CUPTI

DA-05679-001 _v6.5 | August 2014

User's Guide

WHAT'S NEW

CUPTI contains a number of enhancements as part of the CUDA Toolkit 6.5 release.

- Instruction classification is done for source-correlated Instruction Execution activity `CUpti_ActivityInstructionExecution`. See `CUpti_ActivityInstructionClass` for instruction classes.
- Two new device attributes are added to the activity `CUpti_DeviceAttribute`:
    - `CUPTI_DEVICE_ATTR_FLOP_SP_PER_CYCLE` gives peak single precision flop per cycle for the GPU.
    - `CUPTI_DEVICE_ATTR_FLOP_DP_PER_CYCLE` gives peak double precision flop per cycle for the GPU.
- Two new metric properties are added:
    - `CUPTI_METRIC_PROPERTY_FLOP_SP_PER_CYCLE` gives peak single precision flop per cycle for the GPU.
    - `CUPTI_METRIC_PROPERTY_FLOP_DP_PER_CYCLE` gives peak double precision flop per cycle for the GPU.
- Activity record `CUpti_ActivityGlobalAccess` for source level global access information has been deprecated and replaced by new activity record `CUpti_ActivityGlobalAccess2`. New record additionally gives information needed to map SASS assembly instructions to CUDA C source code. And it also provides ideal L2 transactions count based on the access pattern.
- Activity record `CUpti_ActivityBranch` for source level branch information has been deprecated and replaced by new activity record `CUpti_ActivityBranch2`. New record additionally gives information needed to map SASS assembly instructions to CUDA C source code.
- Sample `sass_source_map` is added to demonstrate the mapping of SASS assembly instructions to CUDA C source code.
- Default event collection mode is changed to Kernel (`CUPTI_EVENT_COLLECTION_MODE_KERNEL`) from Continuous (`CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS`). Also Continuous mode is now supported only on Tesla devices.
- Profiling results might be inconsistent when auto boost is enabled. Profiler tries to disable auto boost by default, it might fail to do so in some conditions, but profiling will continue. A new API `cuptiGetAutoBoostState` is added to query the auto boost state of the device. This API returns error `CUPTI_ERROR_NOT_SUPPORTED` on devices that don't support auto boost. Note that auto boost is supported only on certain Tesla devices from the Kepler+ family.
- Activity record `CUpti_ActivityKernel2` for kernel execution has been deprecated and replaced by new activity record `CUpti_ActivityKernel3`. New

record additionally gives information about Global Partitioned Cache Configuration requested and executed. The new fields apply for devices with 5.2 Compute Capability.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. USAGE

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.

1.1. 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.

1.2. CUPTI Initialization

CUPTI initialization occurs lazily the first time you invoke any CUPTI function. For the Activity, 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). See the CUPTI Activity API section for more information on CUPTI initialization requirements for the activity API.

1.3. CUPTI Activity API

The CUPTI Activity API allows you to asynchronously 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_async sample for an example.

Activity Buffer

An activity buffer is used to transfer one or more activity records from CUPTI to the client. CUPTI fills activity buffers with activity records as the corresponding activities occur on the CPU and GPU. The CUPTI client is responsible for providing empty activity buffers as necessary to ensure that no records are dropped.

An *asynchronous* buffering API is implemented by `cuptiActivityRegisterCallbacks` and `cuptiActivityFlushAll`.

It is not required that the activity API be initalized before CUDA, but if the activity API is not initialized before CUDA some activity records may not be collected. You can force initialization of the activity API by enabling one or more activity kinds using `cuptiActivityEnable` or `cuptiActivityEnableContext`, as shown in the `initTrace` function of the activity_trace_async sample. Some activity kinds cannot be directly enabled, see the API documentation for for `CUpti_ActivityKind` for details. Functions `cuptiActivityEnable` and `cuptiActivityEnableContext` will return `CUPTI_ERROR_NOT_COMPATIBLE` if the requested activity kind cannot be enabled.

The activity buffer API uses callbacks to request and return buffers of activity records. To use the asynchronous buffering API you must first register two callbacks using `cuptiActivityRegisterCallbacks`. One of these callbacks will be invoked whenever CUPTI needs an empty activity buffer. The other callback is used to deliver a buffer containing one or more activity records to the client. To minimize profiling overhead the client should return as quickly as possible from these callbacks. Function `cuptiActivityFlushAll` can be used to force CUPTI to deliver any activity buffers that contain completed activity records. Functions `cuptiActivityGetAttribute` and `cuptiActivitySetAttribute` can be used to read and write attributes that control how the buffering API behaves. See the API documentation for more information.

The activity_trace_async sample shows how to use the activity buffer API to collect a trace of CPU and GPU activity for a simple application.

1.3.1. Context Activity Record

In 6.0 the context activity record, `CUpti_ActivityContext`, was changed in a manner that introduced a new field into the structure. This new field was introduced in a way that preserves backward compatibility with any persisted versions of this structure.

The 32-bit `computeApiKind` field was replaced with two 16 bit fields, `computeApiKind` and `defaultStreamId`. Because all valid `computeApiKind` values fit within 16 bits, and because all supported CUDA platforms are little-endian, persisted context record data read with the new structure will have the correct value for `computeApiKind` and have a value of zero for `defaultStreamId`. The CUPTI client is responsible for versioning the persisted context data to recognize when the `defaultStreamId` field is valid.

1.3.2. Legacy Activity Records

In CUPTI 5.5 the `CUpti_ActivityKernel2` structure replaced `CUpti_ActivityKernel` as the activity record used for the CUPTI_ACTIVITY_KIND_KERNEL and CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL activity kinds. The `CUpti_ActivityKernel` definition is retained in CUPTI to enable newer versions of CUPTI to work with presisted activity record data.

The CUPTI client is responsible for versioning the persisted activity record data to recognize when the persisted data is stored using `CUpti_ActivityKernel` or `CUpti_ActivityKernel2`.

1.4. 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. Unless explicitly stated, it is not supported to call any CUDA runtime or driver API from within a callback function. Doing so may cause the application to hang.

1.4.1. 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`.

It is legal to call `cudaThreadSynchronize()`, `cudaDeviceSynchronize()`, `cudaStreamSynchronize()`, `cuCtxSynchronize()`, and `cuStreamSynchronize()` from within a driver or runtime API callback function.

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 `cbdata` 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 the samples page both show how to use the callback API for the driver and runtime API domains.

1.4.2. 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`.

Note that, APIs `cuptiActivityFlush` and `cuptiActivityFlushAll` will result in deadlock when called from stream destroy starting callback identified using callback ID `CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING`.

1.4.3. 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`.

1.4.4. NVIDIA Tools Extension Callbacks

Using the callback API with the `CUPTI_CB_DOMAIN_NVTX` domain, you can associate a callback function with NVIDIA Tools Extension (NVTX) API functions. When an NVTX function is 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_NvtxData`.

The NVTX library has its own convention for discovering the profiling library that will provide the implementation of the NVTX callbacks. To receive callbacks you must set the NVTX environment variables appropriately so that when the application calls an NVTX function, your profiling library recieve the callbacks. The following code sequence shows a typical initialization sequence to enable NVTX callbacks and activity records.

```
/* Set env so CUPTI-based profiling library loads on first nvtx call. */
char *inj32_path = "/path/to/32-bit/version/of/cupti/based/profiling/library";
char *inj64_path = "/path/to/64-bit/version/of/cupti/based/profiling/library";
setenv("NVTX_INJECTION32_PATH", inj32_path, 1);
setenv("NVTX_INJECTION64_PATH", inj64_path, 1);
```

The following code shows a typical sequence used to associate a callback function with one or more NVTX 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_NVTX);
```

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 NVTX functions. Using this code sequence will cause `my_callback` to be called once each time any of the NVTX functions are invoked. CUPTI callback API functions `cuptiEnableCallback` and `cuptiEnableAllDomains` can also be used to associate NVTX 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_NvtxData *nvtxInfo = (CUpti_NvtxData *)cbdata;
  MyDataStruct *my_data = (MyDataStruct *)userdata;

  if ((domain == CUPTI_CB_DOMAIN_NVTX) &&
      (cbid == NVTX_CBID_CORE_NameOsThreadA))  {
    nvtxNameOsThreadA_params *params = (nvtxNameOsThreadA_params *)nvtxInfo->
             functionParams;
    ...
  }
  ...
```

In your callback function, you use the `CUpti_CallbackDomain` and `CUpti_CallbackID` parameters to determine which NVTX API function invocation is causing this callback. In the example above, we are checking for the `nvtxNameOsThreadA` function. The `cbdata` parameter holds a structure of useful information that can be used within the callback. In this case, we use the `functionParams` member to access the parameters that were passed to `nvtxNameOsThreadA`. To access the parameters we first cast `functionParams` to a structure type corresponding to the `nvtxNameOsThreadA` function. These parameter structures are contained in `generated_nvtx_meta.h`.

1.5. 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 occurrence 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`.

You can determine the events available on a device using the `cuptiDeviceEnumEventDomains` and `cuptiEventDomainEnumEvents` functions. The **cupti_query** sample described on the samples page 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 can gather the events across multiple executions of the application or you can enable kernel replay. If you enable kernel replay using `cuptiEnableKernelReplayMode` you will be able to enabled any number of event groups and all the contained events will be collect.

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 the samples page shows how to use these functions to create, enable, and disable event groups, and how to read event counts.

1.5.1. 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) {
    cudaDeviceSynchronize();
    cuptiSetEventCollectionMode(cbInfo->context,
                                CUPTI_EVENT_COLLECTION_MODE_KERNEL);
    cuptiEventGroupEnable(eventGroup);
  }

  if (cbData->callbackSite == CUPTI_API_EXIT) {
    cudaDeviceSynchronize();
    cuptiEventGroupReadEvent(eventGroup,
                             CUPTI_EVENT_READ_FLAG_NONE,
                             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), `cudaDeviceSynchronize` 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 `cudaDeviceSynchronize` is used to cause the CPU thread to wait for the kernel to finish execution. Finally, the event counts are read with `cuptiEventGroupReadEvent`.

1.5.2. 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.

1.6. 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 Property
Each metric is calculated from input values. These input values can be events or properties of the device or system. The available properties are defined by `CUpti_MetricPropertyID`.

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 metric's 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 the samples page shows how to use this function. You can also enumerate all the CUPTI metrics available on any device using the `cuptiEnumMetrics` function.

CUPTI provides two functions for calculating a metric value. `cuptiMetricGetValue2` can be used to calculate a metric value when the device is not available. All required event values and metric properties must be provided by the caller. `cuptiMetricGetValue` can be used to calculate a metric value when the device is available (as a CUdevice object). All required event values must be provided by the caller but CUPTI will determine the appropriate property values from the CUdevice object.

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. When creating event groups in this manner it is important to use the result of `cuptiMetricGetRequiredEventGroupSets` to properly group together events that must be collected in the same pass to ensure proper metric calculation.

Alternatively, you can use the `cuptiMetricCreateEventGroupSets` function to automatically create the event group(s) required for metric's events. When using this function events will be grouped as required to most accurately calculate the metric, as a result it is not necessary to use `cuptiMetricGetRequiredEventGroupSets`.

If you are using `cuptiMetricGetValue2` the you must also collect the required metric property values using `cuptiMetricEnumProperties`.

Collect event counts as described in the CUPTI Event API section, and then use either `cuptiMetricGetValue` or `cuptiMetricGetValue2` to calculate the metric value from the collected event and property values. The **callback_metric** sample described on the samples page shows how to use the functions to calculate event values and calculate a metric using `cuptiMetricGetValue`. 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` or `cuptiMetricGetValue2` 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 the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 1 Capability 1.x Metrics

Metric Name	Description	Scope
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Single-context
gld_efficiency	Ratio of requested global memory load transactions to actual global memory load transactions expressed as percentage	Single-context
gst_efficiency	Ratio of requested global memory store transactions to actual global memory store transactions expressed as percentage	Single-context
gld_requested_throughput	Requested global memory load throughput	Single-context
gst_requested_throughput	Requested global memory store throughput	Single-context

Metric Reference - Compute Capability 2.x

Devices with compute capability between 2.0, inclusive, and 3.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 2 Capability 2.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context
alu_fu_utilization	The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions on a scale of 0 to 10	Multi-context

Metric Name	Description	Scope
atomic_replay_overhead	Average number of replays due to atomic and reduction bank conflicts for each instruction executed	Multi-context
atomic_throughput	Global memory atomic and reduction throughput	Multi-context
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context
cf_issued	Number of issued control-flow instructions	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context

Metric Name	Description	Scope
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_cache_replay_overhead	Average number of replays due to global memory cache misses for each instruction executed	Single-context
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context

Metric Name	Description	Scope
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
ipc_instance	Instructions executed per cycle for a single multiprocessor	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l1_cache_global_hit_rate	Hit rate in L1 cache for global loads	Single-context
l1_cache_local_hit_rate	Hit rate in L1 cache for local loads and stores	Single-context
l1_shared_utilization	The utilization level of the L1/shared memory relative to peak utilization on a scale of 0 to 10	Single-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Sinlge-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_l1_read_hit_rate	Hit rate at L2 cache for all read requests from L1 cache	Sinlge-context
l2_l1_read_throughput	Memory read throughput seen at L2 cache for read requests from L1 cache	Single-context
l2_l1_read_transactions	Memory read transactions seen at L2 cache for all read requests from L1 cache	Single-context

Metric Name	Description	Scope
l2_l1_write_throughput	Memory write throughput seen at L2 cache for write requests from L1 cache	Single-context
l2_l1_write_transactions	Memory write transactions seen at L2 cache for all write requests from L1 cache	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_texture_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_texure_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Sinlge-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context

Metric Name	Description	Scope
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage	Single-context
local_replay_overhead	Average number of replays due to local memory accesses for each instruction executed	Single-context
local_store_throughput	Local memory store throughput	Single-context
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_replay_overhead	Average number of replays due to shared memory conflicts for each instruction executed	Single-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU	Single-context
sm_efficiency_instance	The percentage of time at least one warp is active on a specific multiprocessor	Single-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being	Multi-context

Metric Name	Description	Scope
	available or fully utilized, or because too many requests of a given type are outstanding	
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a __syncthreads() call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
sysmem_read_throughput	System memory read throughput	Single-context
sysmem_read_transactions	System memory read transactions	Single-context
sysmem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10	Single-context
sysmem_write_throughput	System memory write throughput	Single-context
sysmem_write_transactions	System memory write transactions	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp	Multi-context

Metric Name	Description	Scope
	supported on a multiprocessor expressed as percentage	

Metric Reference - Compute Capability 3.x

Devices with compute capability between 3.0, inclusive, and 4.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 3 Capability 3.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context
alu_fu_utilization	The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions on a scale of 0 to 10	Multi-context
atomic_replay_overhead	Average number of replays due to atomic and reduction bank conflicts for each instruction executed	Multi-context
atomic_throughput	Global memory atomic and reduction throughput	Multi-context
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage. This is available for compute capability 3.0.	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context

Metric Name	Description	Scope
cf_issued	Number of issued control-flow instructions	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions expressed as percentage	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_cache_replay_overhead	Average number of replays due to global memory cache misses for each instruction executed	Multi-context
global_replay_overhead	Average number of replays due to global memory cache misses	Multi-context

Metric Name	Description	Scope
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context

Metric Name	Description	Scope
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
ipc_instance	Instructions executed per cycle for a single multiprocessor	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l1_cache_global_hit_rate	Hit rate in L1 cache for global loads	Single-context
l1_cache_local_hit_rate	Hit rate in L1 cache for local loads and stores	Single-context
l1_shared_utilization	The utilization level of the L1/shared memory relative to peak utilization on a scale of 0 to 10. This is available for compute capability 3.0 and 3.5.	Single-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Sinlge-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_l1_read_hit_rate	Hit rate at L2 cache for all read requests from L1 cache. This is available for compute capability 3.0 and 3.5.	Sinlge-context
l2_l1_read_throughput	Memory read throughput seen at L2 cache for read requests from L1 cache. This is available for compute capability 3.0 and 3.5.	Single-context
l2_l1_read_transactions	Memory read transactions seen at L2 cache for all read requests from L1 cache	Single-context
l2_l1_write_throughput	Memory write throughput seen at L2 cache for write requests from L1 cache	Single-context
l2_l1_write_transactions	Memory write transactions seen at L2 cache for all write requests from L1 cache	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context

Metric Name	Description	Scope
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_texture_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_texture_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Sinlge-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage. This is available for compute capability 3.0 and 3.5.	Single-context
local_replay_overhead	Average number of replays due to local memory accesses for each instruction executed	Multi-context
local_store_throughput	Local memory store throughput	Single-context

Metric Name	Description	Scope
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
nc_cache_global_hit_rate	Hit rate in non coherent cache for global loads	Single-context
nc_gld_efficiency	Ratio of requested non coherent global memory load throughput to required non coherent global memory load throughput expressed as percentage	Single-context
nc_gld_requested_throughput	Requested throughput for global memory loaded via non-coherent cache	Multi-context
nc_gld_throughput	Non coherent global memory load throughput	Single-context
nc_l2_read_throughput	Memory read throughput for non coherent global read requests seen at L2 cache	Single-context
nc_l2_read_transactions	Memory read transactions seen at L2 cache for non coherent global read requests	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_replay_overhead	Average number of replays due to shared memory conflicts for each instruction executed	Multi-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context

Metric Name	Description	Scope
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU	Single-context
sm_efficiency_instance	The percentage of time at least one warp is active on a specific multiprocessor	Single-context
stall_compute	Percentage of stalls occurring because a compute operation cannot be performed due to the required resources not being available	Multi-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being available or fully utilized, or because too many requests of a given type are outstanding	Multi-context
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_imc	Percentage of stalls occurring because of immediate constant cache miss	Multi-context
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a __syncthreads() call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
sysmem_read_throughput	System memory read throughput. This is available for compute capability 3.0 and 3.5.	Single-context
sysmem_read_transactions	System memory read transactions. This is available for compute capability 3.0 and 3.5.	Single-context
sysmem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10. This is available for compute capability 3.0 and 3.5.	Single-context

Metric Name	Description	Scope
sysmem_write_throughput	System memory write throughput. This is available for compute capability 3.0 and 3.5.	Single-context
sysmem_write_transactions	System memory write transactions. This is available for compute capability 3.0 and 3.5.	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context
warp_nonpred_execution_efficiency	Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context

Metric Reference - Compute Capability 5.x

Devices with compute capability greater than or equal to 5.0 implement the metrics shown in the following table. A scope value of single-context indicates that the metric can only be accurately collected when a single context (CUDA or graphic) is executing on the GPU. A scope value of multi-context indicates that the metric can be accurately collected when multiple contexts are executing on the GPU.

Table 4 Capability 5.x Metrics

Metric Name	Description	Scope
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor	Multi-context

Metric Name	Description	Scope
atomic_transactions	Global memory atomic and reduction transactions	Multi-context
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction	Multi-context
branch_efficiency	Ratio of non-divergent branches to total branches expressed as percentage	Multi-context
cf_executed	Number of executed control-flow instructions	Multi-context
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10	Multi-context
cf_issued	Number of issued control-flow instructions	Multi-context
double_precision_fu_utilization	The utilization level of the multiprocessor function units that execute double-precision floating-point instructions and integer instructions on a scale of 0 to 10	Multi-context
dram_read_throughput	Device memory read throughput	Single-context
dram_read_transactions	Device memory read transactions	Single-context
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10	Single-context
dram_write_throughput	Device memory write throughput	Single-context
dram_write_transactions	Device memory write transactions	Single-context
ecc_throughput	ECC throughput from L2 to DRAM	Single-context
ecc_transactions	Number of ECC transactions between L2 and DRAM	Single-context
eligible_warps_per_cycle	Average number of warps that are eligible to issue per active cycle	Multi-context
flop_count_dp	Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context

Metric Name	Description	Scope
flop_count_dp_add	Number of double-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_dp_fma	Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_dp_mul	Number of double-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp	Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, multiply-accumulate and special). Each multiply-accumulate operation contributes 2 to the count.	Multi-context
flop_count_sp_add	Number of single-precision floating-point add operations executed by non-predicated threads	Multi-context
flop_count_sp_fma	Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.	Multi-context
flop_count_sp_mul	Number of single-precision floating-point multiply operations executed by non-predicated threads	Multi-context
flop_count_sp_special	Number of single-precision floating-point special operations executed by non-predicated threads	Multi-context
flop_dp_efficiency	Ratio of achieved to peak double-precision floating-point operations	Multi-context
flop_sp_efficiency	Ratio of achieved to peak single-precision floating-point operations	Multi-context
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage	Single-context

Metric Name	Description	Scope
gld_requested_throughput	Requested global memory load throughput	Multi-context
gld_throughput	Global memory load throughput	Single-context
gld_transactions	Number of global memory load transactions	Single-context
gld_transactions_per_request	Average number of global memory load transactions performed for each global memory load	Single-context
global_hit_rate	Hit rate for global loads	Single-context
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage	Single-context
gst_requested_throughput	Requested global memory store throughput	Multi-context
gst_throughput	Global memory store throughput	Single-context
gst_transactions	Number of global memory store transactions	Single-context
gst_transactions_per_request	Average number of global memory store transactions performed for each global memory store	Single-context
inst_bit_convert	Number of bit-conversion instructions executed by non-predicated threads	Multi-context
inst_compute_ld_st	Number of compute load/store instructions executed by non-predicated threads	Multi-context
inst_control	Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)	Multi-context
inst_executed	The number of instructions executed	Multi-context
inst_fp_32	Number of single-precision floating-point instructions executed by non-predicated threads (arithmetric, compare, etc.)	Multi-context
inst_fp_64	Number of double-precision floating-point instructions executed by non-predicated threads (arithmetric, compare, etc.)	Multi-context
inst_integer	Number of integer instructions executed by non-predicated threads	Multi-context

Metric Name	Description	Scope
inst_inter_thread_communication	Number of inter-thread communication instructions executed by non-predicated threads	Multi-context
inst_issued	The number of instructions issued	Multi-context
inst_misc	Number of miscellaneous instructions executed by non-predicated threads	Multi-context
inst_per_warp	Average number of instructions executed by each warp	Multi-context
inst_replay_overhead	Average number of replays for each instruction executed	Multi-context
ipc	Instructions executed per cycle	Multi-context
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles	Multi-context
issue_slots	The number of issue slots used	Multi-context
issued_ipc	Instructions issued per cycle	Multi-context
l2_atomic_throughput	Memory read throughput seen at L2 cache for atomic and reduction requests	Multi-context
l2_atomic_transactions	Memory read transactions seen at L2 cache for atomic and reduction requests	Single-context
l2_read_throughput	Memory read throughput seen at L2 cache for all read requests	Single-context
l2_read_transactions	Memory read transactions seen at L2 cache for all read requests	Single-context
l2_tex_read_hit_rate	Hit rate at L2 cache for all read requests from texture cache	Single-context
l2_tex_read_throughput	Memory read throughput seen at L2 cache for read requests from the texture cache	Sinlge-context
l2_tex_read_transactions	Memory read transactions seen at L2 cache for read requests from the texture cache	Single-context
l2_tex_write_hit_rate	Hit Rate at L2 cache for all write requests from texture cache	Single-context
l2_tex_write_throughput	Memory write throughput seen at L2 cache for write requests from the texture cache	Sinlge-context

Metric Name	Description	Scope
l2_tex_write_transactions	Memory write transactions seen at L2 cache for write requests from the texture cache	Single-context
l2_utilization	The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10	Single-context
l2_write_throughput	Memory write throughput seen at L2 cache for all write requests	Single-context
l2_write_transactions	Memory write transactions seen at L2 cache for all write requests	Single-context
ldst_executed	Number of executed load and store instructions	Multi-context
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10	Multi-context
ldst_issued	Number of issued load and store instructions	Multi-context
local_hit_rate	Hit rate for local loads and stores	Single-context
local_load_throughput	Local memory load throughput	Single-context
local_load_transactions	Number of local memory load transactions	Single-context
local_load_transactions_per_request	Average number of local memory load transactions performed for each local memory load	Single-context
local_memory_overhead	Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage	Single-context
local_store_throughput	Local memory store throughput	Single-context
local_store_transactions	Number of local memory store transactions	Single-context
local_store_transactions_per_request	Average number of local memory store transactions performed for each local memory store	Single-context
shared_efficiency	Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage	Single-context
shared_load_throughput	Shared memory load throughput	Single-context

Metric Name	Description	Scope
shared_load_transactions	Number of shared memory load transactions	Single-context
shared_load_transactions_per_request	Average number of shared memory load transactions performed for each shared memory load	Single-context
shared_store_throughput	Shared memory store throughput	Single-context
shared_store_transactions	Number of shared memory store transactions	Single-context
shared_store_transactions_per_request	Average number of shared memory store transactions performed for each shared memory store	Single-context
shared_utilization	The utilization level of the shared memory relative to peak utilization on a scale of 0 to 10	Single-context
single_precision_fu_utilization	The utilization level of the multiprocessor function units that execute single-precision floating-point instructions and integer instructions on a scale of 0 to 10	Multi-context
sm_efficiency	The percentage of time at least one warp is active on a multiprocessor	Single-context
special_fu_utilization	The utilization level of the multiprocessor function units that execute sin, cos, ex2, popc, flo, and similar instructions on a scale of 0 to 10	Multi-context
stall_compute	Percentage of stalls occurring because a compute operation cannot be performed due to the required resources not being available	Multi-context
stall_data_request	Percentage of stalls occurring because a memory operation cannot be performed due to the required resources not being available or fully utilized, or because too many requests of a given type are outstanding	Multi-context
stall_exec_dependency	Percentage of stalls occurring because an input required by the instruction is not yet available	Multi-context
stall_imc	Percentage of stalls occurring because of immediate constant cache miss	Multi-context

Metric Name	Description	Scope
stall_inst_fetch	Percentage of stalls occurring because the next assembly instruction has not yet been fetched	Multi-context
stall_other	Percentage of stalls occurring due to miscellaneous reasons	Multi-context
stall_sync	Percentage of stalls occurring because the warp is blocked at a __syncthreads() call	Multi-context
stall_texture	Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests	Multi-context
sysmem_read_throughput	System memory read throughput	Single-context
sysmem_read_transactions	System memory read transactions	Single-context
sysmem_utilization	The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10	Single-context
sysmem_write_throughput	System memory write throughput	Single-context
sysmem_write_transactions	System memory write transactions	Single-context
tex_cache_hit_rate	Texture cache hit rate	Single-context
tex_cache_throughput	Texture cache throughput	Single-context
tex_cache_transactions	Texture cache read transactions	Single-context
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10	Multi-context
tex_utilization	The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10	Single-context
warp_execution_efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage	Multi-context
warp_nonpred_execution_efficiency	Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor	Multi-context

1.7. Samples

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

activity_trace_async
: This sample shows how to collect a trace of CPU and GPU activity using the new asynchronous activity buffer APIs.

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.

sass_source_map
: This sample shows how to generate CUpti_ActivityInstructionExecution records and how to map SASS assembly instructions to CUDA C source.

unified_memory
: This sample shows how to collect various counters like page faults and page transfers for unified memory.

Chapter 2. MODULES

Here is a list of all modules:

- CUPTI Version
- CUPTI Result Codes
- CUPTI Activity API
- CUPTI Callback API
- CUPTI Event API
- CUPTI Metric API

2.1. CUPTI Version

Function and macro to determine the CUPTI version.

CUptiResult cuptiGetVersion (uint32_t *version)

Get the CUPTI API version.

Parameters

version
Returns the version

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_INVALID_PARAMETER

  if `version` is NULL

Description

Return the API version in `*version`.

See also:

CUPTI_API_VERSION

#define CUPTI_API_VERSION 7

The API version for this implementation of CUPTI.

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 v3 : CUDA Toolkit 5.0 v4 : CUDA Toolkit 5.5 v5 : CUDA Toolkit 6.0 v6 : CUDA Toolkit 6.5 v7 : CUDA Toolkit 6.5(with sm_52 support)

2.2. CUPTI Result Codes

Error and result codes returned by CUPTI functions.

enum CUptiResult

CUPTI result codes.

Error and result codes returned by CUPTI functions.

Values

CUPTI_SUCCESS = 0
  No error.
CUPTI_ERROR_INVALID_PARAMETER = 1
  One or more of the parameters is invalid.
CUPTI_ERROR_INVALID_DEVICE = 2
  The device does not correspond to a valid CUDA device.
CUPTI_ERROR_INVALID_CONTEXT = 3
  The context is NULL or not valid.
CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID = 4
  The event domain id is invalid.
CUPTI_ERROR_INVALID_EVENT_ID = 5
  The event id is invalid.
CUPTI_ERROR_INVALID_EVENT_NAME = 6
  The event name is invalid.
CUPTI_ERROR_INVALID_OPERATION = 7

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

CUPTI_ERROR_OUT_OF_MEMORY = 8

Unable to allocate enough memory to perform the requested operation.

CUPTI_ERROR_HARDWARE = 9

An error occurred on the performance monitoring hardware.

CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT = 10

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

CUPTI_ERROR_API_NOT_IMPLEMENTED = 11

API is not implemented.

CUPTI_ERROR_MAX_LIMIT_REACHED = 12

The maximum limit is reached.

CUPTI_ERROR_NOT_READY = 13

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

CUPTI_ERROR_NOT_COMPATIBLE = 14

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

CUPTI_ERROR_NOT_INITIALIZED = 15

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

CUPTI_ERROR_INVALID_METRIC_ID = 16

The metric id is invalid.

CUPTI_ERROR_INVALID_METRIC_NAME = 17

The metric name is invalid.

CUPTI_ERROR_QUEUE_EMPTY = 18

The queue is empty.

CUPTI_ERROR_INVALID_HANDLE = 19

Invalid handle (internal?).

CUPTI_ERROR_INVALID_STREAM = 20

Invalid stream.

CUPTI_ERROR_INVALID_KIND = 21

Invalid kind.

CUPTI_ERROR_INVALID_EVENT_VALUE = 22

Invalid event value.

CUPTI_ERROR_DISABLED = 23

CUPTI is disabled due to conflicts with other enabled profilers

CUPTI_ERROR_INVALID_MODULE = 24

Invalid module.

CUPTI_ERROR_INVALID_METRIC_VALUE = 25

Invalid metric value.

CUPTI_ERROR_HARDWARE_BUSY = 26

The performance monitoring hardware is in use by other client.

CUPTI_ERROR_NOT_SUPPORTED = 27

The attempted operation is not supported on the current system or device.

CUPTI_ERROR_UNKNOWN = 999

An unknown internal error has occurred.

CUPTI_ERROR_FORCE_INT = 0x7fffffff

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

Get the descriptive string for a CUptiResult.

Parameters

result
The result to get the string for

str
Returns the string

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_INVALID_PARAMETER

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

Description

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

Thread-safety: this function is thread safe.

2.3. CUPTI Activity API

Functions, types, and enums that implement the CUPTI Activity API.

struct CUpti_Activity
The base activity record.

struct CUpti_ActivityAPI
The activity record for a driver or runtime API invocation.

struct CUpti_ActivityAutoBoostState
Device auto boost state structure.

struct CUpti_ActivityBranch
The activity record for source level result branch. (deprecated).

struct CUpti_ActivityBranch2
The activity record for source level result branch.

struct CUpti_ActivityCdpKernel
The activity record for CDP (CUDA Dynamic Parallelism) kernel.

struct CUpti_ActivityContext
The activity record for a context.

struct CUpti_ActivityDevice
The activity record for a device.

struct CUpti_ActivityDeviceAttribute
The activity record for a device attribute.

struct CUpti_ActivityEnvironment
The activity record for CUPTI environmental data.

struct CUpti_ActivityEvent
The activity record for a CUPTI event.

struct CUpti_ActivityEventInstance
The activity record for a CUPTI event with instance information.

struct CUpti_ActivityFunction
The activity record for global/device functions.

struct CUpti_ActivityGlobalAccess
The activity record for source-level global access. (deprecated).

struct CUpti_ActivityGlobalAccess2
The activity record for source-level global access.

struct CUpti_ActivityInstructionExecution
The activity record for source-level sass/source line-by-line correlation.

struct CUpti_ActivityKernel
The activity record for kernel. (deprecated).

struct CUpti_ActivityKernel2
The activity record for kernel. (deprecated).

struct CUpti_ActivityKernel3
The activity record for a kernel (CUDA 6.5(with sm_52 support) onwards).

struct CUpti_ActivityMarker
The activity record providing a marker which is an instantaneous point in time.

struct CUpti_ActivityMarkerData
The activity record providing detailed information for a marker.

struct CUpti_ActivityMemcpy
The activity record for memory copies.

struct CUpti_ActivityMemcpy2
The activity record for peer-to-peer memory copies.

struct CUpti_ActivityMemset
The activity record for memset.

struct CUpti_ActivityMetric
The activity record for a CUPTI metric.

struct CUpti_ActivityMetricInstance
The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (CUPTI_ACTIVITY_KIND_METRIC_INSTANCE). 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. This activity record should be used when metric domain instance information needs to be associated with the metric.

struct CUpti_ActivityModule

The activity record for a CUDA module.

struct CUpti_ActivityName

The activity record providing a name.

union CUpti_ActivityObjectKindId

Identifiers for object kinds as specified by CUpti_ActivityObjectKind.

struct CUpti_ActivityOverhead

The activity record for CUPTI and driver overheads.

struct CUpti_ActivityPreemption

The activity record for a preemption of a CDP kernel.

struct CUpti_ActivitySharedAccess

The activity record for source-level shared access.

struct CUpti_ActivitySourceLocator

The activity record for source locator.

struct CUpti_ActivityUnifiedMemoryCounter

The activity record for Unified Memory counters.

struct CUpti_ActivityUnifiedMemoryCounterConfig

Unified Memory counters configuration structure.

enum CUpti_ActivityAttribute

Activity attributes.

These attributes are used to control the behavior of the activity API.

Values

CUPTI_ACTIVITY_ATTR_DEVICE_BUFFER_SIZE = 0

The device memory size (in bytes) reserved for storing profiling data for non-CDP operations for each buffer on a context. The value is a size_t.Having larger buffer size means less flush operations but consumes more device memory. Having smaller buffer size increases the risk of dropping timestamps for kernel records if too many kernels are launched/replayed at one time. This value only applies to new buffer allocations.Set this value before initializing CUDA or before creating a context to ensure it is considered for the following allocations.The default value is 4194304 (4MB).Note: The actual amount of device memory per buffer reserved by CUPTI might be larger.

CUPTI_ACTIVITY_ATTR_DEVICE_BUFFER_SIZE_CDP = 1

The device memory size (in bytes) reserved for storing profiling data for CDP operations for each buffer on a context. The value is a size_t.Having larger buffer size means less flush operations but consumes more device memory. This value only applies to new allocations.Set this value before initializing CUDA or before creating a context to ensure it is considered for the following allocations.The default value is 8388608 (8MB).Note: The actual amount of device memory per context reserved by CUPTI might be larger.

CUPTI_ACTIVITY_ATTR_DEVICE_BUFFER_POOL_LIMIT = 2

The maximum number of memory buffers per context. The value is a size_t.Buffers can be reused by the context. Increasing this value reduces the times CUPTI needs to flush the buffers. Setting this value will not modify the number of memory buffers currently stored.Set this value before initializing CUDA to ensure the limit is not exceeded.The default value is 4.

CUPTI_ACTIVITY_ATTR_DEVICE_BUFFER_FORCE_INT = 0x7fffffff

enum CUpti_ActivityComputeApiKind

The kind of a compute API.

Values

CUPTI_ACTIVITY_COMPUTE_API_UNKNOWN = 0

The compute API is not known.

CUPTI_ACTIVITY_COMPUTE_API_CUDA = 1

The compute APIs are for CUDA.

CUPTI_ACTIVITY_COMPUTE_API_CUDA_MPS = 2

The compute APIs are for CUDA running in MPS (Multi-Process Service) environment.

CUPTI_ACTIVITY_COMPUTE_API_FORCE_INT = 0x7fffffff

enum CUpti_ActivityEnvironmentKind

The kind of environment data. Used to indicate what type of data is being reported by an environment activity record.

Values

CUPTI_ACTIVITY_ENVIRONMENT_UNKNOWN = 0

Unknown data.

CUPTI_ACTIVITY_ENVIRONMENT_SPEED = 1
The environment data is related to speed.

CUPTI_ACTIVITY_ENVIRONMENT_TEMPERATURE = 2
The environment data is related to temperature.

CUPTI_ACTIVITY_ENVIRONMENT_POWER = 3
The environment data is related to power.

CUPTI_ACTIVITY_ENVIRONMENT_COOLING = 4
The environment data is related to cooling.

CUPTI_ACTIVITY_ENVIRONMENT_COUNT

CUPTI_ACTIVITY_ENVIRONMENT_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityFlag

Flags associated with activity records.

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.

Values

CUPTI_ACTIVITY_FLAG_NONE = 0
Indicates the activity record has no flags.

CUPTI_ACTIVITY_FLAG_DEVICE_CONCURRENT_KERNELS = 1<<0
Indicates the activity represents a device that supports concurrent kernel execution. Valid for CUPTI_ACTIVITY_KIND_DEVICE.

CUPTI_ACTIVITY_FLAG_DEVICE_ATTRIBUTE_CUDEVICE = 1<<0
Indicates if the activity represents a CUdevice_attribute value or a CUpti_DeviceAttribute value. Valid for CUPTI_ACTIVITY_KIND_DEVICE_ATTRIBUTE.

CUPTI_ACTIVITY_FLAG_MEMCPY_ASYNC = 1<<0
Indicates the activity represents an asynchronous memcpy operation. Valid for CUPTI_ACTIVITY_KIND_MEMCPY.

CUPTI_ACTIVITY_FLAG_MARKER_INSTANTANEOUS = 1<<0
Indicates the activity represents an instantaneous marker. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_START = 1<<1
Indicates the activity represents a region start marker. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_END = 1<<2
Indicates the activity represents a region end marker. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_COLOR_NONE = 1<<0
Indicates the activity represents a marker that does not specify a color. Valid for CUPTI_ACTIVITY_KIND_MARKER_DATA.

CUPTI_ACTIVITY_FLAG_MARKER_COLOR_ARGB = 1<<1
Indicates the activity represents a marker that specifies a color in alpha-red-green-blue format. Valid for CUPTI_ACTIVITY_KIND_MARKER_DATA.

CUPTI_ACTIVITY_FLAG_GLOBAL_ACCESS_KIND_SIZE_MASK = 0xFF<<0
The number of bytes requested by each thread Valid for CUpti_ActivityGlobalAccess2.

CUPTI_ACTIVITY_FLAG_GLOBAL_ACCESS_KIND_LOAD = 1<<8
If bit in this flag is set, the access was load, else it is a store access. Valid for CUpti_ActivityGlobalAccess2.

CUPTI_ACTIVITY_FLAG_GLOBAL_ACCESS_KIND_CACHED = 1<<9
If this bit in flag is set, the load access was cached else it is uncached. Valid for CUpti_ActivityGlobalAccess2.

CUPTI_ACTIVITY_FLAG_METRIC_OVERFLOWED = 1<<0
If this bit in flag is set, the metric value overflowed. Valid for CUpti_ActivityMetric and CUpti_ActivityMetricInstance.

CUPTI_ACTIVITY_FLAG_METRIC_VALUE_INVALID = 1<<1
If this bit in flag is set, the metric value couldn't be calculated. This occurs when a value(s) required to calculate the metric is missing. Valid for CUpti_ActivityMetric and CUpti_ActivityMetricInstance.

CUPTI_ACTIVITY_FLAG_INSTRUCTION_VALUE_INVALID = 1<<0
If this bit in flag is set, the source level metric value couldn't be calculated. This occurs when a value(s) required to calculate the source level metric cannot be evaluated. Valid for CUpti_ActivityInstructionExecution.

CUPTI_ACTIVITY_FLAG_INSTRUCTION_CLASS_MASK = 0xFF<<1
The mask for the instruction class, CUpti_ActivityInstructionClass Valid for CUpti_ActivityInstructionExecution.

CUPTI_ACTIVITY_FLAG_FLUSH_FORCED = 1<<0
When calling cuptiActivityFlushAll, this flag can be set to force CUPTI to flush all records in the buffer, whether finished or not

CUPTI_ACTIVITY_FLAG_SHARED_ACCESS_KIND_SIZE_MASK = 0xFF<<0
The number of bytes requested by each thread Valid for CUpti_ActivitySharedAccess.

CUPTI_ACTIVITY_FLAG_SHARED_ACCESS_KIND_LOAD = 1<<8
If bit in this flag is set, the access was load, else it is a store access. Valid for CUpti_ActivitySharedAccess.

CUPTI_ACTIVITY_FLAG_FORCE_INT = 0x7fffffff

enum CUpti_ActivityInstructionClass

SASS instruction classification.

The sass instruction are broadly divided into different class. Each enum represents a classification.

Values

CUPTI_ACTIVITY_INSTRUCTION_CLASS_UNKNOWN = 0
The instruction class is not known.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_FP_32 = 1
Represents a 32 bit floating point operation.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_FP_64 = 2
Represents a 64 bit floating point operation.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_INTEGER = 3
Represents an integer operation.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_BIT_CONVERSION = 4
Represents a bit conversion operation.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_CONTROL_FLOW = 5
Represents a control flow instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_GLOBAL = 6
Represents a global load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_SHARED = 7
Represents a shared load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_LOCAL = 8
Represents a local load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_GENERIC = 9
Represents a generic load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_SURFACE = 10
Represents a surface load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_CONSTANT = 11
Represents a constant load instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_TEXTURE = 12
Represents a texture load-store instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_GLOBAL_ATOMIC = 13
Represents a global atomic instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_SHARED_ATOMIC = 14
Represents a shared atomic instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_SURFACE_ATOMIC = 15
Represents a surface atomic instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_INTER_THREAD_COMMUNICATION = 16
Represents a inter-thread communication instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_BARRIER = 17
Represents a barrier instruction.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_MISCELLANEOUS = 18
Represents some miscellaneous instructions which do not fit in the above classification.

CUPTI_ACTIVITY_INSTRUCTION_CLASS_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityKind

The kinds of activity records.

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_ActivityContext

CUpti_ActivityDevice

CUpti_ActivityDeviceAttribute

CUpti_ActivityEvent

CUpti_ActivityEventInstance

CUpti_ActivityKernel

CUpti_ActivityKernel2

CUpti_ActivityKernel3

CUpti_ActivityCdpKernel

CUpti_ActivityPreemption

CUpti_ActivityMemcpy

CUpti_ActivityMemcpy2

CUpti_ActivityMemset

CUpti_ActivityMetric

CUpti_ActivityMetricInstance

CUpti_ActivityName

CUpti_ActivityMarker

CUpti_ActivityMarkerData

CUpti_ActivitySourceLocator

CUpti_ActivityGlobalAccess

CUpti_ActivityGlobalAccess2

CUpti_ActivityBranch

CUpti_ActivityBranch2

CUpti_ActivityOverhead

CUpti_ActivityEnvironment

CUpti_ActivityInstructionExecution

CUpti_ActivityUnifiedMemoryCounter

CUpti_ActivityFunction

CUpti_ActivityModule

CUpti_ActivitySharedAccess

Values

CUPTI_ACTIVITY_KIND_INVALID = 0
The activity record is invalid.
CUPTI_ACTIVITY_KIND_MEMCPY = 1
A host<->host, host<->device, or device<->device memory copy. The corresponding activity record structure is CUpti_ActivityMemcpy.
CUPTI_ACTIVITY_KIND_MEMSET = 2
A memory set executing on the GPU. The corresponding activity record structure is CUpti_ActivityMemset.
CUPTI_ACTIVITY_KIND_KERNEL = 3
A kernel executing on the GPU. The corresponding activity record structure is CUpti_ActivityKernel3.
CUPTI_ACTIVITY_KIND_DRIVER = 4
A CUDA driver API function execution. The corresponding activity record structure is CUpti_ActivityAPI.
CUPTI_ACTIVITY_KIND_RUNTIME = 5
A CUDA runtime API function execution. The corresponding activity record structure is CUpti_ActivityAPI.
CUPTI_ACTIVITY_KIND_EVENT = 6
An event value. The corresponding activity record structure is CUpti_ActivityEvent.
CUPTI_ACTIVITY_KIND_METRIC = 7
A metric value. The corresponding activity record structure is CUpti_ActivityMetric.
CUPTI_ACTIVITY_KIND_DEVICE = 8
Information about a device. The corresponding activity record structure is CUpti_ActivityDevice.
CUPTI_ACTIVITY_KIND_CONTEXT = 9
Information about a context. The corresponding activity record structure is CUpti_ActivityContext.
CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL = 10

A (potentially concurrent) kernel executing on the GPU. The corresponding activity record structure is CUpti_ActivityKernel3.

CUPTI_ACTIVITY_KIND_NAME = 11
Thread, device, context, etc. name. The corresponding activity record structure is CUpti_ActivityName.

CUPTI_ACTIVITY_KIND_MARKER = 12
Instantaneous, start, or end marker. The corresponding activity record structure is CUpti_ActivityMarker.

CUPTI_ACTIVITY_KIND_MARKER_DATA = 13
Extended, optional, data about a marker. The corresponding activity record structure is CUpti_ActivityMarkerData.

CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR = 14
Source information about source level result. The corresponding activity record structure is CUpti_ActivitySourceLocator.

CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS = 15
Results for source-level global acccess. The corresponding activity record structure is CUpti_ActivityGlobalAccess2.

CUPTI_ACTIVITY_KIND_BRANCH = 16
Results for source-level branch. The corresponding activity record structure is CUpti_ActivityBranch2.

CUPTI_ACTIVITY_KIND_OVERHEAD = 17
Overhead activity records. The corresponding activity record structure is CUpti_ActivityOverhead.

CUPTI_ACTIVITY_KIND_CDP_KERNEL = 18
A CDP (CUDA Dynamic Parallel) kernel executing on the GPU. The corresponding activity record structure is CUpti_ActivityCdpKernel. This activity can not be directly enabled or disabled. It is enabled and disabled through concurrent kernel activity CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL

CUPTI_ACTIVITY_KIND_PREEMPTION = 19
Preemption activity record indicating a preemption of a CDP (CUDA Dynamic Parallel) kernel executing on the GPU. The corresponding activity record structure is CUpti_ActivityPreemption.

CUPTI_ACTIVITY_KIND_ENVIRONMENT = 20
Environment activity records indicating power, clock, thermal, etc. levels of the GPU. The corresponding activity record structure is CUpti_ActivityEnvironment.

CUPTI_ACTIVITY_KIND_EVENT_INSTANCE = 21
An event value associated with a specific event domain instance. The corresponding activity record structure is CUpti_ActivityEventInstance.

CUPTI_ACTIVITY_KIND_MEMCPY2 = 22
A peer to peer memory copy. The corresponding activity record structure is CUpti_ActivityMemcpy2.

CUPTI_ACTIVITY_KIND_METRIC_INSTANCE = 23

A metric value associated with a specific metric domain instance. The corresponding activity record structure is CUpti_ActivityMetricInstance.

CUPTI_ACTIVITY_KIND_INSTRUCTION_EXECUTION = 24
SASS/Source line-by-line correlation record. The corresponding activity record structure is CUpti_ActivityInstructionExecution.

CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER = 25
Unified Memory counter record. The corresponding activity record structure is CUpti_ActivityUnifiedMemoryCounter.

CUPTI_ACTIVITY_KIND_FUNCTION = 26
Device global/function record. The corresponding activity record structure is CUpti_ActivityFunction.

CUPTI_ACTIVITY_KIND_MODULE = 27
CUDA Module record. The corresponding activity record structure is CUpti_ActivityModule.

CUPTI_ACTIVITY_KIND_DEVICE_ATTRIBUTE = 28
A device attribute value. The corresponding activity record structure is CUpti_ActivityDeviceAttribute.

CUPTI_ACTIVITY_KIND_SHARED_ACCESS = 29
Results for source-level shared acccess. The corresponding activity record structure is CUpti_ActivitySharedAccess.

CUPTI_ACTIVITY_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityMemcpyKind

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

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

Values

CUPTI_ACTIVITY_MEMCPY_KIND_UNKNOWN = 0
The memory copy kind is not known.

CUPTI_ACTIVITY_MEMCPY_KIND_HTOD = 1
A host to device memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_DTOH = 2
A device to host memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_HTOA = 3
A host to device array memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOH = 4
A device array to host memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOA = 5
A device array to device array memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_ATOD = 6
A device array to device memory copy.

CUPTI_ACTIVITY_MEMCPY_KIND_DTOA = 7
A device to device array memory copy.
CUPTI_ACTIVITY_MEMCPY_KIND_DTOD = 8
A device to device memory copy on the same device.
CUPTI_ACTIVITY_MEMCPY_KIND_HTOH = 9
A host to host memory copy.
CUPTI_ACTIVITY_MEMCPY_KIND_PTOP = 10
A peer to peer memory copy across different devices.
CUPTI_ACTIVITY_MEMCPY_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityMemoryKind

The kinds of memory accessed by a memory copy.

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

Values

CUPTI_ACTIVITY_MEMORY_KIND_UNKNOWN = 0
The source or destination memory kind is unknown.
CUPTI_ACTIVITY_MEMORY_KIND_PAGEABLE = 1
The source or destination memory is pageable.
CUPTI_ACTIVITY_MEMORY_KIND_PINNED = 2
The source or destination memory is pinned.
CUPTI_ACTIVITY_MEMORY_KIND_DEVICE = 3
The source or destination memory is on the device.
CUPTI_ACTIVITY_MEMORY_KIND_ARRAY = 4
The source or destination memory is an array.
CUPTI_ACTIVITY_MEMORY_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityObjectKind

The kinds of activity objects.

See also:

CUpti_ActivityObjectKindId

Values

CUPTI_ACTIVITY_OBJECT_UNKNOWN = 0
The object kind is not known.
CUPTI_ACTIVITY_OBJECT_PROCESS = 1
A process.
CUPTI_ACTIVITY_OBJECT_THREAD = 2
A thread.

CUPTI_ACTIVITY_OBJECT_DEVICE = 3
A device.
CUPTI_ACTIVITY_OBJECT_CONTEXT = 4
A context.
CUPTI_ACTIVITY_OBJECT_STREAM = 5
A stream.
CUPTI_ACTIVITY_OBJECT_FORCE_INT = 0x7fffffff

enum CUpti_ActivityOverheadKind

The kinds of activity overhead.

Values

CUPTI_ACTIVITY_OVERHEAD_UNKNOWN = 0
The overhead kind is not known.
CUPTI_ACTIVITY_OVERHEAD_DRIVER_COMPILER = 1
Compiler(JIT) overhead.
CUPTI_ACTIVITY_OVERHEAD_CUPTI_BUFFER_FLUSH = 1<<16
Activity buffer flush overhead.
CUPTI_ACTIVITY_OVERHEAD_CUPTI_INSTRUMENTATION = 2<<16
CUPTI instrumentation overhead.
CUPTI_ACTIVITY_OVERHEAD_CUPTI_RESOURCE = 3<<16
CUPTI resource creation and destruction overhead.
CUPTI_ACTIVITY_OVERHEAD_FORCE_INT = 0x7fffffff

enum CUpti_ActivityPartitionedGlobalCacheConfig

Partitioned global caching option.

Values

CUPTI_ACTIVITY_PARTITIONED_GLOBAL_CACHE_CONFIG_UNKNOWN = 0
Partitioned global cache config unknown.
CUPTI_ACTIVITY_PARTITIONED_GLOBAL_CACHE_CONFIG_NOT_SUPPORTED = 1
Partitioned global cache not supported.
CUPTI_ACTIVITY_PARTITIONED_GLOBAL_CACHE_CONFIG_OFF = 2
Partitioned global cache config off.
CUPTI_ACTIVITY_PARTITIONED_GLOBAL_CACHE_CONFIG_ON = 3
Partitioned global cache config on.
CUPTI_ACTIVITY_PARTITIONED_GLOBAL_CACHE_CONFIG_FORCE_INT = 0x7fffffff

enum CUpti_ActivityPreemptionKind

The kind of a preemption activity.

Values

CUPTI_ACTIVITY_PREEMPTION_KIND_UNKNOWN = 0
The preemption kind is not known.
CUPTI_ACTIVITY_PREEMPTION_KIND_SAVE = 1
Preemption to save CDP block.
CUPTI_ACTIVITY_PREEMPTION_KIND_RESTORE = 2
Preemption to restore CDP block.
CUPTI_ACTIVITY_PREEMPTION_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityUnifiedMemoryCounterKind

Kind of the Unified Memory counter.

Many activities are associated with Unified Memory mechanism; among them are tranfer from host to device, device to host, page fault at host side.

Values

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_UNKNOWN = 0
The unified memory counter kind is not known.
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_HTOD = 1
Number of bytes transfered from host to device
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH = 2
Number of bytes transfered from device to host
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT = 3
Number of CPU page faults
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_COUNT
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityUnifiedMemoryCounterScope

Scope of the unified memory counter.

Values

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_SCOPE_UNKNOWN = 0
The unified memory counter scope is not known.

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_SCOPE_PROCESS_SINGLE_DEVICE = 1
Collect unified memory counter for single process on one device
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_SCOPE_PROCESS_ALL_DEVICES = 2
Collect unified memory counter for single process across all devices
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_SCOPE_COUNT
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_SCOPE_FORCE_INT = 0x7fffffff

enum CUpti_EnvironmentClocksThrottleReason

Reasons for clock throttling.

The possible reasons that a clock can be throttled. There can be more than one reason that a clock is being throttled so these types can be combined by bitwise OR. These are used in the clocksThrottleReason field in the Environment Activity Record.

Values

CUPTI_CLOCKS_THROTTLE_REASON_GPU_IDLE = 0x00000001
Nothing is running on the GPU and the clocks are dropping to idle state.
CUPTI_CLOCKS_THROTTLE_REASON_USER_DEFINED_CLOCKS = 0x00000002
The GPU clocks are limited by a user specified limit.
CUPTI_CLOCKS_THROTTLE_REASON_SW_POWER_CAP = 0x00000004
A software power scaling algorithm is reducing the clocks below requested clocks.
CUPTI_CLOCKS_THROTTLE_REASON_HW_SLOWDOWN = 0x00000008
Hardware slowdown to reduce the clock by a factor of two or more is engaged. This is an indicator of one of the following: 1) Temperature is too high, 2) External power brake assertion is being triggered (e.g. by the system power supply), 3) Change in power state.
CUPTI_CLOCKS_THROTTLE_REASON_UNKNOWN = 0x80000000
Some unspecified factor is reducing the clocks.
CUPTI_CLOCKS_THROTTLE_REASON_UNSUPPORTED = 0x40000000
Throttle reason is not supported for this GPU.
CUPTI_CLOCKS_THROTTLE_REASON_NONE = 0x00000000
No clock throttling.
CUPTI_CLOCKS_THROTTLE_REASON_FORCE_INT = 0x7fffffff

typedef (*CUpti_BuffersCallbackCompleteFunc) (CUcontext context, uint32_t streamId, uint8_t* buffer, size_t size, size_t validSize)

Function type for callback used by CUPTI to return a buffer of activity records.

This callback function returns to the CUPTI client a buffer containing activity records. The buffer contains `validSize` bytes of activity records which should be read using cuptiActivityGetNextRecord. The number of dropped records can be read using cuptiActivityGetNumDroppedRecords. After this call CUPTI relinquished ownership of the buffer and will not use it anymore. The client may return the buffer to CUPTI using the CUpti_BuffersCallbackRequestFunc callback. Note: CUDA 6.0 onwards, all buffers returned by this callback are global buffers i.e. there is no context/stream specific buffer. User needs to parse the global buffer to extract the context/stream specific activity records.

typedef (*CUpti_BuffersCallbackRequestFunc) (uint8_t* *buffer, size_t* size, size_t* maxNumRecords)

Function type for callback used by CUPTI to request an empty buffer for storing activity records.

This callback function signals the CUPTI client that an activity buffer is needed by CUPTI. The activity buffer is used by CUPTI to store activity records. The callback function can decline the request by setting `*buffer` to NULL. In this case CUPTI may drop activity records.

CUptiResult cuptiActivityConfigureUnifiedMemoryCounter (CUpti_ActivityUnifiedMemoryCounterConfig *config, uint32_t count)

Set Unified Memory Counter configuration.

Parameters

config

A pointer to CUpti_ActivityUnifiedMemoryCounterConfig structures containing Unified Memory counter configuration.

count

Number of Unified Memory counter configuration structures

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_PARAMETER

  if `config` is NULL or any parameter in the `config` structures is not a valid value
- CUPTI_ERROR_NOT_SUPPORTED

  Indicates that the system/device does not support the unified memory counters

CUptiResult cuptiActivityDisable (CUpti_ActivityKind kind)

Disable collection of a specific kind of activity record.

Parameters

kind
The kind of activity record to stop collecting

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_KIND

  if the activity kind is not supported

Description

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.

CUptiResult cuptiActivityDisableContext (CUcontext context, CUpti_ActivityKind kind)

Disable collection of a specific kind of activity record for a context.

Parameters

context
The context for which activity is to be disabled
kind
The kind of activity record to stop collecting

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_KIND

  if the activity kind is not supported

Description

Disable collection of a specific kind of activity record for a context. This setting done by this API will supersede the global settings for activity records. Multiple kinds can be enabled by calling this function multiple times.

CUptiResult cuptiActivityEnable (CUpti_ActivityKind kind)

Enable collection of a specific kind of activity record.

Parameters

kind
The kind of activity record to collect

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_NOT_COMPATIBLE

  if the activity kind cannot be enabled
- CUPTI_ERROR_INVALID_KIND

  if the activity kind is not supported

Description

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.

CUptiResult cuptiActivityEnableContext (CUcontext context, CUpti_ActivityKind kind)

Enable collection of a specific kind of activity record for a context.

Parameters

context
The context for which activity is to be enabled

kind
The kind of activity record to collect

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_NOT_COMPATIBLE

  if the activity kind cannot be enabled
- CUPTI_ERROR_INVALID_KIND

  if the activity kind is not supported

Description

Enable collection of a specific kind of activity record for a context. This setting done by this API will supersede the global settings for activity records enabled by cuptiActivityEnable. Multiple kinds can be enabled by calling this function multiple times.

CUptiResult cuptiActivityFlush (CUcontext context, uint32_t streamId, uint32_t flag)

Wait for all activity records are delivered via the completion callback.

Parameters

context
A valid CUcontext or NULL.

streamId
The stream ID.

flag
The flag can be set to indicate a forced flush. See CUpti_ActivityFlag

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_CUPTI_ERROR_INVALID_OPERATION

  if not preceeded by a successful call to cuptiActivityRegisterCallbacks
- CUPTI_ERROR_UNKNOWN

  an internal error occurred

Description

This function does not return until all activity records associated with the specified context/stream are returned to the CUPTI client using the callback registered in cuptiActivityRegisterCallbacks. To ensure that all activity records are complete, the requested stream(s), if any, are synchronized.

If `context` is NULL, the global activity records (i.e. those not associated with a particular stream) are flushed (in this case no streams are synchonized). If `context` is a valid CUcontext and `streamId` is 0, the buffers of all streams of this context are flushed. Otherwise, the buffers of the specified stream in this context is flushed.

Before calling this function, the buffer handling callback api must be activated by calling cuptiActivityRegisterCallbacks.

DEPRECATED This method is deprecated CONTEXT and STREAMID will be ignored. Use cuptiActivityFlushAll to flush all data.

CUptiResult cuptiActivityFlushAll (uint32_t flag)

Wait for all activity records are delivered via the completion callback.

Parameters

flag
The flag can be set to indicate a forced flush. See CUpti_ActivityFlag

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_OPERATION

  if not preceeded by a successful call to cuptiActivityRegisterCallbacks
- CUPTI_ERROR_UNKNOWN

  an internal error occurred

Description

This function does not return until all activity records associated with all contexts/ streams (and the global buffers not associated with any stream) are returned to the CUPTI client using the callback registered in cuptiActivityRegisterCallbacks. To ensure that all activity records are complete, the requested stream(s), if any, are synchronized.

Before calling this function, the buffer handling callback api must be activated by calling cuptiActivityRegisterCallbacks.

CUptiResult cuptiActivityGetAttribute (CUpti_ActivityAttribute attr, size_t *valueSize, void *value)

Read an activity API attribute.

Parameters

attr
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

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_PARAMETER

  if `valueSize` or `value` is NULL, or if `attr` is not an activity attribute
- CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT

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

Description

Read an activity API attribute and return it in `*value`.

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

Iterate over the activity records in a buffer.

Parameters

buffer
The buffer containing activity records

validBufferSizeBytes
The number of valid bytes in the buffer.

record
Inputs the previous record returned by cuptiActivityGetNextRecord and returns the next activity record from the buffer. If input value is NULL, returns the first activity record in the buffer. Records of kind CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL may contain invalid (0) timestamps, indicating that no timing information could be collected for lack of device memory.

Returns

- 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.

Description

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 or by receiving a CUpti_BuffersCallbackCompleteFunc callback.

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);
```

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

Get the number of activity records that were dropped of insufficient buffer space.

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.

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_PARAMETER

  if `dropped` is NULL

Description

Get the number of records that were dropped because of insufficient buffer space. The dropped count includes records that could not be recorded because CUPTI did not have activity buffer space available for the record (because the CUpti_BuffersCallbackRequestFunc callback did not return an empty buffer of sufficient size) and also CDP records that could not be record because the device-size buffer was full (size is controlled by the CUPTI_ACTIVITY_ATTR_DEVICE_BUFFER_SIZE_CDP attribute). The dropped count maintained for the queue is reset to zero when this function is called.

CUptiResult cuptiActivityRegisterCallbacks (CUpti_BuffersCallbackRequestFunc funcBufferRequested, CUpti_BuffersCallbackCompleteFunc funcBufferCompleted)

Registers callback functions with CUPTI for activity buffer handling.

Parameters

funcBufferRequested
callback which is invoked when an empty buffer is requested by CUPTI

funcBufferCompleted
callback which is invoked when a buffer containing activity records is available from CUPTI

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if either `funcBufferRequested` or `funcBufferCompleted` is NULL

Description

This function registers two callback functions to be used in asynchronous buffer handling. If registered, activity record buffers are handled using asynchronous requested/completed callbacks from CUPTI.

Registering these callbacks prevents the client from using CUPTI's blocking enqueue/dequeue functions.

CUptiResult cuptiActivitySetAttribute (CUpti_ActivityAttribute attr, size_t *valueSize, void *value)

Write an activity API attribute.

Parameters

attr
The attribute to write

valueSize
The size, in bytes, of the value

value

The attribute value to write

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_PARAMETER

  if `valueSize` or `value` is NULL, or if `attr` is not an activity attribute
- CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT

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

Description

Write an activity API attribute.

CUptiResult cuptiGetAutoBoostState (CUcontext context, CUpti_ActivityAutoBoostState *state)

Get auto boost state.

Parameters

context

A valid CUcontext.

state

A pointer to CUpti_ActivityAutoBoostState structure which contains the current state and the id of the process that has requested the current state

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `CUcontext` or `state` is NULL
- CUPTI_ERROR_NOT_SUPPORTED

  Indicates that the device does not support auto boost
- CUPTI_ERROR_UNKNOWN

  an internal error occurred

Description

The profiling results can be inconsistent in case auto boost is enabled. CUPTI tries to disable auto boost while profiling. It can fail to disable in cases where user does not have the permissions or CUDA_AUTO_BOOST env variable is set. The function can be used to query whether auto boost is enabled.

CUptiResult cuptiGetContextId (CUcontext context, uint32_t *contextId)

Get the ID of a context.

Parameters

context
The context
contextId
Returns a process-unique ID for the context

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_CONTEXT

  The context is NULL or not valid.
- CUPTI_ERROR_INVALID_PARAMETER

  if `contextId` is NULL

Description

Get the ID of a context.

CUptiResult cuptiGetDeviceId (CUcontext context, uint32_t *deviceId)

Get the ID of a device.

Parameters

context
The context, or NULL to indicate the current context.
deviceId
Returns the ID of the device that is current for the calling thread.

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_DEVICE

  if unable to get device ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `deviceId` is NULL

Description

If `context` is NULL, returns the ID of the device that contains the currently active context. If `context` is non-NULL, returns the ID of the device which contains that context. Operates in a similar manner to cudaGetDevice() or cuCtxGetDevice() but may be called from within callback functions.

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

Get the ID of a stream.

Parameters

context
: If non-NULL then the stream is checked to ensure that it belongs to this context. Typically this parameter should be null.

stream
: The stream

streamId
: Returns a context-unique ID for the stream

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_STREAM

  if unable to get stream ID, or if `context` is non-NULL and `stream` does not belong to the context
- CUPTI_ERROR_INVALID_PARAMETER

  if `streamId` is NULL

Description

Get the ID of a stream. The stream ID is unique within a context (i.e. all streams within a context will have unique stream IDs).

See also:

cuptiActivityEnqueueBuffer

cuptiActivityDequeueBuffer

CUptiResult cuptiGetTimestamp (uint64_t *timestamp)

Get the CUPTI timestamp.

Parameters

timestamp
Returns the CUPTI timestamp

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `timestamp` is NULL

Description

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

#define CUPTI_AUTO_BOOST_INVALID_CLIENT_PID 0

An invalid/unknown process id.

#define CUPTI_CORRELATION_ID_UNKNOWN 0

An invalid/unknown correlation ID. A correlation ID of this value indicates that there is no correlation for the activity record.

#define CUPTI_GRID_ID_UNKNOWN 0LL

An invalid/unknown grid ID.

#define CUPTI_SOURCE_LOCATOR_ID_UNKNOWN 0

The source-locator ID that indicates an unknown source location. There is not an actual CUpti_ActivitySourceLocator object corresponding to this value.

#define CUPTI_TIMESTAMP_UNKNOWN 0LL

An invalid/unknown timestamp for a start, end, queued, submitted, or completed time.

2.4. CUPTI Callback API

Functions, types, and enums that implement the CUPTI Callback API.

struct CUpti_CallbackData

Data passed into a runtime or driver API callback function.

struct CUpti_ModuleResourceData

Module data passed into a resource callback function.

struct CUpti_NvtxData

Data passed into a NVTX callback function.

struct CUpti_ResourceData

Data passed into a resource callback function.

struct CUpti_SynchronizeData

Data passed into a synchronize callback function.

enum CUpti_ApiCallbackSite

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

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

Values

CUPTI_API_ENTER = 0
    The callback is at the entry of the API call.
CUPTI_API_EXIT = 1
    The callback is at the exit of the API call.
CUPTI_API_CBSITE_FORCE_INT = 0x7fffffff

enum CUpti_CallbackDomain

Callback domains.

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

Values

CUPTI_CB_DOMAIN_INVALID = 0
Invalid domain.
CUPTI_CB_DOMAIN_DRIVER_API = 1
Domain containing callback points for all driver API functions.
CUPTI_CB_DOMAIN_RUNTIME_API = 2
Domain containing callback points for all runtime API functions.
CUPTI_CB_DOMAIN_RESOURCE = 3
Domain containing callback points for CUDA resource tracking.
CUPTI_CB_DOMAIN_SYNCHRONIZE = 4
Domain containing callback points for CUDA synchronization.
CUPTI_CB_DOMAIN_NVTX = 5
Domain containing callback points for NVTX API functions.
CUPTI_CB_DOMAIN_SIZE = 6
CUPTI_CB_DOMAIN_FORCE_INT = 0x7fffffff

enum CUpti_CallbackIdResource

Callback IDs for resource domain.

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

Values

CUPTI_CBID_RESOURCE_INVALID = 0
Invalid resource callback ID.
CUPTI_CBID_RESOURCE_CONTEXT_CREATED = 1
A new context has been created.
CUPTI_CBID_RESOURCE_CONTEXT_DESTROY_STARTING = 2
A context is about to be destroyed.
CUPTI_CBID_RESOURCE_STREAM_CREATED = 3
A new stream has been created.
CUPTI_CBID_RESOURCE_STREAM_DESTROY_STARTING = 4
A stream is about to be destroyed.
CUPTI_CBID_RESOURCE_CU_INIT_FINISHED = 5
The driver has finished initializing.
CUPTI_CBID_RESOURCE_MODULE_LOADED = 6

A module has been loaded.

CUPTI_CBID_RESOURCE_MODULE_UNLOAD_STARTING = 7

A module is about to be unloaded.

CUPTI_CBID_RESOURCE_MODULE_PROFILED = 8

The current module which is being profiled.

CUPTI_CBID_RESOURCE_SIZE

CUPTI_CBID_RESOURCE_FORCE_INT = 0x7fffffff

enum CUpti_CallbackIdSync

Callback IDs for synchronization domain.

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

Values

CUPTI_CBID_SYNCHRONIZE_INVALID = 0

Invalid synchronize callback ID.

CUPTI_CBID_SYNCHRONIZE_STREAM_SYNCHRONIZED = 1

Stream synchronization has completed for the stream.

CUPTI_CBID_SYNCHRONIZE_CONTEXT_SYNCHRONIZED = 2

Context synchronization has completed for the context.

CUPTI_CBID_SYNCHRONIZE_SIZE

CUPTI_CBID_SYNCHRONIZE_FORCE_INT = 0x7fffffff

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

Function type for a callback.

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. If `domain` is CUPTI_CB_DOMAIN_NVTX the type of `cbdata` will be CUpti_NvtxData.

typedef uint32_t CUpti_CallbackId

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

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 (these values are defined in cupti_driver_cbid.h). Within a runtime API callback this should be interpreted as a CUpti_runtime_api_trace_cbid value (these values are defined in cupti_runtime_cbid.h). 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.

typedef CUpti_DomainTable

Pointer to an array of callback domains.

typedef struct CUpti_Subscriber_st *CUpti_SubscriberHandle

A callback subscriber.

CUptiResult cuptiEnableAllDomains (uint32_t enable, CUpti_SubscriberHandle subscriber)

Enable or disable all callbacks in all domains.

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

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_NOT_INITIALIZED

  if unable to initialized CUPTI
- CUPTI_ERROR_INVALID_PARAMETER

  if `subscriber` is invalid

Description

Enable or disable all callbacks in all domains.

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.

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.

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

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_NOT_INITIALIZED

  if unable to initialized CUPTI
- CUPTI_ERROR_INVALID_PARAMETER

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

Description

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

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.

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

Enable or disabled all callbacks for a specific domain.

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

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_NOT_INITIALIZED

  if unable to initialized CUPTI
- CUPTI_ERROR_INVALID_PARAMETER

  if `subscriber` or `domain` is invalid

Description

Enable or disabled all callbacks for a specific domain.

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.

CUptiResult cuptiGetCallbackName (CUpti_CallbackDomain domain, uint32_t cbid, const char **name)

Get the name of a callback for a specific domain and callback ID.

Parameters

domain
The domain of the callback

cbid
The ID of the callback

name
Returns pointer to the name string on success, NULL otherwise

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_INVALID_PARAMETER

  if `name` is NULL, or if `domain` or `cbid` is invalid.

Description

Returns a pointer to the name c_string in `**name`.

Names are available only for the DRIVER and RUNTIME domains.

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.

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

Returns

- 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.

Description

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

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.

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

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

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.

Returns

- 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

Description

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.

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

CUptiResult cuptiSupportedDomains (size_t *domainCount, CUpti_DomainTable *domainTable)

Get the available callback domains.

Parameters

domainCount
Returns number of callback domains
domainTable
Returns pointer to array of available callback domains

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_NOT_INITIALIZED

  if unable to initialize CUPTI
- CUPTI_ERROR_INVALID_PARAMETER

  if `domainCount` or `domainTable` are NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiUnsubscribe (CUpti_SubscriberHandle subscriber)

Unregister a callback subscriber.

Parameters

subscriber
Handle to the initialize subscriber

Returns

- CUPTI_SUCCESS

  on success
- CUPTI_ERROR_NOT_INITIALIZED

  if unable to initialized CUPTI
- CUPTI_ERROR_INVALID_PARAMETER

  if `subscriber` is NULL or not initialized

Description

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

Thread-safety: this function is thread safe.

2.5. CUPTI Event API

Functions, types, and enums that implement the CUPTI Event API.

struct CUpti_EventGroupSet

A set of event groups.

struct CUpti_EventGroupSets

A set of event group sets.

enum CUpti_DeviceAttribute

Device attributes.

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

Values

CUPTI_DEVICE_ATTR_MAX_EVENT_ID = 1
Number of event IDs for a device. Value is a uint32_t.
CUPTI_DEVICE_ATTR_MAX_EVENT_DOMAIN_ID = 2
Number of event domain IDs for a device. Value is a uint32_t.
CUPTI_DEVICE_ATTR_GLOBAL_MEMORY_BANDWIDTH = 3
Get global memory bandwidth in Kbytes/sec. Value is a uint64_t.
CUPTI_DEVICE_ATTR_INSTRUCTION_PER_CYCLE = 4
Get theoretical maximum number of instructions per cycle. Value is a uint32_t.
CUPTI_DEVICE_ATTR_INSTRUCTION_THROUGHPUT_SINGLE_PRECISION = 5
Get theoretical maximum number of single precision instructions that can be executed per second. Value is a uint64_t.
CUPTI_DEVICE_ATTR_MAX_FRAME_BUFFERS = 6
Get number of frame buffers for device. Value is a uint64_t.
CUPTI_DEVICE_ATTR_PCIE_LINK_RATE = 7
Get PCIE link rate in Mega bits/sec for device. Return 0 if bus-type is non-PCIE. Value is a uint64_t.
CUPTI_DEVICE_ATTR_PCIE_LINK_WIDTH = 8
Get PCIE link width for device. Return 0 if bus-type is non-PCIE. Value is a uint64_t.
CUPTI_DEVICE_ATTR_PCIE_GEN = 9
Get PCIE generation for device. Return 0 if bus-type is non-PCIE. Value is a uint64_t.
CUPTI_DEVICE_ATTR_DEVICE_CLASS = 10
Get the class for the device. Value is a CUpti_DeviceAttributeDeviceClass.
CUPTI_DEVICE_ATTR_FLOP_SP_PER_CYCLE = 11
Get the peak single precision flop per cycle. Value is a uint64_t.
CUPTI_DEVICE_ATTR_FLOP_DP_PER_CYCLE = 12
Get the peak double precision flop per cycle. Value is a uint64_t.
CUPTI_DEVICE_ATTR_MAX_L2_UNITS = 13
Get number of L2 units. Value is a uint64_t.
CUPTI_DEVICE_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_DeviceAttributeDeviceClass

Device class.

Enumeration of device classes for device attribute CUPTI_DEVICE_ATTR_DEVICE_CLASS.

Values

CUPTI_DEVICE_ATTR_DEVICE_CLASS_TESLA = 0
CUPTI_DEVICE_ATTR_DEVICE_CLASS_QUADRO = 1
CUPTI_DEVICE_ATTR_DEVICE_CLASS_GEFORCE = 2

enum CUpti_EventAttribute

Event attributes.

Event attributes. These attributes can be read using cuptiEventGetAttribute.

Values

CUPTI_EVENT_ATTR_NAME = 0
Event name. Value is a null terminated const c-string.
CUPTI_EVENT_ATTR_SHORT_DESCRIPTION = 1
Short description of event. Value is a null terminated const c-string.
CUPTI_EVENT_ATTR_LONG_DESCRIPTION = 2
Long description of event. Value is a null terminated const c-string.
CUPTI_EVENT_ATTR_CATEGORY = 3
Category of event. Value is CUpti_EventCategory.
CUPTI_EVENT_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_EventCategory

An event category.

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.

Values

CUPTI_EVENT_CATEGORY_INSTRUCTION = 0
An instruction related event.
CUPTI_EVENT_CATEGORY_MEMORY = 1
A memory related event.
CUPTI_EVENT_CATEGORY_CACHE = 2
A cache related event.
CUPTI_EVENT_CATEGORY_PROFILE_TRIGGER = 3

A profile-trigger event.

CUPTI_EVENT_CATEGORY_FORCE_INT = 0x7fffffff

enum CUpti_EventCollectionMethod

The collection method used for an event.

The collection method indicates how an event is collected.

Values

CUPTI_EVENT_COLLECTION_METHOD_PM = 0

Event is collected using a hardware global performance monitor.

CUPTI_EVENT_COLLECTION_METHOD_SM = 1

Event is collected using a hardware SM performance monitor.

CUPTI_EVENT_COLLECTION_METHOD_INSTRUMENTED = 2

Event is collected using software instrumentation.

CUPTI_EVENT_COLLECTION_METHOD_FORCE_INT = 0x7fffffff

enum CUpti_EventCollectionMode

Event collection modes.

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

Values

CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS = 0

Events are collected for the entire duration between the cuptiEventGroupEnable and cuptiEventGroupDisable calls. For devices with compute capability less than 2.0, event values are reset when a kernel is launched. For all other devices event values are only reset when the events are read. For CUDA toolkit v6.0 and older this was the default mode. From CUDA toolkit v6.5 this mode is supported on Tesla devices only.

CUPTI_EVENT_COLLECTION_MODE_KERNEL = 1

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. Event values are reset to zero when each kernel execution begins. 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. This is the default mode from CUDA toolkit v6.5, and it is the only supported mode for non-Tesla (Quadro, GeForce etc.) devices.

CUPTI_EVENT_COLLECTION_MODE_FORCE_INT = 0x7fffffff

enum CUpti_EventDomainAttribute

Event domain attributes.

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

Values

CUPTI_EVENT_DOMAIN_ATTR_NAME = 0
: Event domain name. Value is a null terminated const c-string.

CUPTI_EVENT_DOMAIN_ATTR_INSTANCE_COUNT = 1
: 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 = 3
: 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.

CUPTI_EVENT_DOMAIN_ATTR_COLLECTION_METHOD = 4
: Collection method used for events contained in the event domain. Value is a CUpti_EventCollectionMethod.

CUPTI_EVENT_DOMAIN_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_EventGroupAttribute

Event group attributes.

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

Values

CUPTI_EVENT_GROUP_ATTR_EVENT_DOMAIN_ID = 0
: 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 = 1
: [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 = 2
: [rw] Reserved for user data.

CUPTI_EVENT_GROUP_ATTR_NUM_EVENTS = 3
: Number of events in the group. Value is a uint32_t.

CUPTI_EVENT_GROUP_ATTR_EVENTS = 4
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 = 5
Number of instances of the domain bound to this event group that will be counted. Value is a uint32_t.
CUPTI_EVENT_GROUP_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_ReadEventFlags

Flags for cuptiEventGroupReadEvent an cuptiEventGroupReadAllEvents.

Flags for cuptiEventGroupReadEvent an cuptiEventGroupReadAllEvents.

Values

CUPTI_EVENT_READ_FLAG_NONE = 0
No flags.
CUPTI_EVENT_READ_FLAG_FORCE_INT = 0x7fffffff

typedef uint32_t CUpti_EventDomainID

ID for an event domain.

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

A group of events.

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

ID for an event.

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

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

Get the event domains for a device.

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

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_DEVICE
- CUPTI_ERROR_INVALID_PARAMETER

  if `arraySizeBytes` or `domainArray` are NULL

Description

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`.

Thread-safety: this function is thread safe.

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

Read a device attribute.

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

Returns

- 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.

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiDeviceGetEventDomainAttribute (CUdevice device, CUpti_EventDomainID eventDomain,

CUpti_EventDomainAttribute attrib, size_t *valueSize, void *value)

Read an event domain attribute.

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

Returns

- 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.

Description

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.

Thread-safety: this function is thread safe.

CUptiResult cuptiDeviceGetNumEventDomains (CUdevice device, uint32_t *numDomains)

Get the number of domains for a device.

Parameters

device
  The CUDA device
numDomains
  Returns the number of domains

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_DEVICE
- CUPTI_ERROR_INVALID_PARAMETER

  if `numDomains` is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiDeviceGetTimestamp (CUcontext context, uint64_t *timestamp)

Read a device timestamp.

Parameters

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

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED

- CUPTI_ERROR_INVALID_CONTEXT
- CUPTI_ERROR_INVALID_PARAMETER

  is `timestamp` is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiDisableKernelReplayMode (CUcontext context)

Disable kernel replay mode.

Parameters

context
The context

Returns

- CUPTI_SUCCESS

Description

Set profiling mode for the context to non-replay (default) mode. Event collection mode will be set to CUPTI_EVENT_COLLECTION_MODE_KERNEL. All previously enabled event groups and event group sets will be disabled.

Thread-safety: this function is thread safe.

CUptiResult cuptiEnableKernelReplayMode (CUcontext context)

Enable kernel replay mode.

Parameters

context
The context

Returns

- CUPTI_SUCCESS

Description

Set profiling mode for the context to replay mode. In this mode, any number of events can be collected in one run of the kernel. The event collection mode will automatically switch to CUPTI_EVENT_COLLECTION_MODE_KERNEL. In this mode, cuptiSetEventCollectionMode will return CUPTI_ERROR_INVALID_OPERATION.

- **Kernels** might take longer to run if many events are enabled.
- **Thread-safety**: this function is thread safe.

CUptiResult cuptiEnumEventDomains (size_t *arraySizeBytes, CUpti_EventDomainID *domainArray)

Get the event domains available on any device.

Parameters

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

domainArray
Returns all the event domains

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `arraySizeBytes` or `domainArray` are NULL

Description

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`.

Thread-safety: this function is thread safe.

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

Get the events in a domain.

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

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `arraySizeBytes` or `eventArray` are NULL

Description

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`.

Thread-safety: this function is thread safe.

CUptiResult cuptiEventDomainGetAttribute (CUpti_EventDomainID eventDomain,

CUpti_EventDomainAttribute attrib, size_t *valueSize, void *value)

Read an event domain attribute.

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

Returns

- 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.

Description

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.

Thread-safety: this function is thread safe.

CUptiResult cuptiEventDomainGetNumEvents (CUpti_EventDomainID eventDomain, uint32_t *numEvents)

Get number of events in a domain.

Parameters

eventDomain
ID of the event domain

numEvents
Returns the number of events in the domain

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_EVENT_DOMAIN_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `numEvents` is NULL

Description

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

Thread-safety: this function is thread safe.

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

Get an event attribute.

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

Returns

- 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.

Description

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.

Thread-safety: this function is thread safe.

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

Find an event by name.

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

Returns

- 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

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupAddEvent (CUpti_EventGroup eventGroup, CUpti_EventID event)

Add an event to an event group.

Parameters

eventGroup
The event group

event
The event to add to the group

Returns

- 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

Description

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

Thread-safety: this function is thread safe.

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

Create a new event group for a context.

Parameters

context
The context for the event group

eventGroup
Returns the new event group

flags
Reserved - must be zero

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_CONTEXT
- CUPTI_ERROR_OUT_OF_MEMORY
- CUPTI_ERROR_INVALID_PARAMETER

if `eventGroup` is NULL

Description

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

- `flags` are reserved for future use and should be set to zero.
- **Thread-safety**: this function is thread safe.

CUptiResult cuptiEventGroupDestroy (CUpti_EventGroup eventGroup)

Destroy an event group.

Parameters

eventGroup
The event group to destroy

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_OPERATION

  if the event group is enabled
- CUPTI_ERROR_INVALID_PARAMETER

  if eventGroup is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupDisable (CUpti_EventGroup eventGroup)

Disable an event group.

Parameters

eventGroup
    The event group

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_HARDWARE
- CUPTI_ERROR_INVALID_PARAMETER

  if `eventGroup` is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupEnable (CUpti_EventGroup eventGroup)

Enable an event group.

Parameters

eventGroup
    The event group

Returns

- 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
- CUPTI_ERROR_HARDWARE_BUSY

  if another client is profiling and hardware is busy

Description

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.

Thread-safety: this function is thread safe.

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

Read an event group attribute.

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

Returns

- 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.

Description

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

Thread-safety: this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to cuptiEventGroupDestroy, cuptiEventGroupAddEvent, etc.), and must guard against simultaneous destruction of the context in which `eventGroup` was created (for example, client must guard against simultaneous calls to cudaDeviceReset, cuCtxDestroy, etc.).

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.

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 `eventIdArray`

Returns

- 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

Description

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. After being read, an event's value is reset to zero.

Thread-safety: this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to cuptiEventGroupDestroy, cuptiEventGroupAddEvent, etc.), and must guard against simultaneous destruction of the context in which eventGroup was created (for example, client must guard against simultaneous calls to cudaDeviceReset, cuCtxDestroy, etc.). If cuptiEventGroupResetAllEvents is called simultaneously with this function, then returned event values are undefined.

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.

Parameters

eventGroup
The event group

flags
Flags controlling the reading mode

event
The event to read

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

eventValueBuffer
Returns the event value(s)

Returns

- 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

Description

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. After being read, an event's value is reset to zero.

Thread-safety: this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to cuptiEventGroupDestroy, cuptiEventGroupAddEvent, etc.), and must guard against simultaneous destruction of the context in which `eventGroup` was created (for example, client must guard against simultaneous calls to cudaDeviceReset, cuCtxDestroy, etc.). If cuptiEventGroupResetAllEvents is called simultaneously with this function, then returned event values are undefined.

CUptiResult cuptiEventGroupRemoveAllEvents (CUpti_EventGroup eventGroup)

Remove all events from an event group.

Parameters

eventGroup
The event group

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_OPERATION

  if `eventGroup` is enabled
- CUPTI_ERROR_INVALID_PARAMETER

if `eventGroup` is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupRemoveEvent (CUpti_EventGroup eventGroup, CUpti_EventID event)

Remove an event from an event group.

Parameters

eventGroup
The event group

event
The event to remove from the group

Returns

- 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

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupResetAllEvents (CUpti_EventGroup eventGroup)

Zero all the event counts in an event group.

Parameters

eventGroup
The event group

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_HARDWARE
- CUPTI_ERROR_INVALID_PARAMETER

  if `eventGroup` is NULL

Description

Zero all the event counts in an event group.

Thread-safety: this function is thread safe but client must guard against simultaneous destruction or modification of `eventGroup` (for example, client must guard against simultaneous calls to cuptiEventGroupDestroy, cuptiEventGroupAddEvent, etc.), and must guard against simultaneous destruction of the context in which `eventGroup` was created (for example, client must guard against simultaneous calls to cudaDeviceReset, cuCtxDestroy, etc.).

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

Returns

- 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.

Description

Write an event group attribute.

Thread-safety: this function is thread safe.

CUptiResult cuptiEventGroupSetDisable (CUpti_EventGroupSet *eventGroupSet)

Disable an event group set.

Parameters

eventGroupSet
The pointer to the event group set

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_HARDWARE
- CUPTI_ERROR_INVALID_PARAMETER

  if `eventGroupSet` is NULL

Description

Disable a set of event groups. Disabling a set of event groups stops collection of events contained in the groups.

- **Thread-safety:** this function is thread safe.
- **If** this call fails, some of the event groups in the set may be disabled and other event groups may remain enabled.

CUptiResult cuptiEventGroupSetEnable (CUpti_EventGroupSet *eventGroupSet)

Enable an event group set.

Parameters

eventGroupSet
The pointer to the event group set

Returns

- 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 `eventGroupSet` is NULL
- CUPTI_ERROR_HARDWARE_BUSY

  if other client is profiling and hardware is busy

Description

Enable a set of event groups. Enabling a set of event groups zeros the value of all the events in all the groups and then starts collection of those events.

Thread-safety: this function is thread safe.

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.

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

Returns

- 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

Description

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. Within each event group set, the `sets` array indicates the event groups that should be collected on each pass.

Thread-safety: this function is thread safe, but client must guard against another thread simultaneously destroying `context`.

CUptiResult cuptiEventGroupSetsDestroy (CUpti_EventGroupSets *eventGroupSets)

Destroy a CUpti_EventGroupSets object.

Parameters

eventGroupSets
The object to destroy

Returns

- 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

Description

Destroy a CUpti_EventGroupSets object.

Thread-safety: this function is thread safe.

CUptiResult cuptiGetNumEventDomains (uint32_t *numDomains)

Get the number of event domains available on any device.

Parameters

numDomains
Returns the number of domains

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `numDomains` is NULL

Description

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

Thread-safety: this function is thread safe.

CUptiResult cuptiSetEventCollectionMode (CUcontext context, CUpti_EventCollectionMode mode)

Set the event collection mode.

Parameters

context
The context

mode
The event collection mode

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_CONTEXT
- CUPTI_ERROR_INVALID_OPERATION

  if called when replay mode is enabled
- CUPTI_ERROR_NOT_SUPPORTED

  if mode is not supported on the device

Description

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`. This API is invalid in kernel replay mode.

Thread-safety: this function is thread safe.

#define CUPTI_EVENT_OVERFLOW ((uint64_t)0xFFFFFFFFFFFFFFFFULL)

The overflow value for a CUPTI event.

The CUPTI event value that indicates an overflow.

2.6. CUPTI Metric API

Functions, types, and enums that implement the CUPTI Metric API.

union CUpti_MetricValue

A metric value.

enum CUpti_MetricAttribute

Metric attributes.

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

Values

CUPTI_METRIC_ATTR_NAME = 0
Metric name. Value is a null terminated const c-string.
CUPTI_METRIC_ATTR_SHORT_DESCRIPTION = 1
Short description of metric. Value is a null terminated const c-string.
CUPTI_METRIC_ATTR_LONG_DESCRIPTION = 2
Long description of metric. Value is a null terminated const c-string.
CUPTI_METRIC_ATTR_CATEGORY = 3
Category of the metric. Value is of type CUpti_MetricCategory.
CUPTI_METRIC_ATTR_VALUE_KIND = 4
Value type of the metric. Value is of type CUpti_MetricValueKind.
CUPTI_METRIC_ATTR_EVALUATION_MODE = 5
Metric evaluation mode. Value is of type CUpti_MetricEvaluationMode.
CUPTI_METRIC_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_MetricCategory

A metric category.

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.

Values

CUPTI_METRIC_CATEGORY_MEMORY = 0
A memory related metric.
CUPTI_METRIC_CATEGORY_INSTRUCTION = 1
An instruction related metric.
CUPTI_METRIC_CATEGORY_MULTIPROCESSOR = 2
A multiprocessor related metric.
CUPTI_METRIC_CATEGORY_CACHE = 3
A cache related metric.
CUPTI_METRIC_CATEGORY_TEXTURE = 4
A texture related metric.
CUPTI_METRIC_CATEGORY_FORCE_INT = 0x7fffffff

enum CUpti_MetricEvaluationMode

A metric evaluation mode.

A metric can be evaluated per hardware instance to know the load balancing across instances of a domain or the metric can be evaluated in aggregate mode when the events involved in metric evaluation are from different event domains. It might be possible to evaluate some metrics in both modes for convenience. A metric's evaluation mode is accessed using CUpti_MetricEvaluationMode and the CUPTI_METRIC_ATTR_EVALUATION_MODE attribute.

Values

CUPTI_METRIC_EVALUATION_MODE_PER_INSTANCE = 1
If this bit is set, the metric can be profiled for each instance of the domain. The event values passed to cuptiMetricGetValue can contain values for one instance of the domain. And cuptiMetricGetValue can be called for each instance.
CUPTI_METRIC_EVALUATION_MODE_AGGREGATE = 1<<1
If this bit is set, the metric can be profiled over all instances. The event values passed to cuptiMetricGetValue can be aggregated values of events for all instances of the domain.
CUPTI_METRIC_EVALUATION_MODE_FORCE_INT = 0x7fffffff

enum CUpti_MetricPropertyDeviceClass

Device class.

Enumeration of device classes for metric property CUPTI_METRIC_PROPERTY_DEVICE_CLASS.

Values

CUPTI_METRIC_PROPERTY_DEVICE_CLASS_TESLA = 0
CUPTI_METRIC_PROPERTY_DEVICE_CLASS_QUADRO = 1

CUPTI_METRIC_PROPERTY_DEVICE_CLASS_GEFORCE = 2

enum CUpti_MetricPropertyID

Metric device properties.

Metric device properties describe device properties which are needed for a metric. Some of these properties can be collected using cuDeviceGetAttribute.

Values

CUPTI_METRIC_PROPERTY_MULTIPROCESSOR_COUNT
CUPTI_METRIC_PROPERTY_WARPS_PER_MULTIPROCESSOR
CUPTI_METRIC_PROPERTY_KERNEL_GPU_TIME
CUPTI_METRIC_PROPERTY_CLOCK_RATE
CUPTI_METRIC_PROPERTY_FRAME_BUFFER_COUNT
CUPTI_METRIC_PROPERTY_GLOBAL_MEMORY_BANDWIDTH
CUPTI_METRIC_PROPERTY_PCIE_LINK_RATE
CUPTI_METRIC_PROPERTY_PCIE_LINK_WIDTH
CUPTI_METRIC_PROPERTY_PCIE_GEN
CUPTI_METRIC_PROPERTY_DEVICE_CLASS
CUPTI_METRIC_PROPERTY_FLOP_SP_PER_CYCLE
CUPTI_METRIC_PROPERTY_FLOP_DP_PER_CYCLE
CUPTI_METRIC_PROPERTY_L2_UNITS

enum CUpti_MetricValueKind

Kinds of metric values.

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.

Values

CUPTI_METRIC_VALUE_KIND_DOUBLE = 0
The metric value is a 64-bit double.

CUPTI_METRIC_VALUE_KIND_UINT64 = 1
The metric value is a 64-bit unsigned integer.

CUPTI_METRIC_VALUE_KIND_PERCENT = 2
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 = 3
The metric value is a throughput represented by a 64-bit integer. The unit for throughput values is bytes/second.

CUPTI_METRIC_VALUE_KIND_INT64 = 4

The metric value is a 64-bit signed integer.

CUPTI_METRIC_VALUE_KIND_UTILIZATION_LEVEL = 5

The metric value is a utilization level, as represented by CUpti_MetricValueUtilizationLevel.

CUPTI_METRIC_VALUE_KIND_FORCE_INT = 0x7fffffff

enum CUpti_MetricValueUtilizationLevel

Enumeration of utilization levels for metrics values of kind CUPTI_METRIC_VALUE_KIND_UTILIZATION_LEVEL. Utilization values can vary from IDLE (0) to MAX (10) but the enumeration only provides specific names for a few values.

Values

CUPTI_METRIC_VALUE_UTILIZATION_IDLE = 0
CUPTI_METRIC_VALUE_UTILIZATION_LOW = 2
CUPTI_METRIC_VALUE_UTILIZATION_MID = 5
CUPTI_METRIC_VALUE_UTILIZATION_HIGH = 8
CUPTI_METRIC_VALUE_UTILIZATION_MAX = 10
CUPTI_METRIC_VALUE_UTILIZATION_FORCE_INT = 0x7fffffff

typedef uint32_t CUpti_MetricID

ID for a metric.

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

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

Get the metrics for a device.

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

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED

- CUPTI_ERROR_INVALID_DEVICE
- CUPTI_ERROR_INVALID_PARAMETER

  if `arraySizeBytes` or `metricArray` are NULL

Description

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`.

CUptiResult cuptiDeviceGetNumMetrics (CUdevice device, uint32_t *numMetrics)

Get the number of metrics for a device.

Parameters

device
The CUDA device

numMetrics
Returns the number of metrics available for the device

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_DEVICE
- CUPTI_ERROR_INVALID_PARAMETER

  if `numMetrics` is NULL

Description

Returns the number of metrics available for a device.

CUptiResult cuptiEnumMetrics (size_t *arraySizeBytes, CUpti_MetricID *metricArray)

Get all the metrics available on any device.

Parameters

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

metricArray
Returns the IDs of the metrics

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `arraySizeBytes` or `metricArray` are NULL

Description

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`.

CUptiResult cuptiGetNumMetrics (uint32_t *numMetrics)

Get the total number of metrics available on any device.

Parameters

numMetrics
Returns the number of metrics

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER

  if `numMetrics` is NULL

Description

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

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.

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

Returns

- 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

Description

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.

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

Get the events required to calculating a metric.

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`

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_METRIC_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `eventIdArraySizeBytes` or `eventIdArray` are NULL.

Description

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`.

CUptiResult cuptiMetricEnumProperties (CUpti_MetricID metric, size_t *propIdArraySizeBytes, CUpti_MetricPropertyID *propIdArray)

Get the properties required to calculating a metric.

Parameters

metric
ID of the metric

propIdArraySizeBytes
The size of `propIdArray` in bytes, and returns the number of bytes written to `propIdArray`
propIdArray
Returns the IDs of the properties required to calculate `metric`

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_METRIC_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `propIdArraySizeBytes` or `propIdArray` are NULL.

Description

Gets the property IDs in `propIdArray` required to calculate a `metric`. The size of the `propIdArray` buffer is given by `*propIdArraySizeBytes` and must be at least `numProp` * sizeof(CUpti_DeviceAttribute) or all properties will not be returned. The value returned in `*propIdArraySizeBytes` contains the number of bytes returned in `propIdArray`.

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

Get a metric attribute.

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

Returns

- 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.

Description

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.

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

Find an metric by name.

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

Returns

- 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.

Description

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

CUptiResult cuptiMetricGetNumEvents (CUpti_MetricID metric, uint32_t *numEvents)

Get number of events required to calculate a metric.

Parameters

metric
ID of the metric

numEvents
Returns the number of events required for the metric

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_METRIC_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `numEvents` is NULL

Description

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

CUptiResult cuptiMetricGetNumProperties (CUpti_MetricID metric, uint32_t *numProp)

Get number of properties required to calculate a metric.

Parameters

metric
ID of the metric

numProp
Returns the number of properties required for the metric

Returns

- CUPTI_SUCCESS

- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_METRIC_ID
- CUPTI_ERROR_INVALID_PARAMETER

  if `numProp` is NULL

Description

Returns the number of properties in `numProp` that are required to calculate a metric.

CUptiResult cuptiMetricGetRequiredEventGroupSets (CUcontext context, CUpti_MetricID metric, CUpti_EventGroupSets **eventGroupSets)

For a metric get the groups of events that must be collected in the same pass.

Parameters

context
  The context for event collection
metric
  The metric ID
eventGroupSets
  Returns a CUpti_EventGroupSets object that indicates the events that must be collected in the same pass to ensure the metric is calculated correctly. Returns NULL if no grouping is required for metric

Returns

- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED
- CUPTI_ERROR_INVALID_METRIC_ID

Description

For a metric get the groups of events that must be collected in the same pass to ensure that the metric is calculated correctly. If the events are not collected as specified then the metric value may be inaccurate.

The function returns NULL if a metric does not have any required event group. In this case the events needed for the metric can be grouped in any manner for collection.

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.

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

Returns

- 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_METRIC_VALUE

  if the computed metric value cannot be represented in the metric's value type. For example, if the metric value type is unsigned and the computed metric value is negative
- CUPTI_ERROR_INVALID_PARAMETER

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

Description

Use the events collected for a metric to calculate the metric value. Metric value evaluation depends on the evaluation mode CUpti_MetricEvaluationMode that the metric supports. If a metric has evaluation mode as CUPTI_METRIC_EVALUATION_MODE_PER_INSTANCE, then it assumes that the input event value is for one domain instance. If a metric has evaluation mode as CUPTI_METRIC_EVALUATION_MODE_AGGREGATE, it assumes that input 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).

CUptiResult cuptiMetricGetValue2 (CUpti_MetricID metric, size_t eventIdArraySizeBytes, CUpti_EventID *eventIdArray, size_t eventValueArraySizeBytes, uint64_t *eventValueArray, size_t propIdArraySizeBytes, CUpti_MetricPropertyID *propIdArray, size_t propValueArraySizeBytes, uint64_t *propValueArray, CUpti_MetricValue *metricValue)

Calculate the value for a metric.

Parameters

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`
propIdArraySizeBytes
The size of `propIdArray` in bytes
propIdArray
The metric property IDs required to calculate `metric`
propValueArraySizeBytes
The size of `propValueArray` in bytes
propValueArray
The metric property values required to calculate `metric`. The values must be order to match the order of metric properties in `propIdArray`
metricValue
Returns the value for the metric

Returns

- 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_NOT_COMPATIBLE

  if the computed metric value cannot be represented in the metric's value type. For example, if the metric value type is unsigned and the computed metric value is negative
- CUPTI_ERROR_INVALID_PARAMETER

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

Description

Use the events and properties collected for a metric to calculate the metric value. Metric value evaluation depends on the evaluation mode CUpti_MetricEvaluationMode that the metric supports. If a metric has evaluation mode as CUPTI_METRIC_EVALUATION_MODE_PER_INSTANCE, then it assumes that the input event value is for one domain instance. If a metric has evaluation mode as CUPTI_METRIC_EVALUATION_MODE_AGGREGATE, it assumes that input 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).

Chapter 3. DATA STRUCTURES

Here are the data structures with brief descriptions:

CUpti_Activity
The base activity record

CUpti_ActivityAPI
The activity record for a driver or runtime API invocation

CUpti_ActivityAutoBoostState
Device auto boost state structure

CUpti_ActivityBranch
The activity record for source level result branch. (deprecated)

CUpti_ActivityBranch2
The activity record for source level result branch

CUpti_ActivityCdpKernel
The activity record for CDP (CUDA Dynamic Parallelism) kernel

CUpti_ActivityContext
The activity record for a context

CUpti_ActivityDevice
The activity record for a device

CUpti_ActivityDeviceAttribute
The activity record for a device attribute

CUpti_ActivityEnvironment
The activity record for CUPTI environmental data

CUpti_ActivityEvent
The activity record for a CUPTI event

CUpti_ActivityEventInstance
The activity record for a CUPTI event with instance information

CUpti_ActivityFunction
The activity record for global/device functions

CUpti_ActivityGlobalAccess
The activity record for source-level global access. (deprecated)

CUpti_ActivityGlobalAccess2
The activity record for source-level global access

CUpti_ActivityInstructionExecution
The activity record for source-level sass/source line-by-line correlation

CUpti_ActivityKernel
The activity record for kernel. (deprecated)

CUpti_ActivityKernel2
The activity record for kernel. (deprecated)

CUpti_ActivityKernel3
The activity record for a kernel (CUDA 6.5(with sm_52 support) onwards)

CUpti_ActivityMarker
The activity record providing a marker which is an instantaneous point in time

CUpti_ActivityMarkerData
The activity record providing detailed information for a marker

CUpti_ActivityMemcpy
The activity record for memory copies

CUpti_ActivityMemcpy2
The activity record for peer-to-peer memory copies

CUpti_ActivityMemset
The activity record for memset

CUpti_ActivityMetric
The activity record for a CUPTI metric

CUpti_ActivityMetricInstance
The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (CUPTI_ACTIVITY_KIND_METRIC_INSTANCE). 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. This activity record should be used when metric domain instance information needs to be associated with the metric

CUpti_ActivityModule
The activity record for a CUDA module

CUpti_ActivityName
The activity record providing a name

CUpti_ActivityObjectKindId
Identifiers for object kinds as specified by CUpti_ActivityObjectKind

CUpti_ActivityOverhead
The activity record for CUPTI and driver overheads

CUpti_ActivityPreemption
The activity record for a preemption of a CDP kernel

CUpti_ActivitySharedAccess
The activity record for source-level shared access

CUpti_ActivitySourceLocator
    The activity record for source locator
CUpti_ActivityUnifiedMemoryCounter
    The activity record for Unified Memory counters
CUpti_ActivityUnifiedMemoryCounterConfig
    Unified Memory counters configuration structure
CUpti_CallbackData
    Data passed into a runtime or driver API callback function
CUpti_EventGroupSet
    A set of event groups
CUpti_EventGroupSets
    A set of event group sets
CUpti_MetricValue
    A metric value
CUpti_ModuleResourceData
    Module data passed into a resource callback function
CUpti_NvtxData
    Data passed into a NVTX callback function
CUpti_ResourceData
    Data passed into a resource callback function
CUpti_SynchronizeData
    Data passed into a synchronize callback function

3.1. CUpti_Activity Struct Reference

The base activity record.

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

CUpti_ActivityKind CUpti_Activity::kind

The kind of this activity.

3.2. CUpti_ActivityAPI Struct Reference

The activity record for a driver or runtime API invocation.

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

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the function.

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the function.

uint32_t CUpti_ActivityAPI::threadId

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

3.3. CUpti_ActivityAutoBoostState Struct Reference

Device auto boost state structure.

This structure defines auto boost state for a device. See function /ref cuptiGetAutoBoostState

uint32_t CUpti_ActivityAutoBoostState::enabled

Returned auto boost state. 1 is returned in case auto boost is enabled, 0 otherwise

uint32_t CUpti_ActivityAutoBoostState::pid

Id of process that has set the current boost state. The value will be CUPTI_AUTO_BOOST_INVALID_CLIENT_PID if the user does not have the permission to query process ids or there is an error in querying the process id.

3.4. CUpti_ActivityBranch Struct Reference

The activity record for source level result branch. (deprecated).

This activity record the locations of the branches in the source (CUPTI_ACTIVITY_KIND_BRANCH). Branch activities are now reported using the CUpti_ActivityBranch2 activity record.

uint32_t CUpti_ActivityBranch::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivityBranch::diverged

Number of times this branch diverged

uint32_t CUpti_ActivityBranch::executed

The number of times this branch was executed

CUpti_ActivityKind CUpti_ActivityBranch::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_BRANCH.

uint32_t CUpti_ActivityBranch::pcOffset

The pc offset for the branch.

uint32_t CUpti_ActivityBranch::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivityBranch::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction

3.5. CUpti_ActivityBranch2 Struct Reference

The activity record for source level result branch.

This activity record the locations of the branches in the source (CUPTI_ACTIVITY_KIND_BRANCH).

uint32_t CUpti_ActivityBranch2::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivityBranch2::diverged

Number of times this branch diverged

uint32_t CUpti_ActivityBranch2::executed

The number of times this branch was executed

uint32_t CUpti_ActivityBranch2::functionId

Correlation ID with global/device function name

CUpti_ActivityKind CUpti_ActivityBranch2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_BRANCH.

uint32_t CUpti_ActivityBranch2::pad

Undefined. Reserved for internal use.

uint32_t CUpti_ActivityBranch2::pcOffset

The pc offset for the branch.

uint32_t CUpti_ActivityBranch2::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivityBranch2::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction

3.6. CUpti_ActivityCdpKernel Struct Reference

The activity record for CDP (CUDA Dynamic Parallelism) kernel.

This activity record represents a CDP kernel execution.

int32_t CUpti_ActivityCdpKernel::blockX

The X-dimension block size for the kernel.

int32_t CUpti_ActivityCdpKernel::blockY

The Y-dimension block size for the kernel.

int32_t CUpti_ActivityCdpKernel::blockZ

The Z-dimension grid size for the kernel.

uint64_t CUpti_ActivityCdpKernel::completed

The timestamp when kernel is marked as completed, in ns. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the completion time is unknown.

uint32_t CUpti_ActivityCdpKernel::contextId

The ID of the context where the kernel is executing.

uint32_t CUpti_ActivityCdpKernel::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_ActivityCdpKernel::deviceId

The ID of the device where the kernel is executing.

int32_t CUpti_ActivityCdpKernel::dynamicSharedMemory

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

uint64_t CUpti_ActivityCdpKernel::end

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

uint8_t CUpti_ActivityCdpKernel::executed

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

int64_t CUpti_ActivityCdpKernel::gridId

The grid ID of the kernel. Each kernel execution is assigned a unique grid ID.

int32_t CUpti_ActivityCdpKernel::gridX

The X-dimension grid size for the kernel.

int32_t CUpti_ActivityCdpKernel::gridY

The Y-dimension grid size for the kernel.

int32_t CUpti_ActivityCdpKernel::gridZ

The Z-dimension grid size for the kernel.

CUpti_ActivityKind CUpti_ActivityCdpKernel::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_CDP_KERNEL

uint32_t CUpti_ActivityCdpKernel::localMemoryPerThread

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

uint32_t CUpti_ActivityCdpKernel::localMemoryTotal

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

const char *CUpti_ActivityCdpKernel::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_ActivityCdpKernel::parentBlockX

The X-dimension of the parent block.

uint32_t CUpti_ActivityCdpKernel::parentBlockY

The Y-dimension of the parent block.

uint32_t CUpti_ActivityCdpKernel::parentBlockZ

The Z-dimension of the parent block.

int64_t CUpti_ActivityCdpKernel::parentGridId

The grid ID of the parent kernel.

uint64_t CUpti_ActivityCdpKernel::queued

The timestamp when kernel is queued up, in ns. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the queued time is unknown.

uint16_t CUpti_ActivityCdpKernel::registersPerThread

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

uint8_t CUpti_ActivityCdpKernel::requested

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

uint8_t CUpti_ActivityCdpKernel::sharedMemoryConfig

The shared memory configuration used for the kernel. The value is one of the CUsharedconfig enumeration values from cuda.h.

uint64_t CUpti_ActivityCdpKernel::start

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

int32_t CUpti_ActivityCdpKernel::staticSharedMemory

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

uint32_t CUpti_ActivityCdpKernel::streamId

The ID of the stream where the kernel is executing.

uint64_t CUpti_ActivityCdpKernel::submitted

The timestamp when kernel is submitted to the gpu, in ns. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the submission time is unknown.

3.7. CUpti_ActivityContext Struct Reference

The activity record for a context.

This activity record represents information about a context (CUPTI_ACTIVITY_KIND_CONTEXT).

uint16_t CUpti_ActivityContext::computeApiKind

The compute API kind.

See also:

CUpti_ActivityComputeApiKind

uint32_t CUpti_ActivityContext::contextId

The context ID.

uint32_t CUpti_ActivityContext::deviceId

The device ID.

CUpti_ActivityKind CUpti_ActivityContext::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_CONTEXT.

uint16_t CUpti_ActivityContext::nullStreamId

The ID for the NULL stream in this context

3.8. CUpti_ActivityDevice Struct Reference

The activity record for a device.

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

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.

CUpti_ActivityFlag 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.

3.9. CUpti_ActivityDeviceAttribute Struct Reference

The activity record for a device attribute.

This activity record represents information about a GPU device: either a CUpti_DeviceAttribute or CUdevice_attribute value (CUPTI_ACTIVITY_KIND_DEVICE_ATTRIBUTE).

CUpti_ActivityDeviceAttribute::@8 CUpti_ActivityDeviceAttribute::attribute

The attribute, either a CUpti_DeviceAttribute or CUdevice_attribute. Flag CUPTI_ACTIVITY_FLAG_DEVICE_ATTRIBUTE_CUDEVICE is used to indicate what kind of attribute this is. If CUPTI_ACTIVITY_FLAG_DEVICE_ATTRIBUTE_CUDEVICE is 1 then CUdevice_attribute field is value, otherwise CUpti_DeviceAttribute field is valid.

uint32_t CUpti_ActivityDeviceAttribute::deviceId

The ID of the device that this attribute applies to.

CUpti_ActivityFlag CUpti_ActivityDeviceAttribute::flags

The flags associated with the device.

See also:

CUpti_ActivityFlag

CUpti_ActivityKind CUpti_ActivityDeviceAttribute::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_DEVICE_ATTRIBUTE.

CUpti_ActivityDeviceAttribute::@9 CUpti_ActivityDeviceAttribute::value

The value for the attribute. See CUpti_DeviceAttribute and CUdevice_attribute for the type of the value for a given attribute.

3.10. CUpti_ActivityEnvironment Struct Reference

The activity record for CUPTI environmental data.

This activity record provides CUPTI environmental data, include power, clocks, and thermals. This information is sampled at various rates and returned in this activity record. The consumer of the record needs to check the environmentKind field to figure out what kind of environmental record this is.

CUpti_EnvironmentClocksThrottleReason CUpti_ActivityEnvironment::clocksThrottleReasons

The clocks throttle reasons.

CUpti_ActivityEnvironment::@10::@14 CUpti_ActivityEnvironment::cooling

Data returned for CUPTI_ACTIVITY_ENVIRONMENT_COOLING environment kind.

uint32_t CUpti_ActivityEnvironment::deviceId

The ID of the device

CUpti_ActivityEnvironmentKind CUpti_ActivityEnvironment::environmentKind

The kind of data reported in this record.

uint32_t CUpti_ActivityEnvironment::fanSpeed

The fan speed as percentage of maximum.

uint32_t CUpti_ActivityEnvironment::gpuTemperature

The GPU temperature in degrees C.

CUpti_ActivityKind CUpti_ActivityEnvironment::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_ENVIRONMENT.

uint32_t CUpti_ActivityEnvironment::memoryClock

The memory frequency in MHz

uint32_t CUpti_ActivityEnvironment::pcieLinkGen

The PCIe link generation.

uint32_t CUpti_ActivityEnvironment::pcieLinkWidth

The PCIe link width.

CUpti_ActivityEnvironment::@10::@13 CUpti_ActivityEnvironment::power

Data returned for CUPTI_ACTIVITY_ENVIRONMENT_POWER environment kind.

uint32_t CUpti_ActivityEnvironment::power

The power in milliwatts consumed by GPU and associated circuitry.

uint32_t CUpti_ActivityEnvironment::powerLimit

The power in milliwatts that will trigger power management algorithm.

uint32_t CUpti_ActivityEnvironment::smClock

The SM frequency in MHz

CUpti_ActivityEnvironment::@10::@11 CUpti_ActivityEnvironment::speed

Data returned for CUPTI_ACTIVITY_ENVIRONMENT_SPEED environment kind.

CUpti_ActivityEnvironment::@10::@12 CUpti_ActivityEnvironment::temperature

Data returned for CUPTI_ACTIVITY_ENVIRONMENT_TEMPERATURE environment kind.

uint64_t CUpti_ActivityEnvironment::timestamp

The timestamp when this sample was retrieved, in ns. A value of 0 indicates that timestamp information could not be collected for the marker.

3.11. CUpti_ActivityEvent Struct Reference

The activity record for a CUPTI event.

This activity record represents 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.

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.

3.12. CUpti_ActivityEventInstance Struct Reference

The activity record for a CUPTI event with instance information.

This activity record represents the a CUPTI event value for a specific event domain instance (CUPTI_ACTIVITY_KIND_EVENT_INSTANCE). 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. This activity record should be used when event domain instance information needs to be associated with the event.

uint32_t CUpti_ActivityEventInstance::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_ActivityEventInstance::domain

The event domain ID.

CUpti_EventID CUpti_ActivityEventInstance::id

The event ID.

uint32_t CUpti_ActivityEventInstance::instance

The event domain instance.

CUpti_ActivityKind CUpti_ActivityEventInstance::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_EVENT_INSTANCE.

uint32_t CUpti_ActivityEventInstance::pad

Undefined. Reserved for internal use.

uint64_t CUpti_ActivityEventInstance::value

The event value.

3.13. CUpti_ActivityFunction Struct Reference

The activity record for global/device functions.

This activity records function name and corresponding module information. (CUPTI_ACTIVITY_KIND_FUNCTION).

uint32_t CUpti_ActivityFunction::contextId

The ID of the context where the function is launched.

uint32_t CUpti_ActivityFunction::functionIndex

The function's unique symbol index in the module.

uint32_t CUpti_ActivityFunction::id

ID to uniquely identify the record

CUpti_ActivityKind CUpti_ActivityFunction::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_FUNCTION.

uint32_t CUpti_ActivityFunction::moduleId

The module ID in which this global/device function is present.

const char *CUpti_ActivityFunction::name

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

3.14. CUpti_ActivityGlobalAccess Struct Reference

The activity record for source-level global access. (deprecated).

This activity records the locations of the global accesses in the source (CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS). Global access activities are now reported using the CUpti_ActivityGlobalAccess2 activity record.

uint32_t CUpti_ActivityGlobalAccess::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivityGlobalAccess::executed

The number of times this instruction was executed

CUpti_ActivityFlag CUpti_ActivityGlobalAccess::flags

The properties of this global access.

CUpti_ActivityKind CUpti_ActivityGlobalAccess::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS.

uint64_t CUpti_ActivityGlobalAccess::l2_transactions

The total number of 32 bytes transactions to L2 cache generated by this access

uint32_t CUpti_ActivityGlobalAccess::pcOffset

The pc offset for the access.

uint32_t CUpti_ActivityGlobalAccess::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivityGlobalAccess::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

3.15. CUpti_ActivityGlobalAccess2 Struct Reference

The activity record for source-level global access.

This activity records the locations of the global accesses in the source (CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS).

uint32_t CUpti_ActivityGlobalAccess2::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivityGlobalAccess2::executed

The number of times this instruction was executed

CUpti_ActivityFlag CUpti_ActivityGlobalAccess2::flags

The properties of this global access.

uint32_t CUpti_ActivityGlobalAccess2::functionId

Correlation ID with global/device function name

CUpti_ActivityKind CUpti_ActivityGlobalAccess2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS.

uint64_t CUpti_ActivityGlobalAccess2::l2_transactions

The total number of 32 bytes transactions to L2 cache generated by this access

uint32_t CUpti_ActivityGlobalAccess2::pad

Undefined. Reserved for internal use.

uint32_t CUpti_ActivityGlobalAccess2::pcOffset

The pc offset for the access.

uint32_t CUpti_ActivityGlobalAccess2::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivityGlobalAccess2::theoreticalL2Transactions

The minimum number of L2 transactions possible based on the access pattern.

uint64_t CUpti_ActivityGlobalAccess2::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

3.16. CUpti_ActivityInstructionExecution Struct Reference

The activity record for source-level sass/source line-by-line correlation.

This activity records source level sass/source correlation information. (CUPTI_ACTIVITY_KIND_INSTRUCTION_EXECUTION).

uint32_t CUpti_ActivityInstructionExecution::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivityInstructionExecution::executed

The number of times this instruction was executed.

CUpti_ActivityFlag CUpti_ActivityInstructionExecution::flags

The properties of this instruction execution. Check mask CUPTI_ACTIVITY_FLAG_INSTRUCTION_VALUE_INVALID to determine whether threadsExecuted, notPredOffThreadsExecuted and executed are valid for the instruction. Check mask CUPTI_ACTIVITY_FLAG_INSTRUCTION_CLASS_MASK to identify the instruction class. See CUpti_ActivityInstructionClass.

uint32_t CUpti_ActivityInstructionExecution::functionId

Correlation ID with global/device function name

CUpti_ActivityKind CUpti_ActivityInstructionExecution::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_INSTRUCTION_EXECUTION.

uint64_t CUpti_ActivityInstructionExecution::notPredOffThreadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

uint32_t CUpti_ActivityInstructionExecution::pad

Undefined. Reserved for internal use.

uint32_t CUpti_ActivityInstructionExecution::pcOffset

The pc offset for the instruction.

uint32_t CUpti_ActivityInstructionExecution::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivityInstructionExecution::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction, regardless of predicate or condition code.

3.17. CUpti_ActivityKernel Struct Reference

The activity record for kernel. (deprecated).

This activity record represents a kernel execution (CUPTI_ACTIVITY_KIND_KERNEL and CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL) but is no longer generated by CUPTI. Kernel activities are now reported using the CUpti_ActivityKernel3 activity record.

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

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 or CUPTI_ACTIVITY_KIND_CONCURRENT_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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

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.

3.18. CUpti_ActivityKernel2 Struct Reference

The activity record for kernel. (deprecated).

This activity record represents a kernel execution (CUPTI_ACTIVITY_KIND_KERNEL and CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL) but is no longer generated by CUPTI. Kernel activities are now reported using the CUpti_ActivityKernel3 activity record.

int32_t CUpti_ActivityKernel2::blockX

The X-dimension block size for the kernel.

int32_t CUpti_ActivityKernel2::blockY

The Y-dimension block size for the kernel.

int32_t CUpti_ActivityKernel2::blockZ

The Z-dimension grid size for the kernel.

uint64_t CUpti_ActivityKernel2::completed

The completed timestamp for the kernel execution, in ns. It represents the completion of all it's child kernels and the kernel itself. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the completion time is unknown.

uint32_t CUpti_ActivityKernel2::contextId

The ID of the context where the kernel is executing.

uint32_t CUpti_ActivityKernel2::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 or runtime API activity record that launched the kernel.

uint32_t CUpti_ActivityKernel2::deviceId

The ID of the device where the kernel is executing.

int32_t CUpti_ActivityKernel2::dynamicSharedMemory

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

uint64_t CUpti_ActivityKernel2::end

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

uint8_t CUpti_ActivityKernel2::executed

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

int64_t CUpti_ActivityKernel2::gridId

The grid ID of the kernel. Each kernel is assigned a unique grid ID at runtime.

int32_t CUpti_ActivityKernel2::gridX

The X-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel2::gridY

The Y-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel2::gridZ

The Z-dimension grid size for the kernel.

CUpti_ActivityKind CUpti_ActivityKernel2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_KERNEL or CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL.

uint32_t CUpti_ActivityKernel2::localMemoryPerThread

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

uint32_t CUpti_ActivityKernel2::localMemoryTotal

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

const char *CUpti_ActivityKernel2::name

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

uint16_t CUpti_ActivityKernel2::registersPerThread

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

uint8_t CUpti_ActivityKernel2::requested

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

void *CUpti_ActivityKernel2::reserved0

Undefined. Reserved for internal use.

uint8_t CUpti_ActivityKernel2::sharedMemoryConfig

The shared memory configuration used for the kernel. The value is one of the CUsharedconfig enumeration values from cuda.h.

uint64_t CUpti_ActivityKernel2::start

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

int32_t CUpti_ActivityKernel2::staticSharedMemory

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

uint32_t CUpti_ActivityKernel2::streamId

The ID of the stream where the kernel is executing.

3.19. CUpti_ActivityKernel3 Struct Reference

The activity record for a kernel (CUDA 6.5(with sm_52 support) onwards).

This activity record represents a kernel execution (CUPTI_ACTIVITY_KIND_KERNEL and CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL).

int32_t CUpti_ActivityKernel3::blockX

The X-dimension block size for the kernel.

int32_t CUpti_ActivityKernel3::blockY

The Y-dimension block size for the kernel.

int32_t CUpti_ActivityKernel3::blockZ

The Z-dimension grid size for the kernel.

uint64_t CUpti_ActivityKernel3::completed

The completed timestamp for the kernel execution, in ns. It represents the completion of all it's child kernels and the kernel itself. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the completion time is unknown.

uint32_t CUpti_ActivityKernel3::contextId

The ID of the context where the kernel is executing.

uint32_t CUpti_ActivityKernel3::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 or runtime API activity record that launched the kernel.

uint32_t CUpti_ActivityKernel3::deviceId

The ID of the device where the kernel is executing.

int32_t CUpti_ActivityKernel3::dynamicSharedMemory

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

uint64_t CUpti_ActivityKernel3::end

The end timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

uint8_t CUpti_ActivityKernel3::executed

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

int64_t CUpti_ActivityKernel3::gridId

The grid ID of the kernel. Each kernel is assigned a unique grid ID at runtime.

int32_t CUpti_ActivityKernel3::gridX

The X-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel3::gridY

The Y-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel3::gridZ

The Z-dimension grid size for the kernel.

CUpti_ActivityKind CUpti_ActivityKernel3::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_KERNEL or CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL.

uint32_t CUpti_ActivityKernel3::localMemoryPerThread

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

uint32_t CUpti_ActivityKernel3::localMemoryTotal

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

const char *CUpti_ActivityKernel3::name

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

CUpti_ActivityPartitionedGlobalCacheConfig CUpti_ActivityKernel3::partitionedGlobalCacheExecuted

The partitioned global caching executed for the kernel. Partitioned global caching is required to enable caching on certain chips, such as devices with compute capability 5.2. Partitioned global caching can be automatically disabled if the occupancy requirement of the launch cannot support caching.

CUpti_ActivityPartitionedGlobalCacheConfig CUpti_ActivityKernel3::partitionedGlobalCacheRequested

The partitioned global caching requested for the kernel. Partitioned global caching is required to enable caching on certain chips, such as devices with compute capability 5.2.

uint16_t CUpti_ActivityKernel3::registersPerThread

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

uint8_t CUpti_ActivityKernel3::requested

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

void *CUpti_ActivityKernel3::reserved0

Undefined. Reserved for internal use.

uint8_t CUpti_ActivityKernel3::sharedMemoryConfig

The shared memory configuration used for the kernel. The value is one of the CUsharedconfig enumeration values from cuda.h.

uint64_t CUpti_ActivityKernel3::start

The start timestamp for the kernel execution, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the kernel.

int32_t CUpti_ActivityKernel3::staticSharedMemory

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

uint32_t CUpti_ActivityKernel3::streamId

The ID of the stream where the kernel is executing.

3.20. CUpti_ActivityMarker Struct Reference

The activity record providing a marker which is an instantaneous point in time.

The marker is specified with a descriptive name and unique id (CUPTI_ACTIVITY_KIND_MARKER).

CUpti_ActivityFlag CUpti_ActivityMarker::flags

The flags associated with the marker.

See also:

CUpti_ActivityFlag

uint32_t CUpti_ActivityMarker::id

The marker ID.

CUpti_ActivityKind CUpti_ActivityMarker::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MARKER.

const char *CUpti_ActivityMarker::name

The marker name for an instantaneous or start marker. This will be NULL for an end marker.

CUpti_ActivityMarker::objectId

The identifier for the activity object associated with this marker. 'objectKind' indicates which ID is valid for this record.

CUpti_ActivityObjectKind CUpti_ActivityMarker::objectKind

The kind of activity object associated with this marker.

uint64_t CUpti_ActivityMarker::timestamp

The timestamp for the marker, in ns. A value of 0 indicates that timestamp information could not be collected for the marker.

3.21. CUpti_ActivityMarkerData Struct Reference

The activity record providing detailed information for a marker.

The marker data contains color, payload, and category. (CUPTI_ACTIVITY_KIND_MARKER_DATA).

uint32_t CUpti_ActivityMarkerData::category

The category for the marker.

uint32_t CUpti_ActivityMarkerData::color

The color for the marker.

CUpti_ActivityFlag CUpti_ActivityMarkerData::flags

The flags associated with the marker.

See also:

CUpti_ActivityFlag

uint32_t CUpti_ActivityMarkerData::id

The marker ID.

CUpti_ActivityKind CUpti_ActivityMarkerData::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MARKER_DATA.

CUpti_ActivityMarkerData::payload

The payload value.

CUpti_MetricValueKind CUpti_ActivityMarkerData::payloadKind

Defines the payload format for the value associated with the marker.

3.22. CUpti_ActivityMemcpy Struct Reference

The activity record for memory copies.

This activity record represents a memory copy (CUPTI_ACTIVITY_KIND_MEMCPY).

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

uint32_t CUpti_ActivityMemcpy::streamId

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

3.23. CUpti_ActivityMemcpy2 Struct Reference

The activity record for peer-to-peer memory copies.

This activity record represents a peer-to-peer memory copy (CUPTI_ACTIVITY_KIND_MEMCPY2).

uint64_t CUpti_ActivityMemcpy2::bytes

The number of bytes transferred by the memory copy.

uint32_t CUpti_ActivityMemcpy2::contextId

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

uint8_t CUpti_ActivityMemcpy2::copyKind

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

See also:

CUpti_ActivityMemcpyKind

uint32_t CUpti_ActivityMemcpy2::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 and runtime API activity record that launched the memory copy.

uint32_t CUpti_ActivityMemcpy2::deviceId

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

uint32_t CUpti_ActivityMemcpy2::dstContextId

The ID of the context owning the memory being copied to.

uint32_t CUpti_ActivityMemcpy2::dstDeviceId

The ID of the device where memory is being copied to.

uint8_t CUpti_ActivityMemcpy2::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_ActivityMemcpy2::end

The end timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

uint8_t CUpti_ActivityMemcpy2::flags

The flags associated with the memory copy.

See also:

CUpti_ActivityFlag

CUpti_ActivityKind CUpti_ActivityMemcpy2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MEMCPY2.

uint32_t CUpti_ActivityMemcpy2::pad

Undefined. Reserved for internal use.

void *CUpti_ActivityMemcpy2::reserved0

Undefined. Reserved for internal use.

uint32_t CUpti_ActivityMemcpy2::srcContextId

The ID of the context owning the memory being copied from.

uint32_t CUpti_ActivityMemcpy2::srcDeviceId

The ID of the device where memory is being copied from.

uint8_t CUpti_ActivityMemcpy2::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_ActivityMemcpy2::start

The start timestamp for the memory copy, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory copy.

uint32_t CUpti_ActivityMemcpy2::streamId

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

3.24. CUpti_ActivityMemset Struct Reference

The activity record for memset.

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

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory set.

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. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the memory set.

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.

3.25. CUpti_ActivityMetric Struct Reference

The activity record for a CUPTI metric.

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.

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.

uint8_t CUpti_ActivityMetric::flags

The properties of this metric.

See also:

CUpti_ActivityFlag

CUpti_MetricID CUpti_ActivityMetric::id

The metric ID.

CUpti_ActivityKind CUpti_ActivityMetric::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_METRIC.

uint8_t CUpti_ActivityMetric::pad

Undefined. Reserved for internal use.

CUpti_ActivityMetric::value

The metric value.

3.26. CUpti_ActivityMetricInstance Struct Reference

The activity record for a CUPTI metric with instance information. This activity record represents a CUPTI metric value for a specific metric domain instance (CUPTI_ACTIVITY_KIND_METRIC_INSTANCE). 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. This activity record should be used when metric domain instance information needs to be associated with the metric.

uint32_t CUpti_ActivityMetricInstance::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.

uint8_t CUpti_ActivityMetricInstance::flags

The properties of this metric.

See also:

CUpti_ActivityFlag

CUpti_MetricID CUpti_ActivityMetricInstance::id

The metric ID.

uint32_t CUpti_ActivityMetricInstance::instance

The metric domain instance.

CUpti_ActivityKind CUpti_ActivityMetricInstance::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_METRIC_INSTANCE.

uint8_t CUpti_ActivityMetricInstance::pad

Undefined. Reserved for internal use.

CUpti_ActivityMetricInstance::value

The metric value.

3.27. CUpti_ActivityModule Struct Reference

The activity record for a CUDA module.

This activity record represents a CUDA module (CUPTI_ACTIVITY_KIND_MODULE). 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 module data from the module callback may choose to use this type to store the collected module data.

uint32_t CUpti_ActivityModule::contextId

The ID of the context where the module is loaded.

const void *CUpti_ActivityModule::cubin

The pointer to cubin.

uint32_t CUpti_ActivityModule::cubinSize

The cubin size.

uint32_t CUpti_ActivityModule::id

The module ID.

CUpti_ActivityKind CUpti_ActivityModule::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MODULE.

uint32_t CUpti_ActivityModule::pad

Undefined. Reserved for internal use.

3.28. CUpti_ActivityName Struct Reference

The activity record providing a name.

This activity record provides a name for a device, context, thread, etc. (CUPTI_ACTIVITY_KIND_NAME).

CUpti_ActivityKind CUpti_ActivityName::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_NAME.

const char *CUpti_ActivityName::name

The name.

CUpti_ActivityName::objectId

The identifier for the activity object. 'objectKind' indicates which ID is valid for this record.

CUpti_ActivityObjectKind CUpti_ActivityName::objectKind

The kind of activity object being named.

3.29. CUpti_ActivityObjectKindId Union Reference

Identifiers for object kinds as specified by CUpti_ActivityObjectKind.

See also:

CUpti_ActivityObjectKind

CUpti_ActivityObjectKindId::@1 CUpti_ActivityObjectKindId::dcs

A device object requires that we identify the device ID. A context object requires that we identify both the device and context ID. A stream object requires that we identify device, context, and stream ID.

CUpti_ActivityObjectKindId::@0 CUpti_ActivityObjectKindId::pt

A process object requires that we identify the process ID. A thread object requires that we identify both the process and thread ID.

3.30. CUpti_ActivityOverhead Struct Reference

The activity record for CUPTI and driver overheads.

This activity record provides CUPTI and driver overhead information (CUPTI_ACTIVITY_OVERHEAD).

uint64_t CUpti_ActivityOverhead::end

The end timestamp for the overhead, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the overhead.

CUpti_ActivityKind CUpti_ActivityOverhead::kind

The activity record kind, must be CUPTI_ACTIVITY_OVERHEAD.

CUpti_ActivityOverhead::objectId

The identifier for the activity object. 'objectKind' indicates which ID is valid for this record.

CUpti_ActivityObjectKind CUpti_ActivityOverhead::objectKind

The kind of activity object that the overhead is associated with.

CUpti_ActivityOverheadKind CUpti_ActivityOverhead::overheadKind

The kind of overhead, CUPTI, DRIVER, COMPILER etc.

uint64_t CUpti_ActivityOverhead::start

The start timestamp for the overhead, in ns. A value of 0 for both the start and end timestamps indicates that timestamp information could not be collected for the overhead.

3.31. CUpti_ActivityPreemption Struct Reference

The activity record for a preemption of a CDP kernel.

This activity record represents a preemption of a CDP kernel.

uint32_t CUpti_ActivityPreemption::blockX

The X-dimension of the block that is preempted

uint32_t CUpti_ActivityPreemption::blockY

The Y-dimension of the block that is preempted

uint32_t CUpti_ActivityPreemption::blockZ

The Z-dimension of the block that is preempted

int64_t CUpti_ActivityPreemption::gridId

The grid-id of the block that is preempted

CUpti_ActivityKind CUpti_ActivityPreemption::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_PREEMPTION

uint32_t CUpti_ActivityPreemption::pad

Undefined. Reserved for internal use.

CUpti_ActivityPreemptionKind CUpti_ActivityPreemption::preemptionKind

kind of the preemption

uint64_t CUpti_ActivityPreemption::timestamp

The timestamp of the preemption, in ns. A value of 0 indicates that timestamp information could not be collected for the preemption.

3.32. CUpti_ActivitySharedAccess Struct Reference

The activity record for source-level shared access.

This activity records the locations of the shared accesses in the source (CUPTI_ACTIVITY_KIND_SHARED_ACCESS).

uint32_t CUpti_ActivitySharedAccess::correlationId

The correlation ID of the kernel to which this result is associated.

uint32_t CUpti_ActivitySharedAccess::executed

The number of times this instruction was executed

CUpti_ActivityFlag CUpti_ActivitySharedAccess::flags

The properties of this shared access.

uint32_t CUpti_ActivitySharedAccess::functionId

Correlation ID with global/device function name

CUpti_ActivityKind CUpti_ActivitySharedAccess::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_SHARED_ACCESS.

uint32_t CUpti_ActivitySharedAccess::pad

Undefined. Reserved for internal use.

uint32_t CUpti_ActivitySharedAccess::pcOffset

The pc offset for the access.

uint64_t CUpti_ActivitySharedAccess::sharedTransactions

The total number of shared memory transactions generated by this access

uint32_t CUpti_ActivitySharedAccess::sourceLocatorId

The ID for source locator.

uint64_t CUpti_ActivitySharedAccess::theoreticalSharedTransactions

The minimum number of shared memory transactions possible based on the access pattern.

uint64_t CUpti_ActivitySharedAccess::threadsExecuted

This increments each time when this instruction is executed by number of threads that executed this instruction with predicate and condition code evaluating to true.

3.33. CUpti_ActivitySourceLocator Struct Reference

The activity record for source locator.

This activity record represents a source locator (CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR).

const char *CUpti_ActivitySourceLocator::fileName

The path for the file.

uint32_t CUpti_ActivitySourceLocator::id

The ID for the source path, will be used in all the source level results.

CUpti_ActivityKind CUpti_ActivitySourceLocator::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR.

uint32_t CUpti_ActivitySourceLocator::lineNumber

The line number in the source .

3.34. CUpti_ActivityUnifiedMemoryCounter Struct Reference

The activity record for Unified Memory counters.

This activity record represents a Unified Memory counter (CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER).

CUpti_ActivityUnifiedMemoryCounterKind CUpti_ActivityUnifiedMemoryCounter::counterKind

The Unified Memory counter kind. See /ref CUpti_ActivityUnifiedMemoryCounterKind

uint32_t CUpti_ActivityUnifiedMemoryCounter::deviceId

The ID of the device involved in the memory transfer operation. It is not relevant if the scope of the counter is global (all devices).

CUpti_ActivityKind CUpti_ActivityUnifiedMemoryCounter::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER

uint32_t CUpti_ActivityUnifiedMemoryCounter::pad

Undefined. Reserved for internal use.

uint32_t CUpti_ActivityUnifiedMemoryCounter::processId

The ID of the process to which this record belongs to. In case of global scope, processId is undefined.

CUpti_ActivityUnifiedMemoryCounterScope CUpti_ActivityUnifiedMemoryCounter::scope

Scope of the Unified Memory counter. See /ref CUpti_ActivityUnifiedMemoryCounterScope

uint64_t CUpti_ActivityUnifiedMemoryCounter::timestamp

The timestamp when this sample was retrieved, in ns. A value of 0 indicates that timestamp information could not be collected

uint64_t CUpti_ActivityUnifiedMemoryCounter::value

Value of the counter

3.35. CUpti_ActivityUnifiedMemoryCounterConfig Struct Reference

Unified Memory counters configuration structure.

This structure controls the enable/disable of the various Unified Memory counters consisting of scope, kind and other parameters. See function /ref cuptiActivityConfigureUnifiedMemoryCounter

uint32_t CUpti_ActivityUnifiedMemoryCounterConfig::deviceId

Device id of the traget device. This is relevant only for single device scopes.

uint32_t CUpti_ActivityUnifiedMemoryCounterConfig::enable

Control to enable/disable the counter. To enable the counter set it to non-zero value while disable is indicated by zero.

CUpti_ActivityUnifiedMemoryCounterKind CUpti_ActivityUnifiedMemoryCounterConfig::kind

Unified Memory counter Counter kind

CUpti_ActivityUnifiedMemoryCounterScope CUpti_ActivityUnifiedMemoryCounterConfig::scope

Unified Memory counter Counter scope

3.36. CUpti_CallbackData Struct Reference

Data passed into a runtime or driver API callback function.

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.

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 driver and runtime launch callbacks, where it returns the name of the kernel.

3.37. CUpti_EventGroupSet Struct Reference

A set of event groups.

A set of event groups. When returned by cuptiEventGroupSetsCreate and cuptiMetricCreateEventGroupSets a set indicates that event groups that can be enabled at the same time (i.e. all the events in the set can be collected simultaneously).

CUpti_EventGroup *CUpti_EventGroupSet::eventGroups

An array of `numEventGroups` event groups.

uint32_t CUpti_EventGroupSet::numEventGroups

The number of event groups in the set.

3.38. CUpti_EventGroupSets Struct Reference

A set of event group sets.

A set of event group sets. When returned by cuptiEventGroupSetsCreate and cuptiMetricCreateEventGroupSets a CUpti_EventGroupSets indicates the number of passes required to collect all the events, and the event groups that should be collected during each pass.

uint32_t CUpti_EventGroupSets::numSets

Number of event group sets.

CUpti_EventGroupSet *CUpti_EventGroupSets::sets

An array of `numSets` event group sets.

3.39. CUpti_MetricValue Union Reference

A metric value.

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.

3.40. CUpti_ModuleResourceData Struct Reference

Module data passed into a resource callback function.

CUDA module 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 module 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.

size_t CUpti_ModuleResourceData::cubinSize

The size of the cubin.

uint32_t CUpti_ModuleResourceData::moduleId

Identifier to associate with the CUDA module.

const char *CUpti_ModuleResourceData::pCubin

Pointer to the associated cubin.

3.41. CUpti_NvtxData Struct Reference

Data passed into a NVTX callback function.

Data passed into a NVTX callback function as the `cbdata` argument to CUpti_CallbackFunc. The `cbdata` will be this type for `domain` equal to CUPTI_CB_DOMAIN_NVTX. Unless otherwise notes, 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.

const char *CUpti_NvtxData::functionName

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

const void *CUpti_NvtxData::functionParams

Pointer to the arguments passed to the NVTX API call. See generated_nvtx_meta.h for structure definitions for the parameters for each NVTX API function.

3.42. CUpti_ResourceData Struct Reference

Data passed into a resource callback function.

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.

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.

3.43. CUpti_SynchronizeData Struct Reference

Data passed into a synchronize callback function.

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.

CUcontext CUpti_SynchronizeData::context

The context of the stream being synchronized.

CUstream CUpti_SynchronizeData::stream

The stream being synchronized.

Chapter 4. DATA FIELDS

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field: