CUDA Tools SDK
CUPTI User's Guide

DA-05679-001_v01 | February 2012

Document Change History

Ver	Date	Resp	Reason for change
v01	2011/1/19	DG	Initial revision for CUDA Tools SDK 4.0
v02	2012/1/5	DG	Revisions for CUDA Tools SDK 4.1
v03	2012/2/13	DG	Revisions for CUDA Tools SDK 4.2

CUPTI

The *CUDA Profiling Tools Interface* (CUPTI) enables the creation of profiling and tracing tools that target CUDA applications. CUPTI provides four APIs, the *Activity API*, the *Callback API*, the *Event API*, and the *Metric API*. Using these APIs, you can develop profiling tools that give insight into the CPU and GPU behavior of CUDA applications. CUPTI is delivered as a dynamic library on all platforms supported by CUDA.

CUPTI Compatibility and Requirements

New versions of the CUDA driver are backwards compatible with older versions of CUPTI. For example, a developer using a profiling tool based on CUPTI 4.1 can update to a more recently released CUDA driver. However, new versions of CUPTI are not backwards compatible with older versions of the CUDA driver. For example, a developer using a profiling tool based on CUPTI 4.1 must have a version of the CUDA driver released with CUDA Toolkit 4.1 (or later) installed as well. CUPTI calls will fail with `CUPTI_ERROR_NOT_INITIALIZED` if the CUDA driver version is not compatible with the CUPTI version.

CUPTI Initialization

CUPTI initialization occurs lazily the first time you invoke any CUPTI function. For the Event, Metric, and Callback APIs there are no requirements on when this initialization must occur (i.e. you can invoke the first CUPTI function at any point). For correct operation, the Activity API does require that CUPTI be initialized before any CUDA driver or runtime API is invoked. See the CUPTI Activity API section for more information on CUPTI initialization requirements for the activity API.

CUPTI Activity API

The CUPTI Activity API allows you to asychronously collect a trace of an application's CPU and GPU CUDA activity. The following terminology is used by the activity API.

Activity Record: CPU and GPU activity is reported in C data structures called activity records. There is a different C structure type for each activity kind (e.g. `CUpti_ActivityMemcpy`). Records are generically referred to using the `CUpti_Activity` type. This type contains only a kind field that indicates the kind of the activity record. Using this kind, the object can be cast from the generic `CUpti_Activity` type to the specific type representing the activity. See the `printActivity` function in the `activity_trace` sample for an example.

Activity Buffer: CUPTI fills activity buffers with activity records as the corresponding activities occur on the CPU and GPU. The CUPTI client is responsible for providing activity buffers as necessary to ensure that no records are dropped.

Activity Queue: CUPTI maintains queues of activity buffers. There are three types of queues: global, context, and stream.

Global Queue: The global queue collects all activity records that are not associated with a valid context. All device, context, and API activity records are collected in the global queue. A buffer is enqueued in the global queue by specifying NULL for the `context` argument.

Context Queue: Each context queue collects activity records associated with that context that are not associated with a specific stream or that are associated with the default stream. A buffer is enqueued in a context queue by specifying 0 for the `streamId` argument and a valid context for the `context` argument.

Stream Queue: Each stream queue collects memcpy, memset, and kernel activity records associated with the stream. A buffer is enqueued in a stream queue by specifying a non-zero value for the `streamId` argument and a valid context for the `context` argument. A `streamId` can be obtained from a `CUstream` object by using the `cuptiGetStreamId` function.

CUPTI must be initialized in a specific manner to ensure that activity records are collected correctly. Most importantly, CUPTI must be initialized before any CUDA driver or runtime API is invoked. Initialization can be done by enqueuing one or more buffers in the global queue, as shown in the `initTrace` function of the **activity_trace** sample. Also, to ensure that device activity records are collected, you must enable device records before CUDA is initialized (also shown in the `initTrace` function).

The other important requirement for correct activity API operation is the need to enqueue at least one buffer in the context queue of each context as it is created. Thus, as shown in the `activity_trace` example, the CUPTI client should use the resource callback to enqueue at least one buffer when context creation is indicated by

CUPTI_CBID_RESOURCE_CONTEXT_CREATED. Using the stream queues is optional, but may be useful to reduce or eliminate application perturbations caused by the need to process or save the activity records returned in the buffers. For example, if a stream queue is used, that queue can be flushed when the stream is synchronized.

Each activity buffer must be allocated by the CUPTI client, and passed to CUPTI using the cuptiActivityEnqueueBuffer function. Enqueuing a buffer passes ownership to CUPTI, and so the client should not read or write the contents of a buffer once it is enqueued. Ownership of a buffer is regained by using the cuptiActivityDequeueBuffer function.

As the application executes, the activity buffers will fill. It is the CUPTI client's responsibility to ensure that a sufficient number of appropriately sized buffers are enqueued to avoid dropped activity records. Activity buffers can be enqueued and dequeued at the following points. Enqueuing and dequeuing activity buffers at any other point may result in corrupt activity records.

Before CUDA initialization: Buffers can be enqueued and dequeued to/from the global queue before any CUDA driver or runtime API is called.

In synchronization or resource callbacks: At context creation, destruction, or synchronization, buffers may be enqueued or dequeued to/from the corresponding context queue, and from any stream queues associated with streams in that context. At stream creation, destruction, or synchronization, buffers may be enqueued or dequeued to/from the corresponding stream queue. The global queue may also be enqueued or dequeued at this time.

After device synchronization: After a CUDA device is synchronized or reset (with cudaDeviceSynchronize or cudaDeviceReset), and before any subsequent CUDA driver or runtime API is invoked, buffers can enqueued and dequeued to/from any activity queue.

The **activity_trace** sample described on page 32 shows how to use global, context, and stream queues to collect a trace of CPU and GPU activity for a simple application.

CUPTI Callback API

The CUPTI Callback API allows you to register a callback into your own code. Your callback will be invoked when the application being profiled calls a CUDA runtime or driver function, or when certain events occur in the CUDA driver. The following terminology is used by the callback API.

Callback Domain: Callbacks are grouped into domains to make it easier to associate your callback functions with groups of related CUDA functions or events. There are currently four callback domains, as defined by CUpti_CallbackDomain: a domain for CUDA runtime functions, a domain for CUDA driver functions, a domain for CUDA resource tracking, and a domain for CUDA synchronization notification.

Callback ID: Each callback is given a unique ID within the corresponding callback domain so that you can identify it within your callback function. The CUDA driver API IDs are defined in `cupti_driver_cbid.h` and the CUDA runtime API IDs are defined in `cupti_runtime_cbid.h`. Both of these headers are included for you when you include `cupti.h`. The CUDA resource callback IDs are defined by `CUpti_CallbackIdResource` and the CUDA synchronization callback IDs are defined by `CUpti_CallbackIdSync`.

Callback Function: Your callback function must be of type `CUpti_CallbackFunc`. This function type has two arguments that specify the callback domain and ID so that you know why the callback is occurring. The type also has a `cbdata` argument that is used to pass data specific to the callback.

Subscriber: A subscriber is used to associate each of your callback functions with one or more CUDA API functions. There can be at most one subscriber initialized with `cuptiSubscribe()` at any time. Before initializing a new subscriber, the existing subscriber must be finalized with `cuptiUnsubscribe()`.

Each callback domain is described in detail below.

Driver and Runtime API Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_DRIVER_API` or `CUPTI_CB_DOMAIN_RUNTIME_API` domains, you can associate a callback function with one or more CUDA API functions. When those CUDA functions are invoked in the application, your callback function is invoked as well. For these domains, the `cbdata` argument to your callback function will be of the type `CUpti_CallbackData`.

The following code shows a typical sequence used to associate a callback function with one or more CUDA API functions. To simplify the presentation error checking code has been removed.

```
CUpti_SubscriberHandle subscriber;
MyDataStruct *my_data = ...;
...
cuptiSubscribe(&subscriber,
               (CUpti_CallbackFunc)my_callback , my_data);
cuptiEnableDomain(1, subscriber,
                  CUPTI_CB_DOMAIN_RUNTIME_API);
```

First, `cuptiSubscribe` is used to initialize a subscriber with the `my_callback` callback function. Next, `cuptiEnableDomain` is used to associate that callback with all the CUDA runtime API functions. Using this code sequence will cause `my_callback` to be called twice each time any of the CUDA runtime API functions are invoked, once on entry to the CUDA function and once just before exit from the CUDA function. CUPTI callback API functions `cuptiEnableCallback` and `cuptiEnableAllDomains` can also be used to associate CUDA API functions with a callback (see reference below for more information).

The following code shows a typical callback function.

```
void CUPTIAPI
my_callback(void *userdata, CUpti_CallbackDomain domain,
            CUpti_CallbackId cbid, const void *cbdata)
{
  const CUpti_CallbackData *cbInfo = (CUpti_CallbackData *)↩
     cbdata;
  MyDataStruct *my_data = (MyDataStruct *)userdata;

  if ((domain == CUPTI_CB_DOMAIN_RUNTIME_API) &&
      (cbid == CUPTI_RUNTIME_TRACE_CBID_cudaMemcpy_v3020))  {
    if (cbInfo->callbackSite == CUPTI_API_ENTER) {
        cudaMemcpy_v3020_params *funcParams =
             (cudaMemcpy_v3020_params *)(cbInfo->
                 functionParams);

        size_t count = funcParams->count;
        enum cudaMemcpyKind kind = funcParams->kind;
        ...
      }
  ...
```

In your callback function, you use the `CUpti_CallbackDomain` and `CUpti_CallbackID` parameters to determine which CUDA API function invocation is causing this callback. In the example above, we are checking for the CUDA runtime `cudaMemCpy` function. The `CUpti_CallbackData` parameter holds a structure of useful information that can be used within the callback. In this case we use the `callbackSite` member of the structure to detect that the callback is occurring on entry to `cudaMemCpy`, and we use the `functionParams` member to access the parameters that were passed to `cudaMemCpy`. To access the parameters we first cast `functionParams` to a structure type corresponding to the `cudaMemCpy` function. These parameter structures are contained in `generated_cuda_runtime_api_meta.h`, `generated_cuda_meta.h`, and a number of other files. When possible these files are included for you by `cupti.h`.

The **callback_event** and **callback_timestamp** samples described on page 32 both show how to use the callback API for the driver and runtime API domains.

Resource Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_RESOURCE` domain, you can associate a callback function with some CUDA resource creation and destruction events. For example, when a CUDA context is created, your callback function will be invoked with a callback ID equal to `CUPTI_CBID_RESOURCE_CONTEXT_CREATED`. For this domain, the `cbdata` argument to your callback function will be of the type `CUpti_ResourceData`.

The **activity_trace** sample described on page 32 shows how to use the resource callback.

Synchronization Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_SYNCHRONIZE` domain, you can associate a callback function with CUDA context and stream synchronizations. For example, when a CUDA context is synchronized, your callback function will be invoked with a callback ID equal to `CUPTI_CBID_SYNCHRONIZE_CONTEXT_SYNCHRONIZED`. For this domain, the `cbdata` argument to your callback function will be of the type `CUpti_SynchronizeData`.

The **activity_trace** sample described on page 32 shows how to use the synchronization callback.

CUPTI Event API

The CUPTI Event API allows you to query, configure, start, stop, and read the event counters on a CUDA-enabled device. The following terminology is used by the event API.

Event: An event is a countable activity, action, or occurrence on a device.

Event ID: Each event is assigned a unique identifier. A named event will represent the same activity, action, or occurence on all device types. But the named event may have different IDs on different device families. Use `cuptiEventGetIdFromName` to get the ID for a named event on a particular device.

Event Category: Each event is placed in one of the categories defined by `CUpti_EventCategory`. The category indicates the general type of activity, action, or occurrence measured by the event.

Event Domain: A device exposes one or more event domains. Each event domain represents a group of related events available on that device. A device may have multiple instances of a domain, indicating that the device can simultaneously record multiple instances of each event within that domain.

Event Group: An event group is a collection of events that are managed together. The number and type of events that can be added to an event group are subject to device-specific limits. At any given time, a device may be configured to count events from a limited number of event groups. All events in an event group must belong to the same event domain.

Event Group Set: An event group set is a collection of event groups that can be enabled at the same time. Event group sets are created by `cuptiEventGroupSetsCreate` and `cuptiMetricCreateEventGroupSets`.

The tables included in this section list the events available for each device, as determined by the device's compute capability. You can also determine the events available on a device using the `cuptiDeviceEnumEventDomains` and `cuptiEventDomainEnumEvents` functions. The **cupti_query** sample described on page 32 shows how to use these functions. You can also enumerate all the CUPTI events available on any device using the `cuptiEnumEventDomains` function.

Configuring and reading event counts requires the following steps. First, select your event collection mode. If you want to count events that occur during the execution of a kernel, use `cuptiSetEventCollectionMode` to set mode `CUPTI_EVENT_COLLECTION_MODE_KERNEL`. If you want to continuously sample the event counts, use mode `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS`. Next determine the names of the events that you want to count, and then use the `cuptiEventGroupCreate`, `cuptiEventGetIdFromName`, and `cuptiEventGroupAddEvent` functions to create and initialize an event group with those events. If you are unable to add all the events to a single event group then you will need to create multiple event groups. Alternatively, you can use the `cuptiEventGroupSetsCreate` function to automatically create the event group(s) required for a set of events.

To begin counting a set of events, enable the event group or groups that contain those events by using the `cuptiEventGroupEnable` function. If your events are contained in multiple event groups you may be unable to enable all of the event groups at the same time, due to device limitations. In this case, you will need to gather the events across multiple executions of the application.

Use the `cuptiEventGroupReadEvent` and/or `cuptiEventGroupReadAllEvents` functions to read the event values. When you are done collecting events, use the `cuptiEventGroupDisable` function to stop counting of the events contained in an event group. The **callback_event** sample described on page 32 shows how to use these functions to create, enable, and disable event groups, and how to read event counts.

Collecting Kernel Execution Events

A common use of the event API is to count a set of events during the execution of a kernel (as demonstrated by the **callback_event** sample). The following code shows a typical callback used for this purpose. Assume that the callback was enabled only for a kernel launch using the CUDA runtime (i.e. by `cuptiEnableCallback(1, subscriber, CUPTI_CB_DOMAIN_RUNTIME_API, CUPTI_RUNTIME_TRACE_CBID_cudaLaunch_v3020)`. To simplify the presentation error checking code has been removed.

```
static void CUPTIAPI
getEventValueCallback(void *userdata,
                      CUpti_CallbackDomain domain,
                      CUpti_CallbackId cbid,
                      const void *cbdata)
{
  const CUpti_CallbackData *cbData =
                (CUpti_CallbackData *)cbdata;

  if (cbData->callbackSite == CUPTI_API_ENTER) {
    cudaThreadSynchronize();
    cuptiSetEventCollectionMode(cbInfo->context,
                                CUPTI_EVENT_COLLECTION_MODE_KERNEL↩
                                   );
    cuptiEventGroupEnable(eventGroup);
  }

  if (cbData->callbackSite == CUPTI_API_EXIT) {
    cudaThreadSynchronize();
    cuptiEventGroupReadEvent(eventGroup,
                             CUPTI_EVENT_READ_FLAG_ACCUMULATE,
                             eventId,
                             &bytesRead, &eventVal);

    cuptiEventGroupDisable(eventGroup);
  }
}
```

Two synchronization points are used to ensure that events are counted only for the execution of the kernel. If the application contains other threads that launch kernels, then additional thread-level synchronization must also be introduced to ensure that those threads do not launch kernels while the callback is collecting events. When the cudaLaunch API is entered (that is, before the kernel is actually launched on the device), `cudaThreadSynchronize` is used to wait until the GPU is idle. The event collection mode is set to `CUPTI_EVENT_COLLECTION_MODE_KERNEL` so that the event counters are automatically started and stopped just before and after the kernel executes. Then event collection is enabled with `cuptiEventGroupEnable`.

When the cudaLaunch API is exited (that is, after the kernel is queued for execution on the GPU) another `cudaThreadSynchronize` is used to cause the CPU thread to wait for the kernel to finish execution. Finally, the event counts are read with `cuptiEventGroupReadEvent`.

Sampling Events

The event API can also be used to sample event values while a kernel or kernels are executing (as demonstrated by the **event_sampling** sample). The sample shows one possible way to perform the sampling. The event collection mode is set to `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS` so that the event counters run continuously. Two threads are used in **event_sampling**: one thread schedules the kernels and memcpys that perform the computation, while another thread wakes periodically to sample an event counter. In this sample there is no correlation of the event samples with what is happening on the GPU. To get some coarse correlation, you can use `cuptiDeviceGetTimestamp` to collect the GPU timestamp at the time of the sample and also at other interesting points in your application.

Interpreting Event Values

The tables below describe the events available for each device. Each event has a type that indicates how the activity or action associated with that event is collected. The event types are *SM*, *TPC*, and *FB*.

SM Event Type

The SM event type indicates that the event is collected for an action or activity that occurs on one or more of the device's *streaming multiprocessors* (SMs). A streaming multiprocessor creates, manages, schedules, and executes threads in groups of 32 threads called warps.

The SM event values typically represent activity or action of thread warps, and not the activity or action of individual threads. Details of how each event is incremented are given in the event tables below.

Two factors will impact the accuracy of the values collected for SM type events. First, due to variations in system state, event values can vary across different, identical, runs of the same application. Second, for devices with compute capability less than 2.0, SM events are counted only for one SM. For devices with compute capability greater than 2.0, SM events from *domain_d* are counted for all SMs but for SM events from *domain_a* are counted for multiple but not all, SMs. To get the most consistent results in spite of these factors, it is best to have the number of blocks for each kernel launched to be a multiple of the total number of SMs on a device. In other words, the grid configuration should be chosen such that the number of blocks launched on each SM is the same and also the amount of work of interest per block is the same.

TPC Event Type

The TPC event type indicates that the event is collected for an action or activity that occurs on the SMs within one of the device's *Texture Processing Cluster* (TPC). The number of SMs per TPC varies per device.

Several of the TPC type events measure *coherent* and *incoherent* memory transactions. A coherent (coalesced) access is said to occur when the memory required for a global load or global store instruction can be accessed with a single memory transaction of 32, 64, or 128 bytes. If the memory cannot be accessed with a single memory transaction the access is incoherent. For an incoherent (non-coalesced) access multiple memory transactions are issued, significantly reducing performance. The requirements for coherent access vary based on compute capability. Refer to the CUDA C Programming Guide for details.

FB Event Type

The FB event type indicates that the event is collected for an action or activity that occurs on a DRAM partition.

Event Reference - Compute Capability 1.0 to 1.3

Devices with compute capability less than 2.0 implement two event domains, called *domain_a* and *domain_b*. Table 1 and Table 2 give a description of each event available in these domains. The *Type* column indicates the event type, as described above in the *Interpreting Event Values* section. For the *Capability* columns, a **Y** indicates that the event is available for that compute capability and an **N** indicates that the event is not available.

			Capability			
Event Name	Description	Type	1.0	1.1	1.2	1.3
tex_cache_hit	Number of texture cache hits	SM	Y	Y	Y	Y
tex_cache_miss	Number of texture cache misses	SM	Y	Y	Y	Y

Table 1: Capability 1.x Events For **domain_a**

			Capability			
Event Name	Description	Type	1.0	1.1	1.2	1.3
branch	Number of branches taken by threads executing a kernel. This event is incremented by one if at least one thread in a warp takes the branch. Note that barrier instructions (__-syncThreads()) also get counted as branches	SM	Y	Y	Y	Y

			Capability			
Event Name	Description	Type	1.0	1.1	1.2	1.3
divergent_branch	Number of divergent branches within a warp. This event is incremented by one if at least one thread in a warp diverges (that is, follows a different execution path) via a data dependent conditional branch. The event is incremented by one at each point of divergence in a warp	SM	Y	Y	Y	Y
instructions	Number of instructions executed	SM	Y	Y	Y	Y
warp_serialize	If two addresses of a memory request fall in the same memory bank, there is a bank conflict and the access has to be serialized. This event gives the number of thread warps that serialize on address conflicts to either shared or constant memory	SM	Y	Y	Y	Y
gld_incoherent	Number of non-coalesced global memory loads	TPC	Y	Y	N	N
gld_coherent	Number of coalesced global memory loads	TPC	Y	Y	N	N
gld_32b	Number of 32 byte global memory load transactions; incremented by 1 for each 32 byte transaction	TPC	N	N	Y	Y
gld_64b	Number of 64 byte global memory load transactions; incremented by 1 for each 64 byte transaction	TPC	N	N	Y	Y
gld_128b	Number of 128 byte global memory load transactions; incremented by 1 for each 128 byte transaction	TPC	N	N	Y	Y
gst_incoherent	Number of non-coalesced global memory stores	TPC	Y	Y	N	N
gst_coherent	Number of coalesced global memory stores	TPC	Y	Y	N	N
gst_32b	Number of 32 byte global memory store transactions; incremented by 2 for each 32 byte transaction	TPC	N	N	Y	Y
gst_64b	Number of 64 byte global memory store transactions; incremented by 4 for each 64 byte transaction	TPC	N	N	Y	Y

Event Name	Description	Type	Capability			
			1.0	1.1	1.2	1.3
gst_128b	Number of 128 byte global memory store transactions; incremented by 8 for each 128 byte transaction	TPC	N	N	Y	Y
local_load	Number of local memory load transactions. Each local load request will generate one transaction irrespective of the size of the transaction	TPC	Y	Y	Y	Y
local_store	Number of local memory store transactions; incremented by 2 for each 32-byte transaction, by 4 for each 64-byte transaction and by 8 for each 128-byte transaction	TPC	Y	Y	Y	Y
cta_launched	Number of threads blocks launched on a TPC	TPC	Y	Y	Y	Y
sm_cta_launched	Number of threads blocks launched on an SM	SM	Y	Y	Y	Y
prof_trigger_XX	There are 8 such triggers (00-07) that user can profile. Those are generic and can be inserted in any place of the code to collect the related information	SM	Y	Y	Y	Y

Table 2: Capability 1.x Events For **domain_b**

Event Reference - Compute Capability 2.x

Devices with compute capability 2.x implement four event domains, called *domain_a*, *domain_b*, *domain_c* and *domain_d*. Table 3, Table 4, Table 5 and Table 6 give a description of each event available in these domains. The *Type* column indicates the event type, as described above in the *Interpreting Event Values* section. For the *Capability* columns, a **Y** indicates that the event is available for that compute capability and an **N** indicates that the event is not available.

Event Name	Description	Type	Capability	
			2.0	2.1
sm_cta_launched	Number of thread blocks launched	SM	Y	Y
l1_local_load_hit	Number of local load hits in L1 cache. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y

Event Name	Description	Type	Capability	
			2.0	2.1
l1_local_load_miss	Number of local load misses in L1 cache This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
l1_local_store_hit	Number of local store hits in L1 cache. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
l1_local_store_miss	Number of local store misses in L1 cache. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
l1_global_load_hit	Number of global load hits in L1 cache. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
l1_global_load_miss	Number of global load misses in L1 cache. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
uncached_global_load_transaction	Number of uncached global load transactions. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
global_store_transaction	Number of global store transactions. This increments by 1, 2, or 4 for 32, 64 and 128 bit accesses respectively	SM	Y	Y
l1_shared_bank_conflict	Number of shared bank conflicts caused due to addresses for two or more shared memory requests fall in the same memory bank	SM	Y	Y
tex0_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	Y	Y
tex0_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	Y	Y
tex1_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	N	Y
tex1_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	N	Y

Table 3: Capability 2.x Events For **domain_a**

			Capability	
Event Name	Description	Type	2.0	2.1
l2_subp0_write_sector_misses	Number of write misses in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_write_sector_misses	Number of write misses in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_read_sector_misses	Number of read misses in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_read_sector_misses	Number of read misses in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_write_sector_queries	Number of write requests from L1 to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_write_sector_queries	Number of write requests from L1 to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_read_sector_queries	Number of read requests from L1 to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_read_sector_queries	Number of read requests from L1 to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_read_hit_sectors	Number of read requests from L1 that hit in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_read_hit_sectors	Number of read requests from L1 that hit in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_read_tex_sector_queries	Number of read requests from TEX to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp1_read_tex_sector_queries	Number of read requests from TEX to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
l2_subp0_read_tex_hit_sectors	Number of read requests from L1 that hit in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y

			Capability	
Event Name	Description	Type	2.0	2.1
l2_subp1_read_tex_hit_sectors	Number of read requests from L1 that hit in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y	Y
fb_subp0_read_sectors	Number of DRAM read requests to sub partition 0, increments by 1 for 32 byte access	FB	Y	Y
fb_subp1_read_sectors	Number of DRAM read requests to sub partition 1, increments by 1 for 32 byte access	FB	Y	Y
fb_subp0_write_sectors	Number of DRAM write requests to sub partition 0, increments by 1 for 32 byte access	FB	Y	Y
fb_subp1_write_sectors	Number of DRAM write requests to sub partition 1, increments by 1 for 32 byte access	FB	Y	Y
fb0_subp0_read_sectors	Number of DRAM read requests to sub partition 0 of DRAM unit 0, increments by 1 for 32 byte access	FB	N	Y**
fb0_subp1_read_sectors	Number of DRAM read requests to sub partition 1 of DRAM unit 0, increments by 1 for 32 byte access	FB	N	Y**
fb0_subp0_write_sectors	Number of DRAM write requests to sub partition 0 of DRAM unit 0, increments by 1 for 32 byte access	FB	N	Y**
fb0_subp1_write_sectors	Number of DRAM write requests to sub partition 1 of DRAM unit 0, increments by 1 for 32 byte access	FB	N	Y**
fb1_subp0_read_sectors	Number of DRAM read requests to sub partition 0 of DRAM unit 1, increments by 1 for 32 byte access	FB	N	Y**
fb1_subp1_read_sectors	Number of DRAM read requests to sub partition 1 of DRAM unit 1, increments by 1 for 32 byte access	FB	N	Y**
fb1_subp0_write_sectors	Number of DRAM write requests to sub partition 0 of DRAM unit 1, increments by 1 for 32 byte access	FB	N	Y**
fb1_subp1_write_sectors	Number of DRAM write requests to sub partition 1 of DRAM unit 1, increments by 1 for 32 byte access	FB	N	Y**

Table 4: Capability 2.x Events For **domain_b**

Notes:

- Y**: Devices will have either fb_** counters or fb0_** and fb1_** counters. Total DRAM reads and writes are calculated by adding values for all subpartitions.
- fb* and l2_*_misses events often give a large value when a display is connected to the device. To get accurate values do not connect a display to the device collecting event counts.
- l2_*_queries event values can be greater than l2_*_misses event values because l2_*_queries counts only the requests from L1 to L2 (does not include, for example, texture requests) while l2_*_misses counts all misses
- Initializing device memory on the host fetches data from DRAM to L2, which can modify the fb*_read_sectors event values for a kernel

			Capability	
Event Name	Description	Type	2.0	2.1
gld_inst_8bit	Total number of 8-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gld_inst_16bit	Total number of 16-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gld_inst_32bit	Total number of 32-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gld_inst_64bit	Total number of 64-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gld_inst_128bit	Total number of 128-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gst_inst_8bit	Total number of 8-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gst_inst_16bit	Total number of 16-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gst_inst_32bit	Total number of 32-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y	Y
gst_inst_64bit	Total number of 64-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y	Y

			Capability	
Event Name	Description	Type	2.0	2.1
gst_inst_128bit	Total number of 128-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y	Y

Table 5: Capability 2.x Events For **domain_c**

			Capability	
Event Name	Description	Type	2.0	2.1
branch	Number of branches taken by threads executing a kernel. This counter will be incremented by one if at least one thread in a warp takes the branch	SM	Y	Y
divergent_branch	Number of divergent branches within a warp. This counter will be incremented by one if at least one thread in a warp diverges (that is, follows a different execution path) via a data dependent conditional branch	SM	Y	Y
warps_launched	Number of warps launched	SM	Y	Y
threads_launched	Number of threads launched	SM	Y	Y
active_warps	Accumulated number of active warps per cycle. For every cycle it increments by the number of active warps in the cycle which can be in the range 0 to 48	SM	Y	Y
active_cycles	Number of cycles a multiprocessor has at least one active warp	SM	Y	Y
local_load	Number of local load instructions per warp	SM	Y	Y
local_store	Number of local store instructions per warp	SM	Y	Y
gld_request	Number of global load instructions per warp	SM	Y	Y
gst_request	Number of global store instructions per warp	SM	Y	Y
shared_load	Number of shared load instructions per warp	SM	Y	Y
shared_store	Number of shared store instructions per warp	SM	Y	Y

Event Name	Description	Type	Capability	
			2.0	2.1
prof_trigger_XX	There are 8 such triggers (00-07) that user can profile. The triggers are generic and can be inserted in any place of the code to collect the related information	SM	Y	Y
inst_issued	Number of instructions issued including replays	SM	Y	N
inst_issued1_0	Number of times instruction group 0 issued one instruction	SM	N	Y*
inst_issued2_0	Number of times instruction group 0 issued two instructions	SM	N	Y*
inst_issued1_1	Number of times instruction group 1 issued one instruction	SM	N	Y*
inst_issued2_1	Number of times instruction group 1 issued two instructions	SM	N	Y*
inst_executed	Number of instructions executed, not including replays	SM	Y	Y
thread_inst_-executed_0	Number of instructions executed by all threads, not including replays, in pipeline 0. For each instruction executed increments by the number of threads in the warp	SM	Y	Y
thread_inst_-executed_1	Number of instructions executed by all threads, not including replays, in pipeline 1. For each instruction executed increments by the number of threads in the warp	SM	Y	Y

Table 6: Capability 2.x Events For **domain_d**

Notes:

- Y*: Total instructions issued for compute capability 2.1 can be calculated as: inst_issued1_0 + (inst_issued2_0 * 2) + inst_issued1_1 + (inst_issued2_1 * 2)

Event Reference - Compute Capability 3.x

Devices with compute capability 3.x implement four event domains, called *domain_a*, *domain_b*, *domain_c* and *domain_d*. Table 7, Table 8, Table 9 and Table 10 give a description of each event available in these domains. The *Type* column indicates the event type, as described above in the *Interpreting Event Values* section. For the *Capability* columns, a **Y** indicates that the event is available for that compute capability and an **N** indicates that the event is not available.

			Capability
Event Name	Description	Type	3.0
tex0_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	Y
tex0_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	Y
tex1_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	Y
tex1_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	Y
tex2_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	Y
tex2_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	Y
tex3_cache_sector_queries	Number of texture cache requests. This increments by 1 for each 32-byte access	SM	Y
tex3_cache_sector_misses	Number of texture cache misses. This increments by 1 for each 32-byte access	SM	Y

Table 7: Capability 3.x Events For **domain_a**

			Capability
Event Name	Description	Type	3.0
l2_subp0_write_sector_misses	Number of write misses in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_write_sector_misses	Number of write misses in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_write_sector_misses	Number of write misses in slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_write_sector_misses	Number of write misses in slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y

			Capability
Event Name	Description	Type	3.0
l2_subp0_read_sector_misses	Number of read misses in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_read_sector_misses	Number of read misses in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_read_sector_misses	Number of read misses in slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_read_sector_misses	Number of read misses in slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp0_write_l1_sector_queries	Number of write requests from L1 to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_write_l1_sector_queries	Number of write requests from L1 to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_write_l1_sector_queries	Number of write requests from L1 to slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_write_l1_sector_queries	Number of write requests from L1 to slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp0_read_l1_sector_queries	Number of read requests from L1 to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_read_l1_sector_queries	Number of read requests from L1 to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_read_l1_sector_queries	Number of read requests from L1 to slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_read_l1_sector_queries	Number of read requests from L1 to slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp0_read_l1_hit_sectors	Number of read requests from L1 that hit in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y

			Capability
Event Name	Description	Type	3.0
l2_subp1_read_l1_-hit_sectors	Number of read requests from L1 that hit in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_read_l1_-hit_sectors	Number of read requests from L1 that hit in slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_read_l1_-hit_sectors	Number of read requests from L1 that hit in slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp0_read_-tex_sector_queries	Number of read requests from TEX to slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_read_-tex_sector_queries	Number of read requests from TEX to slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_read_-tex_sector_queries	Number of read requests from TEX to slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_read_-tex_sector_queries	Number of read requests from TEX to slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp0_read_-tex_hit_sectors	Number of read requests from L1 that hit in slice 0 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp1_read_-tex_hit_sectors	Number of read requests from L1 that hit in slice 1 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp2_read_-tex_hit_sectors	Number of read requests from L1 that hit in slice 2 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
l2_subp3_read_-tex_hit_sectors	Number of read requests from L1 that hit in slice 3 of L2 cache. This increments by 1 for each 32-byte access	FB	Y
fb_subp0_read_-sectors	Number of DRAM read requests to sub partition 0, increments by 1 for 32 byte access	FB	Y
fb_subp1_read_-sectors	Number of DRAM read requests to sub partition 1, increments by 1 for 32 byte access	FB	Y

Event Name	Description	Type	Capability
			3.0
fb_subp0_write_-sectors	Number of DRAM write requests to sub partition 0, increments by 1 for 32 byte access	FB	Y
fb_subp1_write_-sectors	Number of DRAM write requests to sub partition 1, increments by 1 for 32 byte access	FB	Y

Table 8: Capability 3.x Events For **domain_b**

Notes:

- fb* and l2_*_misses events often give a large value when a display is connected to the device. To get accurate values do not connect a display to the device collecting event counts.
- l2_*_queries event values can be greater than l2_*_misses event values because l2_*_queries counts only the requests from L1 to L2 (does not include, for example, texture requests) while l2_*_misses counts all misses
- Initializing device memory on the host fetches data from DRAM to L2, which can modify the fb*_read_sectors event values for a kernel

Event Name	Description	Type	Capability
			3.0
gld_inst_8bit	Total number of 8-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y
gld_inst_16bit	Total number of 16-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y
gld_inst_32bit	Total number of 32-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y
gld_inst_64bit	Total number of 64-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y
gld_inst_128bit	Total number of 128-bit global load instructions that are executed by all the threads across all thread blocks	SM	Y
gst_inst_8bit	Total number of 8-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y

			Capability
Event Name	Description	Type	3.0
gst_inst_16bit	Total number of 16-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y
gst_inst_32bit	Total number of 32-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y
gst_inst_64bit	Total number of 64-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y
gst_inst_128bit	Total number of 128-bit global store instructions that are executed by all the threads across all thread blocks	SM	Y

Table 9: Capability 3.x Events For **domain_c**

			Capability
Event Name	Description	Type	3.0
prof_trigger_XX	There are 8 such triggers (00-07) that user can profile. The triggers are generic and can be inserted in any place of the code to collect the related information	SM	Y
warps_launched	Number of warps launched	SM	Y
threads_launched	Number of threads launched	SM	Y
inst_issued1	Number of single instruction issues, including replays	SM	Y*
inst_issued2	Number of dual instruction issues, including replays	SM	Y*
inst_executed	Number of instructions executed, not including replays	SM	Y
inst_executed_lsu_size_128	Number of 128-bit global load and store instructions executed	SM	Y
inst_executed_lsu_size_64	Number of 64-bit global load and store instructions executed	SM	Y
inst_executed_lsu_size_32	Number of 32-bit global load and store instructions executed	SM	Y
inst_executed_lsu_sub_size_32	Number of 8-bit and 16-bit global load and store instructions executed	SM	Y

			Capability
Event Name	Description	Type	3.0
not_predicated_off_-thread_inst_executed	Number of instructions executed by all threads, not including predicated instructions. For each instruction executed increments by the number of threads in the warp	SM	Y
warp_cant_issue_-barrier	Number of active warps that did not issue due to barrier	SM	Y
local_load	Number of local load instructions per warp	SM	Y
local_store	Number of local store instructions per warp	SM	Y
gld_request	Number of global load instructions per warp	SM	Y
gst_request	Number of global store instructions per warp	SM	Y
shared_load	Number of shared load instructions per warp	SM	Y
shared_store	Number of shared store instructions per warp	SM	Y
l1_local_load_transactions	Number of local load transactions per warp	SM	Y
l1_local_store_transactions	Number of local store transactions per warp	SM	Y
l1_shared_load_-transactions	Number of shared load transactions per warp	SM	Y
l1_shared_store_-transactions	Number of shared store transactions per warp	SM	Y
l1_global_load_-transactions	Number of global load transactions per warp	SM	Y
l1_global_store_-transactions	Number of global store transactions per warp	SM	Y
l1_local_load_hit	Number of cache lines that hit in L1 cache for local memory load accesses. In case of perfect coalescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y
l1_local_load_miss	Number of cache lines that miss in L1 cache for local memory load accesses. In case of perfect clescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y

Event Name	Description	Type	Capability
			3.0
l1_local_store_hit	Number of cache lines that hit in L1 cache for local memory store accesses. In case of perfect coalescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y
l1_local_store_miss	Number of cache lines that miss in L1 cache for local memory store accesses. In case of perfect coalescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y
l1_global_load_hit	Number of cache lines that hit in L1 cache for global memory load accesses. In case of perfect coalescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y
l1_global_load_miss	Number of cache lines that miss in L1 cache for global memory load accesses. In case of perfect clescing this increments by 1,2, and 4 for 32, 64 and 128 bit accesses by a warp respectively.	SM	Y
uncached_global_load_transaction	Number of uncached global load transactions. Increments by 1 per transaction. Transaction can be 32/64/128B	SM	Y
global_store_transaction	Number of global store transactions. Increments by 1 per transaction. Transaction can be 32/64/128B	SM	Y
local_ld_mem_divergence_replays	Number of replays due to local load divergence	SM	Y
local_st_mem_divergence_replays	Number of replays due to local store divergence	SM	Y
global_ld_mem_divergence_replays	Number of replays due to global load divergence	SM	Y
global_st_mem_divergence_replays	Number of replays due to global store divergence	SM	Y
shared_load_bank_conflict	Number of shared memory bank conflicts due to shared memory stores	SM	Y
shared_store_bank_conflict	Number of shared memory bank conflicts due to shared memory loads	SM	Y

Event Name	Description	Type	Capability
			3.0
branch	Number of branches taken by threads executing a kernel. This counter will be incremented by one if at least one thread in a warp takes the branch	SM	Y
divergent_branch	Number of divergent branches within a warp. This counter will be incremented by one if at least one thread in a warp diverges (that is, follows a different execution path) via a data dependent conditional branch	SM	Y
active_warps	Accumulated number of active warps per cycle. For every cycle it increments by the number of active warps in the cycle which can be in the range 0 to 48	SM	Y
active_cycles	Number of cycles a multiprocessor has at least one active warp	SM	Y
sm_cta_launched	Number of thread blocks launched	SM	Y

Table 10: Capability 3.x Events For **domain_d**

Notes:

- Y*: Total instructions issued for compute capability 3.x can be calculated as: inst_issued1 + (inst_issued2 * 2)

CUPTI Metric API

The CUPTI Metric API allows you to collect application metrics calculated from one or more event values. The following terminology is used by the metric API.

Metric: An characteristic of an application that is calculated from one or more event values.

Metric ID: Each metric is assigned a unique identifier. A named metric will represent the same characteristic on all device types. But the named metric may have different IDs on different device families. Use `cuptiMetricGetIdFromName` to get the ID for a named metric on a particular device.

Metric Category: Each metric is placed in one of the categories defined by `CUpti_MetricCategory`. The category indicates the general type of the characteristic measured by the metric.

Metric Value: Each metric has a value that represents one of the kinds defined by `CUpti_MetricValueKind`. For each value kind, there is a corresponding member of the `CUpti_MetricValue` union that is used to hold the value.

The tables included in this section list the metrics available for each device, as determined by the device's compute capability. You can also determine the metrics available on a device using the `cuptiDeviceEnumMetrics` function. The **cupti_query** sample described on page 32 shows how to use this function. You can also enumerate all the CUPTI metrics available on any device using the `cuptiEnumMetrics` function.

Configuring and calculating metric values requires the following steps. First, determine the name of the metric that you want to collect, and then use the `cuptiMetricGetIdFromName` to get the metric ID. Use `cuptiMetricEnumEvents` to get the events required to calculate the metric and follow instructions in the CUPTI Event API section to create the event groups for those events. Alternatively, you can use the `cuptiMetricCreateEventGroupSets` function to automatically create the event group(s) required for metric's events.

Collect event counts as described in the CUPTI Event API section, and then use `cuptiMetricGetValue` to calculate the metric value from the collected event values. The **callback_metric** sample described on page 32 shows how to use these functions to calculate event values. Note that, as shown in the example, you should collect event counts from all domain instances and normalize the counts to get the most accurate metric values. It is necessary to normalize the event counts because the number of event counter instances varies by device and by the event being counted.

For example, a device might have 8 multiprocessors but only have event counters for 4 of the multiprocessors, and might have 3 memory units and only have events counters for one memory unit. When calculating a metric that requires a multiprocessor event and a memory unit event, the 4 multiprocessor counters should be summed and multiplied by 2 to normalize the event count across the entire device. Similarly, the one memory unit counter should be multiplied by 3 to normalize the event count across the entire device. The normalized values can then be passed to `cuptiMetricGetValue` to calculate the metric value.

As described, the normalization assumes the kernel executes a sufficient number of blocks to completely load the device. If the kernel has only a small number of blocks, normalizing across the entire device may skew the result.

Metric Reference - Compute Capability 1.x

Devices with compute capability less than 2.0 implement the metrics shown in Table 11.

Metric Name	Description	Formula
branch_efficiency	Ratio of non-divergent branches to total branches	100*(branch-divergent_branch)/branch
gld_efficiency	Ratio of requested global memory load transactions to actual global memory load transactions	For CC 1.2 & 1.3: (gld_request / ((gld_32 + gld_64 + gld_128) / (2 * #SM))) For CC 1.0 & 1.1: gld_coherent / (gld_coherent + gld_incoherent)
gst_efficiency	Ratio of requested global memory store transactions to actual global memory store transactions	For CC 1.2 & 1.3: (gst_request / ((gst_32 + gst_64 + gst_128) / (2 * #SM))) For CC 1.0 & 1.1: gst_coherent / (gst_coherent + gst_incoherent)
gld_requested_-throughput	Requested global memory load throughput	(gld_32 * 32 + gld_64 * 64 + gld_128 * 128) / (gputime)
gst_requested_-throughput	Requested global memory store throughput	(gst_32 * 32 + gst_64 * 64 + gst_128 * 128) / (gputime)

Table 11: Capability 1.x Metrics

Metric Reference - Compute Capability 2.0 and Greater

Devices with compute capability 2.0 or greater implement the metrics shown in Table 12.

Metric Name	Description	Formula
sm_efficiency	The ratio of the time at least one warp is active on a multiprocessor to the total time	100*(active_cycles/#SM) / elapsed_clocks
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	100*(active_warps/active_cycles)/48
ipc	Instructions executed per cycle	(inst_executed/#SM) / elapsed_clocks
branch_efficiency	Ratio of non-divergent branches to total branches	100*(branch-divergent_branch)/branch

Metric Name	Description	Formula
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor	thread_inst_executed/(inst_executed*warp_size)
inst_replay_overhead	Percentage of instruction issues due to memory replays	100*(instructions_issued - instruction_executed)/instruction_issued
shared_replay_overhead	Percentage of instruction issues due to replays for shared memory conflicts	(100*shared_memory_bank_conflicts)/inst_issued
global_cache_replay_overhead	Percentage of instruction issues due to replays for global memory cache misses	100*global_load_miss/ inst_issued
local_replay_overhead	Percentage of instruction issues due to replays for local memory cache misses	100*(local_load_miss+local_store_miss)/ inst_issued
gld_efficiency	Ratio of requested global memory load throughput to actual global memory load throughput	100*gld_requested_throughput/gld_throughput
gst_efficiency	Ratio of requested global memory store throughput to actual global memory store throughput	100*gst_requested_throughput/gst_throughput
gld_throughput	Global memory load throughput	((128*global_load_hit) + (l2_subp0_read_requests + l2_subp1_read_requests) * 32 - (l1_cached_local_ld_misses * 128))/(gputime)
gst_throughput	Global memory store throughput	(l2_subp0_write_requests + l2_subp1_write_requests) * 32 - (l1_cached_local_ld_misses * 128))/(gputime)
gld_requested_throughput	Requested global memory load throughput	(gld_inst_8bit + 2*gld_inst_16bit + 4*gld_inst_32bit + 8*gld_inst_64bit + 16*gld_inst_128bit) / gputime
gst_requested_throughput	Requested global memory store throughput	(gst_inst_8bit + 2*gst_inst_16bit + 4*gst_inst_32bit + 8*gst_inst_64bit + 16*gst_inst_128bit) / gputime
dram_read_throughput	DRAM read throughput	(fb_subp0_read + fb_subp1_read) * 32 / gputime

Metric Name	Description	Formula
dram_write_throughput	DRAM write throughput	(fb_subp0_write + fb_subp1_write) * 32 / gputime
l1_cache_global_hit_rate	Hit rate in L1 cache for global loads	100*l1_cached_global_ld_hits/(l1_cached_global_ld_hits+l1_cached_global_ld_misses)
l1_cache_local_hit_rate	Hit rate in L1 cache for local loads and stores	100*l1_cached_local_ld_hits+l1_cached_local_st_hits/(l1_cached_local_ld_hits+l1_cached_local_ld_misses+l1_cached_local_st_hits+l1_cached_local_st_misses)
tex_cache_hit_rate	Texture cache hit rate	100 * (tex0Queries - tex0Misses)/tex0Queries
tex_cache_throughput	Texture cache throughput	tex_cache_sector_queries * 32 / gputime

Table 12: Capability 2.0 and Greater Metrics

Samples

The CUPTI installation includes several samples that demonstrate the use of the CUPTI APIs.The samples are:

activity_trace: This sample shows how to collect a trace of CPU and GPU activity.

callback_event: This sample shows how to use both the callback and event APIs to record the events that occur during the execution of a simple kernel. The sample shows the required ordering for synchronization, and for event group enabling, disabling and reading.

callback_metric: This sample shows how to use both the callback and metric APIs to record the metric's events during the execution of a simple kernel, and then use those events to calculate the metric value.

callback_timestamp: This sample shows how to use the callback API to record a trace of API start and stop times.

cupti_query: This sample shows how to query CUDA-enabled devices for their event domains, events, and metrics.

event_sampling: This sample shows how to use the event API to sample events using a separate host thread.

CUPTI Reference

CUPTI Version

Defines

- #define CUPTI_API_VERSION 2

  The API version for this implementation of CUPTI.

Functions

- CUptiResult cuptiGetVersion (uint32_t *version)

  Get the CUPTI API version.

Define Documentation

#define CUPTI_API_VERSION 2

The API version for this implementation of CUPTI. This define along with cuptiGetVersion can be used to dynamically detect if the version of CUPTI compiled against matches the version of the loaded CUPTI library.

v1 : CUDAToolsSDK 4.0 v2 : CUDAToolsSDK 4.1

Function Documentation

CUptiResult cuptiGetVersion (uint32_t * version)

Return the API version in `*version`.

Parameters:

version Returns the version

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_INVALID_PARAMETER if `version` is NULL

See also:

CUPTI_API_VERSION

CUPTI Result Codes

Enumerations

- enum CUptiResult {

  CUPTI_SUCCESS = 0,

  CUPTI_ERROR_INVALID_PARAMETER = 1,

  CUPTI_ERROR_INVALID_DEVICE = 2,

  CUPTI_ERROR_INVALID_CONTEXT = 3,

  CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID = 4,

  CUPTI_ERROR_INVALID_EVENT_ID = 5,

  CUPTI_ERROR_INVALID_EVENT_NAME = 6,

  CUPTI_ERROR_INVALID_OPERATION = 7,

  CUPTI_ERROR_OUT_OF_MEMORY = 8,

  CUPTI_ERROR_HARDWARE = 9,

  CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT = 10,

  CUPTI_ERROR_API_NOT_IMPLEMENTED = 11,

  CUPTI_ERROR_MAX_LIMIT_REACHED = 12,

  CUPTI_ERROR_NOT_READY = 13,

  CUPTI_ERROR_NOT_COMPATIBLE = 14,

  CUPTI_ERROR_NOT_INITIALIZED = 15,

  CUPTI_ERROR_INVALID_METRIC_ID = 16,

  CUPTI_ERROR_INVALID_METRIC_NAME = 17,

  CUPTI_ERROR_QUEUE_EMPTY = 18,

  CUPTI_ERROR_INVALID_HANDLE = 19,

  CUPTI_ERROR_INVALID_STREAM = 20,

  CUPTI_ERROR_INVALID_KIND = 21,

  CUPTI_ERROR_INVALID_EVENT_VALUE = 22,

  CUPTI_ERROR_DISABLED = 100,

  CUPTI_ERROR_UNKNOWN = 999 }

Functions

- ▶ CUptiResult cuptiGetResultString (CUptiResult result, const char **str)

  Get the descriptive string for a CUptiResult.

Enumeration Type Documentation

enum **CUptiResult**

Result codes.

Enumerator:

CUPTI_SUCCESS No error.

CUPTI_ERROR_INVALID_PARAMETER One or more of the parameters is invalid.

CUPTI_ERROR_INVALID_DEVICE The device does not correspond to a valid CUDA device.

CUPTI_ERROR_INVALID_CONTEXT The context is NULL or not valid.

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID The event domain id is invalid.

CUPTI_ERROR_INVALID_EVENT_ID The event id is invalid.

CUPTI_ERROR_INVALID_EVENT_NAME The event name is invalid.

CUPTI_ERROR_INVALID_OPERATION The current operation cannot be performed due to dependency on other factors.

CUPTI_ERROR_OUT_OF_MEMORY Unable to allocate enough memory to perform the requested operation.

CUPTI_ERROR_HARDWARE The performance monitoring hardware could not be reserved or some other hardware error occurred.

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT The output buffer size is not sufficient to return all requested data.

CUPTI_ERROR_API_NOT_IMPLEMENTED API is not implemented.

CUPTI_ERROR_MAX_LIMIT_REACHED The maximum limit is reached.

CUPTI_ERROR_NOT_READY The object is not yet ready to perform the requested operation.

CUPTI_ERROR_NOT_COMPATIBLE The current operation is not compatible with the current state of the object

CUPTI_ERROR_NOT_INITIALIZED CUPTI is unable to initialize its connection to the CUDA driver.

CUPTI_ERROR_INVALID_METRIC_ID The metric id is invalid.

CUPTI_ERROR_INVALID_METRIC_NAME The metric name is invalid.

CUPTI_ERROR_QUEUE_EMPTY The queue is empty.

CUPTI_ERROR_INVALID_HANDLE Invalid handle (internal?).

CUPTI_ERROR_INVALID_STREAM Invalid stream.

CUPTI_ERROR_INVALID_KIND Invalid kind.

CUPTI_ERROR_INVALID_EVENT_VALUE Invalid event value.

CUPTI_ERROR_DISABLED CUPTI profiling is not compatible with current profiling mode

CUPTI_ERROR_UNKNOWN An unknown internal error has occurred.

Function Documentation

CUptiResult cuptiGetResultString (**CUptiResult** result, const char ** str)

Return the descriptive string for a CUptiResult in ∗`str`.

Note:

Thread-safety: this function is thread safe.

Parameters:

result The result to get the string for

str Returns the string

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_INVALID_PARAMETER if `str` is NULL or `result` is not a valid CUptiResult

CUPTI Activity API

Data Structures

- struct CUpti_Activity

  The base activity record.

- struct CUpti_ActivityAPI

  The activity record for a driver or runtime API invocation.

- struct CUpti_ActivityContext

  The activity record for a context.

- struct CUpti_ActivityDevice

  The activity record for a device.

- struct CUpti_ActivityEvent

  The activity record for a CUPTI event.

- struct CUpti_ActivityKernel

  The activity record for kernel.

- struct CUpti_ActivityMemcpy

  The activity record for memory copies.

- struct CUpti_ActivityMemset

  The activity record for memset.

- struct CUpti_ActivityMetric

  The activity record for a CUPTI metric.

Enumerations

- enum CUpti_ActivityComputeApiKind {
  CUPTI_ACTIVITY_COMPUTE_API_UNKNOWN = 0,
  CUPTI_ACTIVITY_COMPUTE_API_CUDA = 1,

CUPTI_ACTIVITY_COMPUTE_API_OPENCL = 2 }

The kind of a compute API, indicating if the context was created for CUDA api or OpenCL APIs.

▶ enum CUpti_ActivityFlag {

CUPTI_ACTIVITY_FLAG_NONE = 0,

CUPTI_ACTIVITY_FLAG_DEVICE_CONCURRENT_KERNELS = 1 << 0,

CUPTI_ACTIVITY_FLAG_MEMCPY_ASYNC = 1 << 0 }

Flags associated with activity records.

▶ enum CUpti_ActivityKind {

CUPTI_ACTIVITY_KIND_INVALID = 0,

CUPTI_ACTIVITY_KIND_MEMCPY = 1,

CUPTI_ACTIVITY_KIND_MEMSET = 2,

CUPTI_ACTIVITY_KIND_KERNEL = 3,

CUPTI_ACTIVITY_KIND_DRIVER = 4,

CUPTI_ACTIVITY_KIND_RUNTIME = 5,

CUPTI_ACTIVITY_KIND_EVENT = 6,

CUPTI_ACTIVITY_KIND_METRIC = 7,

CUPTI_ACTIVITY_KIND_DEVICE = 8,

CUPTI_ACTIVITY_KIND_CONTEXT = 9 }

The kinds of activity records.

▶ enum CUpti_ActivityMemcpyKind {

CUPTI_ACTIVITY_MEMCPY_KIND_UNKNOWN = 0,

CUPTI_ACTIVITY_MEMCPY_KIND_HTOD = 1,

CUPTI_ACTIVITY_MEMCPY_KIND_DTOH = 2,

CUPTI_ACTIVITY_MEMCPY_KIND_HTOA = 3,

CUPTI_ACTIVITY_MEMCPY_KIND_ATOH = 4,

CUPTI_ACTIVITY_MEMCPY_KIND_ATOA = 5,

CUPTI_ACTIVITY_MEMCPY_KIND_ATOD = 6,

CUPTI_ACTIVITY_MEMCPY_KIND_DTOA = 7,

CUPTI_ACTIVITY_MEMCPY_KIND_DTOD = 8,

CUPTI_ACTIVITY_MEMCPY_KIND_HTOH = 9 }

The kind of a memory copy, indicating the source and destination targets of the copy.

- enum CUpti_ActivityMemoryKind {
  CUPTI_ACTIVITY_MEMORY_KIND_UNKNOWN = 0,
  CUPTI_ACTIVITY_MEMORY_KIND_PAGEABLE = 1,
  CUPTI_ACTIVITY_MEMORY_KIND_PINNED = 2,
  CUPTI_ACTIVITY_MEMORY_KIND_DEVICE = 3,
  CUPTI_ACTIVITY_MEMORY_KIND_ARRAY = 4 }

  The kinds of memory accessed by a memory copy.

Functions

- CUptiResult cuptiActivityDequeueBuffer (CUcontext context, uint32_t streamId, uint8_t **buffer, size_t *validBufferSizeBytes)

  Dequeue a buffer containing activity records.

- CUptiResult cuptiActivityDisable (CUpti_ActivityKind kind)

  Disable collection of a specific kind of activity record.

- CUptiResult cuptiActivityEnable (CUpti_ActivityKind kind)

  Enable collection of a specific kind of activity record.

- CUptiResult cuptiActivityEnqueueBuffer (CUcontext context, uint32_t streamId, uint8_t *buffer, size_t bufferSizeBytes)

  Queue a buffer for activity record collection.

- CUptiResult cuptiActivityGetNextRecord (uint8_t *buffer, size_t validBufferSizeBytes, CUpti_Activity **record)

  Iterate over the activity records in a buffer.

- CUptiResult cuptiActivityGetNumDroppedRecords (CUcontext context, uint32_t streamId, size_t *dropped)

  Get the number of activity records that were dropped from a queue because of insufficient buffer space.

- CUptiResult cuptiActivityQueryBuffer (CUcontext context, uint32_t streamId, size_t *validBufferSizeBytes)

Query the status of the buffer at the head of a queue.

- CUptiResult cuptiGetStreamId (CUcontext context, CUstream stream, uint32_t *streamId)

  Get the ID of a stream.

- CUptiResult cuptiGetTimestamp (uint64_t *timestamp)

  Get the CUPTI timestamp.

Enumeration Type Documentation

enum **CUpti_ActivityComputeApiKind**

Enumerator:

CUPTI_ACTIVITY_COMPUTE_API_UNKNOWN The compute API is not known.

CUPTI_ACTIVITY_COMPUTE_API_CUDA The compute APIs are for CUDA.

CUPTI_ACTIVITY_COMPUTE_API_OPENCL The compute APIs are for OpenCL.

enum **CUpti_ActivityFlag**

Activity record flags. Flags can be combined by bitwise OR to associated multiple flags with an activity record. Each flag is specific to a certain activity kind, as noted below.

Enumerator:

CUPTI_ACTIVITY_FLAG_NONE Indicates the activity record has no flags.

CUPTI_ACTIVITY_FLAG_DEVICE_CONCURRENT_KERNELS Indicates the activity represents a device that supports concurrent kernel execution. Valid for CUPTI_ACTIVITY_KIND_DEVICE.

CUPTI_ACTIVITY_FLAG_MEMCPY_ASYNC Indicates the activity represents an asychronous memcpy operation. Valid for CUPTI_ACTIVITY_KIND_MEMCPY.

enum **CUpti_ActivityKind**

Each activity record kind represents information about a GPU or an activity occurring on a CPU or GPU. Each kind is associated with a activity record structure that holds the information associated with the kind.

See also:

CUpti_Activity
CUpti_ActivityAPI
CUpti_ActivityDevice
CUpti_ActivityEvent
CUpti_ActivityKernel
CUpti_ActivityMemcpy
CUpti_ActivityMemset
CUpti_ActivityMetric

Enumerator:

CUPTI_ACTIVITY_KIND_INVALID The activity record is invalid.

CUPTI_ACTIVITY_KIND_MEMCPY A host<->host, host<->device, or device<->device memory copy. The corresponding activity record structure is CUpti_ActivityMemcpy.

CUPTI_ACTIVITY_KIND_MEMSET A memory set executing on the GPU. The corresponding activity record structure is CUpti_ActivityMemset.

CUPTI_ACTIVITY_KIND_KERNEL A kernel executing on the GPU. The corresponding activity record structure is CUpti_ActivityKernel.

CUPTI_ACTIVITY_KIND_DRIVER A CUDA driver API function execution. The corresponding activity record structure is CUpti_ActivityAPI.

CUPTI_ACTIVITY_KIND_RUNTIME A CUDA runtime API function execution. The corresponding activity record structure is CUpti_ActivityAPI.

CUPTI_ACTIVITY_KIND_EVENT An event value. The corresponding activity record structure is CUpti_ActivityEvent.

CUPTI_ACTIVITY_KIND_METRIC A metric value. The corresponding activity record structure is CUpti_ActivityMetric.

CUPTI_ACTIVITY_KIND_DEVICE Information about a device. The corresponding activity record structure is CUpti_ActivityDevice.

CUPTI_ACTIVITY_KIND_CONTEXT Information about a context. The corresponding activity record structure is CUpti_ActivityContext.

enum **CUpti_ActivityMemcpyKind**

Each kind represents the source and destination targets of a memory copy. Targets are host, device, and array.

Enumerator:

CUPTI_ACTIVITY_MEMCPY_KIND_UNKNOWN The memory copy kind is not known.

CUPTI_ACTIVITY_MEMCPY_KIND_HTOD A host to device memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_DTOH A device to host memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_HTOA A host to device array memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOH A device array to host memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOA A device array to device array memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOD A device array to device memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_DTOA A device to device array memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_DTOD A device to device memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_HTOH A host to host memory copy.

enum **CUpti_ActivityMemoryKind**

Each kind represents the type of the source or destination memory accessed by a memory copy.

Enumerator:

CUPTI_ACTIVITY_MEMORY_KIND_UNKNOWN The source or destination memory kind is unknown.

CUPTI_ACTIVITY_MEMORY_KIND_PAGEABLE The source or destination memory is pageable.

CUPTI_ACTIVITY_MEMORY_KIND_PINNED The source or destination memory is pinned.

CUPTI_ACTIVITY_MEMORY_KIND_DEVICE The source or destination memory is on the device.

CUPTI_ACTIVITY_MEMORY_KIND_ARRAY The source or destination memory is an array.

Function Documentation

CUptiResult cuptiActivityDequeueBuffer (CUcontext context, uint32_t streamId, uint8_t ** buffer, size_t * validBufferSizeBytes)

Remove the buffer from the head of the specified queue. See cuptiActivityEnqueueBuffer() for description of queues. Calling this function transfers ownership of the buffer from CUPTI. CUPTI will no add any activity records to the buffer after it is dequeued.

Parameters:

context The context, or NULL to dequeue from the global queue

streamId The stream ID

buffer Returns the dequeued buffer

validBufferSizeBytes Returns the number of bytes in the buffer that contain activity records

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `buffer` or `validBufferSizeBytes` are NULL

CUPTI_ERROR_QUEUE_EMPTY the queue is empty, `buffer` returns NULL and `validBufferSizeBytes` returns 0

CUptiResult cuptiActivityDisable (**CUpti_ActivityKind** kind)

Disable collection of a specific kind of activity record. Multiple kinds can be disabled by calling this function multiple times. By default all activity kinds are disabled for collection.

Parameters:

kind The kind of activity record to stop collecting

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUptiResult cuptiActivityEnable (**CUpti_ActivityKind** kind)

Enable collection of a specific kind of activity record. Multiple kinds can be enabled by calling this function multiple times. By default all activity kinds are disabled for collection.

Parameters:

kind The kind of activity record to collect

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_NOT_COMPATIBLE if the activity kind cannot be enabled

CUptiResult cuptiActivityEnqueueBuffer (CUcontext context, uint32_t streamId, uint8_t * buffer, size_t bufferSizeBytes)

Queue a buffer for activity record collection. Calling this function transfers ownership of the buffer to CUPTI. The buffer should not be accessed or modified until ownership is regained by calling cuptiActivityDequeueBuffer().

There are three types of queues:

Global Queue: The global queue collects all activity records that are not associated with a valid context. All device and API activity records are collected in the global queue. A buffer is enqueued in the global queue by specifying `context` == NULL.

Context Queue: Each context queue collects activity records associated with that context that are not associated with a specific stream or that are associated with the default stream. A buffer is enqueued in a context queue by specifying the context and a `streamId` of 0.

Stream Queue: Each stream queue collects memcpy, memset, and kernel activity records associated with the stream. A buffer is enqueued in a stream queue by specifying a context and a non-zero stream ID.

Multiple buffers can be enqueued on each queue, and buffers can be enqueue on multiple queues.

When a new activity record needs to be recorded, CUPTI searches for a non-empty queue to hold the record in this order: 1) the appropriate stream queue, 2) the appropriate context queue, and 3) the global queue. If the search does not find any queue with a buffer then the activity record is dropped. If the search finds a queue containing a buffer, but that buffer is full, then the activity record is dropped and the dropped record count for the queue is incremented. If the search finds a queue containing a buffer with space available to hold the record, then the record is recorded in the buffer.

At a minimum, one or more buffers must be queued in the global queue at all times to avoid dropping activity records. For correct operation it is also necessary to enqueue at least one buffer in the context queue of each context as it is created. The stream queues are optional and can be used to reduce or eliminate application perturbations caused by the need to process or save the activity records returned in the buffers. For example, if a stream queue is used, that queue can be flushed when the stream is synchronized.

Parameters:

context The context, or NULL to enqueue on the global queue

streamId The stream ID

buffer The pointer to user supplied buffer for storing activity records.The buffer must be at least 8 byte aligned, and the size of the buffer must be at least 1024 bytes.

bufferSizeBytes The size of the buffer, in bytes. The size of the buffer must be at least 1024 bytes.

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `buffer` is NULL, does not have alignment of at least 8 bytes, or is not at least 1024 bytes in size

CUptiResult cuptiActivityGetNextRecord (uint8_t * buffer, size_t validBufferSizeBytes, **CUpti_Activity** ** record)

This is a helper function to iterate over the activity records in a buffer. A buffer of activity records is typically obtained by using the cuptiActivityDequeueBuffer() function.

An example of typical usage:

```
CUpti_Activity *record = NULL;
CUptiResult status = CUPTI_SUCCESS;
  do {
     status = cuptiActivityGetNextRecord(buffer, validSize, &record);
     if(status == CUPTI_SUCCESS) {
          // Use record here...
     }
     else if (status == CUPTI_ERROR_MAX_LIMIT_REACHED)
         break;
     else {
         goto Error;
     }
   } while (1);
```

Parameters:

buffer The buffer containing activity records

record Inputs the previous record returned by cuptiActivityGetNextRecord and returns the next activity record from the buffer. If input value if NULL, returns the first activity record in the buffer.

validBufferSizeBytes The number of valid bytes in the buffer.

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_MAX_LIMIT_REACHED if no more records in the buffer

CUPTI_ERROR_INVALID_PARAMETER if `buffer` is NULL.

CUptiResult cuptiActivityGetNumDroppedRecords (CUcontext context, uint32_t streamId, size_t * dropped)

Get the number of records that were dropped from a queue because all the buffers in the queue are full. See cuptiActivityEnqueueBuffer() for description of queues. Calling this function does not transfer ownership of the buffer. The dropped count maintained for the queue is reset to zero when this function is called.

Parameters:

context The context, or NULL to get dropped count from global queue

streamId The stream ID

dropped The number of records that were dropped since the last call to this function.

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `dropped` is NULL

CUptiResult cuptiActivityQueryBuffer (CUcontext context, uint32_t streamId, size_t * validBufferSizeBytes)

Query the status of buffer at the head in the queue. See cuptiActivityEnqueueBuffer() for description of queues. Calling this function does not transfer ownership of the buffer.

Parameters:

context The context, or NULL to query the global queue

streamId The stream ID

validBufferSizeBytes Returns the number of bytes in the buffer that contain activity records

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `buffer` or `validBufferSizeBytes` are NULL

CUPTI_ERROR_MAX_LIMIT_REACHED if buffer is full

CUPTI_ERROR_QUEUE_EMPTY the queue is empty, `validBufferSizeBytes` returns 0

CUptiResult cuptiGetStreamId (CUcontext context, CUstream stream, uint32_t * streamId)

Get the ID of a stream. The stream ID is needed to enqueue and dequeue activity record buffers on a stream queue.

Parameters:

context The context containing the stream

stream The stream

streamId Returns the ID for the stream

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `streamId` is NULL

See also:

cuptiActivityEnqueueBuffer
cuptiActivityDequeueBuffer

CUptiResult cuptiGetTimestamp (uint64_t * timestamp)

Returns a timestamp normalized to correspond with the start and end timestamps reported in the CUPTI activity records. The timestamp is reported in nanoseconds.

Parameters:

timestamp Returns the CUPTI timestamp

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_INVALID_PARAMETER if `timestamp` is NULL

CUpti_Activity Type Reference

The base activity record.

Data Fields

- CUpti_ActivityKind kind

Detailed Description

The activity API uses a CUpti_Activity as a generic representation for any activity. The 'kind' field is used to determine the specific activity kind, and from that the CUpti_Activity object can be cast to the specific activity record type appropriate for that kind.

Note that all activity record types are padded and aligned to ensure that each member of the record is naturally aligned.

See also:

CUpti_ActivityKind

Field Documentation

CUpti_ActivityKind CUpti_Activity::kind

The kind of this activity.

CUpti_ActivityAPI Type Reference

The activity record for a driver or runtime API invocation.

Data Fields

- CUpti_CallbackId cbid
- uint32_t correlationId
- uint64_t end
- CUpti_ActivityKind kind
- uint32_t processId
- uint32_t returnValue
- uint64_t start
- uint32_t threadId

Detailed Description

This activity record represents an invocation of a driver or runtime API (CUPTI_ACTIVITY_KIND_DRIVER and CUPTI_ACTIVITY_KIND_RUNTIME).

Field Documentation

CUpti_CallbackId CUpti_ActivityAPI::cbid

The ID of the driver or runtime function.

uint32_t **CUpti_ActivityAPI::correlationId**

The correlation ID of the driver or runtime CUDA function. Each function invocation is assigned a unique correlation ID that is identical to the correlation ID in the memcpy, memset, or kernel activity record that is associated with this function.

uint64_t **CUpti_ActivityAPI::end**

The end timestamp for the function, in ns.

CUpti_ActivityKind CUpti_ActivityAPI::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_DRIVER or CUPTI_ACTIVITY_KIND_RUNTIME.

uint32_t **CUpti_ActivityAPI::processId**

The ID of the process where the driver or runtime CUDA function is executing.

uint32_t **CUpti_ActivityAPI::returnValue**

The return value for the function. For a CUDA driver function with will be a CUresult value, and for a CUDA runtime function this will be a cudaError_t value.

uint64_t **CUpti_ActivityAPI::start**

The start timestamp for the function, in ns.

uint32_t **CUpti_ActivityAPI::threadId**

The ID of the thread where the driver or runtime CUDA function is executing.

CUpti_ActivityDevice Type Reference

The activity record for a device.

Data Fields

- uint32_t computeCapabilityMajor
- uint32_t computeCapabilityMinor
- uint32_t constantMemorySize
- uint32_t coreClockRate
- uint32_t flags
- uint64_t globalMemoryBandwidth
- uint64_t globalMemorySize
- uint32_t id
- CUpti_ActivityKind kind
- uint32_t l2CacheSize
- uint32_t maxBlockDimX
- uint32_t maxBlockDimY
- uint32_t maxBlockDimZ
- uint32_t maxBlocksPerMultiprocessor
- uint32_t maxGridDimX
- uint32_t maxGridDimY
- uint32_t maxGridDimZ
- uint32_t maxIPC
- uint32_t maxRegistersPerBlock
- uint32_t maxSharedMemoryPerBlock
- uint32_t maxThreadsPerBlock
- uint32_t maxWarpsPerMultiprocessor
- const char ∗ name
- uint32_t numMemcpyEngines
- uint32_t numMultiprocessors
- uint32_t numThreadsPerWarp

Detailed Description

This activity record represents information about a GPU device (CUPTI_ACTIVITY_KIND_DEVICE).

Field Documentation

uint32_t **CUpti_ActivityDevice::computeCapabilityMajor**

Compute capability for the device, major number.

uint32_t **CUpti_ActivityDevice::computeCapabilityMinor**

Compute capability for the device, minor number.

uint32_t **CUpti_ActivityDevice::constantMemorySize**

The amount of constant memory on the device, in bytes.

uint32_t **CUpti_ActivityDevice::coreClockRate**

The core clock rate of the device, in kHz.

uint32_t **CUpti_ActivityDevice::flags**

The flags associated with the device.

See also:

CUpti_ActivityFlag

uint64_t **CUpti_ActivityDevice::globalMemoryBandwidth**

The global memory bandwidth available on the device, in kBytes/sec.

uint64_t **CUpti_ActivityDevice::globalMemorySize**

The amount of global memory on the device, in bytes.

uint32_t **CUpti_ActivityDevice::id**

The device ID.

CUpti_ActivityKind CUpti_ActivityDevice::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_DEVICE.

uint32_t **CUpti_ActivityDevice::l2CacheSize**

The size of the L2 cache on the device, in bytes.

uint32_t **CUpti_ActivityDevice::maxBlockDimX**

Maximum allowed X dimension for a block.

uint32_t **CUpti_ActivityDevice::maxBlockDimY**

Maximum allowed Y dimension for a block.

uint32_t **CUpti_ActivityDevice::maxBlockDimZ**

Maximum allowed Z dimension for a block.

uint32_t **CUpti_ActivityDevice::maxBlocksPerMultiprocessor**

Maximum number of blocks that can be present on a multiprocessor at any given time.

uint32_t **CUpti_ActivityDevice::maxGridDimX**

Maximum allowed X dimension for a grid.

uint32_t **CUpti_ActivityDevice::maxGridDimY**

Maximum allowed Y dimension for a grid.

uint32_t **CUpti_ActivityDevice::maxGridDimZ**

Maximum allowed Z dimension for a grid.

uint32_t **CUpti_ActivityDevice::maxIPC**

The maximum "instructions per cycle" possible on each device multiprocessor.

uint32_t **CUpti_ActivityDevice::maxRegistersPerBlock**

Maximum number of registers that can be allocated to a block.

uint32_t **CUpti_ActivityDevice::maxSharedMemoryPerBlock**

Maximum amount of shared memory that can be assigned to a block, in bytes.

uint32_t **CUpti_ActivityDevice::maxThreadsPerBlock**

Maximum number of threads allowed in a block.

uint32_t **CUpti_ActivityDevice::maxWarpsPerMultiprocessor**

Maximum number of warps that can be present on a multiprocessor at any given time.

const char∗ **CUpti_ActivityDevice::name**

The device name. This name is shared across all activity records representing instances of the device, and so should not be modified.

uint32_t **CUpti_ActivityDevice::numMemcpyEngines**

Number of memory copy engines on the device.

uint32_t **CUpti_ActivityDevice::numMultiprocessors**

Number of multiprocessors on the device.

uint32_t **CUpti_ActivityDevice::numThreadsPerWarp**

The number of threads per warp on the device.

CUpti_ActivityEvent Type Reference

The activity record for a CUPTI event.

Data Fields

- uint32_t correlationId
- CUpti_EventDomainID domain
- CUpti_EventID id
- CUpti_ActivityKind kind
- uint64_t value

Detailed Description

This activity record represents the collection of a CUPTI event value (CUPTI_ACTIVITY_KIND_EVENT). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect event data may choose to use this type to store the collected event data.

Field Documentation

uint32_t **CUpti_ActivityEvent::correlationId**

The correlation ID of the event. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the event was gathered.

CUpti_EventDomainID CUpti_ActivityEvent::domain

The event domain ID.

CUpti_EventID CUpti_ActivityEvent::id

The event ID.

CUpti_ActivityKind CUpti_ActivityEvent::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_EVENT.

uint64_t **CUpti_ActivityEvent::value**

The event value.

CUpti_ActivityKernel Type Reference

The activity record for kernel.

Data Fields

- int32_t blockX
- int32_t blockY
- int32_t blockZ
- uint8_t cacheConfigExecuted
- uint8_t cacheConfigRequested
- uint32_t contextId
- uint32_t correlationId
- uint32_t deviceId
- int32_t dynamicSharedMemory
- uint64_t end
- int32_t gridX
- int32_t gridY
- int32_t gridZ
- CUpti_ActivityKind kind
- uint32_t localMemoryPerThread
- uint32_t localMemoryTotal
- const char * name
- uint32_t pad
- uint16_t registersPerThread
- void * reserved0
- uint32_t runtimeCorrelationId
- uint64_t start
- int32_t staticSharedMemory
- uint32_t streamId

Detailed Description

This activity record represents a kernel execution (CUPTI_ACTIVITY_KIND_KERNEL).

Field Documentation

int32_t **CUpti_ActivityKernel::blockX**

The X-dimension block size for the kernel.

int32_t **CUpti_ActivityKernel::blockY**

The Y-dimension block size for the kernel.

int32_t **CUpti_ActivityKernel::blockZ**

The Z-dimension grid size for the kernel.

uint8_t **CUpti_ActivityKernel::cacheConfigExecuted**

The cache configuration used for the kernel. The value is one of the CUfunc_cache enumeration values from cuda.h.

uint8_t **CUpti_ActivityKernel::cacheConfigRequested**

The cache configuration requested by the kernel. The value is one of the CUfunc_cache enumeration values from cuda.h.

uint32_t **CUpti_ActivityKernel::contextId**

The ID of the context where the kernel is executing.

uint32_t **CUpti_ActivityKernel::correlationId**

The correlation ID of the kernel. Each kernel execution is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the kernel.

uint32_t **CUpti_ActivityKernel::deviceId**

The ID of the device where the kernel is executing.

int32_t **CUpti_ActivityKernel::dynamicSharedMemory**

The dynamic shared memory reserved for the kernel, in bytes.

uint64_t **CUpti_ActivityKernel::end**

The end timestamp for the kernel execution, in ns.

int32_t **CUpti_ActivityKernel::gridX**

The X-dimension grid size for the kernel.

int32_t **CUpti_ActivityKernel::gridY**

The Y-dimension grid size for the kernel.

int32_t **CUpti_ActivityKernel::gridZ**

The Z-dimension grid size for the kernel.

CUpti_ActivityKind CUpti_ActivityKernel::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_KERNEL.

uint32_t **CUpti_ActivityKernel::localMemoryPerThread**

The amount of local memory reserved for each thread, in bytes.

uint32_t **CUpti_ActivityKernel::localMemoryTotal**

The total amount of local memory reserved for the kernel, in bytes.

const char* **CUpti_ActivityKernel::name**

The name of the kernel. This name is shared across all activity records representing the same kernel, and so should not be modified.

uint32_t **CUpti_ActivityKernel::pad**

Undefined. Reserved for internal use.

uint16_t **CUpti_ActivityKernel::registersPerThread**

The number of registers required for each thread executing the kernel.

void∗ **CUpti_ActivityKernel::reserved0**

Undefined. Reserved for internal use.

uint32_t **CUpti_ActivityKernel::runtimeCorrelationId**

The runtime correlation ID of the kernel. Each kernel execution is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the kernel.

uint64_t **CUpti_ActivityKernel::start**

The start timestamp for the kernel execution, in ns.

int32_t **CUpti_ActivityKernel::staticSharedMemory**

The static shared memory allocated for the kernel, in bytes.

uint32_t **CUpti_ActivityKernel::streamId**

The ID of the stream where the kernel is executing.

CUpti_ActivityMemcpy Type Reference

The activity record for memory copies.

Data Fields

- uint64_t bytes
- uint32_t contextId
- uint8_t copyKind
- uint32_t correlationId
- uint32_t deviceId
- uint8_t dstKind
- uint64_t end
- uint8_t flags
- CUpti_ActivityKind kind
- void ∗ reserved0
- uint32_t runtimeCorrelationId
- uint8_t srcKind
- uint64_t start
- uint32_t streamId

Detailed Description

This activity record represents a memory copy (CUPTI_ACTIVITY_KIND_MEMCPY).

Field Documentation

uint64_t **CUpti_ActivityMemcpy::bytes**

The number of bytes transferred by the memory copy.

uint32_t **CUpti_ActivityMemcpy::contextId**

The ID of the context where the memory copy is occurring.

uint8_t **CUpti_ActivityMemcpy::copyKind**

The kind of the memory copy, stored as a byte to reduce record size.

See also:

CUpti_ActivityMemcpyKind

uint32_t **CUpti_ActivityMemcpy::correlationId**

The correlation ID of the memory copy. Each memory copy is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the memory copy.

uint32_t **CUpti_ActivityMemcpy::deviceId**

The ID of the device where the memory copy is occurring.

uint8_t **CUpti_ActivityMemcpy::dstKind**

The destination memory kind read by the memory copy, stored as a byte to reduce record size.

See also:

CUpti_ActivityMemoryKind

uint64_t **CUpti_ActivityMemcpy::end**

The end timestamp for the memory copy, in ns.

uint8_t **CUpti_ActivityMemcpy::flags**

The flags associated with the memory copy.

See also:

CUpti_ActivityFlag

CUpti_ActivityKind CUpti_ActivityMemcpy::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MEMCPY.

void* **CUpti_ActivityMemcpy::reserved0**

Undefined. Reserved for internal use.

uint32_t **CUpti_ActivityMemcpy::runtimeCorrelationId**

The runtime correlation ID of the memory copy. Each memory copy is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the memory copy.

uint8_t **CUpti_ActivityMemcpy::srcKind**

The source memory kind read by the memory copy, stored as a byte to reduce record size.

See also:

CUpti_ActivityMemoryKind

uint64_t **CUpti_ActivityMemcpy::start**

The start timestamp for the memory copy, in ns.

uint32_t **CUpti_ActivityMemcpy::streamId**

The ID of the stream where the memory copy is occurring.

CUpti_ActivityMemset Type Reference

The activity record for memset.

Data Fields

- uint64_t bytes
- uint32_t contextId
- uint32_t correlationId
- uint32_t deviceId
- uint64_t end
- CUpti_ActivityKind kind
- void ∗ reserved0
- uint32_t runtimeCorrelationId
- uint64_t start
- uint32_t streamId
- uint32_t value

Detailed Description

This activity record represents a memory set operation (CUPTI_ACTIVITY_KIND_MEMSET).

Field Documentation

uint64_t **CUpti_ActivityMemset::bytes**

The number of bytes being set by the memory set.

uint32_t **CUpti_ActivityMemset::contextId**

The ID of the context where the memory set is occurring.

uint32_t **CUpti_ActivityMemset::correlationId**

The correlation ID of the memory set. Each memory set is assigned a unique correlation ID that is identical to the correlation ID in the driver API activity record that launched the memory set.

uint32_t **CUpti_ActivityMemset::deviceId**

The ID of the device where the memory set is occurring.

uint64_t **CUpti_ActivityMemset::end**

The end timestamp for the memory set, in ns.

CUpti_ActivityKind CUpti_ActivityMemset::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MEMSET.

void∗ **CUpti_ActivityMemset::reserved0**

Undefined. Reserved for internal use.

uint32_t **CUpti_ActivityMemset::runtimeCorrelationId**

The runtime correlation ID of the memory set. Each memory set is assigned a unique runtime correlation ID that is identical to the correlation ID in the runtime API activity record that launched the memory set.

uint64_t **CUpti_ActivityMemset::start**

The start timestamp for the memory set, in ns.

uint32_t **CUpti_ActivityMemset::streamId**

The ID of the stream where the memory set is occurring.

uint32_t **CUpti_ActivityMemset::value**

The value being assigned to memory by the memory set.

CUpti_ActivityMetric Type Reference

The activity record for a CUPTI metric.

Data Fields

- uint32_t correlationId
- CUpti_MetricID id
- CUpti_ActivityKind kind
- uint32_t pad
- CUpti_MetricValue value

Detailed Description

This activity record represents the collection of a CUPTI metric value (CUPTI_ACTIVITY_KIND_METRIC). This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profile frameworks built on top of CUPTI that collect metric data may choose to use this type to store the collected metric data.

Field Documentation

uint32_t **CUpti_ActivityMetric::correlationId**

The correlation ID of the metric. Use of this ID is user-defined, but typically this ID value will equal the correlation ID of the kernel for which the metric was gathered.

CUpti_MetricID CUpti_ActivityMetric::id

The metric ID.

CUpti_ActivityKind CUpti_ActivityMetric::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_METRIC.

uint32_t **CUpti_ActivityMetric::pad**

Undefined. Reserved for internal use.

CUpti_MetricValue CUpti_ActivityMetric::value

The metric value.

CUPTI Callback API

Data Structures

- struct CUpti_CallbackData

  Data passed into a runtime or driver API callback function.

- struct CUpti_ResourceData

  Data passed into a resource callback function.

- struct CUpti_SynchronizeData

  Data passed into a synchronize callback function.

Typedefs

- typedef void(∗ CUpti_CallbackFunc)(void ∗userdata, CUpti_CallbackDomain domain, CUpti_CallbackId cbid, const void ∗cbdata)

  Function type for a callback.

- typedef uint32_t CUpti_CallbackId

  An ID for a driver API, runtime API, resource or synchronization callback.

- typedef CUpti_CallbackDomain ∗ CUpti_DomainTable

  Pointer to an array of callback domains.

- typedef struct CUpti_Subscriber_st ∗ CUpti_SubscriberHandle

  A callback subscriber.

Enumerations

- enum CUpti_ApiCallbackSite {

  CUPTI_API_ENTER = 0,

  CUPTI_API_EXIT = 1 }

  Specifies the point in an API call that a callback is issued.

- enum CUpti_CallbackDomain {
  CUPTI_CB_DOMAIN_INVALID = 0,
  CUPTI_CB_DOMAIN_DRIVER_API = 1,
  CUPTI_CB_DOMAIN_RUNTIME_API = 2,
  CUPTI_CB_DOMAIN_RESOURCE = 3,
  CUPTI_CB_DOMAIN_SYNCHRONIZE = 4 }

  Callback domains.

- enum CUpti_CallbackIdResource {
  CUPTI_CBID_RESOURCE_INVALID = 0,
  CUPTI_CBID_RESOURCE_CONTEXT_CREATED = 1,
  CUPTI_CBID_RESOURCE_CONTEXT_DESTROY_STARTING = 2,
  CUPTI_CBID_RESOURCE_STREAM_CREATED = 3,
  CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING = 4 }

  Callback IDs for resource domain.

- enum CUpti_CallbackIdSync {
  CUPTI_CBID_SYNCHRONIZE_INVALID = 0,
  CUPTI_CBID_SYNCHRONIZE_STREAM_SYNCHRONIZED = 1,
  CUPTI_CBID_SYNCHRONIZE_CONTEXT_SYNCHRONIZED = 2 }

  Callback IDs for synchronization domain.

Functions

- CUptiResult cuptiEnableAllDomains (uint32_t enable, CUpti_SubscriberHandle subscriber)

  Enable or disable all callbacks in all domains.

- CUptiResult cuptiEnableCallback (uint32_t enable, CUpti_SubscriberHandle subscriber, CUpti_CallbackDomain domain, CUpti_CallbackId cbid)

  Enable or disabled callbacks for a specific domain and callback ID.

- CUptiResult cuptiEnableDomain (uint32_t enable, CUpti_SubscriberHandle subscriber, CUpti_CallbackDomain domain)

Enable or disabled all callbacks for a specific domain.

- CUptiResult cuptiGetCallbackState (uint32_t *enable, CUpti_SubscriberHandle subscriber, CUpti_CallbackDomain domain, CUpti_CallbackId cbid)

  Get the current enabled/disabled state of a callback for a specific domain and function ID.

- CUptiResult cuptiSubscribe (CUpti_SubscriberHandle *subscriber, CUpti_CallbackFunc callback, void *userdata)

  Initialize a callback subscriber with a callback function and user data.

- CUptiResult cuptiSupportedDomains (size_t *domainCount, CUpti_DomainTable *domainTable)

  Get the available callback domains.

- CUptiResult cuptiUnsubscribe (CUpti_SubscriberHandle subscriber)

  Unregister a callback subscriber.

Typedef Documentation

typedef void(* **CUpti_CallbackFunc**)(void *userdata, **CUpti_CallbackDomain** domain, **CUpti_CallbackId** cbid, const void *cbdata)

Function type for a callback. The type of the data passed to the callback in `cbdata` depends on the `domain`. If `domain` is CUPTI_CB_DOMAIN_DRIVER_API or CUPTI_CB_DOMAIN_RUNTIME_API the type of `cbdata` will be CUpti_CallbackData. If `domain` is CUPTI_CB_DOMAIN_RESOURCE the type of `cbdata` will be CUpti_ResourceData. If `domain` is CUPTI_CB_DOMAIN_SYNCHRONIZE the type of `cbdata` will be CUpti_SynchronizeData.

Parameters:

userdata User data supplied at subscription of the callback

domain The domain of the callback

cbid The ID of the callback

cbdata Data passed to the callback.

typedef uint32_t **CUpti_CallbackId**

An ID for a driver API, runtime API, resource or synchronization callback. Within a driver API callback this should be interpreted as a CUpti_driver_api_trace_cbid value. Within a runtime API callback this should be interpreted as a CUpti_runtime_api_trace_cbid value. Within a resource API callback this should be interpreted as a CUpti_CallbackIdResource value. Within a synchronize API callback this should be interpreted as a CUpti_CallbackIdSync value.

Enumeration Type Documentation

enum **CUpti_ApiCallbackSite**

Specifies the point in an API call that a callback is issued. This value is communicated to the callback function via CUpti_CallbackData::callbackSite.

Enumerator:

CUPTI_API_ENTER The callback is at the entry of the API call.

CUPTI_API_EXIT The callback is at the exit of the API call.

enum **CUpti_CallbackDomain**

Callback domains. Each domain represents callback points for a group of related API functions or CUDA driver activity.

Enumerator:

CUPTI_CB_DOMAIN_INVALID Invalid domain.

CUPTI_CB_DOMAIN_DRIVER_API Domain containing callback points for all driver API functions.

CUPTI_CB_DOMAIN_RUNTIME_API Domain containing callback points for all runtime API functions.

CUPTI_CB_DOMAIN_RESOURCE Domain containing callback points for CUDA resource tracking.

CUPTI_CB_DOMAIN_SYNCHRONIZE Domain containing callback points for CUDA synchronization.

enum **CUpti_CallbackIdResource**

Callback IDs for resource domain, CUPTI_CB_DOMAIN_RESOURCE. This value is communicated to the callback function via the cbid parameter.

Enumerator:

CUPTI_CBID_RESOURCE_INVALID Invalid resource callback ID.

CUPTI_CBID_RESOURCE_CONTEXT_CREATED A new context has been created.

CUPTI_CBID_RESOURCE_CONTEXT_DESTROY_STARTING A context is about to be destroyed.

CUPTI_CBID_RESOURCE_STREAM_CREATED A new stream has been created.

CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING A stream is about to be destroyed.

enum **CUpti_CallbackIdSync**

Callback IDs for synchronization domain, CUPTI_CB_DOMAIN_SYNCHRONIZE. This value is communicated to the callback function via the cbid parameter.

Enumerator:

CUPTI_CBID_SYNCHRONIZE_INVALID Invalid synchronize callback ID.

CUPTI_CBID_SYNCHRONIZE_STREAM_SYNCHRONIZED Stream synchronization has completed for the stream.

CUPTI_CBID_SYNCHRONIZE_CONTEXT_SYNCHRONIZED Context synchronization has completed for the context.

Function Documentation

CUptiResult cuptiEnableAllDomains (uint32_t enable, **CUpti_SubscriberHandle** subscriber)

Enable or disable all callbacks in all domains.

Note:

Thread-safety: a subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackState(sub, d, $*$) and cuptiEnableAllDomains(sub) are called concurrently, the results are undefined.

Parameters:

enable New enable state for all callbacks in all domain. Zero disables all callbacks, non-zero enables all callbacks.

subscriber - Handle to callback subscription

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialized CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `subscriber` is invalid

CUptiResult cuptiEnableCallback (uint32_t enable, **CUpti_SubscriberHandle** subscriber, **CUpti_CallbackDomain** domain, **CUpti_CallbackId** cbid)

Enable or disabled callbacks for a subscriber for a specific domain and callback ID.

Note:

Thread-safety: a subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackState(sub, d, c) and cuptiEnableCallback(sub, d, c) are called concurrently, the results are undefined.

Parameters:

enable New enable state for the callback. Zero disables the callback, non-zero enables the callback.

subscriber - Handle to callback subscription

domain The domain of the callback

cbid The ID of the callback

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialized CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `subscriber`, `domain` or `cbid` is invalid.

CUptiResult cuptiEnableDomain (uint32_t enable, **CUpti_SubscriberHandle** subscriber, **CUpti_CallbackDomain** domain)

Enable or disabled all callbacks for a specific domain.

Note:

Thread-safety: a subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackEnabled(sub, d, ∗) and cuptiEnableDomain(sub, d) are called concurrently, the results are undefined.

Parameters:

enable New enable state for all callbacks in the domain. Zero disables all callbacks, non-zero enables all callbacks.

subscriber - Handle to callback subscription

domain The domain of the callback

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialized CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `subscriber` or `domain` is invalid

CUptiResult cuptiGetCallbackState (uint32_t * enable, **CUpti_SubscriberHandle** subscriber, **CUpti_CallbackDomain** domain, **CUpti_CallbackId** cbid)

Returns non-zero in *`enable` if the callback for a domain and callback ID is enabled, and zero if not enabled.

Note:

Thread-safety: a subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackState(sub, d, c) and cuptiEnableCallback(sub, d, c) are called concurrently, the results are undefined.

Parameters:

enable Returns non-zero if callback enabled, zero if not enabled

subscriber Handle to the initialize subscriber

domain The domain of the callback

cbid The ID of the callback

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialized CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `enabled` is NULL, or if `subscriber`, `domain` or `cbid` is invalid.

CUptiResult cuptiSubscribe (**CUpti_SubscriberHandle** * subscriber, **CUpti_CallbackFunc** callback, void * userdata)

Initializes a callback subscriber with a callback function and (optionally) a pointer to user data. The returned subscriber handle can be used to enable and disable the callback for specific domains and callback IDs.

Note:

Only a single subscriber can be registered at a time.
This function does not enable any callbacks.
Thread-safety: this function is thread safe.

Parameters:

subscriber Returns handle to initialize subscriber

callback The callback function

userdata A pointer to user data. This data will be passed to the callback function via the `userdata` paramater.

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialize CUPTI

CUPTI_ERROR_MAX_LIMIT_REACHED if there is already a CUPTI subscriber

CUPTI_ERROR_INVALID_PARAMETER if `subscriber` is NULL

CUptiResult cuptiSupportedDomains (size_t ∗ domainCount, **CUpti_DomainTable** ∗ domainTable)

Returns in ∗`domainTable` an array of size ∗`domainCount` of all the available callback domains.

Note:

Thread-safety: this function is thread safe.

Parameters:

domainCount Returns number of callback domains

domainTable Returns pointer to array of available callback domains

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialize CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `domainCount` or `domainTable` are NULL

CUptiResult cuptiUnsubscribe (**CUpti_SubscriberHandle** subscriber)

Removes a callback subscriber so that no future callbacks will be issued to that subscriber.

Note:

Thread-safety: this function is thread safe.

Parameters:

subscriber Handle to the initialize subscriber

Return values:

CUPTI_SUCCESS on success

CUPTI_ERROR_NOT_INITIALIZED if unable to initialized CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `subscriber` is NULL or not initialized

CUpti_CallbackData Type Reference

Data passed into a runtime or driver API callback function.

Data Fields

- CUpti_ApiCallbackSite callbackSite
- CUcontext context
- uint32_t contextUid
- uint64_t * correlationData
- uint32_t correlationId
- const char * functionName
- const void * functionParams
- void * functionReturnValue
- const char * symbolName

Detailed Description

Data passed into a runtime or driver API callback function as the `cbdata` argument to CUpti_CallbackFunc. The `cbdata` will be this type for `domain` equal to CUPTI_CB_DOMAIN_DRIVER_API or CUPTI_CB_DOMAIN_RUNTIME_API. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data. For example, if you make a shallow copy of CUpti_CallbackData within a callback, you cannot dereference `functionParams` outside of that callback to access the function parameters. `functionName` is an exception: the string pointed to by `functionName` is a global constant and so may be accessed outside of the callback.

Field Documentation

CUpti_ApiCallbackSite CUpti_CallbackData::callbackSite

Point in the runtime or driver function from where the callback was issued.

CUcontext **CUpti_CallbackData::context**

Driver context current to the thread, or null if no context is current. This value can change from the entry to exit callback of a runtime API function if the runtime initializes a context.

uint32_t **CUpti_CallbackData::contextUid**

Unique ID for the CUDA context associated with the thread. The UIDs are assigned sequentially as contexts are created and are unique within a process.

uint64_t* **CUpti_CallbackData::correlationData**

Pointer to data shared between the entry and exit callbacks of a given runtime or drive API function invocation. This field can be used to pass 64-bit values from the entry callback to the corresponding exit callback.

uint32_t **CUpti_CallbackData::correlationId**

The activity record correlation ID for this callback. For a driver domain callback (i.e. `domain` CUPTI_CB_DOMAIN_DRIVER_API) this ID will equal the correlation ID in the CUpti_ActivityAPI record corresponding to the CUDA driver function call. For a runtime domain callback (i.e. `domain` CUPTI_CB_DOMAIN_RUNTIME_API) this ID will equal the correlation ID in the CUpti_ActivityAPI record corresponding to the CUDA runtime function call. Within the callback, this ID can be recorded to correlate user data with the activity record. This field is new in 4.1.

const char* **CUpti_CallbackData::functionName**

Name of the runtime or driver API function which issued the callback. This string is a global constant and so may be accessed outside of the callback.

const void* **CUpti_CallbackData::functionParams**

Pointer to the arguments passed to the runtime or driver API call. See generated_cuda_runtime_api_meta::h and generated_cuda_meta::h for structure definitions for the parameters for each runtime and driver API function.

void* **CUpti_CallbackData::functionReturnValue**

Pointer to the return value of the runtime or driver API call. This field is only valid within the exit::CUPTI_API_EXIT callback. For a runtime API `functionReturnValue` points to a `cudaError_t`. For a driver API `functionReturnValue` points to a `CUresult`.

const char* **CUpti_CallbackData::symbolName**

Name of the symbol operated on by the runtime or driver API function which issued the callback. This entry is valid only for the runtime cudaLaunch callback (i.e. CUPTI_RUNTIME_TRACE_CBID_cudaLaunch_v3020), where it returns the name of the kernel.

CUpti_ResourceData Type Reference

Data passed into a resource callback function.

Data Fields

- CUcontext context
- void ∗ resourceDescriptor
- CUstream stream

Detailed Description

Data passed into a resource callback function as the `cbdata` argument to CUpti_CallbackFunc. The `cbdata` will be this type for `domain` equal to CUPTI_CB_DOMAIN_RESOURCE. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

Field Documentation

CUcontext **CUpti_ResourceData::context**

For CUPTI_CBID_RESOURCE_CONTEXT_CREATED and CUPTI_CBID_RESOURCE_CONTEXT_DESTROY_STARTING, the context being created or destroyed. For CUPTI_CBID_RESOURCE_STREAM_CREATED and CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING, the context containing the stream being created or destroyed.

void∗ **CUpti_ResourceData::resourceDescriptor**

Reserved for future use.

CUstream **CUpti_ResourceData::stream**

For CUPTI_CBID_RESOURCE_STREAM_CREATED and CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING, the stream being created or destroyed.

CUpti_SynchronizeData Type Reference

Data passed into a synchronize callback function.

Data Fields

- CUcontext context
- CUstream stream

Detailed Description

Data passed into a synchronize callback function as the `cbdata` argument to CUpti_CallbackFunc. The `cbdata` will be this type for `domain` equal to CUPTI_CB_DOMAIN_SYNCHRONIZE. The callback data is valid only within the invocation of the callback function that is passed the data. If you need to retain some data for use outside of the callback, you must make a copy of that data.

Field Documentation

CUcontext **CUpti_SynchronizeData::context**

The context of the stream being synchronized.

CUstream **CUpti_SynchronizeData::stream**

The stream being synchronized.

CUPTI Event API

Data Structures

- struct CUpti_EventGroupSet

  A set of event groups.

- struct CUpti_EventGroupSets

  A set of event group sets.

Defines

- #define CUPTI_EVENT_OVERFLOW ((uint64_t)0xFFFFFFFFFFFFFFFFULL)

  The overflow value for a CUPTI event.

Typedefs

- typedef uint32_t CUpti_EventDomainID

  ID for an event domain.

- typedef void ∗ CUpti_EventGroup

  A group of events.

- typedef uint32_t CUpti_EventID

  ID for an event.

Enumerations

- enum CUpti_DeviceAttribute {

  CUPTI_DEVICE_ATTR_MAX_EVENT_ID = 1,

  CUPTI_DEVICE_ATTR_MAX_EVENT_DOMAIN_ID = 2,

  CUPTI_DEVICE_ATTR_GLOBAL_MEMORY_BANDWIDTH = 3, CUPTI_DEVICE_ATTR_INSTRUCTION_PER_CYCLE = 4,

CUPTI_DEVICE_ATTR_INSTRUCTION_THROUGHPUT_SINGLE_PRECISION = 5 }

Device attributes.

- enum CUpti_EventAttribute {

CUPTI_EVENT_ATTR_NAME = 0,

CUPTI_EVENT_ATTR_SHORT_DESCRIPTION = 1,

CUPTI_EVENT_ATTR_LONG_DESCRIPTION = 2,

CUPTI_EVENT_ATTR_CATEGORY = 3 }

Event attributes.

- enum CUpti_EventCategory {

CUPTI_EVENT_CATEGORY_INSTRUCTION = 0,

CUPTI_EVENT_CATEGORY_MEMORY = 1,

CUPTI_EVENT_CATEGORY_CACHE = 2,

CUPTI_EVENT_CATEGORY_PROFILE_TRIGGER = 3 }

An event category.

- enum CUpti_EventCollectionMode {

CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS = 0,

CUPTI_EVENT_COLLECTION_MODE_KERNEL = 1 }

Event collection modes.

- enum CUpti_EventDomainAttribute {

CUPTI_EVENT_DOMAIN_ATTR_NAME = 0,

CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT = 1,

CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT = 3 }

Event domain attributes.

- enum CUpti_EventGroupAttribute {

CUPTI_EVENT_GROUP_ATTR_EVENT_DOMAIN_ID = 0,

CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES = 1,

CUPTI_EVENT_GROUP_ATTR_USER_DATA = 2,

CUPTI_EVENT_GROUP_ATTR_NUM_EVENTS = 3,

CUPTI_EVENT_GROUP_ATTR_EVENTS = 4,

CUPTI_EVENT_GROUP_ATTR_INSTANCE_COUNT = 5 }

Event group attributes.

- enum CUpti_ReadEventFlags { CUPTI_EVENT_READ_FLAG_NONE = 0 }

  Flags for cuptiEventGroupReadEvent an cuptiEventGroupReadAllEvents.

Functions

- CUptiResult cuptiDeviceEnumEventDomains (CUdevice device, size_t ∗arraySizeBytes, CUpti_EventDomainID ∗domainArray)

  Get the event domains for a device.

- CUptiResult cuptiDeviceGetAttribute (CUdevice device, CUpti_DeviceAttribute attrib, size_t ∗valueSize, void ∗value)

  Read a device attribute.

- CUptiResult cuptiDeviceGetEventDomainAttribute (CUdevice device, CUpti_EventDomainID eventDomain, CUpti_EventDomainAttribute attrib, size_t ∗valueSize, void ∗value)

  Read an event domain attribute.

- CUptiResult cuptiDeviceGetNumEventDomains (CUdevice device, uint32_t ∗numDomains)

  Get the number of domains for a device.

- CUptiResult cuptiDeviceGetTimestamp (CUcontext context, uint64_t ∗timestamp)

  Read a device timestamp.

- CUptiResult cuptiEnumEventDomains (size_t ∗arraySizeBytes, CUpti_EventDomainID ∗domainArray)

  Get the event domains available on any device.

- CUptiResult cuptiEventDomainEnumEvents (CUpti_EventDomainID eventDomain, size_t ∗arraySizeBytes, CUpti_EventID ∗eventArray)

  Get the events in a domain.

- CUptiResult cuptiEventDomainGetAttribute (CUpti_EventDomainID eventDomain, CUpti_EventDomainAttribute attrib, size_t ∗valueSize, void ∗value)

  Read an event domain attribute.

- CUptiResult cuptiEventDomainGetNumEvents (CUpti_EventDomainID eventDomain, uint32_t ∗numEvents)

  Get number of events in a domain.

- CUptiResult cuptiEventGetAttribute (CUpti_EventID event, CUpti_EventAttribute attrib, size_t ∗valueSize, void ∗value)

  Get an event attribute.

- CUptiResult cuptiEventGetIdFromName (CUdevice device, const char ∗eventName, CUpti_EventID ∗event)

  Find an event by name.

- CUptiResult cuptiEventGroupAddEvent (CUpti_EventGroup eventGroup, CUpti_EventID event)

  Add an event to an event group.

- CUptiResult cuptiEventGroupCreate (CUcontext context, CUpti_EventGroup ∗eventGroup, uint32_t flags)

  Create a new event group for a context.

- CUptiResult cuptiEventGroupDestroy (CUpti_EventGroup eventGroup)

  Destroy an event group.

- CUptiResult cuptiEventGroupDisable (CUpti_EventGroup eventGroup)

  Disable an event group.

- CUptiResult cuptiEventGroupEnable (CUpti_EventGroup eventGroup)

  Enable an event group.

- CUptiResult cuptiEventGroupGetAttribute (CUpti_EventGroup eventGroup, CUpti_EventGroupAttribute attrib, size_t ∗valueSize, void ∗value)

  Read an event group attribute.

- CUptiResult cuptiEventGroupReadAllEvents (CUpti_EventGroup eventGroup, CUpti_ReadEventFlags flags, size_t ∗eventValueBufferSizeBytes, uint64_t *eventValueBuffer, size_t *eventIdArraySizeBytes, CUpti_EventID *eventIdArray, size_t *numEventIdsRead)

  Read the values for all the events in an event group.

- CUptiResult cuptiEventGroupReadEvent (CUpti_EventGroup eventGroup, CUpti_ReadEventFlags flags, CUpti_EventID event, size_t *eventValueBufferSizeBytes, uint64_t *eventValueBuffer)

  Read the value for an event in an event group.

- CUptiResult cuptiEventGroupRemoveAllEvents (CUpti_EventGroup eventGroup)

  Remove all events from an event group.

- CUptiResult cuptiEventGroupRemoveEvent (CUpti_EventGroup eventGroup, CUpti_EventID event)

  Remove an event from an event group.

- CUptiResult cuptiEventGroupResetAllEvents (CUpti_EventGroup eventGroup)

  Zero all the event counts in an event group.

- CUptiResult cuptiEventGroupSetAttribute (CUpti_EventGroup eventGroup, CUpti_EventGroupAttribute attrib, size_t valueSize, void *value)

  Write an event group attribute.

- CUptiResult cuptiEventGroupSetsCreate (CUcontext context, size_t eventIdArraySizeBytes, CUpti_EventID *eventIdArray, CUpti_EventGroupSets **eventGroupPasses)

  For a set of events, get the grouping that indicates the number of passes and the event groups necessary to collect the events.

- CUptiResult cuptiEventGroupSetsDestroy (CUpti_EventGroupSets *eventGroupSets)

  Destroy a CUpti_EventGroupSets object.

- CUptiResult cuptiGetNumEventDomains (uint32_t *numDomains)

  Get the number of event domains available on any device.

- CUptiResult cuptiSetEventCollectionMode (CUcontext context, CUpti_EventCollectionMode mode)

  Set the event collection mode.

Define Documentation

#define CUPTI_EVENT_OVERFLOW ((uint64_t)0xFFFFFFFFFFFFFFFFULL)

The CUPTI event value that indicates an overflow.

Typedef Documentation

typedef uint32_t **CUpti_EventDomainID**

ID for an event domain. An event domain represents a group of related events. A device may have multiple instances of a domain, indicating that the device can simultaneously record multiple instances of each event within that domain.

typedef void∗ **CUpti_EventGroup**

An event group is a collection of events that are managed together. All events in an event group must belong to the same domain.

typedef uint32_t **CUpti_EventID**

An event represents a countable activity, action, or occurrence on the device.

Enumeration Type Documentation

enum **CUpti_DeviceAttribute**

CUPTI device attributes. These attributes can be read using cuptiDeviceGetAttribute.

Enumerator:

CUPTI_DEVICE_ATTR_MAX_EVENT_ID Number of event IDs for a device. Value is a uint32_t.

CUPTI_DEVICE_ATTR_MAX_EVENT_DOMAIN_ID Number of event domain IDs for a device. Value is a uint32_t.

CUPTI_DEVICE_ATTR_GLOBAL_MEMORY_BANDWIDTH Get global memory bandwidth in Kbytes/sec. Value is a uint64_t.

CUPTI_DEVICE_ATTR_INSTRUCTION_PER_CYCLE Get theoretical instructions per cycle. Value is a uint32_t.

CUPTI_DEVICE_ATTR_INSTRUCTION_THROUGHPUT_SINGLE_PRECISION Get theoretical number of single precision instructions that can be executed per second. Value is a uint64_t.

enum **CUpti_EventAttribute**

Event attributes. These attributes can be read using cuptiEventGetAttribute.

Enumerator:

CUPTI_EVENT_ATTR_NAME Event name. Value is a null terminated const c-string.

CUPTI_EVENT_ATTR_SHORT_DESCRIPTION Short description of event. Value is a null terminated const c-string.

CUPTI_EVENT_ATTR_LONG_DESCRIPTION Long description of event. Value is a null terminated const c-string.

CUPTI_EVENT_ATTR_CATEGORY Category of event. Value is CUpti_EventCategory.

enum **CUpti_EventCategory**

Each event is assigned to a category that represents the general type of the event. A event's category is accessed using cuptiEventGetAttribute and the CUPTI_EVENT_ATTR_CATEGORY attribute.

Enumerator:

CUPTI_EVENT_CATEGORY_INSTRUCTION An instruction related event.

CUPTI_EVENT_CATEGORY_MEMORY A memory related event.

CUPTI_EVENT_CATEGORY_CACHE A cache related event.

CUPTI_EVENT_CATEGORY_PROFILE_TRIGGER A profile-trigger event.

enum **CUpti_EventCollectionMode**

The event collection mode determines the period over which the events within the enabled event groups will be collected.

Enumerator:

CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS Events are collected for the entire duration between the cuptiEventGroupEnable and cuptiEventGroupDisable calls. This is the default mode.

CUPTI_EVENT_COLLECTION_MODE_KERNEL Events are collected only for the durations of kernel executions that occur between the cuptiEventGroupEnable and cuptiEventGroupDisable calls. Event collection begins when a kernel execution begins, and stops when kernel execution completes. If multiple kernel executions occur between the cuptiEventGroupEnable and cuptiEventGroupDisable calls then the event values must be read after each kernel launch if those events need to be associated with the specific kernel launch.

enum **CUpti_EventDomainAttribute**

Event domain attributes. Except where noted, all the attributes can be read using either cuptiDeviceGetEventDomainAttribute or cuptiEventDomainGetAttribute.

Enumerator:

CUPTI_EVENT_DOMAIN_ATTR_NAME Event domain name. Value is a null terminated const c-string.

CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT Number of instances of the domain for which event counts will be collected. The domain may have additional instances that cannot be profiled (see CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT). Can be read only with cuptiDeviceGetEventDomainAttribute. Value is a uint32_t.

CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT Total number of instances of the domain, including instances that cannot be profiled. Use CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT to get the number of instances that can be profiled. Can be read only with cuptiDeviceGetEventDomainAttribute. Value is a uint32_t.

enum **CUpti_EventGroupAttribute**

Event group attributes. These attributes can be read using cuptiEventGroupGetAttribute. Attributes marked [rw] can also be written using cuptiEventGroupSetAttribute.

Enumerator:

CUPTI_EVENT_GROUP_ATTR_EVENT_DOMAIN_ID The domain to which the event group is bound. This attribute is set when the first event is added to the group. Value is a CUpti_EventDomainID.

CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES [rw] Profile all the instances of the domain for this eventgroup. This feature can be used to get load balancing across all instances of a domain. Value is an integer.

CUPTI_EVENT_GROUP_ATTR_USER_DATA [rw] Reserved for user data.

CUPTI_EVENT_GROUP_ATTR_NUM_EVENTS Number of events in the group. Value is a uint32_t.

CUPTI_EVENT_GROUP_ATTR_EVENTS Enumerates events in the group. Value is a pointer to buffer of size sizeof(CUpti_EventID) ∗ num_of_events in the eventgroup. num_of_events can be queried using CUPTI_EVENT_GROUP_ATTR_NUM_EVENTS.

CUPTI_EVENT_GROUP_ATTR_INSTANCE_COUNT Number of instances of the domain bound to this event group that will be counted. Value is a uint32_t.

enum **CUpti_ReadEventFlags**

Flags for cuptiEventGroupReadEvent an cuptiEventGroupReadAllEvents.

Enumerator:

CUPTI_EVENT_READ_FLAG_NONE No flags.

Function Documentation

CUptiResult cuptiDeviceEnumEventDomains (CUdevice device, size_t ∗ arraySizeBytes, **CUpti_EventDomainID** ∗ domainArray)

Returns the event domains IDs in `domainArray` for a device. The size of the `domainArray` buffer is given by ∗`arraySizeBytes`. The size of the `domainArray` buffer must be at least `numdomains` ∗ sizeof(CUpti_EventDomainID) or else all domains will not be returned. The value returned in ∗`arraySizeBytes` contains the number of bytes returned in `domainArray`.

Parameters:

device The CUDA device

arraySizeBytes The size of `domainArray` in bytes, and returns the number of bytes written to `domainArray`

domainArray Returns the IDs of the event domains for the device

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `domainArray` are NULL

CUptiResult cuptiDeviceGetAttribute (CUdevice device, **CUpti_DeviceAttribute** attrib, size_t * valueSize, void * value)

Read a device attribute and return it in *`value`.

Parameters:

device The CUDA device

attrib The attribute to read

valueSize Size of buffer pointed by the value, and returns the number of bytes written to `value`

value Returns the value of the attribute

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not a device attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiDeviceGetEventDomainAttribute (CUdevice device, **CUpti_EventDomainID** eventDomain, **CUpti_EventDomainAttribute** attrib, size_t * valueSize, void * value)

Returns an event domain attribute in *`value`. The size of the `value` buffer is given by *`valueSize`. The value returned in *`valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than *`valueSize`, then only the first *`valueSize` characters will be returned and there will be no terminating null byte.

Parameters:

device The CUDA device

eventDomain ID of the event domain

attrib The event domain attribute to read

valueSize The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

value Returns the attribute's value

Return values:

CUPTI_SUCCESS
CUPTI_ERROR_NOT_INITIALIZED
CUPTI_ERROR_INVALID_DEVICE
CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID
CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not an event domain attribute
CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiDeviceGetNumEventDomains (CUdevice device, uint32_t * numDomains)

Returns the number of domains in `numDomains` for a device.

Parameters:

device The CUDA device
numDomains Returns the number of domains

Return values:

CUPTI_SUCCESS
CUPTI_ERROR_NOT_INITIALIZED
CUPTI_ERROR_INVALID_DEVICE
CUPTI_ERROR_INVALID_PARAMETER if `numDomains` is NULL

CUptiResult cuptiDeviceGetTimestamp (CUcontext context, uint64_t * timestamp)

Returns the device timestamp in `*timestamp`. The timestamp is reported in nanoseconds and indicates the time since the device was last reset.

Parameters:

context A context on the device from which to get the timestamp
timestamp Returns the device timestamp

Return values:

CUPTI_SUCCESS
CUPTI_ERROR_NOT_INITIALIZED
CUPTI_ERROR_INVALID_CONTEXT
CUPTI_ERROR_INVALID_PARAMETER is `timestamp` is NULL

CUptiResult cuptiEnumEventDomains (size_t * arraySizeBytes, **CUpti_EventDomainID** * domainArray)

Returns all the event domains available on any CUDA-capable device. Event domain IDs are returned in `domainArray`. The size of the `domainArray` buffer is given by *`arraySizeBytes`. The size of the `domainArray` buffer must be at least `numDomains` * sizeof(CUpti_EventDomainID) or all domains will not be returned. The value returned in *`arraySizeBytes` contains the number of bytes returned in `domainArray`.

Parameters:

arraySizeBytes The size of `domainArray` in bytes, and returns the number of bytes written to `domainArray`

domainArray Returns all the event domains

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `domainArray` are NULL

CUptiResult cuptiEventDomainEnumEvents (**CUpti_EventDomainID** eventDomain, size_t * arraySizeBytes, **CUpti_EventID** * eventArray)

Returns the event IDs in `eventArray` for a domain. The size of the `eventArray` buffer is given by *`arraySizeBytes`. The size of the `eventArray` buffer must be at least `numdomainevents` * sizeof(CUpti_EventID) or else all events will not be returned. The value returned in *`arraySizeBytes` contains the number of bytes returned in `eventArray`.

Parameters:

eventDomain ID of the event domain

arraySizeBytes The size of `eventArray` in bytes, and returns the number of bytes written to `eventArray`

eventArray Returns the IDs of the events in the domain

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `eventArray` are NULL

CUptiResult cuptiEventDomainGetAttribute (**CUpti_EventDomainID** eventDomain, **CUpti_EventDomainAttribute** attrib, size_t ∗ valueSize, void ∗ value)

Returns an event domain attribute in ∗`value`. The size of the `value` buffer is given by ∗`valueSize`. The value returned in ∗`valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than ∗`valueSize`, then only the first ∗`valueSize` characters will be returned and there will be no terminating null byte.

Parameters:

eventDomain ID of the event domain

attrib The event domain attribute to read

valueSize The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

value Returns the attribute's value

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not an event domain attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiEventDomainGetNumEvents (**CUpti_EventDomainID** eventDomain, uint32_t ∗ numEvents)

Returns the number of events in `numEvents` for a domain.

Parameters:

eventDomain ID of the event domain

numEvents Returns the number of events in the domain

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID

CUPTI_ERROR_INVALID_PARAMETER if `numEvents` is NULL

CUptiResult cuptiEventGetAttribute (**CUpti_EventID** event, **CUpti_EventAttribute** attrib, size_t * valueSize, void * value)

Returns an event attribute in *`value`. The size of the `value` buffer is given by *`valueSize`. The value returned in *`valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than *`valueSize`, then only the first *`valueSize` characters will be returned and there will be no terminating null byte.

Parameters:

event ID of the event

attrib The event attribute to read

valueSize The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

value Returns the attribute's value

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_ID

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not an event attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiEventGetIdFromName (CUdevice device, const char * eventName, **CUpti_EventID** * event)

Find an event by name and return the event ID in *`event`.

Parameters:

device The CUDA device

eventName The name of the event to find

event Returns the ID of the found event or undefined if unable to find the event

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_EVENT_NAME if unable to find an event with name `eventName`. In this case *`event` is undefined

CUPTI_ERROR_INVALID_PARAMETER if `eventName` or `event` are NULL

CUptiResult cuptiEventGroupAddEvent (**CUpti_EventGroup** eventGroup, **CUpti_EventID** event)

Add an event to an event group. The event add can fail for a number of reasons:

- The event group is enabled
- The event does not belong to the same event domain as the events that are already in the event group
- Device limitations on the events that can belong to the same group
- The event group is full

Parameters:

eventGroup The event group

event The event to add to the group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_ID

CUPTI_ERROR_OUT_OF_MEMORY

CUPTI_ERROR_INVALID_OPERATION if `eventGroup` is enabled

CUPTI_ERROR_NOT_COMPATIBLE if `event` belongs to a different event domain than the events already in `eventGroup`, or if a device limitation prevents `event` from being collected at the same time as the events already in `eventGroup`

CUPTI_ERROR_MAX_LIMIT_REACHED if `eventGroup` is full

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupCreate (CUcontext context, **CUpti_EventGroup** * eventGroup, uint32_t flags)

Creates a new event group for `context` and returns the new group in *`eventGroup`.

Note:

`flags` are reserved for future use and should be set to zero.

Parameters:

context The context for the event group

eventGroup Returns the new event group

flags Reserved - must be zero

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_CONTEXT

CUPTI_ERROR_OUT_OF_MEMORY

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupDestroy (**CUpti_EventGroup** eventGroup)

Destroy an `eventGroup` and free its resources. An event group cannot be destroyed if it is enabled.

Parameters:

eventGroup The event group to destroy

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_OPERATION if the event group is enabled

CUPTI_ERROR_INVALID_PARAMETER if eventGroup is NULL

CUptiResult cuptiEventGroupDisable (**CUpti_EventGroup** eventGroup)

Disable an event group. Disabling an event group stops collection of events contained in the group.

Parameters:

eventGroup The event group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_HARDWARE

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupEnable (**CUpti_EventGroup** eventGroup)

Enable an event group. Enabling an event group zeros the value of all the events in the group and then starts collection of those events.

Parameters:

eventGroup The event group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_HARDWARE

CUPTI_ERROR_NOT_READY if `eventGroup` does not contain any events

CUPTI_ERROR_NOT_COMPATIBLE if `eventGroup` cannot be enabled due to other already enabled event groups

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupGetAttribute (**CUpti_EventGroup** eventGroup, **CUpti_EventGroupAttribute** attrib, size_t * valueSize, void * value)

Read an event group attribute and return it in *`value`.

Parameters:

eventGroup The event group

attrib The attribute to read

valueSize Size of buffer pointed by the value, and returns the number of bytes written to `value`

value Returns the value of the attribute

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not an eventgroup attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiEventGroupReadAllEvents (**CUpti_EventGroup** eventGroup, **CUpti_ReadEventFlags** flags, size_t * eventValueBufferSizeBytes, uint64_t * eventValueBuffer, size_t * eventIdArraySizeBytes, **CUpti_EventID** * eventIdArray, size_t * numEventIdsRead)

Read the values for all the events in an event group. The event values are returned in the `eventValueBuffer` buffer. `eventValueBufferSizeBytes` indicates the size of `eventValueBuffer`. The buffer must be at least (sizeof(uint64) * number of events in group) if CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES is not set on the group containing the events. The buffer must be at least (sizeof(uint64) * number of domain instances * number of events in group) if CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES is set on the group.

The data format returned in `eventValueBuffer` is:

- domain instance 0: event0 event1 ... eventN
- domain instance 1: event0 event1 ... eventN
- ...
- domain instance M: event0 event1 ... eventN

The event order in `eventValueBuffer` is returned in `eventIdArray`. The size of `eventIdArray` is specified in `eventIdArraySizeBytes`. The size should be at least (sizeof(CUpti_EventID) * number of events in group).

If any instance of any event counter overflows, the value returned for that event instance will be CUPTI_EVENT_OVERFLOW.

The only allowed value for `flags` is CUPTI_EVENT_READ_FLAG_NONE.

Reading events from a disabled event group is not allowed.

Parameters:

eventGroup The event group

flags Flags controlling the reading mode

eventValueBufferSizeBytes The size of `eventValueBuffer` in bytes, and returns the number of bytes written to `eventValueBuffer`

eventValueBuffer Returns the event values

eventIdArraySizeBytes The size of `eventIdArray` in bytes, and returns the number of bytes written to `eventIdArray`

eventIdArray Returns the IDs of the events in the same order as the values return in eventValueBuffer.

numEventIdsRead Returns the number of event IDs returned in

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_HARDWARE

CUPTI_ERROR_INVALID_OPERATION if `eventGroup` is disabled

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup`, `eventValueBufferSizeBytes`, `eventValueBuffer`, `eventIdArraySizeBytes`, `eventIdArray` or `numEventIdsRead` is NULL

CUptiResult cuptiEventGroupReadEvent (**CUpti_EventGroup** eventGroup, **CUpti_ReadEventFlags** flags, **CUpti_EventID** event, size_t * eventValueBufferSizeBytes, uint64_t * eventValueBuffer)

Read the value for an event in an event group. The event value is returned in the `eventValueBuffer` buffer. `eventValueBufferSizeBytes` indicates the size of the `eventValueBuffer` buffer. The buffer must be at least sizeof(uint64) if CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES is not set on the group containing the event. The buffer must be at least (sizeof(uint64) * number of domain instances) if CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES is set on the group.

If any instance of an event counter overflows, the value returned for that event instance will be CUPTI_EVENT_OVERFLOW.

The only allowed value for `flags` is CUPTI_EVENT_READ_FLAG_NONE.

Reading an event from a disabled event group is not allowed.

Parameters:

eventGroup The event group

flags Flags controlling the reading mode

event The event to read

eventValueBbufferSizeBytes The size of `eventValueBuffer` in bytes, and returns the number of bytes written to `eventValueBuffer`

eventValueBuffer Returns the event value(s)

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_ID

CUPTI_ERROR_HARDWARE

CUPTI_ERROR_INVALID_OPERATION if `eventGroup` is disabled

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup`, `eventValueBufferSizeBytes` or `eventValueBuffer` is NULL

CUptiResult cuptiEventGroupRemoveAllEvents (**CUpti_EventGroup** eventGroup)

Remove all events from an event group. Events cannot be removed if the event group is enabled.

Parameters:

eventGroup The event group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_OPERATION if `eventGroup` is enabled

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupRemoveEvent (**CUpti_EventGroup** eventGroup, **CUpti_EventID** event)

Remove `event` from the an event group. The event cannot be removed if the event group is enabled.

Parameters:

eventGroup The event group

event The event to remove from the group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_EVENT_ID

CUPTI_ERROR_INVALID_OPERATION if `eventGroup` is enabled

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupResetAllEvents (**CUpti_EventGroup** eventGroup)

Zero all the event counts in an event group.

Parameters:

eventGroup The event group

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_HARDWARE

CUPTI_ERROR_INVALID_PARAMETER if `eventGroup` is NULL

CUptiResult cuptiEventGroupSetAttribute (**CUpti_EventGroup** eventGroup, **CUpti_EventGroupAttribute** attrib, size_t valueSize, void ∗ value)

Write an event group attribute.

Parameters:

eventGroup The event group

attrib The attribute to write

valueSize The size, in bytes, of the value

value The attribute value to write

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not an event group attribute, or if `attrib` is not a writable attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT Indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiEventGroupSetsCreate (CUcontext context, size_t eventIdArraySizeBytes, **CUpti_EventID** ∗ eventIdArray, **CUpti_EventGroupSets** ∗∗ eventGroupPasses)

The number of events that can be collected simultaneously varies by device and by the type of the events. When events can be collected simultaneously, they may need to be grouped into multiple event groups because they are from different event domains. This function takes a set of events and determines how many passes are required to collect all those events, and which events can be collected simultaneously in each pass.

The CUpti_EventGroupSets returned in `eventGroupPasses` indicates how many passes are required to collect the events with the `numSets` field. The `sets` array indicates the event groups that should be collected on each pass.

Parameters:

context The context for event collection

eventIdArraySizeBytes Size of eventIdArray in bytes

eventIdArray Array of event IDs that need to be grouped

eventGroupPasses Returns a CUpti_EventGroupSets object that indicates the number of passes required to collect the events and the events to collect on each pass

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_CONTEXT

CUPTI_ERROR_INVALID_EVENT_ID

CUPTI_ERROR_INVALID_PARAMETER if `eventIdArray` or `eventGroupPasses` is NULL

CUptiResult cuptiEventGroupSetsDestroy (**CUpti_EventGroupSets** * eventGroupSets)

Destroy a CUpti_EventGroupSets object.

Parameters:

eventGroupSets The object to destroy

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_OPERATION if any of the event groups contained in the sets is enabled

CUPTI_ERROR_INVALID_PARAMETER if `eventGroupSets` is NULL

CUptiResult cuptiGetNumEventDomains (uint32_t * numDomains)

Returns the total number of event domains available on any CUDA-capable device.

Parameters:

numDomains Returns the number of domains

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_INVALID_PARAMETER if `numDomains` is NULL

CUptiResult cuptiSetEventCollectionMode (CUcontext context, **CUpti_EventCollectionMode** mode)

Set the event collection mode for a `context`. The `mode` controls the event collection behavior of all events in event groups created in the `context`.

Parameters:

context The context

mode The event collection mode

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_CONTEXT

CUPTI Metric API

Data Structures

- union CUpti_MetricValue

  A metric value.

Typedefs

- typedef uint32_t CUpti_MetricID

  ID for a metric.

Enumerations

- enum CUpti_MetricAttribute {
  CUPTI_METRIC_ATTR_NAME = 0,
  CUPTI_METRIC_ATTR_SHORT_DESCRIPTION = 1,
  CUPTI_METRIC_ATTR_LONG_DESCRIPTION = 2,
  CUPTI_METRIC_ATTR_CATEGORY = 3,
  CUPTI_METRIC_ATTR_VALUE_KIND = 4 }

  Metric attributes.

- enum CUpti_MetricCategory {
  CUPTI_METRIC_CATEGORY_MEMORY = 0,
  CUPTI_METRIC_CATEGORY_INSTRUCTION = 1,
  CUPTI_METRIC_CATEGORY_MULTIPROCESSOR = 2,
  CUPTI_METRIC_CATEGORY_CACHE = 3,
  CUPTI_METRIC_CATEGORY_TEXTURE = 4 }

  A metric category.

- enum CUpti_MetricValueKind {
  CUPTI_METRIC_VALUE_KIND_DOUBLE = 0,
  CUPTI_METRIC_VALUE_KIND_UINT64 = 1,

CUPTI_METRIC_VALUE_KIND_PERCENT = 2,

CUPTI_METRIC_VALUE_KIND_THROUGHPUT = 3 }

Kinds of metric values.

Functions

- CUptiResult cuptiDeviceEnumMetrics (CUdevice device, size_t ∗arraySizeBytes, CUpti_MetricID ∗metricArray)

  Get the metrics for a device.

- CUptiResult cuptiDeviceGetNumMetrics (CUdevice device, uint32_t ∗numMetrics)

  Get the number of metrics for a device.

- CUptiResult cuptiEnumMetrics (size_t ∗arraySizeBytes, CUpti_MetricID ∗metricArray)

  Get all the metrics available on any device.

- CUptiResult cuptiGetNumMetrics (uint32_t ∗numMetrics)

  Get the total number of metrics available on any device.

- CUptiResult cuptiMetricCreateEventGroupSets (CUcontext context, size_t metricIdArraySizeBytes, CUpti_MetricID ∗metricIdArray, CUpti_EventGroupSets ∗∗eventGroupPasses)

  For a set of metrics, get the grouping that indicates the number of passes and the event groups necessary to collect the events required for those metrics.

- CUptiResult cuptiMetricEnumEvents (CUpti_MetricID metric, size_t ∗eventIdArraySizeBytes, CUpti_EventID ∗eventIdArray)

  Get the events required to calculating a metric.

- CUptiResult cuptiMetricGetAttribute (CUpti_MetricID metric, CUpti_MetricAttribute attrib, size_t ∗valueSize, void ∗value)

  Get a metric attribute.

- CUptiResult cuptiMetricGetIdFromName (CUdevice device, const char ∗metricName, CUpti_MetricID ∗metric)

  Find an metric by name.

- CUptiResult cuptiMetricGetNumEvents (CUpti_MetricID metric, uint32_t ∗numEvents)

  Get number of events required to calculate a metric.

- CUptiResult cuptiMetricGetValue (CUdevice device, CUpti_MetricID metric, size_t eventIdArraySizeBytes, CUpti_EventID ∗eventIdArray, size_t eventValueArraySizeBytes, uint64_t ∗eventValueArray, uint64_t timeDuration, CUpti_MetricValue ∗metricValue)

  Calculate the value for a metric.

Typedef Documentation

typedef uint32_t **CUpti_MetricID**

A metric provides a measure of some aspect of the device.

Enumeration Type Documentation

enum **CUpti_MetricAttribute**

Metric attributes describe properties of a metric. These attributes can be read using cuptiMetricGetAttribute.

Enumerator:

CUPTI_METRIC_ATTR_NAME Metric name. Value is a null terminated const c-string.

CUPTI_METRIC_ATTR_SHORT_DESCRIPTION Short description of metric. Value is a null terminated const c-string.

CUPTI_METRIC_ATTR_LONG_DESCRIPTION Long description of metric. Value is a null terminated const c-string.

CUPTI_METRIC_ATTR_CATEGORY Category of the metric. Value is of type CUpti_MetricCategory.

CUPTI_METRIC_ATTR_VALUE_KIND Value type of the metric. Value is of type CUpti_MetricValueKind.

enum **CUpti_MetricCategory**

Each metric is assigned to a category that represents the general type of the metric. A metric's category is accessed using cuptiMetricGetAttribute and the CUPTI_METRIC_ATTR_CATEGORY attribute.

Enumerator:

CUPTI_METRIC_CATEGORY_MEMORY A memory related metric.

CUPTI_METRIC_CATEGORY_INSTRUCTION An instruction related metric.

CUPTI_METRIC_CATEGORY_MULTIPROCESSOR A multiprocessor related metric.

CUPTI_METRIC_CATEGORY_CACHE A cache related metric.

CUPTI_METRIC_CATEGORY_TEXTURE A texture related metric.

enum **CUpti_MetricValueKind**

Metric values can be one of several different kinds. Corresponding to each kind is a member of the CUpti_MetricValue union. The metric value returned by cuptiMetricGetValue should be accessed using the appropriate member of that union based on its value kind.

Enumerator:

CUPTI_METRIC_VALUE_KIND_DOUBLE The metric value is a 64-bit double.

CUPTI_METRIC_VALUE_KIND_UINT64 The metric value is a 64-bit integer.

CUPTI_METRIC_VALUE_KIND_PERCENT The metric value is a percentage represented by a 64-bit double. For example, 57.5% is represented by the value 57.5.

CUPTI_METRIC_VALUE_KIND_THROUGHPUT The metric value is a throughput represented by a 64-bit integer. The unit for throughput values is bytes/second.

Function Documentation

CUptiResult cuptiDeviceEnumMetrics (CUdevice device, size_t * arraySizeBytes, **CUpti_MetricID** * metricArray)

Returns the metric IDs in `metricArray` for a device. The size of the `metricArray` buffer is given by *`arraySizeBytes`. The size of the `metricArray` buffer must be at least `numMetrics` * sizeof(CUpti_MetricID) or else all metric IDs will not be returned. The value returned in *`arraySizeBytes` contains the number of bytes returned in `metricArray`.

Parameters:

device The CUDA device

arraySizeBytes The size of `metricArray` in bytes, and returns the number of bytes written to `metricArray`

metricArray Returns the IDs of the metrics for the device

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `metricArray` are NULL

CUptiResult cuptiDeviceGetNumMetrics (CUdevice device, uint32_t ∗ numMetrics)

Returns the number of metrics available for a device.

Parameters:

device The CUDA device

numMetrics Returns the number of metrics available for the device

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_PARAMETER if `numMetrics` is NULL

CUptiResult cuptiEnumMetrics (size_t ∗ arraySizeBytes, **CUpti_MetricID** ∗ metricArray)

Returns the metric IDs in `metricArray` for all CUDA-capable devices. The size of the `metricArray` buffer is given by ∗`arraySizeBytes`. The size of the `metricArray` buffer must be at least `numMetrics` ∗ sizeof(CUpti_MetricID) or all metric IDs will not be returned. The value returned in ∗`arraySizeBytes` contains the number of bytes returned in `metricArray`.

Parameters:

arraySizeBytes The size of `metricArray` in bytes, and returns the number of bytes written to `metricArray`

metricArray Returns the IDs of the metrics

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `metricArray` are NULL

CUptiResult cuptiGetNumMetrics (uint32_t * numMetrics)

Returns the total number of metrics available on any CUDA-capable devices.

Parameters:

numMetrics Returns the number of metrics

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_INVALID_PARAMETER if `numMetrics` is NULL

CUptiResult cuptiMetricCreateEventGroupSets (CUcontext context, size_t metricIdArraySizeBytes, **CUpti_MetricID** * metricIdArray, **CUpti_EventGroupSets** ** eventGroupPasses)

For a set of metrics, get the grouping that indicates the number of passes and the event groups necessary to collect the events required for those metrics.

See also:

cuptiEventGroupSetsCreate for details on event group set creation.

Parameters:

context The context for event collection

metricIdArraySizeBytes Size of the metricIdArray in bytes

metricIdArray Array of metric IDs

eventGroupPasses Returns a CUpti_EventGroupSets object that indicates the number of passes required to collect the events and the events to collect on each pass

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_CONTEXT

CUPTI_ERROR_INVALID_METRIC_ID

CUPTI_ERROR_INVALID_PARAMETER if `metricIdArray` or `eventGroupPasses` is NULL

CUptiResult cuptiMetricEnumEvents (**CUpti_MetricID** metric, size_t ∗ eventIdArraySizeBytes, **CUpti_EventID** ∗ eventIdArray)

Gets the event IDs in `eventIdArray` required to calculate a `metric`. The size of the `eventIdArray` buffer is given by ∗`eventIdArraySizeBytes` and must be at least `numEvents` ∗ sizeof(CUpti_EventID) or all events will not be returned. The value returned in ∗`eventIdArraySizeBytes` contains the number of bytes returned in `eventIdArray`.

Parameters:

metric ID of the metric

eventIdArraySizeBytes The size of `eventIdArray` in bytes, and returns the number of bytes written to `eventIdArray`

eventIdArray Returns the IDs of the events required to calculate `metric`

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_METRIC_ID

CUPTI_ERROR_INVALID_PARAMETER if `eventIdArraySizeBytes` or `eventIdArray` are NULL.

CUptiResult cuptiMetricGetAttribute (**CUpti_MetricID** metric, **CUpti_MetricAttribute** attrib, size_t ∗ valueSize, void ∗ value)

Returns a metric attribute in ∗`value`. The size of the `value` buffer is given by ∗`valueSize`. The value returned in ∗`valueSize` contains the number of bytes returned in `value`.

If the attribute value is a c-string that is longer than ∗`valueSize`, then only the first ∗`valueSize` characters will be returned and there will be no terminating null byte.

Parameters:

metric ID of the metric

attrib The metric attribute to read

valueSize The size of the `value` buffer in bytes, and returns the number of bytes written to `value`

value Returns the attribute's value

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_METRIC_ID

CUPTI_ERROR_INVALID_PARAMETER if `valueSize` or `value` is NULL, or if `attrib` is not a metric attribute

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT For non-c-string attribute values, indicates that the `value` buffer is too small to hold the attribute value.

CUptiResult cuptiMetricGetIdFromName (CUdevice device, const char ∗ metricName, **CUpti_MetricID** ∗ metric)

Find a metric by name and return the metric ID in ∗`metric`.

Parameters:

device The CUDA device

metricName The name of metric to find

metric Returns the ID of the found metric or undefined if unable to find the metric

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

CUPTI_ERROR_INVALID_METRIC_NAME if unable to find a metric with name `metricName`. In this case ∗`metric` is undefined

CUPTI_ERROR_INVALID_PARAMETER if `metricName` or `metric` are NULL.

CUptiResult cuptiMetricGetNumEvents (**CUpti_MetricID** metric, uint32_t ∗ numEvents)

Returns the number of events in `numEvents` that are required to calculate a metric.

Parameters:

metric ID of the metric

numEvents Returns the number of events required for the metric

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_METRIC_ID

CUPTI_ERROR_INVALID_PARAMETER if `numEvents` is NULL

CUptiResult cuptiMetricGetValue (CUdevice device, **CUpti_MetricID** metric, size_t eventIdArraySizeBytes, **CUpti_EventID** * eventIdArray, size_t eventValueArraySizeBytes, uint64_t * eventValueArray, uint64_t timeDuration, **CUpti_MetricValue** * metricValue)

Use the events collected for a metric to calculate the metric value. Metric value calculation assumes that event values are normalized to represent all domain instances on a device. For the most accurate metric collection, the events required for the metric should be collected for all profiled domain instances. For example, to collect all instances of an event, set the CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES attribute on the group containing the event to 1. The normalized value for the event is then: (`sum_event_values` * `totalInstanceCount`) / `instanceCount`, where `sum_event_values` is the summation of the event values across all profiled domain instances, `totalInstanceCount` is obtained from querying CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT and `instanceCount` is obtained from querying CUPTI_EVENT_GROUP_ATTR_INSTANCE_COUNT (or CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT).

Parameters:

device The CUDA device that the metric is being calculated for

metric The metric ID

eventIdArraySizeBytes The size of `eventIdArray` in bytes

eventIdArray The event IDs required to calculate `metric`

eventValueArraySizeBytes The size of `eventValueArray` in bytes

eventValueArray The normalized event values required to calculate `metric`. The values must be order to match the order of events in `eventIdArray`

timeDuration The duration over which the events were collected, in ns

metricValue Returns the value for the metric

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_METRIC_ID

CUPTI_ERROR_INVALID_OPERATION

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT if the eventIdArray does not contain all the events needed for metric

CUPTI_ERROR_INVALID_EVENT_VALUE if any of the event values required for the metric is CUPTI_EVENT_OVERFLOW

CUPTI_ERROR_INVALID_PARAMETER if `metricValue`, `eventIdArray` or `eventValueArray` is NULL

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

Copyright

www.nvidia.com