximum size for 1D textures bound to linear memory.
- int maxTexture1D

    Maximum number of elements in 1D images.
- int maxTexture2D [2]

    Maximum dimensions (width, height) of 2D images, in image elements.
- int maxTexture3D [3]

    Maximum dimensions (width, height, depth) of 3D images, in image elements.
- unsigned int ∗ hdpMemFlushCntl

    Addres of HDP_MEM_COHERENCY_FLUSH_CNTL register.
- unsigned int ∗ hdpRegFlushCntl

    Addres of HDP_REG_COHERENCY_FLUSH_CNTL register.
- size_t memPitch

    Maximum pitch in bytes allowed by memory copies.
- size_t textureAlignment

    Alignment requirement for textures.
- size_t texturePitchAlignment

    Pitch alignment requirement for texture references bound to pitched memory.
- int kernelExecTimeoutEnabled

    Run time limit for kernels executed on the device.
- int ECCEnabled

    Device has ECC support enabled.
- int tccDriver

    1:If device is Tesla device using TCC driver, else 0
- int cooperativeMultiDeviceUnmatchedFunc
- int cooperativeMultiDeviceUnmatchedGridDim
- int cooperativeMultiDeviceUnmatchedBlockDim
- int cooperativeMultiDeviceUnmatchedSharedMem
- int isLargeBar

    1: if it is a large PCI bar device, else 0
- int asicRevision

    Revision of the GPU in this device.
- int managedMemory

    Device supports allocating managed memory on this system.
- int directManagedMemAccessFromHost

    Host can directly access managed memory on the device without migration.
- int concurrentManagedAccess

    Device can coherently access managed memory concurrently with the CPU.
- int pageableMemoryAccess
- int pageableMemoryAccessUsesHostPageTables

    Device accesses pageable memory via the host's page tables.

5.39.1 Detailed Description

hipDeviceProp

5.39.2 Member Data Documentation

5.39.2.1 clockInstructionRate

`int hipDeviceProp_t::clockInstructionRate`
Frequency in khz of the timer used by the device-side "clock*" instructions. New for HIP.

5.39.2.2 cooperativeMultiDeviceUnmatchedBlockDim

`int hipDeviceProp_t::cooperativeMultiDeviceUnmatchedBlockDim`
HIP device supports cooperative launch on multiple devices with unmatched block dimensions

5.39.2.3 cooperativeMultiDeviceUnmatchedFunc

`int hipDeviceProp_t::cooperativeMultiDeviceUnmatchedFunc`
HIP device supports cooperative launch on multiple devices with unmatched functions

5.39.2.4 cooperativeMultiDeviceUnmatchedGridDim

`int hipDeviceProp_t::cooperativeMultiDeviceUnmatchedGridDim`
HIP device supports cooperative launch on multiple devices with unmatched grid dimensions

5.39.2.5 cooperativeMultiDeviceUnmatchedSharedMem

`int hipDeviceProp_t::cooperativeMultiDeviceUnmatchedSharedMem`
HIP device supports cooperative launch on multiple devices with unmatched shared memories

5.39.2.6 major

`int hipDeviceProp_t::major`
Major compute capability. On HCC, this is an approximation and features may differ from CUDA CC. See the arch feature flags for portable ways to query feature caps.

5.39.2.7 minor

`int hipDeviceProp_t::minor`
Minor compute capability. On HCC, this is an approximation and features may differ from CUDA CC. See the arch feature flags for portable ways to query feature caps.

5.39.2.8 pageableMemoryAccess

`int hipDeviceProp_t::pageableMemoryAccess`
Device supports coherently accessing pageable memory without calling hipHostRegister on it

5.40 hipExtent Struct Reference

Public Attributes

- size_t **width**
- size_t **height**
- size_t **depth**

5.41 hipExternalMemoryBufferDesc_st Struct Reference

Public Attributes

- unsigned long long **offset**
- unsigned long long **size**
- unsigned int **flags**

5.42 hipExternalMemoryHandleDesc_st Struct Reference

Public Attributes

- hipExternalMemoryHandleType **type**
-

```
union {
  int fd
  struct {
    void * handle
    const void * name
  } win32
} handle
```

- unsigned long long **size**
- unsigned int **flags**

5.43 hipExternalSemaphoreHandleDesc_st Struct Reference

Public Attributes

- hipExternalSemaphoreHandleType **type**
-

```
union {
  int fd
  struct {
    void * handle
    const void * name
  } win32
} handle
```

- unsigned int **flags**

5.44 hipExternalSemaphoreSignalParams_st Struct Reference

Public Attributes

-

```
struct {
  struct {
    unsigned long long value
  } fence
  struct {
    unsigned long long key
  } keyedMutex
  unsigned int reserved [12]
} params
```

- unsigned int **flags**

5.45 hipExternalSemaphoreWaitParams_st Struct Reference

Public Attributes

-

```
struct {
   struct {
      unsigned long long value
   } fence
   struct {
      unsigned long long key
      unsigned int timeoutMs
   } keyedMutex
   unsigned int reserved [10]
} params
```

- unsigned int **flags**

5.45.1 Detailed Description

External semaphore wait parameters, compatible with driver type

5.46 hipFuncAttributes Struct Reference

Public Attributes

- int **binaryVersion**
- int **cacheModeCA**
- size_t **constSizeBytes**
- size_t **localSizeBytes**
- int **maxDynamicSharedSizeBytes**
- int **maxThreadsPerBlock**
- int **numRegs**
- int **preferredShmemCarveout**
- int **ptxVersion**
- size_t **sharedSizeBytes**

5.47 hipHostNodeParams Struct Reference

Public Attributes

- hipHostFn_t **fn**
- void ∗ **userData**

5.48 hipIpcEventHandle_st Struct Reference

Public Attributes

- char **reserved** [HIP_IPC_HANDLE_SIZE]

5.49 hipIpcMemHandle_st Struct Reference

Public Attributes

- char **reserved** [HIP_IPC_HANDLE_SIZE]

5.50 hipKernelNodeParams Struct Reference

Public Attributes

- dim3 **blockDim**
- void ∗∗ **extra**
- void ∗ **func**
- dim3 **gridDim**
- void ∗∗ **kernelParams**
- unsigned int **sharedMemBytes**

5.51 hipLaunchParams_t Struct Reference

Public Attributes

- void ∗ func

    Device function symbol.
- dim3 gridDim

    Grid dimentions.
- dim3 blockDim

    Block dimentions.
- void ∗∗ args

    Arguments.
- size_t sharedMem

    Shared memory.
- hipStream_t stream

    Stream identifier.

5.52 hipMemcpy3DParms Struct Reference

Public Attributes

- hipArray_t **srcArray**
- struct hipPos **srcPos**
- struct hipPitchedPtr **srcPtr**
- hipArray_t **dstArray**
- struct hipPos **dstPos**
- struct hipPitchedPtr **dstPtr**
- struct hipExtent **extent**
- enum hipMemcpyKind **kind**

5.53 hipMemsetParams Struct Reference

Public Attributes

- void ∗ **dst**
- unsigned int **elementSize**
- size_t **height**

- size_t **pitch**
- unsigned int **value**
- size_t **width**

5.54 hipMipmappedArray Struct Reference

Public Attributes

- void ∗ **data**
- struct hipChannelFormatDesc **desc**
- unsigned int **type**
- unsigned int **width**
- unsigned int **height**
- unsigned int **depth**
- unsigned int **min_mipmap_level**
- unsigned int **max_mipmap_level**
- unsigned int **flags**
- enum hipArray_Format **format**

5.55 hipPitchedPtr Struct Reference

Public Attributes

- void ∗ **ptr**
- size_t **pitch**
- size_t **xsize**
- size_t **ysize**

5.56 hipPointerAttribute_t Struct Reference

Public Attributes

- enum hipMemoryType **memoryType**
- int **device**
- void ∗ **devicePointer**
- void ∗ **hostPointer**
- int **isManaged**
- unsigned **allocationFlags**

5.56.1 Detailed Description

Pointer attributes

5.57 hipPos Struct Reference

Public Attributes

- size_t **x**
- size_t **y**
- size_t **z**

5.58 hipResourceDesc Struct Reference

Public Attributes

- enum hipResourceType **resType**
-
```
union {
  struct {
    hipArray_t array
  } array
  struct {
    hipMipmappedArray_t mipmap
  } mipmap
  struct {
    void ∗ devPtr
    struct hipChannelFormatDesc desc
    size_t sizeInBytes
  } linear
  struct {
    void ∗ devPtr
    struct hipChannelFormatDesc desc
    size_t width
    size_t height
    size_t pitchInBytes
  } pitch2D
} res
```

5.58.1 Detailed Description

HIP resource descriptor

5.59 hipResourceViewDesc Struct Reference

Public Attributes

- enum hipResourceViewFormat **format**
- size_t **width**
- size_t **height**
- size_t **depth**
- unsigned int **firstMipmapLevel**
- unsigned int **lastMipmapLevel**
- unsigned int **firstLayer**
- unsigned int **lastLayer**

5.59.1 Detailed Description

hip resource view descriptor

5.60 hipTextureDesc Struct Reference

Public Attributes

- enum hipTextureAddressMode **addressMode** [3]
- enum hipTextureFilterMode **filterMode**
- enum hipTextureReadMode **readMode**
- int **sRGB**

- float **borderColor** [4]
- int **normalizedCoords**
- unsigned int **maxAnisotropy**
- enum hipTextureFilterMode **mipmapFilterMode**
- float **mipmapLevelBias**
- float **minMipmapLevelClamp**
- float **maxMipmapLevelClamp**

5.60.1 Detailed Description

hip texture descriptor

5.61 int1 Union Reference

Public Attributes

- int **data**

5.62 int16 Union Reference

Public Attributes

- int **data** [16]

5.63 int2 Union Reference

Public Attributes

- int **data** [2]

5.64 int3 Union Reference

Public Attributes

- int4 **data**

5.65 int4 Union Reference

Public Attributes

- int **data** [4]

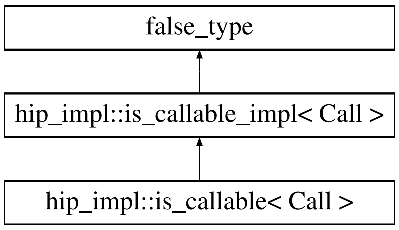
5.66 int8 Union Reference

Public Attributes

- int **data** [8]

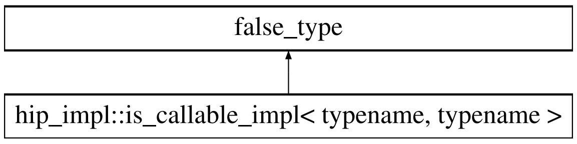
5.67 hip_impl::is_callable< Call > Struct Template Reference

Inheritance diagram for hip_impl::is_callable< Call >:


false_type
hip_impl::is_callable_impl< Call >
hip_impl::is_callable< Call >

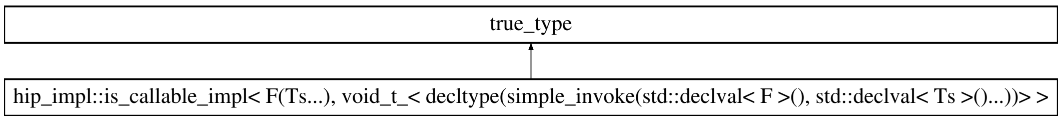
5.68 hip_impl::is_callable_impl< typename, typename > Struct Template Reference

Inheritance diagram for hip_impl::is_callable_impl< typename, typename >:



5.69 hip_impl::is_callable_impl< F(Ts...), void_t_< decltype(simple_invoke(std::declval< F >(), std::declval< Ts >()...))> > Struct Template Reference

Inheritance diagram for hip_impl::is_callable_impl< F(Ts...), void_t_< decltype(simple_invoke(std::declval< F >(), std::declval< Ts >()...))> >:


true_type
hip_impl::is_callable_impl< F(Ts...), void_t_< decltype(simple_invoke(std::declval< F >(), std::declval< Ts >()...))> >

5.70 hip_impl::kernarg Class Reference

Public Member Functions

- **kernarg** (kernarg &&)
- std::uint8_t ∗ **data** ()
- std::size_t **size** ()
- void **reserve** (std::size_t)
- void **resize** (std::size_t)
- **kernarg** (kernarg &&)
- std::uint8_t ∗ **data** ()
- std::size_t **size** ()
- void **reserve** (std::size_t)
- void **resize** (std::size_t)

5.71 hip_impl::kernargs_size_align Class Reference

Public Member Functions

- std::size_t **size** (std::size_t n) const

- std::size_t **alignment** (std::size_t n) const
- const void ∗ **getHandle** () const
- std::size_t **size** (std::size_t n) const
- std::size_t **alignment** (std::size_t n) const
- const void ∗ **getHandle** () const

Friends

- kernargs_size_align **program_state::get_kernargs_size_align** (std::uintptr_t)
- kernargs_size_align **program_state::get_kernargs_size_align** (std::uintptr_t)

5.72 long1 Union Reference

Public Attributes

- long **data**

5.73 long16 Union Reference

Public Attributes

- long **data** [16]

5.74 long2 Union Reference

Public Attributes

- long **data** [2]

5.75 long3 Union Reference

Public Attributes

- long4 **data**

5.76 long4 Union Reference

Public Attributes

- long **data** [4]

5.77 long8 Union Reference

Public Attributes

- long **data** [8]

5.78 longlong1 Union Reference

Public Attributes

- long long **data**

5.79 longlong16 Union Reference

Public Attributes

- long long **data** [16]

5.80 longlong2 Union Reference

Public Attributes

- long long **data** [2]

5.81 longlong3 Union Reference

Public Attributes

- longlong4 **data**

5.82 longlong4 Union Reference

Public Attributes

- long long **data** [4]

5.83 longlong8 Union Reference

Public Attributes

- long long **data** [8]

5.84 hip_impl::program_state Class Reference

Public Member Functions

- **program_state** (const program_state &)=delete
- hipFunction_t **kernel_descriptor** (std::uintptr_t, hsa_agent_t)
- kernargs_size_align **get_kernargs_size_align** (std::uintptr_t)
- hsa_executable_t **load_executable** (const char ∗, const size_t, hsa_executable_t, hsa_agent_t)
- hsa_executable_t **load_executable_no_copy** (const char ∗, const size_t, hsa_executable_t, hsa_agent_t)
- void ∗ **global_addr_by_name** (const char ∗name)
- **program_state** (const program_state &)=delete
- hipFunction_t **kernel_descriptor** (std::uintptr_t, hsa_agent_t)
- kernargs_size_align **get_kernargs_size_align** (std::uintptr_t)
- hsa_executable_t **load_executable** (const char ∗, const size_t, hsa_executable_t, hsa_agent_t)
- hsa_executable_t **load_executable_no_copy** (const char ∗, const size_t, hsa_executable_t, hsa_agent_t)
- void ∗ **global_addr_by_name** (const char ∗name)

Friends

- class **agent_globals_impl**

5.85 short1 Union Reference

Public Attributes

- short **data**

5.86 short16 Union Reference

Public Attributes

- short **data** [16]

5.87 short2 Union Reference

Public Attributes

- short **data** [2]

5.88 short3 Union Reference

Public Attributes

- short4 **data**

5.89 short4 Union Reference

Public Attributes

- short **data** [4]

5.90 short8 Union Reference

Public Attributes

- short **data** [8]

5.91 surfaceReference Struct Reference

Public Attributes

- hipSurfaceObject_t **surfaceObject**

5.91.1 Detailed Description

hip surface reference

5.92 TData Union Reference

Public Attributes

- __hip_float4_vector_value_type **f**
- __hip_int4_vector_value_type **i**
- __hip_uint4_vector_value_type **u**

5.93 textureReference Struct Reference

Public Attributes

- int **normalized**
- enum hipTextureReadMode **readMode**
- enum hipTextureFilterMode **filterMode**

- enum hipTextureAddressMode **addressMode** [3]
- struct hipChannelFormatDesc **channelDesc**
- int **sRGB**
- unsigned int **maxAnisotropy**
- enum hipTextureFilterMode **mipmapFilterMode**
- float **mipmapLevelBias**
- float **minMipmapLevelClamp**
- float **maxMipmapLevelClamp**
- hipTextureObject_t **textureObject**
- int **numChannels**
- enum hipArray_Format **format**

5.93.1 Detailed Description

hip texture reference

5.94 uchar1 Union Reference

Public Attributes

- unsigned char **data**

5.95 uchar16 Union Reference

Public Attributes

- unsigned char **data** [16]

5.96 uchar2 Union Reference

Public Attributes

- unsigned char **data** [2]

5.97 uchar2Holder Struct Reference

Public Attributes

-

```
union {
   unsigned int ui [2]
   unsigned char c [8]
};
```

5.98 uchar3 Union Reference

Public Attributes

- uchar4 **data**

5.99 uchar4 Union Reference

Public Attributes

- unsigned char **data** [4]

5.100 uchar8 Union Reference

Public Attributes

- unsigned char **data** [8]

5.101 ucharHolder Struct Reference

Public Attributes

-

  union {
     unsigned char **c** [4]
     unsigned int **ui**
  } **__attribute__**

5.102 uint1 Union Reference

Public Attributes

- unsigned int **data**

5.103 uint16 Union Reference

Public Attributes

- unsigned int **data** [16]

5.104 uint2 Union Reference

Public Attributes

- unsigned int **data** [2]

5.105 uint3 Union Reference

Public Attributes

- uint4 **data**

5.106 uint4 Union Reference

Public Attributes

- unsigned int **data** [4]

5.107 uint8 Union Reference

Public Attributes

- unsigned int **data** [8]

5.108 ulong1 Union Reference

Public Attributes

- unsigned long **data**

5.109 ulong16 Union Reference

Public Attributes

- unsigned long **data** [16]

5.110 ulong2 Union Reference

Public Attributes

- unsigned long **data** [2]

5.111 ulong3 Union Reference

Public Attributes

- ulong4 **data**

5.112 ulong4 Union Reference

Public Attributes

- unsigned long **data** [4]

5.113 ulong8 Union Reference

Public Attributes

- unsigned long **data** [8]

5.114 ulonglong1 Union Reference

Public Attributes

- unsigned long long **data**

5.115 ulonglong16 Union Reference

Public Attributes

- unsigned long long **data** [16]

5.116 ulonglong2 Union Reference

Public Attributes

- unsigned long long **data** [2]

5.117 ulonglong3 Union Reference

Public Attributes

- ulonglong4 **data**

5.118 ulonglong4 Union Reference

Public Attributes

- unsigned long long **data** [4]

5.119 ulonglong8 Union Reference

Public Attributes

- unsigned long long **data** [8]

5.120 ushort1 Union Reference

Public Attributes

- unsigned short **data**

5.121 ushort16 Union Reference

Public Attributes

- unsigned short **data** [16]

5.122 ushort2 Union Reference

Public Attributes

- unsigned short **data** [2]

5.123 ushort3 Union Reference

Public Attributes

- ushort4 **data**

5.124 ushort4 Union Reference

Public Attributes

- unsigned short **data** [4]

5.125 ushort8 Union Reference

Public Attributes

- unsigned short **data** [8]