CUDA Tools SDK CUPTI User's Guide

DA-05679-001_v01 | February 2011

Document Change History

Ver	Date	Resp	Reason for change
v01	2011/1/19	DG	Initial revision for CUDA Tools SDK 4.0

CUPTI

The *CUDA Profiling Tools Interface* (CUPTI) enables the creation of profiling and tracing tools that target CUDA applications. CUPTI provides two APIs, the *Callback API* and the *Event 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 Callback API

The CUPTI Callback API allows you to interject your own code at the entry and exit to each CUDA runtime and driver API call. Using the callback API, you 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. The following terminology is used by the callback API.

Callback ID: Each CUDA API function is given a unique ID so that you can identify it within your callback. 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`.

Callback Site: A location in a CUDA API function where your callback code is invoked. Currently there are two callback sites; one at CUDA API function entry and one at function exit.

Callback Domain: CUDA API functions are grouped into domains to make it easier to associate your callback functions with groups of related CUDA functions. There are currently two callback domains, as defined by `CUpti_CallbackDomain`; one for CUDA runtime functions and one for CUDA driver functions.

Subscriber: A subscriber is used to associate each of your callback functions with one or more CUDA API functions.

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 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 CUpti_CallbackData *↩
               cbInfo)
{
  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 15 both show how to use the callback API.

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 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.

The tables included in this guide list the events available for each device, as determined by the device's compute capability. You can also use the `cuptiDeviceEnumEventDomains` and `cuptiEventDomainEnumEvents` functions to enumerate the domains and events available on a device. The **cupti_query** sample described on page 15 shows how to use these functions.

Configuring and reading event counts requires the following steps. First 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.

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 15 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 *params)
{
  const CUpti_CallbackData *cbData =
               (CUpti_CallbackData *)params;

  if (cbData->callbackSite == CUPTI_API_ENTER) {
    cudaThreadSynchronize();
    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. Then event collection is start 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. 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

are counted for multiple, but not all, SMs. To get the most consistent results inspite of these factors, it is best to have 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 the device's first *Texture Processing Cluster* (TPC). Devices with compute capabiliity less than 1.3 have two SMs per TPC, and devices with compute capability 1.3 have three SMs per TPC.

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 half-warp's execution of a single 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 a separate memory transaction is issued for each thread in the half-warp, 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.

Event Name	Description	Type	Capability			
			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**

Event Name	Description	Type	Capability			
			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
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

Event Name	Description	Type	Capability			
			1.0	1.1	1.2	1.3
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
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 2.0 or greater implement two event domains, called *domain_a* and *domain_b*. Table 3 and Table 4 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
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
sm_cta_launched	Number of thread blocks launched	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
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
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

Event Name	Description	Type	Capability	
			2.0	2.1
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
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

Event Name	Description	Type	Capability	
			2.0	2.1
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
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**

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 Name	Description	Type	Capability	
			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

Event Name	Description	Type	Capability	
			2.0	2.1
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_-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
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**

Event Name	Description	Type	Capability	
			2.0	2.1
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

Samples

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

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

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

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 1

  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 1

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.

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_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_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 Callback API

Data Structures

- struct CUpti_CallbackData

  Data passed into a runtime or driver API callback function.

Typedefs

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

  Function type for an API callback.

- typedef uint32_t CUpti_CallbackId

  An ID for a driver or runtime API function.

- 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 }

  Callback domains.

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 function 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 **CUpti_CallbackData** *cbdata)

Function type for an API callback

Parameters:

userdata User data supplied at subscription of the callback

domain The domain of the callback

cbid The ID of the API function associated with this callback

cbdata Data passed to the callback.

typedef uint32_t **CUpti_CallbackId**

An ID for a driver or runtime API function. 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.

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.

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.

Function Documentation

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

Enable or disable all callbacks in all domains.

Note:

Thread-safety: function is thread safe for different subscribers, but each subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackEnabled(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 function ID.

Note:

Thread-safety: function is thread safe for different subscribers, but each subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackEnabled(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 API function

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: function is thread safe for different subscribers, but each 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 API function is enabled, and zero if not enabled.

Note:

Thread-safety: function is thread safe for different subscribers, but each subscriber must serialize access to cuptiGetCallbackState, cuptiEnableCallback, cuptiEnableDomain, and cuptiEnableAllDomains. For example, if cuptiGetCallbackEnabled(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 API function

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 API functions. data can also be provided.

Note:

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_OUT_OF_MEMORY

CUPTI_ERROR_NOT_INITIALIZED if unable to initialize CUPTI

CUPTI_ERROR_INVALID_PARAMETER if `subscriber` is NULL

CUPTI_ERROR_UNKNOWN if the subscriber limit is reached

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

Returns in `*domainTable` an array of size `*domainCount` of all the available callback domains. A callback domain is a set of callback points for a related group of API functions.

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 Reference

Data passed into a runtime or driver API callback function.

Data Fields

- CUpti_ApiCallbackSite callbackSite
- CUcontext context
- uint64_t contextUid
- uint64_t * correlationData
- 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 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 in any context.

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.

uint64_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. A UID of zero indicates that there is no context current to the thread, or that a context has not yet been attached to the thread during runtime-driver interop.

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.

const char* CUpti_CallbackData::functionName

Name of the runtime or driver API function which issued 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 Event API

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 }

  Device attributes.

- enum CUpti_EventAttribute {
  CUPTI_EVENT_ATTR_NAME = 0,
  CUPTI_EVENT_ATTR_SHORT_DESCRIPTION = 1,
  CUPTI_EVENT_ATTR_LONG_DESCRIPTION = 2 }

  Event attributes.

- enum CUpti_EventDomainAttribute {
  CUPTI_EVENT_DOMAIN_ATTR_NAME = 0,
  CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT = 1,
  CUPTI_EVENT_DOMAIN_MAX_EVENTS = 2 }

  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 }

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, uint64_t *value)

  Read a device 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 cuptiEventDomainEnumEvents (CUdevice device, CUpti_EventDomainID eventDomain, size_t *arraySizeBytes, CUpti_EventID *eventArray)

  Get the events in a domain.

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

  Read an event domain attribute.

- CUptiResult cuptiEventDomainGetNumEvents (CUdevice device,

CUpti_EventDomainID eventDomain, uint32_t *numevents)

Get number of events in a domain.

- CUptiResult cuptiEventGetAttribute (CUdevice device, 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, uint64_t *value)

  Read an event group attribute.

- CUptiResult cuptiEventGroupReadAllEvents (CUpti_EventGroup eventGroup, CUpti_ReadEventFlags flags, size_t *bufferSizeBytes, uint64_t *counterDataBuffer, size_t *arraySizeBytes, CUpti_EventID *eventArray, size_t *numCountersRead)

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

- CUptiResult cuptiEventGroupReadEvent (CUpti_EventGroup eventGroup, CUpti_ReadEventFlags flags, CUpti_EventID event, size_t *bufferSizeBytes, uint64_t *counterData)

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, uint64_t value)

  Write an event group attribute.

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 attriutes can be read using cuptiDeviceGetAttribute.

Enumerator:

CUPTI_DEVICE_ATTR_MAX_EVENT_ID Number of event IDs for a device. Value is an integer

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

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

enum CUpti_EventDomainAttribute

Event domain attributes. These attributes can be read using 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. Value is an integer

CUPTI_EVENT_DOMAIN_MAX_EVENTS Maximum number of events available in the domain. Value is an integer

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 an integer.

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, uint64_t * value)

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

Parameters:

device The CUDA device

attrib The attribute to read

value Returns the value of the attribute

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

CUPTI_ERROR_INVALID_DEVICE

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

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 cuptiEventDomainEnumEvents (CUdevice device, **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:

device The CUDA device

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_DEVICE

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID

CUPTI_ERROR_INVALID_PARAMETER if `arraySizeBytes` or `eventArray` are NULL

CUptiResult cuptiEventDomainGetAttribute (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 cuptiEventDomainGetNumEvents (CUdevice device, **CUpti_EventDomainID** eventDomain, uint32_t * numevents)

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

Parameters:

device The CUDA device

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_DEVICE

CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID

CUPTI_ERROR_INVALID_PARAMETER if `numevents` is NULL

CUptiResult cuptiEventGetAttribute (CUdevice device, **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:

device The CUDA device

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_DEVICE

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)

Finds a returns an event by name 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, uint64_t * value)

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

Parameters:

eventGroup The event group

attrib The attribute to read

value Returns the value of the attribute

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

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

CUptiResult cuptiEventGroupReadAllEvents (**CUpti_EventGroup** eventGroup, **CUpti_ReadEventFlags** flags, size_t * bufferSizeBytes, uint64_t * counterDataBuffer, size_t * arraySizeBytes, **CUpti_EventID** * eventArray, size_t * numCountersRead)

Read the values for all the events in an event group. The counter values are returned in the `counterData` buffer. `bufferSizeBytes` indicates the size of the `counterData` buffer. 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 domain 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 domain.

The data format returned in `counterData` is:

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

The event order in `counterData` is returned in `eventArray`. The size of `eventArray` is specified in `arraySizeBytes`. The size should be at least (sizeof(CUpti_EventID) * number of events in group).

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

bufferSizeBytes The size of `counterData` in bytes, and returns the number of bytes written to `counterData`

counterData Returns the event counter values

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

numCountersRead Returns the number of event counts 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`, `bufferSizeBytes`, `counterData`, `arraySizeBytes`, `eventArray` or `numCountersRead` is NULL

CUptiResult cuptiEventGroupReadEvent (**CUpti_EventGroup** eventGroup, **CUpti_ReadEventFlags** flags, **CUpti_EventID** event, size_t * bufferSizeBytes, uint64_t * counterData)

Read the value for an event in an event group. The counter value is returned in the `counterData` buffer. `bufferSizeBytes` indicates the size of the `counterData` buffer. The buffer must be at least sizeof(uint64) if CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES is not set on the domain 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 domain.

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

bufferSizeBytes The size of `counterData` in bytes, and returns the number of bytes written to `counterData`

counterData Returns the event counter values

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`, `bufferSizeBytes` or `counterData` 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, uint64_t value)

Write an event group attribute.

Parameters:

eventGroup The event group

attrib The attribute to write

value The attribute value to write

Return values:

CUPTI_SUCCESS

CUPTI_ERROR_NOT_INITIALIZED

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

Notice

Trademarks

Copyright