WHAT'S NEW

CUPTI contains below changes as part of the CUDA Toolkit 9.2 release.

‣ Added support to query PCI devices information which can be used to construct the PCIE topology. See activity kind `CUPTI_ACTIVITY_KIND_PCIE` and related activity record `CUpti_ActivityPcie`.

‣ To view and analyze bandwidth of memory transfers over PCIe topologies, new set of metrics to collect total data bytes transmitted and received through PCIe are added. Those give accumulated count for all devices in the system. These metrics are collected at the device level for the entire application. And those are made available for devices with compute capability 5.2 and higher.

‣ CUPTI added support for new metrics:
  ➤ Instruction executed for different types of load and store
  ➤ Total number of cached global/local load requests from SM to texture cache
  ➤ Global atomic/non-atomic/reduction bytes written to L2 cache from texture cache
  ➤ Surface atomic/non-atomic/reduction bytes written to L2 cache from texture cache
  ➤ Hit rate at L2 cache for all requests from texture cache
  ➤ Device memory (DRAM) read and write bytes
  ➤ The utilization level of the multiprocessor function units that execute tensor core instructions for devices with compute capability 7.0

‣ A new attribute `CUPTI_EVENT_ATTR_PROFILING_SCOPE` is added under enum `CUpti_EventAttribute` to query the profiling scope of an event. Profiling scope indicates if the event can be collected at the context level or device level or both. See Enum `CUpti_EventProfilingScope` for available profiling scopes.

‣ A new error code `CUPTI_ERROR_VIRTUALIZED_DEVICE_NOT_SUPPORTED` is added to indicate that tracing and profiling on virtualized GPU is not supported.
# TABLE OF CONTENTS

Chapter 1. Usage.................................................................................................................. 1
  1.1. CUPTI Compatibility and Requirements................................................................. 1
  1.2. CUPTI Initialization............................................................................................. 1
  1.3. CUPTI Activity API............................................................................................. 1
    1.3.1. SASS Source Correlation............................................................................. 2
    1.3.2. PC Sampling........................................................................................... 3
    1.3.3. NVLink................................................................................................. 4
    1.3.4. OpenACC.............................................................................................. 5
    1.3.5. External Correlation................................................................................ 5
  1.4. CUPTI Callback API............................................................................................ 6
    1.4.1. Driver and Runtime API Callbacks............................................................... 7
    1.4.2. Resource Callbacks.................................................................................. 8
    1.4.3. Synchronization Callbacks.......................................................................... 8
    1.4.4. NVIDIA Tools Extension Callbacks................................................................. 8
  1.5. CUPTI Event API................................................................................................ 10
    1.5.1. Collecting Kernel Execution Events............................................................. 12
    1.5.2. Sampling Events.................................................................................... 13
  1.6. CUPTI Metric API............................................................................................... 13
    1.6.1. Metrics Reference.................................................................................. 15
      1.6.1.1. Metrics for Capability 3.x................................................................. 15
      1.6.1.2. Metrics for Capability 5.x................................................................. 22
      1.6.1.3. Metrics for Capability 6.x................................................................. 23
      1.6.1.4. Metrics for Capability 7.0................................................................. 32
  1.7. Samples........................................................................................................ 41

Chapter 2. Modules......................................................................................................... 42
  2.1. CUPTI Version.................................................................................................. 42
    cuptiGetVersion............................................................................................. 42
    CUPTI_API_VERSION...................................................................................... 43
  2.2. CUPTI Result Codes......................................................................................... 43
    CUptiResult................................................................................................... 43
    cuptiGetResultString........................................................................................ 45
  2.3. CUPTI Activity API............................................................................................ 46
    CUpti_Activity............................................................................................... 47
    CUpti_ActivityAPI........................................................................................... 47
    CUpti_ActivityAutoBoostState.......................................................................... 47
    CUpti_ActivityBranch....................................................................................... 47
    CUpti_ActivityBranch2..................................................................................... 47
    CUpti_ActivityCdpKernel.................................................................................. 47
    CUpti_ActivityContext..................................................................................... 47
    CUpti_ActivityCudaEvent................................................................................. 47
CUpti_CallbackDomain

CUpti_CallbackIdResource

CUpti_CallbackIdSync

CUpti_CallbackFunc

CUpti_CallbackId

CUpti_DomainTable

CUpti_SubscriberHandle

cuptiEnableAllDomains

cuptiEnableCallback

cuptiEnableDomain

cuptiGetCallbackName

cuptiGetCallbackState

cuptiSubscribe

cuptiSupportedDomains

cuptiUnsubscribe

2.5. CUPTI Event API

CUpti_EventGroupSet

CUpti_EventGroupSets

CUpti_DeviceAttribute

CUpti_DeviceAttributeDeviceClass

CUpti_EventAttribute

CUpti_EventCategory

CUpti_EventCollectionMethod

CUpti_EventCollectionMode

CUpti_EventDomainAttribute

CUpti_EventGroupAttribute

CUpti_EventProfilingScope

CUpti_ReadEventFlags

CUpti_EventDomainID

CUpti_EventGroup

CUpti_EventID

CUpti_KernelReplayUpdateFunc

cuptiDeviceEnumEventDomains

cuptiDeviceGetAttribute

cuptiDeviceGetEventDomainAttribute

cuptiDeviceGetNumEventDomains

cuptiDeviceGetTimestamp

cuptiDisableKernelReplayMode

cuptiEnableKernelReplayMode

cuptiEnumEventDomains

cuptiEventDomainEnumEvents

cuptiEventDomainGetAttribute

cuptiEventDomainGetNumEvents
2.6. CUPTI Metric API

CUpti_MetricValue
CUpti_MetricAttribute
CUpti_MetricCategory
CUpti_MetricEvaluationMode
CUpti_MetricPropertyDeviceClass
CUpti_MetricPropertyID
CUpti_MetricValueKind
CUpti_MetricValueUtilizationLevel
CUpti_MetricID

cuptiDeviceEnumMetrics
cuptiDeviceGetNumMetrics
cuptiEnumMetrics
cuptiGetNumMetrics
cuptiMetricCreateEventGroupSets
cuptiMetricEnumEvents
cuptiMetricEnumProperties
cuptiMetricGetAttribute
cuptiMetricGetIdFromName
cuptiMetricGetNumEvents
cuptiMetricGetNumProperties........................................................................... 146

cuptiMetricGetRequiredEventGroupSets............................................................ 147

cuptiMetricGetValue...................................................................................... 148

cuptiMetricGetValue2.................................................................................... 149

Chapter 3. Data Structures................................................................................. 152

CUpti_Activity................................................................................................ 155

kind.......................................................................................................... 156

CUpti_ActivityAPI............................................................................................ 156

cbid................................................................. 156
correlationId.................................................... 156
diverged........................................................ 156

end.......................................................................................................... 156

kind................................................................. 156

processId......................................................... 156

returnValue...................................................... 156

start................................................................. 157

threadId........................................................ 157

CUpti_ActivityAutoBoostState........................................................................ 157

enabled........................................................ 157

pid........................................................................................................ 157

CUpti_ActivityBranch..................................................................................... 157

correlationId.................................................... 157

diverged........................................................ 157

eated............................................................................................... 157

kind.......................................................................................................... 158

pcOffset........................................................ 158

sourceLocatorId.............................................. 158

threadsExecuted.............................................. 158

CUpti_ActivityBranch2..................................................................................... 158

correlationId.................................................... 158

diverged........................................................ 158

executed............................................................................................... 158

functionId...................................................... 158

kind.......................................................................................................... 159

pad.......................................................................................................... 159

pcOffset........................................................ 159

sourceLocatorId.............................................. 159

threadsExecuted.............................................. 159

CUpti_ActivityCdpKernel................................................................................... 159

blockX........................................................ 159

blockY........................................................ 159

blockZ........................................................ 159

completed....................................................... 159

contextId....................................................... 160
correlationId................................................................. 160
deviceId ............................................................... 160
dynamicSharedMemory ........................................... 160
end ............................................................................ 160
executed ..................................................................... 160
gridId ........................................................................ 160
gridX ................................................................. 160
gridY ........................................................................ 160
gridZ ........................................................................ 160
kind ........................................................................... 161
localMemoryPerThread ........................................... 161
localMemoryTotal ................................................. 161
name ......................................................................... 161
parentBlockX ....................................................... 161
parentBlockY ....................................................... 161
parentBlockZ ....................................................... 161
parentGridId ........................................................... 161
queued .................................................................... 161
registersPerThread .............................................. 161
requested ................................................................... 161
sharedMemoryConfig ............................................ 162
start ......................................................................... 162
staticSharedMemory ............................................. 162
streamId .................................................................. 162
submitted .................................................................. 162
CUpti_ActivityContext ........................................... 162
computeApiKind .................................................... 162
contextId ............................................................ 163
deviceId ............................................................. 163
kind ....................................................................... 163
nullStreamId .......................................................... 163
CUpti_ActivityCudaEvent ........................................ 163
contextId ............................................................ 163
correlationId ........................................................ 163
eventId ................................................................... 163
kind ....................................................................... 163
pad .......................................................................... 163
streamId .............................................................. 164
CUpti_ActivityDevice .............................................. 164
computeCapabilityMajor ..................................... 164
computeCapabilityMinor .................................... 164
constantMemorySize ........................................... 164
coreClockRate ...................................................... 164
flags ................................................................. 164
globalMemoryBandwidth .................................................. 164
globalMemorySize .......................................................... 164
id ................................................................. 165
kind ............................................................. 165
l2CacheSize .......................................................... 165
maxBlockDimX ......................................................... 165
maxBlockDimY ......................................................... 165
maxBlockDimZ ......................................................... 165
maxBlocksPerMultiprocessor ......................................... 165
maxGridDimX ........................................................ 165
maxGridDimY ........................................................ 165
maxGridDimZ ........................................................ 165
maxIPC ............................................................. 165
maxRegistersPerBlock .................................................. 166
maxSharedMemoryPerBlock ............................................ 166
maxThreadsPerBlock ................................................... 166
maxWarpsPerMultiprocessor ......................................... 166
name ............................................................. 166
numMemcpyEngines ..................................................... 166
numMultiprocessors .................................................... 166
numThreadsPerWarp ................................................... 166
CUpti_ActivityDevice2 .................................................. 166
computeCapabilityMajor ............................................... 167
computeCapabilityMinor ............................................... 167
constantMemorySize .................................................. 167
coreClockRate ........................................................ 167
eccEnabled ............................................................ 167
flags ................................................................. 167
globalMemoryBandwidth .................................................. 167
globalMemorySize ........................................................ 167
id ................................................................. 167
kind ............................................................. 168
l2CacheSize .......................................................... 168
maxBlockDimX ......................................................... 168
maxBlockDimY ......................................................... 168
maxBlockDimZ ......................................................... 168
maxBlocksPerMultiprocessor ......................................... 168
maxGridDimX ........................................................ 168
maxGridDimY ........................................................ 168
maxGridDimZ ........................................................ 168
maxIPC ............................................................. 168
maxRegistersPerBlock .................................................. 168

domain...................................................................................................... 174
id.............................................................................................................174
instance.....................................................................................................174
kind..........................................................................................................174
pad.......................................................................................................... 174
value........................................................................................................174
CUpti_ActivityExternalCorrelation........................................................................ 174
correlationId............................................................................................... 175
externalId...................................................................................................175
externalKind................................................................................................175
kind..........................................................................................................175
reserved.................................................................................................... 175
CUpti_ActivityFunction......................................................................................175
contextId................................................................................................... 175
functionIndex.............................................................................................. 175
id.............................................................................................................176
kind..........................................................................................................176
moduleId....................................................................................................176
name........................................................................................................ 176
CUpti_ActivityGlobalAccess.................................................................................176
correlationId............................................................................................... 176
executed....................................................................................................176
flags......................................................................................................... 176
kind..........................................................................................................176
l2_transactions............................................................................................ 177
pcOffset.....................................................................................................177
sourceLocatorId............................................................................................177
threadsExecuted...........................................................................................177
CUpti_ActivityGlobalAccess2............................................................................... 177
correlationId............................................................................................... 177
executed....................................................................................................177
flags......................................................................................................... 177
functionId...................................................................................................177
kind..........................................................................................................178
l2_transactions............................................................................................ 178
pad.......................................................................................................... 178
pcOffset.....................................................................................................178
sourceLocatorId............................................................................................178
theoreticalL2Transactions................................................................................178
threadsExecuted...........................................................................................178
CUpti_ActivityGlobalAccess3............................................................................... 178
correlationId............................................................................................... 178
executed....................................................................................................178
flags......................................................................................................... 179
functionId...................................................................................................179
kind..........................................................................................................179
l2_transactions............................................................................................ 179
pcOffset.....................................................................................................179
sourceLocatorId............................................................................................179
theoreticalL2Transactions................................................................................179
threadsExecuted...........................................................................................179

CUpti_ActivityInstantaneousEvent.....................................................................180

deviceld......................................................................................................180
id.............................................................................................................180
kind..........................................................................................................180
reserved.................................................................................................... 180
timestamp.................................................................................................. 180
value........................................................................................................ 180

CUpti_ActivityInstantaneousEventInstance.........................................................181

deviceld......................................................................................................181
id.............................................................................................................181
instance.....................................................................................................181
kind..........................................................................................................181
pad.......................................................................................................... 181
timestamp.................................................................................................. 182
value........................................................................................................ 182

CUpti_ActivityInstantaneousMetric....................................................................182

deviceld......................................................................................................182
flags......................................................................................................... 182
id.............................................................................................................182
kind..........................................................................................................182
pad.......................................................................................................... 183
timestamp.................................................................................................. 183
value........................................................................................................ 183

CUpti_ActivityInstantaneousMetricInstance.........................................................183

deviceld......................................................................................................183
flags......................................................................................................... 183
id.............................................................................................................183
instance.....................................................................................................183
kind..........................................................................................................184
pad.......................................................................................................... 184
timestamp.................................................................................................. 184
value........................................................................................................ 184

CUpti_ActivityInstructionCorrelation.................................................................184

flags......................................................................................................... 184
functionId...................................................................................................184
kind.......................................................................................................... 185
pad.......................................................................................................... 185
pcOffset.....................................................................................................185
sourceLocatorId............................................................................................185
CUppti_ActivityInstructionExecution................................................................... 185
correlationId............................................................................................... 185
executed....................................................................................................185
flags......................................................................................................... 185
functionId...................................................................................................186
kind..........................................................................................................186
notPredOffThreadsExecuted.............................................................................186
pad.......................................................................................................... 186
pcOffset.....................................................................................................186
sourceLocatorId............................................................................................186
threadsExecuted...........................................................................................186
CUppti_ActivityKernel......................................................................................... 186
blockX....................................................................................................... 187
blockY....................................................................................................... 187
blockZ....................................................................................................... 187
cacheConfigExecuted..................................................................................... 187
cacheConfigRequested....................................................................................187
correlationId............................................................................................... 187
deviceld.....................................................................................................187
dynamicSharedMemory................................................................................... 187
dude......................................................................................................... 187
gridX.........................................................................................................188
gidY.........................................................................................................188
gidZ.........................................................................................................188
kind..........................................................................................................188
localMemoryPerThread................................................................................... 188
localMemoryTotal..........................................................................................188
name........................................................................................................ 188
pad.......................................................................................................... 188
registersPerThread........................................................................................ 188
reserved0................................................................................................... 188
runtimeCorrelationId..................................................................................... 189
start......................................................................................................... 189
streamId.................................................................................................... 189
CUppti_ActivityKernel2....................................................................................... 189
blockX....................................................................................................... 189
blockY....................................................................................................... 189
partitionedGlobalCacheRequested......................................................... 194
registersPerThread.................................................................................. 194
requested............................................................................................... 194
reserved0.............................................................................................. 194
sharedMemoryConfig.............................................................................. 194
start......................................................................................................... 195
staticSharedMemory............................................................................... 195
streamId................................................................................................. 195
CUpti_ActivityKernel4................................................................................ 195
blockX..................................................................................................... 195
blockY..................................................................................................... 195
blockZ..................................................................................................... 195
cacheConfig............................................................................................ 195
completed............................................................................................... 195
correlationId......................................................................................... 196
deviceld................................................................................................... 196
dynamicSharedMemory............................................................................. 196
end.......................................................................................................... 196
executed................................................................................................. 196
gridId...................................................................................................... 196
gridX....................................................................................................... 196
gridY....................................................................................................... 196
gridZ....................................................................................................... 196
isSharedMemoryCarveoutRequested....................................................... 197
kind....................................................................................................... 197
launchType............................................................................................. 197
localMemoryPerThread.......................................................................... 197
localMemoryTotal................................................................................... 197
name....................................................................................................... 197
partitionedGlobalCacheExecuted............................................................. 197
partitionedGlobalCacheRequested........................................................ 198
queued................................................................................................... 198
registersPerThread............................................................................... 198
requested............................................................................................... 198
reserved0.............................................................................................. 198
sharedMemoryCarveoutRequested.......................................................... 198
sharedMemoryConfig............................................................................. 198
start......................................................................................................... 199
staticSharedMemory............................................................................... 199
streamId................................................................................................. 199
submitted............................................................................................... 199
CUpti_ActivityMarker........................................................................................ 199
    flags......................................................................................................... 199
    id.............................................................................................................199
    kind..........................................................................................................199
    name........................................................................................................ 200
    objectId.....................................................................................................200
    objectKind..................................................................................................200
    timestamp.................................................................................................. 200
CUpti_ActivityMarker2.......................................................................................200
    domain...................................................................................................... 200
    flags......................................................................................................... 200
    id.............................................................................................................200
    kind..........................................................................................................201
    name........................................................................................................ 201
    objectId.....................................................................................................201
    objectKind..................................................................................................201
    pad.......................................................................................................... 201
    timestamp.................................................................................................. 201
CUpti_ActivityMarkerData.................................................................................. 201
    category.................................................................................................... 201
    color.........................................................................................................201
    flags......................................................................................................... 202
    id.............................................................................................................202
    kind..........................................................................................................202
    payload..................................................................................................... 202
    payloadKind................................................................................................ 202
CUpti_ActivityMemcpy...................................................................................... 202
    bytes........................................................................................................ 202
    contextId................................................................................................... 202
    copyKind.................................................................................................... 202
    correlationId............................................................................................... 203
    deviceId.....................................................................................................203
    dstKind...................................................................................................... 203
    end.......................................................................................................... 203
    flags......................................................................................................... 203
    kind..........................................................................................................203
    reserved0................................................................................................... 203
    runtimeCorrelationId..................................................................................... 203
    srcKind...................................................................................................... 203
    start......................................................................................................... 204
    streamId.................................................................................................... 204
CUpti_ActivityMemcpy2..................................................................................... 204
    bytes........................................................................................................ 204
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextId</td>
<td>204</td>
</tr>
<tr>
<td>copyKind</td>
<td>204</td>
</tr>
<tr>
<td>correlationId</td>
<td>204</td>
</tr>
<tr>
<td>deviceld</td>
<td>205</td>
</tr>
<tr>
<td>dstContextId</td>
<td>205</td>
</tr>
<tr>
<td>dstDeviceId</td>
<td>205</td>
</tr>
<tr>
<td>dstKind</td>
<td>205</td>
</tr>
<tr>
<td>end</td>
<td>205</td>
</tr>
<tr>
<td>flags</td>
<td>205</td>
</tr>
<tr>
<td>kind</td>
<td>205</td>
</tr>
<tr>
<td>pad</td>
<td>206</td>
</tr>
<tr>
<td>reserved0</td>
<td>206</td>
</tr>
<tr>
<td>srcContextId</td>
<td>206</td>
</tr>
<tr>
<td>srcDeviceId</td>
<td>206</td>
</tr>
<tr>
<td>srcKind</td>
<td>206</td>
</tr>
<tr>
<td>start</td>
<td>206</td>
</tr>
<tr>
<td>streamId</td>
<td>206</td>
</tr>
<tr>
<td>CUpti_ActivityMemory</td>
<td>206</td>
</tr>
<tr>
<td>address</td>
<td>206</td>
</tr>
<tr>
<td>allocPC</td>
<td>207</td>
</tr>
<tr>
<td>bytes</td>
<td>207</td>
</tr>
<tr>
<td>contextId</td>
<td>207</td>
</tr>
<tr>
<td>deviceld</td>
<td>207</td>
</tr>
<tr>
<td>end</td>
<td>207</td>
</tr>
<tr>
<td>freePC</td>
<td>207</td>
</tr>
<tr>
<td>kind</td>
<td>207</td>
</tr>
<tr>
<td>memoryKind</td>
<td>207</td>
</tr>
<tr>
<td>name</td>
<td>207</td>
</tr>
<tr>
<td>processId</td>
<td>207</td>
</tr>
<tr>
<td>start</td>
<td>207</td>
</tr>
<tr>
<td>CUpti_ActivityMemset</td>
<td>208</td>
</tr>
<tr>
<td>bytes</td>
<td>208</td>
</tr>
<tr>
<td>contextId</td>
<td>208</td>
</tr>
<tr>
<td>correlationId</td>
<td>208</td>
</tr>
<tr>
<td>deviceld</td>
<td>208</td>
</tr>
<tr>
<td>end</td>
<td>208</td>
</tr>
<tr>
<td>flags</td>
<td>208</td>
</tr>
<tr>
<td>kind</td>
<td>209</td>
</tr>
<tr>
<td>memoryKind</td>
<td>209</td>
</tr>
<tr>
<td>reserved0</td>
<td>209</td>
</tr>
<tr>
<td>start</td>
<td>209</td>
</tr>
<tr>
<td>streamId</td>
<td>209</td>
</tr>
<tr>
<td>value</td>
<td>209</td>
</tr>
</tbody>
</table>
CUpti_ActivityPCSamplingRecordInfo..................................................................... 230
   correlationId............................................................................................... 230
droppedSamples........................................................................................... 230
   kind..........................................................................................................230
   samplingPeriodInCycles..............................................................................230
totalSamples............................................................................................... 230

CUpti_ActivityPreemption.............................................................................. 230
   blockX....................................................................................................... 231
   blockY....................................................................................................... 231
   blockZ....................................................................................................... 231
gridId.......................................................................................................... 231
   kind..........................................................................................................231
   pad.......................................................................................................... 231
   preemptionKind..........................................................................................231
timestamp................................................................................................... 231

CUpti_ActivitySharedAccess........................................................................... 231
   correlationId............................................................................................... 232
excecuted....................................................................................................232
   flags......................................................................................................... 232
   functionId.................................................................................................232
   kind..........................................................................................................232
   pad.......................................................................................................... 232
   pcOffset.....................................................................................................232
   sharedTransactions........................................................................................232
   sourceLocatorId............................................................................................232
   theoreticalSharedTransactions...................................................................... 232
   threadsExecuted...........................................................................................233

CUpti_ActivitySourceLocator......................................................................... 233
   fileName.................................................................................................... 233
   id.............................................................................................................233
   kind..........................................................................................................233
   lineNumber.................................................................................................233

CUpti_ActivityStream..................................................................................... 233
   contextId................................................................................................... 233
   correlationId............................................................................................... 233
   flag.......................................................................................................... 234
   kind..........................................................................................................234
   priority......................................................................................................234
   streamId.................................................................................................... 234

CUpti_ActivitySynchronization..................................................................... 234
   contextId................................................................................................... 234
   correlationId............................................................................................... 234
LIST OF TABLES

Table 1  Capability 3.x Metrics ................................................................................15
Table 2  Capability 6.x Metrics ................................................................................23
Table 3  Capability 7.0 Metrics ................................................................................32
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 7.0 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 7.0 must have a version of the CUDA driver released with CUDA Toolkit 7.0 (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

\texttt{cuptiActivityRegisterCallbacks} and \texttt{cuptiActivityFlushAll}.

It is not required that the activity API be initialized before CUDA initialization. All related activities occurring after initializing the activity API are collected. You can force initialization of the activity API by enabling one or more activity kinds using \texttt{cuptiActivityEnable} or \texttt{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 \texttt{cuptiActivityEnable} and \texttt{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 \texttt{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 \texttt{cuptiActivityFlushAll} can be used to force CUPTI to deliver any activity buffers that contain completed activity records. Functions \texttt{cuptiActivityGetAttribute} and \texttt{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. SASS Source Correlation

While high-level languages for GPU programming like CUDA C offer a useful level of abstraction, convenience, and maintainability, they inherently hide some of the details of the execution on the hardware. It is sometimes helpful to analyze performance problems
for a kernel at the assembly instruction level. Reading assembly language is tedious and challenging; CUPTI can help you to build the correlation between lines in your high-level source code and the executed assembly instructions.

Building SASS source correlation for a PC can be split into two parts -

- **Correlation of the PC to SASS instruction** - subscribe to any one of `CUPTI_CBID_RESOURCE_MODULE_LOADED` or `CUPTI_CBID_RESOURCE_MODULE_UNLOAD_STARTING` or `CUPTI_CBID_RESOURCE_MODULE_PROFILED` callbacks. This returns a `CUpti_ModuleResourceData` structure having the CUDA binary. The binary can be disassembled using `nvdisasm` utility that comes with the CUDA toolkit. An application can have multiple functions and modules, to uniquely identify there is a `functionId` field in all source level activity records. This uniquely corresponds to a `CUPTI_ACTIVITY_KIND_FUNCTION` which has the unique module ID and function ID in the module.

- **Correlation of the SASS instruction to CUDA source line** - every source level activity has a `sourceLocatorId` field which uniquely maps to a record of kind `CUPTI_ACTIVITY_KIND_SOURCE_LOCATOR` containing the line and file name information. Please note that multiple PCs can correspond to single source line.

When any source level activity (global access, branch, PC Sampling etc) is enabled, source locator record is generated for the PCs that have the source level results. Record `CUpti_ActivityInstructionCorrelation` can be used along with source level activities to generate SASS assembly instructions to CUDA C source code mapping for all the PCs of the function and not just the PCs that have the source level results. This can be enabled using activity kind `CUPTI_ACTIVITY_KIND_INSTRUCTION_CORRELATION`.

The `sass_source_map` sample shows how to map SASS assembly instructions to CUDA C source.

### 1.3.2. PC Sampling

CUPTI supports device-wide sampling of the program counter (PC). The PC Sampling gives the number of samples for each source and assembly line with various stall reasons. Using this information you can pinpoint portions of your kernel that are introducing latencies and the reason for the latency. Samples are taken in round robin order for all active warps at a fixed number of cycles regardless of whether the warp is issuing an instruction or not.

Devices with compute capability 6.0 and higher have a new feature that gives latency reasons. The latency samples indicate the reasons for holes in the issue pipeline. While collecting these samples, there is no instruction issued in the respective warp scheduler and hence these give the latency reasons. The latency reasons will be one of the stall
reasons listed in the enum \texttt{CUpti\_ActivityPCSamplingStallReason} except stall reason \texttt{CUPTI\_ACTIVITY\_PC\_SAMPLING\_STALL\_NOT\_SELECTED}.

Activity record \texttt{CUpti\_ActivityPCSampling3} enabled using activity kind \texttt{CUPTI\_ACTIVITY\_KIND\_PC\_SAMPLING} outputs stall reason along with PC and other related information. Enum \texttt{CUpti\_ActivityPCSamplingStallReason} lists all the stall reasons. Sampling period is configurable and can be tuned using API \texttt{cuptiActivityConfigurePCSampling}. A wide range of sampling periods ranging from \(2^5\) cycles to \(2^{31}\) cycles per sample is supported. This can be controlled through field \texttt{samplingPeriod2} in the PC sampling configuration struct \texttt{CUpti\_ActivityPCSamplingConfig}. Activity record \texttt{CUpti\_ActivityPCSamplingRecordInfo} provides the total and dropped samples for each kernel profiled for PC sampling.

This feature is available on devices with compute capability 5.2 and higher, excluding mobile devices.

The \texttt{pc\_sampling} sample shows how to use these APIs to collect PC Sampling profiling information for a kernel.

### 1.3.3. NVLink

NVIDIA NVLink is a high-bandwidth, energy-efficient interconnect that enables fast communication between the CPU and GPU, and between GPUs. CUPTI provides NVLink topology information and NVLink transmit/receive throughput metrics.

Activity record \texttt{CUpti\_ActivityNVLink2} enabled using activity kind \texttt{CUPTI\_ACTIVITY\_KIND\_NVLink} outputs NVLink topology information in terms of logical NVLinks. A logical NVLink is connected between 2 devices, the device can be of type NPU (NVLink Processing Unit which can be CPU) or GPU. Each device can support upto 6 NVLinks hence one logical link can comprise of 1 to 6 physical NVLinks. Field \texttt{physicalNvLinkCount} gives number of physical links in this logical link. Fields \texttt{portDev0} and \texttt{portDev1} give information about the slot in which physical NVLinks are connected for a logical link. This port is same as instance of NVLink metrics profiled from a device. So port and instance information should be used to correlate the per-instance metric values with the physical NVLinks and in turn to the topology. Field \texttt{flag} gives the properties of a logical link, whether the link has access to system memory or peer device memory, and have capabilities to do system memory or peer memmory atomics. Field \texttt{bandwidth} gives the bandwidth of the logical link in kilobytes/sec.

CUPTI also provides some metrics for each physical links. Metrics are provided for data transmitted/received, transmit/receive throughput and header versus user data overhead for each physical NVLink. These metrics are also provided per packet type (read/write/ atomics/response) to get more detailed insight in the NVLink traffic.

This feature is available on devices with compute capability 6.0 and 7.0.
The `nvlink_bandwidth` sample shows how to use these APIs to collect NVLink metrics and topology and how to correlate metrics with the topology.

### 1.3.4. OpenACC

On Linux x86_64, CUPTI supports collecting information for OpenACC applications using the OpenACC tools interface implementation of the PGI runtime. In addition to being available only on 64bit Linux platforms, this feature also requires PGI runtime version 15.7 or higher.

Activity records `CUpti_ActivityOpenAccData`, `CUpti_ActivityOpenAccLaunch` and `CUpti_ActivityOpenAccOther` are created, representing the three groups of callback events specified in the OpenACC tools interface. `CUPTI_ACTIVITY_KIND_OPENACC_DATA`, `CUPTI_ACTIVITY_KIND_OPENACC_LAUNCH` and `CUPTI_ACTIVITY_KIND_OPENACC_OTHER` can be enabled to collect the respective activity records.

Due to restrictions of the OpenACC tools interface, CUPTI cannot record OpenACC records from within the client application. Instead, a shared library that exports the `acc_register_library` function defined in the OpenACC tools interface specification must be implemented. Parameters passed into this function from the OpenACC runtime can be used to initialize CUPTI OpenACC measurement using `cuptiOpenACCInitialize`. Before starting the client application, the environment variable `ACC_PROFLIB` must be set to point to this shared library.

`cuptiOpenACCInitialize` is defined in `cupti_openacc.h`, which is included by `cupti_activity.h`. Since the CUPTI OpenACC header is only available on supported platforms, CUPTI clients must define `CUPTI_OPENACC_SUPPORT` when compiling.

The `openacc_trace` sample shows how to use CUPTI APIs for OpenACC data collection.

### 1.3.5. External Correlation

Starting with CUDA 8.0, CUPTI supports correlation of CUDA API activity records with external APIs. Such APIs include e.g. OpenACC, OpenMP and MPI. The correlation associates CUPTI correlation IDs with IDs provided by the external API. Both IDs are stored in a new activity record of type `CUpti_ActivityExternalCorrelation`.

CUPTI maintains a stack of external correlation IDs per CPU thread and per `CUpti_ExternalCorrelationKind`. Clients must use `cuptiActivityPushExternalCorrelationId` to push an external ID of a specific kind to this stack and `cuptiActivityPopExternalCorrelationId` to remove the latest ID. If a CUDA API activity record is generated while any `CUpti_ExternalCorrelationKind-stack` on the same CPU thread is non-empty, one `CUpti_ActivityExternalCorrelation` record per `CUpti_ExternalCorrelationKind-stack` is inserted into the activity buffer before...
the respective CUDA API activity record. The CUPTI client is responsible for tracking passed external API correlation IDs in order to eventually associate external API calls with CUDA API calls.

If both `CUPTI_ACTIVITY_KIND_EXTERNAL_CORRELATION` and any of `CUPTI_ACTIVITY_KIND_OPENACC_*` activity kinds are enabled, CUPTI will generate external correlation activity records for OpenACC with `externalKind` `CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC`.

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.

```c
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.

```c
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.

```c
/* 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.

```c
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.

```c
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.

For event collection mode `CUPTI_EVENT_COLLECTION_MODE_KERNEL`, events or metrics collection may significantly change the overall performance characteristics of the application because all kernel executions that occur between the `cuptiEventGroupEnable` and `cuptiEventGroupDisable` calls are serialized on the GPU. This can be avoided by using mode `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS` and restricting profiling to events and metrics that can be collected in a single pass.

All the events and metrics except NVLink metrics are collected at the context level irrespective of the event collection mode. That is, events or metrics can be attributed to the context being profiled and values can be accurately collected when multiple contexts are executing on the GPU. NVLink metrics are collected at device level for all event collection modes.

In a system with multiple GPUs, events can be collected simultaneously on all the GPUs i.e. event profiling doesn’t enforce any serialization of work across GPUs. The `event_multi_gpu` sample shows how to use the CUPTI event and CUDA APIs on such setups.
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 cupiEnableCallback(1, subscriber, CUPTI_CB_DOMAIN_RUNTIME_API, CUPTI_RUNTIME_TRACE_CBID_cudalaunch_v3020)). To simplify the presentation, error checking code has been removed.

```c
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();
        cupiSetEventCollectionMode(cbInfo->context,
                                    CUPTI_EVENT_COLLECTION_MODE_KERNEL);
        cupiEventGroupEnable(eventGroup);
    }

    if (cbData->callbackSite == CUPTI_API_EXIT) {
        cudaDeviceSynchronize();
        cupiEventGroupReadEvent(eventGroup,
                                 CUPTI_EVENT_READ_FLAG_NONE,
                                 eventId,
                                 &bytesRead, &eventVal);
        cupiEventGroupDisable(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 cupiEventGroupEnable.

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 cupiEventGroupReadEvent.
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 up 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 cupi_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` then 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.
1.6.1. Metrics Reference

This section contains detailed descriptions of the metrics that can be collected by theCUPTI. A scope value of "Single-context" indicates that the metric can only be accurately collected when a single context (CUDA or graphics) 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. A scope value of "Device" indicates that the metric will be collected at device level, that is, it will include values for all the contexts executing on the GPU. The events for these metrics can be collected at device level using CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS. When these metrics are collected for a kernel using CUPTI_EVENT_COLLECTION_MODE_KERNEL, they exhibit the behavior of single-context. Note that NVLink metrics collected for kernel mode exhibit the behavior of "Single-context".

1.6.1.1. Metrics for Capability 3.x

Devices with compute capability 3.x implement the metrics shown in the following table. Note that for some metrics the "Multi-context" scope is supported only for specific devices. Such metrics are marked with "Multi-context *" under the "Scope" column. Refer to the note at the bottom of the table.

Table 1 Capability 3.x Metrics

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>achieved_occupancy</td>
<td>Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>alu_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_replay_overhead</td>
<td>Average number of replays due to atomic and reduction bank conflicts for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_throughput</td>
<td>Global memory atomic and reduction throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions</td>
<td>Global memory atomic and reduction transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions_per_request</td>
<td>Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction</td>
<td>Multi-context</td>
</tr>
<tr>
<td>branch_efficiency</td>
<td>Ratio of non-divergent branches to total branches expressed as percentage. This is available for compute capability 3.0.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_executed</td>
<td>Number of executed control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>cf_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_issued</td>
<td>Number of issued control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_throughput</td>
<td>Device memory read throughput. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_transactions</td>
<td>Device memory read transactions. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_utilization</td>
<td>The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_throughput</td>
<td>Device memory write throughput. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_transactions</td>
<td>Device memory write transactions. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ecc_throughput</td>
<td>ECC throughput from L2 to DRAM. This is available for compute capability 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ecc_transactions</td>
<td>Number of ECC transactions between L2 and DRAM. This is available for compute capability 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>eligible_warps_per_cycle</td>
<td>Average number of warps that are eligible to issue per active cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp</td>
<td>Number of double-precision floating-point operations executed by non-predicated threads (add, multiply and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_add</td>
<td>Number of double-precision floating-point add operations executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_fma</td>
<td>Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_mul</td>
<td>Number of double-precision floating-point multiply operations executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp</td>
<td>Number of single-precision floating-point operations executed by non-predicated threads (add, multiply and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count. The count does not include special operations.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>flop_count_sp_add</td>
<td>Number of single-precision floating-point add operations executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_fma</td>
<td>Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_mul</td>
<td>Number of single-precision floating-point multiply operations executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_special</td>
<td>Number of single-precision floating-point special operations executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_dp_efficiency</td>
<td>Ratio of achieved to peak double-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_sp_efficiency</td>
<td>Ratio of achieved to peak single-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_efficiency</td>
<td>Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_requested_throughput</td>
<td>Requested global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_throughput</td>
<td>Global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions</td>
<td>Number of global memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions_per_request</td>
<td>Average number of global memory load transactions performed for each global memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_cache_replay_overhead</td>
<td>Average number of replays due to global memory cache misses for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_replay_overhead</td>
<td>Average number of replays due to global memory cache misses</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_efficiency</td>
<td>Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gstRequested_throughput</td>
<td>Requested global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_throughput</td>
<td>Global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_transactions</td>
<td>Number of global memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_transactions_per_request</td>
<td>Average number of global memory store transactions performed for each global memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_bit_convert</td>
<td>Number of bit-conversion instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>inst_compute_ld_st</td>
<td>Number of compute load/store instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_control</td>
<td>Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed</td>
<td>The number of instructions executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_fp_32</td>
<td>Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_fp_64</td>
<td>Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_integer</td>
<td>Number of integer instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_inter_thread_communication</td>
<td>Number of inter-thread communication instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>instissued</td>
<td>The number of instructions issued</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_misc</td>
<td>Number of miscellaneous instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_per_warp</td>
<td>Average number of instructions executed by each warp</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_replay_overhead</td>
<td>Average number of replays for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ipc</td>
<td>Instructions executed per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ipc_instance</td>
<td>Instructions executed per cycle for a single multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slot_utilization</td>
<td>Percentage of issue slots that issued at least one instruction, averaged across all cycles</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slots</td>
<td>The number of issue slots used</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issued_ipc</td>
<td>Instructions issued per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l1_cache_global_hit_rate</td>
<td>Hit rate in L1 cache for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l1_cache_local_hit_rate</td>
<td>Hit rate in L1 cache for local loads and stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l1_shared_utilization</td>
<td>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, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_atomic_throughput</td>
<td>Memory read throughput seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_atomic_transactions</td>
<td>Memory read transactions seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>l2_l1_read_hit_rate</td>
<td>Hit rate at L2 cache for all read requests from L1 cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_l1_read_throughput</td>
<td>Memory read throughput seen at L2 cache for read requests from L1 cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_l1_read_transactions</td>
<td>Memory read transactions seen at L2 cache for all read requests from L1 cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_l1_write_throughput</td>
<td>Memory write throughput seen at L2 cache for write requests from L1 cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_l1_write_transactions</td>
<td>Memory write transactions seen at L2 cache for all write requests from L1 cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_throughput</td>
<td>Memory read throughput seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_transactions</td>
<td>Memory read transactions seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_transactions</td>
<td>Memory read transactions seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_hit_rate</td>
<td>Hit rate at L2 cache for all read requests from texture cache. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_throughput</td>
<td>Memory read throughput seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_utilization</td>
<td>The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_throughput</td>
<td>Memory write throughput seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_transactions</td>
<td>Memory write transactions seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_executed</td>
<td>Number of executed local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute global, local and shared memory instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_issued</td>
<td>Number of issued local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_throughput</td>
<td>Local memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_transactions</td>
<td>Number of local memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>local_load_transactions_per_request</td>
<td>Average number of local memory load transactions performed for each local memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_memory_overhead</td>
<td>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, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_replay_overhead</td>
<td>Average number of replays due to local memory accesses for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_throughput</td>
<td>Local memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions</td>
<td>Number of local memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions_per_request</td>
<td>Average number of local memory store transactions performed for each local memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_cache_global_hit_rate</td>
<td>Hit rate in non coherent cache for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_gld_efficiency</td>
<td>Ratio of requested non coherent global memory load throughput to required non coherent global memory load throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_gld_requested_throughput</td>
<td>Requested throughput for global memory loaded via non-coherent cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_gld_throughput</td>
<td>Non coherent global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_l2_read_throughput</td>
<td>Memory read throughput for non coherent global read requests seen at L2 cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nc_l2_read_transactions</td>
<td>Memory read transactions seen at L2 cache for non coherent global read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_efficiency</td>
<td>Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_throughput</td>
<td>Shared memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions</td>
<td>Number of shared memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions_per_request</td>
<td>Average number of shared memory load transactions performed for each shared memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_replay_overhead</td>
<td>Average number of replays due to shared memory conflicts for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_throughput</td>
<td>Shared memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>shared_store_transactions</td>
<td>Number of shared memory store transactions</td>
<td></td>
</tr>
<tr>
<td>shared_store_transactions_per_request</td>
<td>Average number of shared memory store transactions performed for each shared memory store</td>
<td></td>
</tr>
<tr>
<td>sm_efficiency</td>
<td>The percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU</td>
<td></td>
</tr>
<tr>
<td>sm_efficiency_instance</td>
<td>The percentage of time at least one warp is active on a specific multiprocessor</td>
<td></td>
</tr>
<tr>
<td>stall_constant_memory_dependency</td>
<td>Percentage of stalls occurring because of immediate constant cache miss. This is available for compute capability 3.2, 3.5 and 3.7.</td>
<td></td>
</tr>
<tr>
<td>stall_exec_dependency</td>
<td>Percentage of stalls occurring because an input required by the instruction is not yet available</td>
<td></td>
</tr>
<tr>
<td>stall_inst_fetch</td>
<td>Percentage of stalls occurring because the next assembly instruction has not yet been fetched</td>
<td></td>
</tr>
<tr>
<td>stall_memory_dependency</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>stall_memory_throttle</td>
<td>Percentage of stalls occurring because of memory throttle.</td>
<td></td>
</tr>
<tr>
<td>stall_not_selected</td>
<td>Percentage of stalls occurring because warp was not selected.</td>
<td></td>
</tr>
<tr>
<td>stall_other</td>
<td>Percentage of stalls occurring due to miscellaneous reasons</td>
<td></td>
</tr>
<tr>
<td>stall_pipe_busy</td>
<td>Percentage of stalls occurring because a compute operation cannot be performed because the compute pipeline is busy. This is available for compute capability 3.2, 3.5 and 3.7.</td>
<td></td>
</tr>
<tr>
<td>stall_sync</td>
<td>Percentage of stalls occurring because the warp is blocked at a __syncthreads() call</td>
<td></td>
</tr>
<tr>
<td>stall_texture</td>
<td>Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests</td>
<td></td>
</tr>
<tr>
<td>sysmem_read_throughput</td>
<td>System memory read throughput. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_transactions</td>
<td>System memory read transactions. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_utilization</td>
<td>The read utilization level of the system memory relative to the peak utilization on a scale of 0 to 1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td><strong>10. This is available for compute capability 3.0, 3.5 and 3.7.</strong></td>
<td></td>
</tr>
<tr>
<td>sysmem_utilization</td>
<td>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, 3.5 and 3.7.</td>
<td>Multi-context*</td>
</tr>
<tr>
<td>sysmem_write_throughput</td>
<td>System memory write throughput. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_transactions</td>
<td>System memory write transactions. This is available for compute capability 3.0, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_utilization</td>
<td>The write 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, 3.5 and 3.7.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_hit_rate</td>
<td>Texture cache hit rate</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_throughput</td>
<td>Texture cache throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_transactions</td>
<td>Texture cache read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute texture instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_utilization</td>
<td>The utilization level of the texture cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_execution_efficiency</td>
<td>Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_nonpred_execution_efficiency</td>
<td>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</td>
<td>Multi-context</td>
</tr>
</tbody>
</table>

* The "Multi-context" scope for this metric is supported only for devices with compute capability 3.0, 3.5 and 3.7.

1.6.1.2. Metrics for Capability 5.x

Devices with compute capability 5.x implement the metrics shown in the following table. Note that for some metrics the "Multi-context" scope is supported only for specific devices. Such metrics are marked with "Multi-context *" under the "Scope" column. Refer to the note at the bottom of the table.
* The "Multi-context" scope for this metric is supported only for devices with compute capability 5.0 and 5.2.

1.6.1.3. Metrics for Capability 6.x

Devices with compute capability 6.x implement the metrics shown in the following table.

Table 2  Capability 6.x Metrics

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>achieved_occupancy</td>
<td>Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions</td>
<td>Global memory atomic and reduction transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions_per_request</td>
<td>Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction</td>
<td>Multi-context</td>
</tr>
<tr>
<td>branch_efficiency</td>
<td>Ratio of non-divergent branches to total branches expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_executed</td>
<td>Number of executed control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_issued</td>
<td>Number of issued control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>double_precision_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute double-precision floating-point instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_bytes</td>
<td>Total bytes read from DRAM to L2 cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_throughput</td>
<td>Device memory read throughput. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_transactions</td>
<td>Device memory read transactions. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_utilization</td>
<td>The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_bytes</td>
<td>Total bytes written from L2 cache to DRAM</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_throughput</td>
<td>Device memory write throughput. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_transactions</td>
<td>Device memory write transactions. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ecc_throughput</td>
<td>ECC throughput from L2 to DRAM. This is available for compute capability 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>ecc_transactions</td>
<td>Number of ECC transactions between L2 and DRAM. This is available for compute capability 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>eligible_warps_per_cycle</td>
<td>Average number of warps that are eligible to issue per active cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>executed_ipc</td>
<td>Instructions executed per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp</td>
<td>Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_add</td>
<td>Number of double-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_fma</td>
<td>Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_mul</td>
<td>Number of double-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp</td>
<td>Number of half-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_add</td>
<td>Number of half-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_fma</td>
<td>Number of half-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_mul</td>
<td>Number of half-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp</td>
<td>Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count. The count does not include special operations.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_add</td>
<td>Number of single-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_fma</td>
<td>Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>flop_count_sp_mul</td>
<td>Number of single-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_special</td>
<td>Number of single-precision floating-point special operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_dp_efficiency</td>
<td>Ratio of achieved to peak double-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_hp_efficiency</td>
<td>Ratio of achieved to peak half-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_sp_efficiency</td>
<td>Ratio of achieved to peak single-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_efficiency</td>
<td>Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_requested_throughput</td>
<td>Requested global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_throughput</td>
<td>Global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions</td>
<td>Number of global memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions_per_request</td>
<td>Average number of global memory load transactions performed for each global memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_atomic_requests</td>
<td>Total number of global atomic(Atom and Atom CAS) requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_hit_rate</td>
<td>Hit rate for global loads in unified l1/tex cache. Metric value maybe wrong if malloc is used in kernel.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_load_requests</td>
<td>Total number of global load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_reduction_requests</td>
<td>Total number of global reduction requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_store_requests</td>
<td>Total number of global store requests from Multiprocessor. This does not include atomic requests.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_efficiency</td>
<td>Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_requested_throughput</td>
<td>Requested global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_throughput</td>
<td>Global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_transactions</td>
<td>Number of global memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_transactions_per_request</td>
<td>Average number of global memory store transactions performed for each global memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>half_precision_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute 16 bit floating-point instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>inst_bit_convert</td>
<td>Number of bit-conversion instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_compute_lod_st</td>
<td>Number of compute load/store instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_control</td>
<td>Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed</td>
<td>The number of instructions executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_global_atomics</td>
<td>Warp level instructions for global atom and atom cas</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_global_loads</td>
<td>Warp level instructions for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_global_reductions</td>
<td>Warp level instructions for global reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_global_stores</td>
<td>Warp level instructions for global stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_local_loads</td>
<td>Warp level instructions for local loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_local_stores</td>
<td>Warp level instructions for local stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_shared_atomics</td>
<td>Warp level shared instructions for atom and atom CAS</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_shared_loads</td>
<td>Warp level instructions for shared loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_shared_stores</td>
<td>Warp level instructions for shared stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_surface_atomics</td>
<td>Warp level instructions for surface atom and atom cas</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_surface_loads</td>
<td>Warp level instructions for surface loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_surface_reductions</td>
<td>Warp level instructions for surface reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_surface_stores</td>
<td>Warp level instructions for surface stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_executed_tex_ops</td>
<td>Warp level instructions for texture</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_fp_16</td>
<td>Number of half-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_fp_32</td>
<td>Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_fp_64</td>
<td>Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_integer</td>
<td>Number of integer instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_inter_thread_communication</td>
<td>Number of inter-thread communication instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_issued</td>
<td>The number of instructions issued</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_misc</td>
<td>Number of miscellaneous instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>inst_per_warp</td>
<td>Average number of instructions executed by each warp</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_replay_overhead</td>
<td>Average number of replays for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ipc</td>
<td>Instructions executed per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slot_utilization</td>
<td>Percentage of issue slots that issued at least one instruction, averaged across all cycles</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slots</td>
<td>The number of issue slots used</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issued_ipc</td>
<td>Instructions issued per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2.atomic_throughput</td>
<td>Memory read throughput seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2.atomic_transactions</td>
<td>Memory read transactions seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_global_atomic_store_bytes</td>
<td>Bytes written to L2 from Unified cache for global atomics (ATOM and ATOM CAS)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_global_load_bytes</td>
<td>Bytes read from L2 for misses in Unified Cache for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_global_reduction_bytes</td>
<td>Bytes written to L2 from Unified cache for global reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_local_global_store_bytes</td>
<td>Bytes written to L2 from Unified Cache for local and global stores. This does not include global atomics.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_local_load_bytes</td>
<td>Bytes read from L2 for misses in Unified Cache for local loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_throughput</td>
<td>Memory read throughput seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_transactions</td>
<td>Memory read transactions seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_atomic_store_bytes</td>
<td>Bytes transferred between Unified Cache and L2 for surface atomics (ATOM and ATOM CAS)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_load_bytes</td>
<td>Bytes read from L2 for misses in Unified Cache for surface loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_reduction_bytes</td>
<td>Bytes written to L2 from Unified Cache for surface reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_store_bytes</td>
<td>Bytes written to L2 from Unified Cache for surface stores. This does not include surface atomics.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tes_hit_rate</td>
<td>Hit rate at L2 cache for all requests from texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tes_read_hit_rate</td>
<td>Hit rate at L2 cache for all read requests from texture cache. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>l2_tex_read_throughput</td>
<td>Memory read throughput seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_transactions</td>
<td>Memory read transactions seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_hit_rate</td>
<td>Hit Rate at L2 cache for all write requests from texture cache. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_throughput</td>
<td>Memory write throughput seen at L2 cache for write requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_transactions</td>
<td>Memory write transactions seen at L2 cache for write requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_utilization</td>
<td>The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_throughput</td>
<td>Memory write throughput seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_transactions</td>
<td>Memory write transactions seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_executed</td>
<td>Number of executed local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute shared load, shared store and constant load instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_issued</td>
<td>Number of issued local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_hit_rate</td>
<td>Hit rate for local loads and stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_requests</td>
<td>Total number of local load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_throughput</td>
<td>Local memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_transactions</td>
<td>Number of local memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_transactions_per_request</td>
<td>Average number of local memory load transactions performed for each local memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_memory_overhead</td>
<td>Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_requests</td>
<td>Total number of local store requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_throughput</td>
<td>Local memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions</td>
<td>Number of local memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions_per_request</td>
<td>Average number of local memory store transactions performed for each local memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>nvlink_overhead_data_received</td>
<td>Ratio of overhead data to the total data, received through NVLink. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_overhead_data_transmitted</td>
<td>Ratio of overhead data to the total data, transmitted through NVLink. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_receive_throughput</td>
<td>Number of bytes received per second through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_data_received</td>
<td>Total data bytes received through NVLinks including headers. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_data_transmitted</td>
<td>Total data bytes transmitted through NVLinks including headers. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_nratom_data_transmitted</td>
<td>Total non-reduction atomic data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_ratom_data_transmitted</td>
<td>Total reduction atomic data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_response_data_received</td>
<td>Total response data bytes received through NVLink, response data includes data for read requests and result of non-reduction atomic requests. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_write_data_transmitted</td>
<td>Total write data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_transmit_throughput</td>
<td>Number of Bytes Transmitted per second through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_data_received</td>
<td>User data bytes received through NVLinks, doesn't include headers. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_data_transmitted</td>
<td>User data bytes transmitted through NVLinks, doesn't include headers. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_nratom_data_transmitted</td>
<td>Total non-reduction atomic user data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_ratom_data_transmitted</td>
<td>Total reduction atomic user data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_response_data_received</td>
<td>Total user response data bytes received through NVLink, response data includes data for read requests and result of non-reduction</td>
<td>Device</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>atomic_requests</td>
<td>Atomic requests. This is available for compute capability 6.0.</td>
<td></td>
</tr>
<tr>
<td>nvlink_user_write_data_transmitted</td>
<td>User write data bytes transmitted through NVLinks. This is available for compute capability 6.0.</td>
<td>Device</td>
</tr>
<tr>
<td>pcie_total_data_received</td>
<td>Total data bytes received through PCIe</td>
<td>Device</td>
</tr>
<tr>
<td>pcie_total_data_transmitted</td>
<td>Total data bytes transmitted through PCIe</td>
<td>Device</td>
</tr>
<tr>
<td>shared_efficiency</td>
<td>Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_throughput</td>
<td>Shared memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions</td>
<td>Number of shared memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions_per_request</td>
<td>Average number of shared memory load transactions performed for each shared memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_throughput</td>
<td>Shared memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_transactions</td>
<td>Number of shared memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_transactions_per_request</td>
<td>Average number of shared memory store transactions performed for each shared memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_utilization</td>
<td>The utilization level of the shared memory relative to peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>single_precision_fu_utilization</td>
<td>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</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sm_efficiency</td>
<td>The percentage of time at least one warp is active on a specific multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>special_fu_utilization</td>
<td>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</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_constant_memory_dependency</td>
<td>Percentage of stalls occurring because of immediate constant cache miss</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_exec_dependency</td>
<td>Percentage of stalls occurring because an input required by the instruction is not yet available</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_inst_fetch</td>
<td>Percentage of stalls occurring because the next assembly instruction has not yet been fetched</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_memory_dependency</td>
<td>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</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_memory_throttle</td>
<td>Percentage of stalls occurring because of memory throttle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>stall_not_selected</td>
<td>Percentage of stalls occurring because warp was not selected</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_other</td>
<td>Percentage of stalls occurring due to miscellaneous reasons</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_pipe_busy</td>
<td>Percentage of stalls occurring because a compute operation cannot be performed because the compute pipeline is busy</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_sync</td>
<td>Percentage of stalls occurring because the warp is blocked at a __syncthreads() call</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_texture</td>
<td>Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_atomic_requests</td>
<td>Total number of surface atomic(Atom and Atom CAS) requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_load_requests</td>
<td>Total number of surface load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_reduction_requests</td>
<td>Total number of surface reduction requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_store_requests</td>
<td>Total number of surface store requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_bytes</td>
<td>Number of bytes read from system memory</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_throughput</td>
<td>System memory read throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_transactions</td>
<td>Number of system memory read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_utilization</td>
<td>The read utilization level of the system memory relative to the peak utilization on a scale of 0 to 10. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_utilization</td>
<td>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 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_bytes</td>
<td>Number of bytes written to system memory</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_throughput</td>
<td>System memory write throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_transactions</td>
<td>Number of system memory write transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_utilization</td>
<td>The write utilization level of the system memory relative to the peak utilization on a scale of 0 to 10. This is available for compute capability 6.0 and 6.1.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_hit_rate</td>
<td>Unified cache hit rate</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_throughput</td>
<td>Unified cache throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_transactions</td>
<td>Unified cache read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute global, local and</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>achieved_occupancy</td>
<td>Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions</td>
<td>Global memory atomic and reduction transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>atomic_transactions_per_request</td>
<td>Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction</td>
<td>Multi-context</td>
</tr>
<tr>
<td>branch_efficiency</td>
<td>Ratio of branch instruction to sum of branch and divergent branch instruction</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_executed</td>
<td>Number of executed control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>cf_issued</td>
<td>Number of issued control-flow instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>double_precision_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute double-precision floating-point instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_bytes</td>
<td>Total bytes read from DRAM to L2 cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_read_throughput</td>
<td>Device memory read throughput</td>
<td>Multi-context</td>
</tr>
</tbody>
</table>

### 1.6.1.4. Metrics for Capability 7.0

Devices with compute capability 7.0 implement the metrics shown in the following table.

**Table 3  Capability 7.0 Metrics**

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>tex_utilization</td>
<td>The utilization level of the unified cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>texture_load_requests</td>
<td>Total number of texture Load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>unique_warps_launched</td>
<td>Number of warps launched. Value is unaffected by compute preemption.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_execution_efficiency</td>
<td>Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_nonpred_execution_efficiency</td>
<td>Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>dram_read_transactions</td>
<td>Device memory read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_utilization</td>
<td>The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_bytes</td>
<td>Total bytes written from L2 cache to DRAM</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_throughput</td>
<td>Device memory write throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>dram_write_transactions</td>
<td>Device memory write transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>eligible_warps_per_cycle</td>
<td>Average number of warps that are eligible to issue per active cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp</td>
<td>Number of double-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_add</td>
<td>Number of double-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_fma</td>
<td>Number of double-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_dp_mul</td>
<td>Number of double-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp</td>
<td>Number of half-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 or 4 to the count based on the number of inputs.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_add</td>
<td>Number of half-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_fma</td>
<td>Number of half-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate contributes 2 or 4 to the count based on the number of inputs.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_hp_mul</td>
<td>Number of half-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp</td>
<td>Number of single-precision floating-point operations executed by non-predicated threads (add, multiply, and multiply-accumulate). Each multiply-accumulate operation contributes 2 to the count. The count does not include special operations.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_add</td>
<td>Number of single-precision floating-point add operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>flop_count_sp_fma</td>
<td>Number of single-precision floating-point multiply-accumulate operations executed by non-predicated threads. Each multiply-accumulate operation contributes 1 to the count.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_mul</td>
<td>Number of single-precision floating-point multiply operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_count_sp_special</td>
<td>Number of single-precision floating-point special operations executed by non-predicated threads.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_dp_efficiency</td>
<td>Ratio of achieved to peak double-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_hp_efficiency</td>
<td>Ratio of achieved to peak half-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>flop_sp_efficiency</td>
<td>Ratio of achieved to peak single-precision floating-point operations</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_efficiency</td>
<td>Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_requested_throughput</td>
<td>Requested global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_throughput</td>
<td>Global memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions</td>
<td>Number of global memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gld_transactions_per_request</td>
<td>Average number of global memory load transactions performed for each global memory load.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_atomic_requests</td>
<td>Total number of global atomic(Atom and Atom CAS) requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_hit_rate</td>
<td>Hit rate for global load and store in unified l1/ tex cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_load_requests</td>
<td>Total number of global load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_reduction_requests</td>
<td>Total number of global reduction requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>global_store_requests</td>
<td>Total number of global store requests from Multiprocessor. This does not include atomic requests.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_efficiency</td>
<td>Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_requested_throughput</td>
<td>Requested global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_throughput</td>
<td>Global memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>gst_transactions</td>
<td>Number of global memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><code>gst_transactions_per_request</code></td>
<td>Average number of global memory store transactions performed for each global memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>half_precision_fu_utilization</code></td>
<td>The utilization level of the multiprocessor function units that execute 16 bit floating-point instructions on a scale of 0 to 10. Note that this doesn't specify the utilization level of tensor core unit</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_bit_convert</code></td>
<td>Number of bit-conversion instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_compute_ld_st</code></td>
<td>Number of compute load/store instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_control</code></td>
<td>Number of control-flow instructions executed by non-predicated threads (jump, branch, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed</code></td>
<td>The number of instructions executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_global_atomics</code></td>
<td>Warp level instructions for global atom and atom cas</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_global_loads</code></td>
<td>Warp level instructions for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_global_reductions</code></td>
<td>Warp level instructions for global reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_global_stores</code></td>
<td>Warp level instructions for global stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_local_loads</code></td>
<td>Warp level instructions for local loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_local_stores</code></td>
<td>Warp level instructions for local stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_shared_atomics</code></td>
<td>Warp level shared instructions for atom and atom CAS</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_shared_loads</code></td>
<td>Warp level instructions for shared loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_shared_stores</code></td>
<td>Warp level instructions for shared stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_surface_atomics</code></td>
<td>Warp level instructions for surface atom and atom cas</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_surface_loads</code></td>
<td>Warp level instructions for surface loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_surface_reductions</code></td>
<td>Warp level instructions for surface reductions</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_surface_stores</code></td>
<td>Warp level instructions for surface stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_executed_tex_ops</code></td>
<td>Warp level instructions for texture</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_fp_16</code></td>
<td>Number of half-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_fp_32</code></td>
<td>Number of single-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td><code>inst_fp_64</code></td>
<td>Number of double-precision floating-point instructions executed by non-predicated threads (arithmetic, compare, etc.)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>inst_integer</td>
<td>Number of integer instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_inter_thread_communication</td>
<td>Number of inter-thread communication instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_issued</td>
<td>The number of instructions issued</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_m misc</td>
<td>Number of miscellaneous instructions executed by non-predicated threads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_per_warp</td>
<td>Average number of instructions executed by each warp</td>
<td>Multi-context</td>
</tr>
<tr>
<td>inst_replay_overhead</td>
<td>Average number of replays for each instruction executed</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ipc</td>
<td>Instructions executed per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slot_utilization</td>
<td>Percentage of issue slots that issued at least one instruction, averaged across all cycles</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issue_slots</td>
<td>The number of issue slots used</td>
<td>Multi-context</td>
</tr>
<tr>
<td>issued_ipc</td>
<td>Instructions issued per cycle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_atomic_throughput</td>
<td>Memory read throughput seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_atomic_transactions</td>
<td>Memory read transactions seen at L2 cache for atomic and reduction requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_global_atomic_store_bytes</td>
<td>Bytes written to L2 from L1 for global atomics (ATOM and ATOM CAS)</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_global_load_bytes</td>
<td>Bytes read from L2 for misses in L1 for global loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_local_global_store_bytes</td>
<td>Bytes written to L2 from L1 for local and global stores. This does not include global atomics.</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_local_load_bytes</td>
<td>Bytes read from L2 for misses in L1 for local loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_throughput</td>
<td>Memory read throughput seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_read_transactions</td>
<td>Memory read transactions seen at L2 cache for all read requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_load_bytes</td>
<td>Bytes read from L2 for misses in L1 for surface loads</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_surface_store_bytes</td>
<td>Bytes read from L2 for misses in L1 for surface stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_hit_rate</td>
<td>Hit rate at L2 cache for all requests from texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_hit_rate</td>
<td>Hit rate at L2 cache for all read requests from texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_read_throughput</td>
<td>Memory read throughput seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>l2_tex_read_transactions</td>
<td>Memory read transactions seen at L2 cache for read requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_hit_rate</td>
<td>Hit Rate at L2 cache for all write requests from texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_throughput</td>
<td>Memory write throughput seen at L2 cache for write requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_tex_write_transactions</td>
<td>Memory write transactions seen at L2 cache for write requests from the texture cache</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_utilization</td>
<td>The utilization level of the L2 cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_throughput</td>
<td>Memory write throughput seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>l2_write_transactions</td>
<td>Memory write transactions seen at L2 cache for all write requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_executed</td>
<td>Number of executed local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute shared load, shared store and constant load instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>ldst_issued</td>
<td>Number of issued local, global, shared and texture memory load and store instructions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_hit_rate</td>
<td>Hit rate for local loads and stores</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_requests</td>
<td>Total number of local load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_throughput</td>
<td>Local memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_transactions</td>
<td>Number of local memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_load_transactions_per_request</td>
<td>Average number of local memory load transactions performed for each local memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_memory_overhead</td>
<td>Ratio of local memory traffic to total memory traffic between the L1 and L2 caches expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_requests</td>
<td>Total number of local store requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_throughput</td>
<td>Local memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions</td>
<td>Number of local memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>local_store_transactions_per_request</td>
<td>Average number of local memory store transactions performed for each local memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>nvlink_overhead_data_received</td>
<td>Ratio of overhead data to the total data, received through NVLink.</td>
<td>Device</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>nvlink_overhead_data_transmitted</td>
<td>Ratio of overhead data to the total data, transmitted through NVLink.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_receive_throughput</td>
<td>Number of bytes received per second through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_data_received</td>
<td>Total data bytes received through NVLinks including headers.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_data_transmitted</td>
<td>Total data bytes transmitted through NVLinks including headers.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_nratom_data_transmitted</td>
<td>Total non-reduction atomic data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_ratom_data_transmitted</td>
<td>Total reduction atomic data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_response_data_received</td>
<td>Total response data bytes received through NVLink, response data includes data for read requests and result of non-reduction atomic requests.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_total_write_data_transmitted</td>
<td>Total write data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_transmit_throughput</td>
<td>Number of Bytes Transmitted per second through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_data_received</td>
<td>User data bytes received through NVLinks, doesn’t include headers.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_data_transmitted</td>
<td>User data bytes transmitted through NVLinks, doesn’t include headers.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_nratom_data_transmitted</td>
<td>Total non-reduction atomic user data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_ratom_data_transmitted</td>
<td>Total reduction atomic user data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_response_data_received</td>
<td>Total user response data bytes received through NVLink, response data includes data for read requests and result of non-reduction atomic requests.</td>
<td>Device</td>
</tr>
<tr>
<td>nvlink_user_write_data_transmitted</td>
<td>User write data bytes transmitted through NVLinks.</td>
<td>Device</td>
</tr>
<tr>
<td>pcie_total_data_received</td>
<td>Total data bytes received through PCIe</td>
<td>Device</td>
</tr>
<tr>
<td>pcie_total_data_transmitted</td>
<td>Total data bytes transmitted through PCIe</td>
<td>Device</td>
</tr>
<tr>
<td>shared_efficiency</td>
<td>Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_throughput</td>
<td>Shared memory load throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions</td>
<td>Number of shared memory load transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_load_transactions_per_request</td>
<td>Average number of shared memory load transactions performed for each shared memory load</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>shared_store_throughput</td>
<td>Shared memory store throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_transactions</td>
<td>Number of shared memory store transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_store_transactions_per_request</td>
<td>Average number of shared memory store transactions performed for each shared memory store</td>
<td>Multi-context</td>
</tr>
<tr>
<td>shared_utilization</td>
<td>The utilization level of the shared memory relative to peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>single_precision_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute single-precision floating-point instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sm_efficiency</td>
<td>The percentage of time at least one warp is active on a specific multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>special_fu_utilization</td>
<td>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</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_constant_memory_dependency</td>
<td>Percentage of stalls occurring because of immediate constant cache miss</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_exec_dependency</td>
<td>Percentage of stalls occurring because an input required by the instruction is not yet available</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_inst_fetch</td>
<td>Percentage of stalls occurring because the next assembly instruction has not yet been fetched</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_memory_dependency</td>
<td>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</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_memory_throttle</td>
<td>Percentage of stalls occurring because of memory throttle</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_not_selected</td>
<td>Percentage of stalls occurring because warp was not selected</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_other</td>
<td>Percentage of stalls occurring due to miscellaneous reasons</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_pipe_busy</td>
<td>Percentage of stalls occurring because a compute operation cannot be performed because the compute pipeline is busy</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_sleeping</td>
<td>Percentage of stalls occurring because warp was sleeping</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_sync</td>
<td>Percentage of stalls occurring because the warp is blocked at a __syncthreads() call</td>
<td>Multi-context</td>
</tr>
<tr>
<td>stall_texture</td>
<td>Percentage of stalls occurring because the texture sub-system is fully utilized or has too many outstanding requests</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_atomic_requests</td>
<td>Total number of surface atomic(Atom and Atom CAS) requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Scope</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>surface_load_requests</td>
<td>Total number of surface load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_reduction_requests</td>
<td>Total number of surface reduction requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>surface_store_requests</td>
<td>Total number of surface store requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_bytes</td>
<td>Number of bytes read from system memory</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_throughput</td>
<td>System memory read throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_transactions</td>
<td>Number of system memory read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_read_utilization</td>
<td>The read utilization level of the system memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_utilization</td>
<td>The utilization level of the system memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_bytes</td>
<td>Number of bytes written to system memory</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_throughput</td>
<td>System memory write throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_transactions</td>
<td>Number of system memory write transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>sysmem_write_utilization</td>
<td>The write utilization level of the system memory relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tensor_precision_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute tensor core instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_hit_rate</td>
<td>Unified cache hit rate</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_throughput</td>
<td>Unified cache to Multiprocessor read throughput</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_cache_transactions</td>
<td>Unified cache to Multiprocessor read transactions</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_fu_utilization</td>
<td>The utilization level of the multiprocessor function units that execute global, local and texture memory instructions on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>tex_utilization</td>
<td>The utilization level of the unified cache relative to the peak utilization on a scale of 0 to 10</td>
<td>Multi-context</td>
</tr>
<tr>
<td>texture_load_requests</td>
<td>Total number of texture Load requests from Multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_execution_efficiency</td>
<td>Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
<tr>
<td>warp_nonpred_execution_efficiency</td>
<td>Ratio of the average active threads per warp executing non-predicated instructions to the maximum number of threads per warp supported on a multiprocessor</td>
<td>Multi-context</td>
</tr>
</tbody>
</table>
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 APIs to sample events using a separate host thread.

**event_multi_gpu**
This sample shows how to use the CUPTI event and CUDA APIs to sample events on a setup with multiple GPUs. The sample shows the required ordering for synchronization, and for event group enabling, disabling and reading.

**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 information about page transfers for unified memory.

**pc_sampling**
This sample shows how to collect PC Sampling profiling information for a kernel.

**nvlink_bandwidth**
This sample shows how to collect NVLink topology and NVLink throughput metrics in continuous mode.

**openacc_trace**
This sample shows how to use CUPTI APIs for OpenACC data collection.
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.

```c
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 11
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.


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_UM_PROFILING_NOT_SUPPORTED = 28
Unified memory profiling is not supported on the system. Potential reason could be unsupported OS or architecture.

CUPTI_ERROR_UM_PROFILING_NOT_SUPPORTED_ON_DEVICE = 29
Unified memory profiling is not supported on the device

CUPTI_ERROR_UM_PROFILING_NOT_SUPPORTED_ON_NON_P2P_DEVICES = 30
Unified memory profiling is not supported on a multi-GPU configuration without P2P support between any pair of devices

CUPTI_ERROR_UM_PROFILING_NOT_SUPPORTED_WITH_MPS = 31
Unified memory profiling is not supported under the Multi-Process Service (MPS) environment. CUDA 7.5 removes this restriction.

CUPTI_ERROR_CDP_TRACING_NOT_SUPPORTED = 32
In CUDA 9.0, devices with compute capability 7.0 don't support CDP tracing

CUPTI_ERROR_VIRTUALIZED_DEVICE_NOT_SUPPORTED = 33
Profiling on virtualized GPU is not supported.

CUPTI_ERROR_CUDA_COMPILER_NOT_COMPATIBLE = 34
Profiling results might be incorrect for CUDA applications compiled with nvcc version older than 9.0 for devices with compute capability 6.0 and 6.1. Profiling session will continue and CUPTI will notify it using this error code. User is advised to recompile the application code with nvcc version 9.0 or later. Ignore this warning if code is already compiled with the recommended nvcc version.

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_ActivityCudaEvent
The activity record for CUDA event.

struct CUpti_ActivityDevice
The activity record for a device. (deprecated).

struct CUpti_ActivityDevice2
The activity record for a device. (CUDA 7.0 onwards).

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_ActivityExternalCorrelation
The activity record for correlation with external records.

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. (deprecated in CUDA 9.0).

struct CUpti_ActivityGlobalAccess3
The activity record for source-level global access.

struct CUpti_ActivityInstantaneousEvent
The activity record for an instantaneous CUPTI event.

struct CUpti_ActivityInstantaneousEventInstance
The activity record for an instantaneous CUPTI event with event domain instance information.

struct CUpti_ActivityInstantaneousMetric
The activity record for an instantaneous CUPTI metric.

struct CUpti_ActivityInstantaneousMetricInstance
The instantaneous activity record for a CUPTI metric with instance information.

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

struct CUpti_ActivityInstructionExecution
The activity record for source-level instruction execution.

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). (deprecated in CUDA 9.0).

struct CUpti_ActivityKernel4
The activity record for a kernel.

struct CUpti_ActivityMarker
The activity record providing a marker which is an instantaneous point in time. (deprecated in CUDA 8.0).

struct CUpti_ActivityMarker2
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_ActivityMemory
The activity record for memory.

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.

struct CUpti_ActivityModule
The activity record for a CUDA module.
struct CUpti_ActivityManager
The activity record providing a name.

struct CUpti_ActivityNvLink
NVLink information. (deprecated in CUDA 9.0).

struct CUpti_ActivityNvLink2
NVLink information.

union CUpti_ActivityObjectKindId
Identifiers for object kinds as specified by CUpti_ActivityObjectKind.

struct CUpti_ActivityOpenAcc
The base activity record for OpenAcc records.

struct CUpti_ActivityOpenAccData
The activity record for OpenACC data.

struct CUpti_ActivityOpenAccLaunch
The activity record for OpenACC launch.

struct CUpti_ActivityOpenAccOther
The activity record for OpenACC other.

struct CUpti_ActivityOverhead
The activity record for CUPTI and driver overheads.

struct CUpti_ActivityPcie
PCI devices information required to construct topology.

struct CUpti_ActivityPCSampling
The activity record for PC sampling. (deprecated in CUDA 8.0).

struct CUpti_ActivityPCSampling2
The activity record for PC sampling. (deprecated in CUDA 9.0).

struct CUpti_ActivityPCSampling3
The activity record for PC sampling.
struct CUpti_ActivityPCSamplingConfig
PC sampling configuration structure.

struct CUpti_ActivityPCSamplingRecordInfo
The activity record for record status for PC sampling.

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_ActivityStream
The activity record for CUDA stream.

struct CUpti_ActivitySynchronization
The activity record for synchronization management.

struct CUpti_ActivityUnifiedMemoryCounter
The activity record for Unified Memory counters (deprecated in CUDA 7.0).

struct CUpti_ActivityUnifiedMemoryCounter2
The activity record for Unified Memory counters (CUDA 7.0 and beyond).

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, especially for concurrent kernel tracing, 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 8388608 (8MB). 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 number of 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 100.

CUPTI_ACTIVITY_ATTR_PROFILING_SEMAPHORE_POOL_SIZE = 3
The profiling semaphore pool size reserved for storing profiling data for serialized kernels and memory operations for each context. The value is a size_t. Having larger pool size means less semaphore query operations but consumes more device resources. Having smaller pool size increases the risk of dropping timestamps for kernel and memcpy records if too many kernels or memcpy are launched/replayed at one time. This value only applies to new pool 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 65536.

CUPTI_ACTIVITY_ATTR_PROFILING_SEMAPHORE_POOL_LIMIT = 4
The maximum number of profiling semaphore pools per context. The value is a size_t. Profiling semaphore pool can be reused by the context. Increasing this value reduces the number of times CUPTI needs to query semaphores in the pool. Setting this value will not modify the number of semaphore pools currently stored. Set this value before initializing CUDA to ensure the limit is not exceeded. The default value is 100.

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_SYNC_ACQUIRE = 1<<3
Indicates the activity represents an attempt to acquire a user defined synchronization object. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_SYNC_ACQUIRE_SUCCESS = 1<<4
Indicates the activity represents success in acquiring the user defined synchronization object. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_SYNC_ACQUIRE_FAILED = 1<<5
Indicates the activity represents failure in acquiring the user defined synchronization object. Valid for CUPTI_ACTIVITY_KIND_MARKER.

CUPTI_ACTIVITY_FLAG_MARKER_SYNC_RELEASE = 1<<6
Indicates the activity represents releasing a reservation on user defined synchronization object. 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_ActivityGlobalAccess3.

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_ActivityGlobalAccess3.

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_ActivityGlobalAccess3.

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 and CUpti_ActivityInstructionCorrelation

**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_MEMSET_ASYNC = 1<<0**

Indicates the activity represents an asynchronous memset operation. Valid for CUPTI_ACTIVITY_KIND_MEMSET.

**CUPTI_ACTIVITY_FLAG_THRASHING_IN_CPU = 1<<0**

Indicates the activity represents thrashing in CPU. Valid for counter of kind CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING in CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER

**CUPTI_ACTIVITY_FLAG_THROTTLING_IN_CPU = 1<<0**

Indicates the activity represents page throttling in CPU. Valid for counter of kind CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING in CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER

**CUPTI_ACTIVITY_FLAG_FORCE_INT = 0x7fffffff**


definition of 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_FP_16 = 19
    Represents a 16 bit floating point operation.

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_ActivityDevice2
CUpti_ActivityDeviceAttribute
CUpti_ActivityEvent
CUpti_ActivityEventInstance
CUpti_ActivityKernel
CUpti_ActivityKernel2
CUpti_ActivityKernel3
CUpti_ActivityKernel4
CUpti_ActivityCdpKernel
CUpti_ActivityPreemption
CUpti_ActivityMemcpy
CUpti_ActivityMemcpy2
CUpti_ActivityMemset
CUpti_ActivityMetric
CUpti_ActivityMetricInstance
CUpti_ActivityName
CUpti_ActivityMarker
CUpti_ActivityMarker2
CUpti_ActivityMarkerData
CUpti_ActivitySourceLocator
CUpti_ActivityGlobalAccess
CUpti_ActivityGlobalAccess2
CUpti_ActivityGlobalAccess3
CUpti_ActivityBranch
CUpti_ActivityBranch2
CUpti_ActivityOverhead
CUpti_ActivityEnvironment
CUpti_ActivityInstructionExecution
CUpti_ActivityUnifiedMemoryCounter
CUpti_ActivityFunction
CUpti_ActivityModule
CUpti_ActivitySharedAccess
CUpti_ActivityPCSampling
CUpti_ActivityPCSampling2
CUpti_ActivityPCSampling3
CUpti_ActivityPCSamplingRecordInfo
CUpti_ActivityCudaEvent
CUpti_ActivityStream
CUpti_ActivitySynchronization
CUpti_ActivityInstructionCorrelation
CUpti_ActivityExternalCorrelation
CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityOpenAccData
CUpti_ActivityOpenAccLaunch
CUpti_ActivityOpenAccOther
CUpti_ActivityNvLink
CUpti_ActivityNvLink2
CUpti_ActivityMemory
CUpti_ActivityPcie
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_ActivityKernel4.

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_ActivityDevice2.

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_ActivityKernel4.

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_ActivityMarker2.

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 access. The corresponding activity record structure is 
CUpti_ActivityGlobalAccess3.
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 i.e. 
_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
Results for source-level instruction execution. 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_ActivityUnifiedMemoryCounter2.
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 access. The corresponding activity record structure is `CUpti_ActivitySharedAccess`.

**CUPTI_ACTIVITY_KIND_PC_SAMPLING = 30**
Enable PC sampling for kernels. This will serialize kernels. The corresponding activity record structure is `CUpti_ActivityPCSampling3`.

**CUPTI_ACTIVITY_KIND_PC_SAMPLING_RECORD_INFO = 31**
Summary information about PC sampling records. The corresponding activity record structure is `CUpti_ActivityPCSamplingRecordInfo`.

**CUPTI_ACTIVITY_KIND_INSTRUCTION_CORRELATION = 32**
SASS/Source line-by-line correlation record. This will generate sass/source correlation for functions that have source level analysis or pc sampling results. The records will be generated only when either of source level analysis or pc sampling activity is enabled. The corresponding activity record structure is `CUpti_ActivityInstructionCorrelation`.

**CUPTI_ACTIVITY_KIND_OPENACC_DATA = 33**
OpenACC data events. The corresponding activity record structure is `CUpti_ActivityOpenAccData`.

**CUPTI_ACTIVITY_KIND_OPENACC_LAUNCH = 34**
OpenACC launch events. The corresponding activity record structure is `CUpti_ActivityOpenAccLaunch`.

**CUPTI_ACTIVITY_KIND_OPENACC_OTHER = 35**
OpenACC other events. The corresponding activity record structure is `CUpti_ActivityOpenAccOther`.

**CUPTI_ACTIVITY_KIND_CUDA_EVENT = 36**
Information about a CUDA event. The corresponding activity record structure is `CUpti_ActivityCudaEvent`.

**CUPTI_ACTIVITY_KIND_STREAM = 37**
Information about a CUDA stream. The corresponding activity record structure is `CUpti_ActivityStream`.

**CUPTI_ACTIVITY_KIND_SYNCHRONIZATION = 38**
Records for synchronization management. The corresponding activity record structure is `CUpti_ActivitySynchronization`.

**CUPTI_ACTIVITY_KIND_EXTERNAL_CORRELATION = 39**
Records for correlation of different programming APIs. The corresponding activity record structure is `CUpti_ActivityExternalCorrelation`.

**CUPTI_ACTIVITY_KIND_NVLINK = 40**
NVLink information. The corresponding activity record structure is `CUpti_ActivityNvLink2`.

**CUPTI_ACTIVITY_KIND_INSTANTANEOUS_EVENT = 41**
Instantaneous Event information. The corresponding activity record structure is `CUpti_ActivityInstantaneousEvent`.

CUPTI_ACTIVITY_KIND_INSTANTANEOUS_EVENT_INSTANCE = 42
Instantaneous Event information for a specific event domain instance. The corresponding activity record structure is `CUpti_ActivityInstantaneousEventInstance`.

CUPTI_ACTIVITY_KIND_INSTANTANEOUS_METRIC = 43
Instantaneous Metric information. The corresponding activity record structure is `CUpti_ActivityInstantaneousMetric`.

CUPTI_ACTIVITY_KIND_INSTANTANEOUS_METRIC_INSTANCE = 44
Instantaneous Metric information for a specific metric domain instance. The corresponding activity record structure is `CUpti_ActivityInstantaneousMetricInstance`.

CUPTI_ACTIVITY_KIND_MEMORY = 45
CUPTI_ACTIVITY_KIND_PCIE = 46
CUPTI_ACTIVITY_KIND_FORCE_INT = 0x7fffffff

enum CUpti_ActivityLaunchType
The type of the CUDA kernel launch.

Values

CUPTI_ACTIVITY_LAUNCH_TYPE_REGULAR = 0
The kernel was launched via a regular kernel call.

CUPTI_ACTIVITY_LAUNCH_TYPE_COOPERATIVE_SINGLE_DEVICE = 1
The kernel was launched via API `cudaLaunchCooperativeKernel()` or `cuLaunchCooperativeKernel()`.

CUPTI_ACTIVITY_LAUNCH_TYPE_COOPERATIVE_MULTI_DEVICE = 2
The kernel was launched via API `cudaLaunchCooperativeKernelMultiDevice()` or `cuLaunchCooperativeKernelMultiDevice()`.

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 CUpsti_ActivityMemoryKind
The kinds of memory accessed by a memory operation/copy.
Each kind represents the type of the memory accessed by a memory operation/copy.

Values

CUPTI_ACTIVITY_MEMORY_KIND_UNKNOWN = 0
The memory kind is unknown.

CUPTI_ACTIVITY_MEMORY_KIND_PAGEABLE = 1
The memory is pageable.

CUPTI_ACTIVITY_MEMORY_KIND_PINNED = 2
The memory is pinned.

CUPTI_ACTIVITY_MEMORY_KIND_DEVICE = 3
The memory is on the device.

CUPTI_ACTIVITY_MEMORY_KIND_ARRAY = 4
The memory is an array.

CUPTI_ACTIVITY_MEMORY_KIND_MANAGED = 5
The memory is managed

CUPTI_ACTIVITY_MEMORY_KIND_DEVICE_STATIC = 6
The memory is device static

CUPTI_ACTIVITY_MEMORY_KIND_MANAGED_STATIC = 7
The memory is managed static

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_CUPTIRESOURCE = 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_ActivityPCSamplingPeriod
Sampling period for PC sampling method. Sampling period can be set using /ref
cuptiActivityConfigurePCSampling.

Values

CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_INVALID = 0
    The PC sampling period is not set.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_MIN = 1
    Minimum sampling period available on the device.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_LOW = 2
    Sampling period in lower range.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_MID = 3
    Medium sampling period.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_HIGH = 4
    Sampling period in higher range.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_MAX = 5
    Maximum sampling period available on the device.
CUPTI_ACTIVITY_PC_SAMPLING_PERIOD_FORCE_INT = 0x7fffffff

enum CUpti_ActivityPCSamplingStallReason
The stall reason for PC sampling activity.

Values

CUPTI_ACTIVITY_PC_SAMPLING_STALL_INVALID = 0
    Invalid reason
CUPTI_ACTIVITY_PC_SAMPLING_STALL_NONE = 1
    No stall, instruction is selected for issue
CUPTI_ACTIVITY_PC_SAMPLING_STALL_INST_FETCH = 2
Warp is blocked because next instruction is not yet available, because of instruction cache miss, or because of branching effects

CUPTI_ACTIVITY_PC_SAMPLING_STALL_EXEC_DEPENDENCY = 3
Instruction is waiting on an arithmetic dependency

CUPTI_ACTIVITY_PC_SAMPLING_STALL_MEMORY_DEPENDENCY = 4
Warp is blocked because it is waiting for a memory access to complete.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_TEXTURE = 5
Texture sub-system is fully utilized or has too many outstanding requests.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_SYNC = 6
Warp is blocked as it is waiting at __syncthreads() or at memory barrier.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_CONSTANT_MEMORY_DEPENDENCY = 7
Warp is blocked waiting for __constant__ memory and immediate memory access to complete.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_PIPE_BUSY = 8
Compute operation cannot be performed due to the required resources not being available.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_MEMORY_THROTTLE = 9
Warp is blocked because there are too many pending memory operations. In Kepler architecture it often indicates high number of memory replays.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_NOT_SELECTED = 10
Warp was ready to issue, but some other warp issued instead.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_OTHER = 11
Miscellaneous reasons

CUPTI_ACTIVITY_PC_SAMPLING_STALL_SLEEPING = 12
Sleeping.

CUPTI_ACTIVITY_PC_SAMPLING_STALL_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_ActivityStreamFlag

stream type.
The types of stream to be used with CUpti_ActivityStream.

Values

CUPTI_ACTIVITY_STREAM_CREATE_FLAG_UNKNOWN = 0
Unknown data.
CUPTI_ACTIVITY_STREAM_CREATE_FLAG_DEFAULT = 1
Default stream.
CUPTI_ACTIVITY_STREAM_CREATE_FLAG_NON_BLOCKING = 2
Non-blocking stream.
CUPTI_ACTIVITY_STREAM_CREATE_FLAG_NULL = 3
Null stream.
CUPTI_ACTIVITY_STREAM_CREATE_MASK = 0xFFFF
Stream create Mask
CUPTI_ACTIVITY_STREAM_CREATE_FLAG_FORCE_INT = 0x7fffffff

enum CUpti_ActivitySynchronizationType

Synchronization type.
The types of synchronization to be used with CUpti_ActivitySynchronization.

Values

CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_UNKNOWN = 0
Unknown data.
CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_EVENT_SYNCHRONIZE = 1
Event synchronize API.
CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_STREAM_WAIT_EVENT = 2
Stream wait event API.
CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_STREAM_SYNCHRONIZE = 3
Stream synchronize API.
CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_CONTEXT_SYNCHRONIZE = 4
Context synchronize API.
CUPTI_ACTIVITY_SYNCHRONIZATION_TYPE_FORCE_INT = 0x7fffffff

enum CUpti_ActivityThreadIdType

Thread-Id types.
CUPTI uses different methods to obtain the thread-id depending on the support and
the underlying platform. This enum documents these methods for each type. APIs
cuptiSetThreadIdType and cuptiGetThreadIdType can be used to set and get the thread-
id type.
Values

CUPTI_ACTIVITY_THREAD_ID_TYPE_DEFAULT = 0
   Default type Windows uses API GetCurrentThreadId() Linux/Mac/Android/QNX use
   POSIX pthread API pthread_self()
CUPTI_ACTIVITY_THREAD_ID_TYPE_SYSTEM = 1
   This type is based on the system API available on the underlying platform and
   thread-id obtained is supposed to be unique for the process lifetime. Windows
   uses API GetCurrentThreadId() Linux uses syscall SYS_gettid Mac uses syscall
   SYS_thread_selfid Android/QNX use gettid()
CUPTI_ACTIVITY_THREAD_ID_TYPE_FORCE_INT = 0x7fffffff

enum CUpti_ActivityUnifiedMemoryAccessType
Memory access type for unified memory page faults.
   This is valid for
   CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT and
   CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT

Values

CUPTI_ACTIVITY_UNIFIED_MEMORY_ACCESS_TYPE_UNKNOWN = 0
   The unified memory access type is not known
CUPTI_ACTIVITY_UNIFIED_MEMORY_ACCESS_TYPE_READ = 1
   The page fault was triggered by read memory instruction
CUPTI_ACTIVITY_UNIFIED_MEMORY_ACCESS_TYPE_WRITE = 2
   The page fault was triggered by write memory instruction
CUPTI_ACTIVITY_UNIFIED_MEMORY_ACCESS_TYPE_ATOMIC = 3
   The page fault was triggered by atomic memory instruction
CUPTI_ACTIVITY_UNIFIED_MEMORY_ACCESS_TYPE_PREFETCH = 4
   The page fault was triggered by memory prefetch operation

enum CUpti_ActivityUnifiedMemoryCounterKind
Kind of the Unified Memory counter.
   Many activities are associated with Unified Memory mechanism; among them are
   transfer 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 transferred from host to device
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH = 2
   Number of bytes transferred from device to host

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT = 3
   Number of CPU page faults, this is only supported on 64 bit Linux and Mac platforms

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT = 4
   Number of GPU page faults, this is only supported on devices with compute capability 6.0 and higher and 64 bit Linux platforms

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING = 5
   Thrashing occurs when data is frequently accessed by multiple processors and has to be constantly migrated around to achieve data locality. In this case the overhead of migration may exceed the benefits of locality. This is only supported on 64 bit Linux platforms.

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING = 6
   Throttling is a prevention technique used by the driver to avoid further thrashing. Here, the driver doesn't service the fault for one of the contending processors for a specific period of time, so that the other processor can run at full-speed. This is only supported on 64 bit Linux platforms.

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_REMOTE_MAP = 7
   In case throttling does not help, the driver tries to pin the memory to a processor for a specific period of time. One of the contending processors will have slow access to the memory, while the other will have fast access. This is only supported on 64 bit Linux platforms.

CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOD = 8
   Number of bytes transferred from one device to another device. This is only supported on 64 bit Linux platforms.

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 (deprecated in CUDA 7.0).

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_ActivityUnifiedMemoryMigrationCause
Migration cause of the Unified Memory counter.

This is valid for
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_HTOD
and
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH

Values

CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_UNKNOWN = 0
The unified memory migration cause is not known
CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_USER = 1
The unified memory migrated due to an explicit call from the user e.g. cudaMemPrefetchAsync
CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_COHERENCE = 2
The unified memory migrated to guarantee data coherence e.g. CPU/GPU faults on Pascal+ and kernel launch on pre-Pascal GPUs
CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_PREFETCH = 3
The unified memory was speculatively migrated by the UVM driver before being accessed by the destination processor to improve performance
CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_EVICTION = 4
The unified memory migrated to the CPU because it was evicted to make room for another block of memory on the GPU
CUPTI_ACTIVITY_UNIFIED_MEMORY_MIGRATION_CAUSE_ACCESS_COUNTERS = 5
The unified memory migrated to another processor because of access counter notifications

enum CUpTI_DeviceSupport
Device support.
Describes device support returned by API cuptiDeviceSupported.

Values

CUPTI_DEVICE_UNSUPPORTED = 0
If device is not supported.
CUPTI_DEVICE_SUPPORTED = 1
    If device is supported.
CUPTI_DEVICE_VIRTUAL = 2
    If device is a virtual GPU.

**enum CUpti_DevType**

The device type for device connected to NVLink.

**Values**

CUPTI_DEV_TYPE_INVALID = 0
CUPTI_DEV_TYPE_GPU = 1
    The device type is GPU.
CUPTI_DEV_TYPE_NPU = 2
    The device type is NVLink processing unit in CPU.
CUPTI_DEV_TYPE_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
enum CUpti_ExternalCorrelationKind
The kind of external APIs supported for correlation.
Custom correlation kinds are reserved for usage in external tools.

See also:
CUpti_ActivityExternalCorrelation

Values
CUPTI_EXTERNAL_CORRELATION_KIND_INVALID = 0
CUPTI_EXTERNAL_CORRELATION_KIND_UNKNOWN = 1
CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC = 2
CUPTI_EXTERNAL_CORRELATION_KIND_CUSTOM0 = 3
CUPTI_EXTERNAL_CORRELATION_KIND_CUSTOM1 = 4
CUPTI_EXTERNAL_CORRELATION_KIND_CUSTOM2 = 5
CUPTI_EXTERNAL_CORRELATION_KIND_SIZE
CUPTI_EXTERNAL_CORRELATION_KIND_FORCE_INT = 0x7fffffff

enum CUpti_LinkFlag
Link flags.
Describes link properties, to be used with CUpti_ActivityNvLink.

Values
CUPTI_LINK_FLAG_INVALID = 0
CUPTI_LINK_FLAG_PEER_ACCESS = (1<<1)  # Is peer to peer access supported by this link.
CUPTI_LINK_FLAG_SYSTEMEM_ACCESS = (1<<2)  # Is system memory access supported by this link.
CUPTI_LINK_FLAG_PEER_ATOMICS = (1<<3)  # Is peer atomic access supported by this link.
CUPTI_LINK_FLAG_SYSTEMEM_ATOMICS = (1<<4)  # Is system memory atomic access supported by this link.
CUPTI_LINK_FLAG_FORCE_INT = 0x7fffffff

enum CUpti_OpenAccConstructKind
The OpenAcc parent construct kind for OpenAcc activity records.

Values
CUPTI_OPENACC_CONSTRUCT_KIND_UNKNOWN = 0
CUPTI_OPENACC_CONSTRUCT_KIND_PARALLEL = 1
CUPTI_OPENACC_CONSTRUCT_KIND_KERNELS = 2
CUPTI_OPENACC_CONSTRUCT_KIND_LOOP = 3
CUPTI_OPENACC_CONSTRUCT_KIND_DATA = 4
CUPTI_OPENACC_CONSTRUCT_KIND_ENTER_DATA = 5
CUPTI_OPENACC_CONSTRUCT_KIND_EXIT_DATA = 6
CUPTI_OPENACC_CONSTRUCT_KIND_HOST_DATA = 7
CUPTI_OPENACC_CONSTRUCT_KIND.Atomic = 8
CUPTI_OPENACC_CONSTRUCT_KIND_DECLARE = 9
CUPTI_OPENACC_CONSTRUCT_KIND_INIT = 10
CUPTI_OPENACC_CONSTRUCT_KIND_SHUTDOWN = 11
CUPTI_OPENACC_CONSTRUCT_KIND_SET = 12
CUPTI_OPENACC_CONSTRUCT_KIND_UPDATE = 13
CUPTI_OPENACC_CONSTRUCT_KIND_ROUTINE = 14
CUPTI_OPENACC_CONSTRUCT_KIND_WAIT = 15
CUPTI_OPENACC_CONSTRUCT_KIND_RUNTIME_API = 16
CUPTI_OPENACC_CONSTRUCT_KIND_FORCE_INT = 0x7fffffff

defined CUpti_OpenAccEventKind

The OpenAcc event kind for OpenAcc activity records.

See also:
CUpti_ActivityKindOpenAcc

Values

CUPTI_OPENACC_EVENT_KIND_INVALID = 0
CUPTI_OPENACC_EVENT_KIND_DEVICE_INIT = 1
CUPTI_OPENACC_EVENT_KIND_DEVICE_SHUTDOWN = 2
CUPTI_OPENACC_EVENT_KIND_RUNTIME_SHUTDOWN = 3
CUPTI_OPENACC_EVENT_KIND_ENQUEUE_LAUNCH = 4
CUPTI_OPENACC_EVENT_KIND_ENQUEUE_UPLOAD = 5
CUPTI_OPENACC_EVENT_KIND_ENQUEUE_DOWNLOAD = 6
CUPTI_OPENACC_EVENT_KIND_WAIT = 7
CUPTI_OPENACC_EVENT_KIND_IMPLICIT_WAIT = 8
CUPTI_OPENACC_EVENT_KIND_COMPUTE_CONSTRUCT = 9
CUPTI_OPENACC_EVENT_KIND_UPDATE = 10
CUPTI_OPENACC_EVENT_KIND_ENTER_DATA = 11
CUPTI_OPENACC_EVENT_KIND_EXIT_DATA = 12
CUPTI_OPENACC_EVENT_KIND_CREATE = 13
CUPTI_OPENACC_EVENT_KIND_DELETE = 14
CUPTI_OPENACC_EVENT_KIND_ALLOC = 15
CUPTI_OPENACC_EVENT_KIND_FREE = 16
CUPTI_OPENACC_EVENT_KIND_FORCE_INT = 0x7fffffff
enum CUpti_PcieDeviceType

Field to differentiate whether PCIE Activity record is of a GPU or a PCI Bridge

Values

CUPTI_PCIE_DEVICE_TYPE_GPU = 0  
    PCIE GPU record
CUPTI_PCIE_DEVICE_TYPE_BRIDGE = 1  
    PCIE Bridge record
CUPTI_PCIE_DEVICE_TYPE_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 cupiActivityGetNextRecord. The number of dropped records can be read using cupiActivityGetNumDroppedRecords. 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 cuptiActivityConfigurePCSampling (CUcontext ctx, CUpti_ActivityPCSamplingConfig *config)
Set PC sampling configuration.

Parameters
ctx
The context
config
A pointer to CUpti_ActivityPCSamplingConfig structure containing PC sampling configuration.

Returns
- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_OPERATION
  if this api is called while some valid event collection method is set.
- 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

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.

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_UM_PROFILING_NOT_SUPPORTED
One potential reason is that platform (OS/arch) does not support the unified memory counters

CUPTI_ERROR_UM_PROFILING_NOT_SUPPORTED_ON_DEVICE
Indicates that the device does not support the unified memory counters

CUPTI_ERROR_UM_PROFILING_NOT_SUPPORTED_ON_NON_P2P_DEVICES
Indicates that multi-GPU configuration without P2P support between any pair of devices 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.
CUpțiResult 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.

CUpțiResult 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 cuptiActivityEnableLatencyTimestamps (uint8_t enable)
Controls the collection of queued and submitted timestamps for kernels.

Parameters
enable
is a boolean, denoting whether these timestamps should be collected

Returns
- CUPTI_SUCCESS
- CUPTI_ERROR_NOT_INITIALIZED

Description
This API is used to control the collection of queued and submitted timestamps for kernels whose records are provided through the struct CUpti_ActivityKernel4. Default value is 0, i.e. these timestamps are not collected. This API needs to be called before initialization of CUDA and this setting should not be changed during the profiling session.

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 synchronized). 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 preceded 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`.

**CupiResult cuptiActivityGetAttribute (Cupi_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`.

**CupiResult cuptiActivityGetNextRecord (uint8_t *buffer, size_t validBufferSizeBytes, Cupi_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 receiving a CUpti_BuffersCallbackCompleteFunc callback.

An example of typical usage:

```c
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 cuptiActivityPopExternalCorrelationId
(CUpti_ExternalCorrelationKind kind, uint64_t *lastId)
Pop an external correlation id for the calling thread.

Parameters
kind
   The kind of external API activities should be correlated with.
lastId
   If the function returns successful, contains the last external correlation id for this kind, can be NULL.

Returns
▪ CUPTI_SUCCESS
▪ CUPTI_ERROR_INVALID_PARAMETER
   The external API kind is invalid.
▪ CUPTI_ERROR_QUEUE_EMPTY
   No external id is currently associated with kind.
Description
This function notifies CUPTI that the calling thread is leaving an external API region.

\textbf{CUptiResult cuptiActivityPushExternalCorrelationId (CUpti\_ExternalCorrelationKind kind, uint64\_t id)}

Push an external correlation id for the calling thread.

Parameters
\begin{itemize}
\item \textbf{kind} \quad The kind of external API activities should be correlated with.
\item \textbf{id} \quad External correlation id.
\end{itemize}

Returns
\begin{itemize}
\item CUPTI\_SUCCESS
\item CUPTI\_ERROR\_INVALID\_PARAMETER \quad The external API kind is invalid
\end{itemize}

Description
This function notifies CUPTI that the calling thread is entering an external API region. When a CUPTI activity API record is created while within an external API region and CUPTI\_ACTIVITY\_KIND\_EXTERNAL\_CORRELATION is enabled, the activity API record will be preceeded by a \texttt{CUpti\_ActivityExternalCorrelation} record for each \texttt{CUpti\_ExternalCorrelationKind}.

\textbf{CUptiResult cuptiActivityRegisterCallbacks (CUpti\_BuffersCallbackRequestFunc funcBufferRequested, CUpti\_BuffersCallbackCompleteFunc funcBufferCompleted)}

Registers callback functions with CUPTI for activity buffer handling.

Parameters
\begin{itemize}
\item \textbf{funcBufferRequested} \quad 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 cuptiComputeCapabilitySupported (int major, int minor, int *support)**

Check support for a compute capability.

**Parameters**

- **major**
  The major revision number of the compute capability

- **minor**
  The minor revision number of the compute capability

- **support**
  Pointer to an integer to return the support status

**Returns**

- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER
  if support is NULL

Description

This function is used to check the support for a device based on its compute capability. It sets the **support** when the compute capability is supported by the current version of CUPTI, and clears it otherwise. This version of CUPTI might not support all GPUs sharing the same compute capability. It is suggested to use API cuptiDeviceSupported which provides correct information.

See also:

- cuptiDeviceSupported

**CUptiResult cuptiDeviceSupported (CUdevice dev, int *support)**

Check support for a compute device.

**Parameters**

- **dev**
  The device handle returned by CUDA Driver API cuDeviceGet
support
    Pointer to an integer to return the support status

Returns

➤ CUPTI_SUCCESS
➤ CUPTI_ERROR_INVALID_PARAMETER
    if support is NULL
➤ CUPTI_ERROR_INVALID_DEVICE
    if dev is not a valid device

Description

This function is used to check the support for a compute device. It sets the support
when the device is supported by the current version of CUPTI.

See also:

CUpti_DeviceSupport

See also:

cuptiComputeCapabilitySupported

CUptiResult cuptiFinalize (void)

Cleanup CUPTI.

Description

Explicitly destroys and cleans up all resources associated with CUPTI in the current
process. Any subsequent CUPTI API call will reinitialize CUPTI. The CUPTI client needs
to make sure that required CUDA synchronization and CUPTI activity buffer flush is
done before calling cuptiFinalize.

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 cuptiGetLastError (void)
Returns the last error from a cupti call or callback.

Description
Returns the last error that has been produced by any of the cupti api calls or the callback in the same host thread and resets it to CUPTI_SUCCESS.
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).

**DEPRECATED** This method is deprecated as of CUDA 8.0. Use method cuptiGetStreamIdEx instead.

CUptiResult cuptiGetStreamIdEx (CUcontext context, CUstream stream, uint8_t perThreadStream, 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
perThreadStream
    Flag to indicate if program is compiled for per-thread streams
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).

CUpTIResult cuptiGetThreadIdType
(CUpti_ActivityThreadIdType *type)
Get the thread-id type.

Returns
- CUPTI_SUCCESS
- CUPTI_ERROR_INVALID_PARAMETER
  if type is NULL

Description
Returns the thread-id type used in CUPTI
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.

CUptiResult cuptiSetThreadIdType (CUpti_ActivityThreadIdType type)
Set the thread-id type.

Returns
  ‣ CUPTI_SUCCESS
  ‣ CUPTI_ERROR_NOT_SUPPORTED
    if type is not supported on the platform

Description
CUPTI uses the method corresponding to set type to generate the thread-id. See enum / ref CUpti_ActivityThreadIdType for the list of methods. Activity records having thread-id field contain the same value. Thread id type must not be changed during the profiling session to avoid thread-id value mismatch across activity records.

#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_MAX_NVLINK_PORTS 16
Maximum NVLink port numbers.

#define CUPTI_NVLINK_INVALID_PORT -1
Invalid/unknown NVLink port number.

#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_SYNCHRONIZATION_INVALID_VALUE -1
An invalid/unknown 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 = 0xffffffff

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_CBIDRESOURCE_INVALID = 0
   Invalid resource callback ID.
CUPTI_CBIDRESOURCE_CONTEXT_CREATED = 1
   A new context has been created.
CUPTI_CBIDRESOURCECONTEXT_DESTROYSTARTING = 2
   A context is about to be destroyed.
CUPTI_CBIDRESOURCESTREAM_CREATED = 3
   A new stream has been created.
CUPTI_CBIDRESOURCESTREAM_DESTROYSTARTING = 4
   A stream is about to be destroyed.
CUPTI_CBIDRESOURCECU_INITFINISHED = 5
   The driver has finished initializing.
CUPTI_CBIDRESOURCE_MODULE_LOADED = 6
   A module has been loaded.
CUPTI_CBIDRESOURCE_MODULE_UNLOAD_STARTING = 7
   A module is about to be unloaded.
CUPTI_CBIDRESOURCE_MODULE_PROFILED = 8
   The current module which is being profiled.
CUPTI_CBIDRESOURCE_SIZE
CUPTI_CBIDRESOURCE_FORCE_INT = 0xffffffff

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` parameter.

**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.

class 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.

CUPTIDEVICE_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_MAX_SHARED_MEMORY_CACHE_CONFIG_PREFER_SHARED = 14
Get the maximum shared memory for the CU_FUNC_CACHE_PREFER_SHARED preference. Value is a uint64_t.

CUPTI_DEVICE_ATTR_MAX_SHARED_MEMORY_CACHE_CONFIG_PREFER_L1 = 15
Get the maximum shared memory for the CU_FUNC_CACHE_PREFER_L1 preference. Value is a uint64_t.

CUPTI_DEVICE_ATTR_MAX_SHARED_MEMORY_CACHE_CONFIG_PREFER_EQUAL = 16
Get the maximum shared memory for the CU_FUNC_CACHE_PREFER_EQUAL preference. Value is a uint64_t.

CUPTI_DEVICE_ATTR_FLOP_HP_PER_CYCLE = 17
Get the peak half precision flop per cycle. Value is a uint64_t.

CUPTI_DEVICE_ATTR_NVLINK_PRESENT = 18
Check if Nvlink is connected to device. Returns 1, if at least one Nvlink is connected to the device, returns 0 otherwise. Value is a uint32_t.

CUPTI_DEVICE_ATTR_GPU_CPU_NVLINK_BW = 19
Check if Nvlink is present between GPU and CPU. Returns Bandwidth, in Bytes/sec, if Nvlink is present, returns 0 otherwise. 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
CUPTI_DEVICE_ATTR_DEVICE_CLASS_TEGRA = 3

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_PROFILING_SCOPE = 5
    Profiling scope of the events. It can be either device or context or both. Value is a
    CUpti_EventProfilingScope.
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_NVLINK_TC = 3
Event is collected using NvLink throughput counter method.

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. Event values are 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. Note that collection in this mode may significantly change the overall performance characteristics of the application because kernel executions that occur between the cuptiEventGroupEnable and cuptiEventGroupDisable calls are serialized on the GPU. 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_PROFILING_SCOPE = 6
Event group scope can be set to CUPTI_EVENT_PROFILING_SCOPE_DEVICE or CUPTI_EVENT_PROFILING_SCOPE_CONTEXT for an eventGroup, before adding any event. Sets the scope of eventgroup as CUPTI_EVENT_PROFILING_SCOPE_DEVICE or CUPTI_EVENT_PROFILING_SCOPE_CONTEXT when the scope of the events that will be added is CUPTI_EVENT_PROFILING_SCOPE_BOTH. If profiling scope of event is either CUPTI_EVENT_PROFILING_SCOPE_DEVICE or CUPTI_EVENT_PROFILING_SCOPE_CONTEXT then setting this attribute will not affect the default scope. It is not allowed to add events of different scope to same eventgroup. Value is a uint32_t.

CUPTI_EVENT_GROUP_ATTR_FORCE_INT = 0x7fffffff

enum CUpti_EventProfilingScope
Profiling scope for event.

Profiling scope of event indicates if the event can be collected at context scope or device scope or both i.e. it can be collected at any of context or device scope.

Values

CUPTI_EVENT_PROFILING_SCOPE_CONTEXT = 0
Event is collected at context scope.

CUPTI_EVENT_PROFILING_SCOPE_DEVICE = 1
Event is collected at device scope.

CUPTI_EVENT_PROFILING_SCOPE_BOTH = 2
Event can be collected at device or context scope. The scope can be set using /ref cuptiEventGroupSetAttribute API.

CUPTI_EVENT_PROFILING_SCOPE_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.

typedef (*CUpti_KernelReplayUpdateFunc) (const char* kernelName, int numReplaysDone, void* customData)
Function type for getting updates on kernel replay.

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.
CUp tiResult 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: if 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.

CUp tiResult 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.

CUp티Result 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.

CUp티Result cuptiEnumEventDomains (size_t *arraySizeBytes, CUp티_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.

```c
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.

CUp tiRe sult cuptiEventDomainGetNumEvents (CUp ti_E ven tDomainID 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>ID of the event</td>
</tr>
<tr>
<td>attrib</td>
<td>The event attribute to read</td>
</tr>
<tr>
<td>valueSize</td>
<td>The size of the value buffer in bytes, and returns the number of bytes written to value</td>
</tr>
<tr>
<td>value</td>
<td>Returns the attribute's value</td>
</tr>
</tbody>
</table>

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 (CUppti_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
- CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT
  if size of eventValueBuffer or eventIdArray is not sufficient

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.

```c
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
- CUPTI_ERROR_PARAMETER_SIZE_NOT_SUFFICIENT
  - if size of `eventValueBuffer` is not sufficient

**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.
CUp티Result cuptiEventGroupSetDisable (CUp티_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.

CUp티Result cuptiEventGroupSetEnable (CUp티_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

eventIdArray
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 cuptiKernelReplaySubscribeUpdate (CUpTi_KernelReplayUpdateFunc updateFunc, void *customData)
Subscribe to kernel replay updates.

**Parameters**
updateFunc
The update function pointer
customData
Pointer to any custom data

**Returns**
- CUpTi_SUCCESS
Description
When subscribed, the function pointer passed in will be called each time a kernel run is finished during kernel replay. Previously subscribed function pointer will be replaced. Pass in NULL as the function pointer unsubscribes the update.

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_INVALID
((uint64_t)0xFFFFFFFFFFFFFFFEULL)
The value that indicates the event value is invalid.
#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_NVLINK = 5
A Nvlink related metric.
CUPTI_METRIC_CATEGORY_PCIE = 6
A PCIe related metric.
CUPTI_METRIC_CATEGORY_FORCE_INT = 0xffffffff

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
CUPTI_METRIC_PROPERTY_DEVICE_CLASS_TEGRA = 3

c 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
CUPTI_METRIC_PROPERTY_ECC_ENABLED
CUPTI_METRIC_PROPERTY_FLOP_HP_PER_CYCLE
CUPTI_METRIC_PROPERTY_GPU_CPU_NVLINK_BANDWIDTH

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

CUPTI_MetricValueKind_Int64 = 4
The metric value is a 64-bit signed integer.

CUPTI_MetricValueKind_UtilizationLevel = 5
The metric value is a utilization level, as represented by CUpti_MetricValueUtilizationLevel.

CUPTI_MetricValueKind_ForceInt = 0x7fffffff

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

Values

CUPTI_MetricValueUtilization_Idle = 0
CUPTI_MetricValueUtilization_Low = 2
CUPTI_MetricValueUtilization_Mid = 5
CUPTI_MetricValueUtilization_High = 8
CUPTI_MetricValueUtilization_Max = 10
CUPTI_MetricValueUtilization_ForceInt = 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).

CUpptResult 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: \((\text{sum\_event\_values} \times \text{totalInstanceCount}) / \text{instanceCount}\), where \text{sum\_event\_values} is the summation of the event values across all profiled domain instances, \text{totalInstanceCount} is obtained from querying CUPTI_EVENT_DOMAIN_ATTR_TOTAL_INSTANCE_COUNT and \text{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_ActivityCudaEvent**
The activity record for CUDA event

**CUpti_ActivityDevice**
The activity record for a device. (deprecated)

**CUpti_ActivityDevice2**
The activity record for a device. (CUDA 7.0 onwards)

**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_ActivityExternalCorrelation
The activity record for correlation with external records

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. (deprecated in CUDA 9.0)

CUpti_ActivityGlobalAccess3
The activity record for source-level global access

CUpti_ActivityInstantaneousEvent
The activity record for an instantaneous CUPTI event

CUpti_ActivityInstantaneousEventInstance
The activity record for an instantaneous CUPTI event with event domain instance information

CUpti_ActivityInstantaneousMetric
The activity record for an instantaneous CUPTI metric

CUpti_ActivityInstantaneousMetricInstance
The instantaneous activity record for a CUPTI metric with instance information

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

CUpti_ActivityInstructionExecution
The activity record for source-level instruction execution

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). (deprecated in CUDA 9.0)

CUpti_ActivityKernel4
The activity record for a kernel

CUpti_ActivityMarker
The activity record providing a marker which is an instantaneous point in time. (deprecated in CUDA 8.0)

CUpti_ActivityMarker2
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_ActivityMemory
   The activity record for memory
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
CUpti_ActivityModule
   The activity record for a CUDA module
CUpti_ActivityName
   The activity record providing a name
CUpti_ActivityNvLink
   NVLink information. (deprecated in CUDA 9.0)
CUpti_ActivityNvLink2
   NVLink information
CUpti_ActivityObjectKindId
   Identifiers for object kinds as specified by CUpti_ActivityObjectKind
CUpti_ActivityOpenAcc
   The base activity record for OpenAcc records
CUpti_ActivityOpenAccData
   The activity record for OpenACC data
CUpti_ActivityOpenAccLaunch
   The activity record for OpenACC launch
CUpti_ActivityOpenAccOther
   The activity record for OpenACC other
CUpti_ActivityOverhead
   The activity record for CUPTI and driver overheads
CUpti_ActivityPcie
   PCI devices information required to construct topology
CUpti_ActivityPCSampling
   The activity record for PC sampling. (deprecated in CUDA 8.0)
CUpti_ActivityPCSampling2
   The activity record for PC sampling. (deprecated in CUDA 9.0)
CUpti_ActivityPCSampling3
   The activity record for PC sampling
CUpti_ActivityPCSamplingConfig
   PC sampling configuration structure
CUpti_ActivityPCSamplingRecordInfo
   The activity record for record status for PC sampling
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_ActivityStream
  The activity record for CUDA stream
CUpti_ActivitySynchronization
  The activity record for synchronization management
CUpti_ActivityUnifiedMemoryCounter
  The activity record for Unified Memory counters (deprecated in CUDA 7.0)
CUpti_ActivityUnifiedMemoryCounter2
  The activity record for Unified Memory counters (CUDA 7.0 and beyond)
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 instruction was executed per warp. It will be incremented regardless of predicate or condition code.

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 instruction was executed per warp. It will be incremented regardless of predicate or condition code.

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
**3.8. CUpti_ActivityCudaEvent Struct Reference**

The activity record for CUDA event.

This activity is used to track recorded events. (CUPTI_ACTIVITY_KIND_CUDA_EVENT).

**uint32_t CUpti_ActivityCudaEvent::contextId**

The ID of the context where the event was recorded.

**uint32_t CUpti_ActivityCudaEvent::correlationId**

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

**uint32_t CUpti_ActivityCudaEvent::eventId**

A unique event ID to identify the event record.

**CUpti_ActivityKind CUpti_ActivityCudaEvent::kind**

The activity record kind, must be CUPTI_ACTIVITY_KIND_CUDA_EVENT.

**uint32_t CUpti_ActivityCudaEvent::pad**

Undefined. Reserved for internal use.
uint32_t CUpti_ActivityCudaEvent::streamId
The compute stream where the event was recorded.

3.9. CUpti_ActivityDevice Struct Reference
The activity record for a device. (deprecated).
This activity record represents information about a GPU device (CUPTI_ACTIVITY_KIND_DEVICE). Device activity is now reported using the CUpti_ActivityDevice2 activity record.

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

uint32_t CUpti_ACTIVITY_DEVICE::maxSharedMemoryPerBlock
Maximum amount of shared memory that can be assigned to a block, in bytes.

uint32_t CUpti_ACTIVITY_DEVICE::maxThreadsPerBlock
Maximum number of threads allowed in a block.

uint32_t CUpti_ACTIVITY_DEVICE::maxWarpsPerMultiprocessor
Maximum number of warps that can be present on a multiprocessor at any given time.

const char *CUpti_ACTIVITY_DEVICE::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_ACTIVITY_DEVICE::numMemcpyEngines
Number of memory copy engines on the device.

uint32_t CUpti_ACTIVITY_DEVICE::numMultiprocessors
Number of multiprocessors on the device.

uint32_t CUpti_ACTIVITY_DEVICE::numThreadsPerWarp
The number of threads per warp on the device.

3.10. CUpti_ACTIVITY_DEVICE2 Struct Reference
The activity record for a device. (CUDA 7.0 onwards).
This activity record represents information about a GPU device (CUPTI_ACTIVITY_KIND_DEVICE).
**uint32_t**
**CUpti_ActivityDevice2::computeCapabilityMajor**
Compute capability for the device, major number.

**uint32_t**
**CUpti_ActivityDevice2::computeCapabilityMinor**
Compute capability for the device, minor number.

**uint32_t** **CUpti_ActivityDevice2::constantMemorySize**
The amount of constant memory on the device, in bytes.

**uint32_t** **CUpti_ActivityDevice2::coreClockRate**
The core clock rate of the device, in kHz.

**uint32_t** **CUpti_ActivityDevice2::eccEnabled**
ECC enabled flag for device

**CUpti_ActivityFlag** **CUpti_ActivityDevice2::flags**
The flags associated with the device.

**See also:**
**CUpti_ActivityFlag**

**uint64_t**
**CUpti_ActivityDevice2::globalMemoryBandwidth**
The global memory bandwidth available on the device, in kBytes/sec.

**uint64_t** **CUpti_ActivityDevice2::globalMemorySize**
The amount of global memory on the device, in bytes.

**uint32_t** **CUpti_ActivityDevice2::id**
The device ID.
CUpti_ActivityKind CUpti_ActivityDevice2::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_DEVICE.

uint32_t CUpti_ActivityDevice2::l2CacheSize
The size of the L2 cache on the device, in bytes.

uint32_t CUpti_ActivityDevice2::maxBlockDimX
Maximum allowed X dimension for a block.

uint32_t CUpti_ActivityDevice2::maxBlockDimY
Maximum allowed Y dimension for a block.

uint32_t CUpti_ActivityDevice2::maxBlockDimZ
Maximum allowed Z dimension for a block.

uint32_t CUpti_ActivityDevice2::maxBlocksPerMultiprocessor
Maximum number of blocks that can be present on a multiprocessor at any given time.

uint32_t CUpti_ActivityDevice2::maxGridDimX
Maximum allowed X dimension for a grid.

uint32_t CUpti_ActivityDevice2::maxGridDimY
Maximum allowed Y dimension for a grid.

uint32_t CUpti_ActivityDevice2::maxGridDimZ
Maximum allowed Z dimension for a grid.

uint32_t CUpti_ActivityDevice2::maxIPC
The maximum "instructions per cycle" possible on each device multiprocessor.

uint32_t CUpti_ActivityDevice2::maxRegistersPerBlock
Maximum number of registers that can be allocated to a block.
uint32_t
CUpti_ActivityDevice2::maxRegistersPerMultiprocessor
Maximum number of 32-bit registers available per multiprocessor.

uint32_t
CUpti_ActivityDevice2::maxSharedMemoryPerBlock
Maximum amount of shared memory that can be assigned to a block, in bytes.

uint32_t
CUpti_ActivityDevice2::maxSharedMemoryPerMultiprocessor
Maximum amount of shared memory available per multiprocessor, in bytes.

uint32_t CUpti_ActivityDevice2::maxThreadsPerBlock
Maximum number of threads allowed in a block.

uint32_t
CUpti_ActivityDevice2::maxWarpsPerMultiprocessor
Maximum number of warps that can be present on a multiprocessor at any given time.

const char *CUpti_ActivityDevice2::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_ActivityDevice2::numMemcpyEngines
Number of memory copy engines on the device.

uint32_t CUpti_ActivityDevice2::numMultiprocessors
Number of multiprocessors on the device.

uint32_t CUpti_ActivityDevice2::numThreadsPerWarp
The number of threads per warp on the device.
uint32_t CUpti_ActivityDevice2::pad

Undefined. Reserved for internal use.

CUuuid CUpti_ActivityDevice2::uuid

The device UUID. This value is the globally unique immutable alphanumeric identifier of the device.

3.11. 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::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::value

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

### 3.12. 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::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::@12::@15
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::@12::@13
CUpti_ActivityEnvironment::speed
Data returned for CUPTI_ACTIVITY_ENVIRONMENT_SPEED environment kind.

CUpti_ActivityEnvironment::@12::@14
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.

3.13. 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.

3.14. 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.

```c
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.

```c
CUpti_EventDomainID CUpti_ActivityEventInstance::domain
```

The event domain ID.

```c
CUpti_EventID CUpti_ActivityEventInstance::id
```

The event ID.

```c
uint32_t CUpti_ActivityEventInstance::instance
```

The event domain instance.

```c
CUpti_ActivityKind CUpti_ActivityEventInstance::kind
```

The activity record kind, must be CUPTI_ACTIVITY_KIND_EVENT_INSTANCE.

```c
uint32_t CUpti_ActivityEventInstance::pad
```

Undefined. Reserved for internal use.

```c
uint64_t CUpti_ActivityEventInstance::value
```

The event value.

### 3.15. CUpti_ActivityExternalCorrelation Struct

The activity record for correlation with external records.

This activity record correlates native CUDA records (e.g. CUDA Driver API, kernels, memcpys, ...) with records from external APIs such as OpenACC. (CUPTI_ACTIVITY_KIND_EXTERNAL_CORRELATION).

**See also:**
CUpti_ActivityKind

uint32_t CUpti_ActivityExternalCorrelation::correlationId

The correlation ID of the associated CUDA driver or runtime API record.

uint64_t CUpti_ActivityExternalCorrelation::externalId

The correlation ID of the associated non-CUDA API record. The exact field in the associated external record depends on that record’s activity kind (See also: externalKind).

CUpti_ExternalCorrelationKind
CUpti_ActivityExternalCorrelation::externalKind

The kind of external API this record correlated to.

CUpti_ActivityKind
CUpti_ActivityExternalCorrelation::kind

The kind of this activity.

uint32_t CUpti_ActivityExternalCorrelation::reserved

Undefined. Reserved for internal use.

3.16. 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.

3.17. 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_ActivityGlobalAccess3 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 per warp. It will be incremented when at least one of thread among warp is active with predicate and condition code evaluating to true.

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.

3.18. CUpti_ActivityGlobalAccess2 Struct
Reference
The activity record for source-level global access. (deprecated in CUDA 9.0).
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_ActivityGlobalAccess3 activity record.

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 per warp. It will be incremented when at least one of thread among warp is active with predicate and condition code evaluating to true.

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.

3.19. CUpti_ActivityGlobalAccess3 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_ActivityGlobalAccess3::correlationId
The correlation ID of the kernel to which this result is associated.
**uint32_t CUpti_ActivityGlobalAccess3::executed**

The number of times this instruction was executed per warp. It will be incremented when at least one of thread among warp is active with predicate and condition code evaluating to true.

**CUPti_ActivityFlag CUpti_ActivityGlobalAccess3::flags**

The properties of this global access.

**uint32_t CUpti_ActivityGlobalAccess3::functionId**

Correlation ID with global/device function name

**CUPti_ActivityKind CUpti_ActivityGlobalAccess3::kind**

The activity record kind, must be CUPTI_ACTIVITY_KIND_GLOBAL_ACCESS.

**uint64_t CUpti_ActivityGlobalAccess3::l2_transactions**

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

**uint64_t CUpti_ActivityGlobalAccess3::pcOffset**

The pc offset for the access.

**uint32_t CUpti_ActivityGlobalAccess3::sourceLocatorId**

The ID for source locator.

**uint64_t CUpti_ActivityGlobalAccess3::theoreticalL2Transactions**

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

**uint64_t CUpti_ActivityGlobalAccess3::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.
3.20. CUpti_ActivityInstantaneousEvent Struct Reference

The activity record for an instantaneous CUPTI event.

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

**uint32_t CUpti_ActivityInstantaneousEvent::deviceId**

The device id

**CUpti_EventID CUpti_ActivityInstantaneousEvent::id**

The event ID.

**CUpti_ActivityKind CUpti_ActivityInstantaneousEvent::kind**

The activity record kind, must be CUPTI_ACTIVITY_KIND_INSTANTANEOUS_EVENT.

**uint32_t CUpti_ActivityInstantaneousEvent::reserved**

Undefined. reserved for internal use

**uint64_t CUpti_ActivityInstantaneousEvent::timestamp**

The timestamp at which event is sampled

**uint64_t CUpti_ActivityInstantaneousEvent::value**

The event value.
3.21. CUpti_ActivityInstantaneousEventInstance Struct Reference

The activity record for an instantaneous CUPTI event with event domain instance information.

This activity record represents the a CUPTI event value for a specific event domain instance (CUPTI_ACTIVITY_KIND_EVENT_INSTANCE) sampled at a particular instant. This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profiler 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.

```c
uint32_t
CUpti_ActivityInstantaneousEventInstance::deviceId
```

The device id

```c
CUpti_EventID
CUpti_ActivityInstantaneousEventInstance::id
```

The event ID.

```c
uint8_t
CUpti_ActivityInstantaneousEventInstance::instance
```

The event domain instance

```c
CUpti_ActivityKind
CUpti_ActivityInstantaneousEventInstance::kind
```

The activity record kind, must be

CUPTI_ACTIVITY_KIND_INSTANTANEOUS_EVENT_INSTANCE.

```c
uint8_t CUpti_ActivityInstantaneousEventInstance::pad
```

Undefined. reserved for internal use
The timestamp at which event is sampled.

**uint64_t**
CUpti_ActivityInstantaneousEventInstance::timestamp

The event value.

**uint64_t**
CUpti_ActivityInstantaneousEventInstance::value

### 3.22. CUpti_ActivityInstantaneousMetric Struct Reference

The activity record for an instantaneous CUPTI metric.

This activity record represents the collection of a CUPTI metric value (CUPTI_ACTIVITY_KIND_METRIC) at a particular instance. This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profiler 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_ActivityInstantaneousMetric::deviceId

The device id

**uint8_t**
CUpti_ActivityInstantaneousMetric::flags

The properties of this metric.

**See also:**

CUpti_ActivityFlag

**CUpti_MetricID**
CUpti_ActivityInstantaneousMetric::id

The metric ID.

**CUpti_ActivityKind**
CUpti_ActivityInstantaneousMetric::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_INSTANTANEOUS_METRIC.
3.23. **CUpti_ActivityInstantaneousMetricInstance Struct Reference**

The instantaneous 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) sampled at a particular time. This activity record kind is not produced by the activity API but is included for completeness and ease-of-use. Profiler 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_ActivityInstantaneousMetricInstance::deviceId`

The device id

**uint8_t**

`CUpti_ActivityInstantaneousMetricInstance::flags`

The properties of this metric.

See also:

`CUpti_ActivityFlag`

**CUpti_MetricID**

`CUpti_ActivityInstantaneousMetricInstance::id`

The metric ID.
uint8_t CUpti_ActivityInstantaneousMetricInstance::instance

The metric domain instance

CUpti_ActivityKind
CUpti_ActivityInstantaneousMetricInstance::kind

The activity record kind, must be
CUPTI_ACTIVITY_KIND_INSTANTANEOUS_METRIC_INSTANCE.

uint8_t CUpti_ActivityInstantaneousMetricInstance::pad

Undefined. reserved for internal use

uint64_t CUpti_ActivityInstantaneousMetricInstance::timestamp

The timestamp at which metric is sampled

CUpti_ActivityInstantaneousMetricInstance::value

The metric value.

3.24. CUpti_ActivityInstructionCorrelation 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_CORRELATION).

CUpti_ActivityFlag
CUpti_ActivityInstructionCorrelation::flags

The properties of this instruction.

uint32_t CUpti_ActivityInstructionCorrelation::functionId

Correlation ID with global/device function name
CUpti_ActivityKind
CUpti_ActivityInstructionCorrelation::kind
The activity record kind, must be
CUPTI_ACTIVITY_KIND_INSTRUCTION_CORRELATION.

uint32_t CUpti_ActivityInstructionCorrelation::pad
Undefined. Reserved for internal use.

uint32_t CUpti_ActivityInstructionCorrelation::pcOffset
The pc offset for the instruction.

uint32_t
CUpti_ActivityInstructionCorrelation::sourceLocatorId
The ID for source locator.

3.25. CUpti_ActivityInstructionExecution Struct
Reference
The activity record for source-level instruction execution.
This activity records result for source level instruction execution.
(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 per warp. It will be incremented
regardless of predicate or condition code.

CUpti_ActivityFlag
CUpti_ActivityInstructionExecution::flags
The properties of this instruction execution.
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.

3.26. 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_ActivityKernel4 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.

3.27. 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_ActivityKernel4 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.

3.28. CUpti_ActivityKernel3 Struct Reference
The activity record for a kernel (CUDA 6.5(with sm_52 support) onwards). (deprecated in CUDA 9.0).
This activity record represents a kernel execution (CUPTI_ACTIVITY_KIND_KERNEL and CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL). Kernel activities are now reported using the CUpti_ActivityKernel4 activity record.

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.

3.29. CUpti_ActivityKernel4 Struct Reference

The activity record for a kernel.

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

int32_t CUpti_ActivityKernel4::blockX

The X-dimension block size for the kernel.

int32_t CUpti_ActivityKernel4::blockY

The Y-dimension block size for the kernel.

int32_t CUpti_ActivityKernel4::blockZ

The Z-dimension grid size for the kernel.

CUpti_ActivityKernel4::@6

CUpti_ActivityKernel4::cacheConfig

For devices with compute capability 7.0+ cacheConfig values are not updated in case field isSharedMemoryCarveoutRequested is set

uint64_t CUpti_ActivityKernel4::completed

The completed timestamp for the kernel execution, in ns. It represents the completion of all its child kernels and the kernel itself. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the completion time is unknown.
uint32_t CUpti_ActivityKernel4::contextId

The ID of the context where the kernel is executing.

uint32_t CUpti_ActivityKernel4::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_ActivityKernel4::deviceId

The ID of the device where the kernel is executing.

int32_t CUpti_ActivityKernel4::dynamicSharedMemory

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

uint64_t CUpti_ActivityKernel4::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_ActivityKernel4::executed

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

int64_t CUpti_ActivityKernel4::gridId

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

int32_t CUpti_ActivityKernel4::gridX

The X-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel4::gridY

The Y-dimension grid size for the kernel.

int32_t CUpti_ActivityKernel4::gridZ

The Z-dimension grid size for the kernel.
```c
uint8_t
CUpti_ActivityKernel4::isSharedMemoryCarveoutRequested
```
This indicates if
CU_FUNC_ATTRIBUTE_PREFERRED_SHARED_MEMORY_CARVEOUT was updated
for the kernel launch

```c
CUpti_ActivityKind CUpti_ActivityKernel4::kind
```
The activity record kind, must be CUPTI_ACTIVITY_KIND_KERNEL or
CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL.

```c
uint8_t CUpti_ActivityKernel4::launchType
```
The indicates if the kernel was executed via a regular launch or via a single/multi device
cooperative launch.

**See also:**
CUpti_ActivityLaunchType

```c
uint32_t CUpti_ActivityKernel4::localMemoryPerThread
```
The amount of local memory reserved for each thread, in bytes.

```c
uint32_t CUpti_ActivityKernel4::localMemoryTotal
```
The total amount of local memory reserved for the kernel, in bytes.

```c
const char *CUpti_ActivityKernel4::name
```
The name of the kernel. This name is shared across all activity records representing the
same kernel, and so should not be modified.

```c
uint8_t CUpti_ActivityKernel4::padding
```
Undefined. Reserved for internal use.

```c
CUpti_ActivityPartitionedGlobalCacheConfig
CUpti_ActivityKernel4::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_ActivityKernel4::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.

**uint64_t CUpti_ActivityKernel4::queued**

The timestamp when the kernel is queued up in the command buffer, in ns. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the queued time could not be collected for the kernel. This timestamp is not collected by default. Use API `cuptiActivityEnableLatencyTimestamps()` to enable collection.

Command buffer is a buffer written by CUDA driver to send commands like kernel launch, memory copy etc to the GPU. All launches of CUDA kernels are asynchronous with respect to the host, the host requests the launch by writing commands into the command buffer, then returns without checking the GPU’s progress.

**uint16_t CUpti_ActivityKernel4::registersPerThread**

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

**uint8_t CUpti_ActivityKernel4::requested**

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

**void *CUpti_ActivityKernel4::reserved0**

Undefined. Reserved for internal use.

**uint8_t CUpti_ActivityKernel4::sharedMemoryCarveoutRequested**

Shared memory carveout value requested for the function in percentage of the total resource. The value will be updated only if field `isSharedMemoryCarveoutRequested` is set.

**uint8_t CUpti_ActivityKernel4::sharedMemoryConfig**

The shared memory configuration used for the kernel. The value is one of the `CUsharedconfig` enumeration values from cuda.h.
uint64_t CUpti_ActivityKernel4::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_ActivityKernel4::staticSharedMemory

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

uint32_t CUpti_ActivityKernel4::streamId

The ID of the stream where the kernel is executing.

uint64_t CUpti_ActivityKernel4::submitted

The timestamp when the command buffer containing the kernel launch is submitted to the GPU, in ns. A value of CUPTI_TIMESTAMP_UNKNOWN indicates that the submitted time could not be collected for the kernel. This timestamp is not collected by default. Use API cuptiActivityEnableLatencyTimestamps() to enable collection.

3.30. CUpti_ActivityMarker Struct Reference

The activity record providing a marker which is an instantaneous point in time. (deprecated in CUDA 8.0).

The marker is specified with a descriptive name and unique id (CUPTI_ACTIVITY_KIND_MARKER). Marker activity is now reported using the CUpti_ActivityMarker2 activity record.

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.

3.31. CUpti_ActivityMarker2 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).

const char *CUpti_ActivityMarker2::domain

The name of the domain to which this marker belongs to. This will be NULL for default domain.

CUpti_ActivityFlag CUpti_ActivityMarker2::flags

The flags associated with the marker.

See also:

CUpti_ActivityFlag

uint32_t CUpti_ActivityMarker2::id

The marker ID.
CUpti_ActivityKind CUpti_ActivityMarker2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_MARKER.

const char *CUpti_ActivityMarker2::name

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

CUpti_ActivityMarker2::objectId

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

CUpti_ActivityObjectKind
CUpti_ActivityMarker2::objectKind

The kind of activity object associated with this marker.

uint32_t CUpti_ActivityMarker2::pad

Undefined. Reserved for internal use.

uint64_t CUpti_ActivityMarker2::timestamp

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

3.32. 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.

3.33. 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.

3.34. 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.

3.35. CUpti_ActivityMemory Struct Reference
The activity record for memory.
This activity record represents a memory allocation and free operation (CUPTI_ACTIVITY_KIND_MEMORY).

uint64_t CUpti_ActivityMemory::address
The virtual address of the allocation
uint64_t CUpti_ActivityMemory::allocPC
The program counter of the allocation of memory

uint64_t CUpti_ActivityMemory::bytes
The number of bytes of memory allocated.

uint32_t CUpti_ActivityMemory::contextId
The ID of the context

uint32_t CUpti_ActivityMemory::deviceId
The ID of the device where the memory allocation is taking place.

uint64_t CUpti_ActivityMemory::end
The end timestamp for the memory operation, i.e. the time when memory was freed, in
ns. This will be 0 if memory is not freed in the application

uint64_t CUpti_ActivityMemory::freePC
The program counter of the freeing of memory. This will be 0 if memory is not freed in
the application

CUpti_ActivityKind CUpti_ActivityMemory::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_MEMORY

CUpti_ActivityMemoryKind
CUpti_ActivityMemory::memoryKind
The memory kind requested by the user

const char *CUpti_ActivityMemory::name
Variable name. This name is shared across all activity records representing the same
symbol, and so should not be modified.

uint32_t CUpti_ActivityMemory::processId
The ID of the process to which this record belongs to.
uint64_t CUpti_ActivityMemory::start

The start timestamp for the memory operation, i.e. the time when memory was allocated, in ns.

3.36. 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.

uint16_t CUpti_ActivityMemset::flags

The flags associated with the memset.

See also:

CUpti_ActivityFlag
CUpti_ActivityKind CUpti_ActivityMemset::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_MEMSET.

uint16_t CUpti_ActivityMemset::memoryKind
The memory kind of the memory set
See also:
CUpti_ActivityMemoryKind

void *CUpti_ActivityMemset::reserved0
Undefined. Reserved for internal use.

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.

3.37. 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.

3.38. 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.

3.39. 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.
3.40. 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.

3.41. CUpti_ActivityNvLink Struct Reference

NVLink information. (deprecated in CUDA 9.0).

This structure gives capabilities of each logical NVLink connection between two devices, gpu <-> gpu or gpu <-> CPU which can be used to understand the topology. NVLink information are now reported using the CUpti_ActivityNvLink2 activity record.
uint64_t CUpti_ActivityNvLink::bandwidth
Banwidth of NVLink in kbytes/sec

uint32_t CUpti_ActivityNvLink::domainId
Domain ID of NPU. On Linux, this can be queried using lspci.

uint32_t CUpti_ActivityNvLink::flag
Flag gives capabilities of the link

See also:
CUpti_LinkFlag

CUpti_ActivityNvLink::@17 CUpti_ActivityNvLink::idDev0
If typeDev0 is CUPTI_DEV_TYPE_GPU, UUID for device 0. CUpti_ActivityDevice2. If typeDev0 is CUPTI_DEV_TYPE_NPU, struct npu for NPU.

CUpti_ActivityNvLink::@18 CUpti_ActivityNvLink::idDev1
If typeDev1 is CUPTI_DEV_TYPE_GPU, UUID for device 1. CUpti_ActivityDevice2. If typeDev1 is CUPTI_DEV_TYPE_NPU, struct npu for NPU.

uint32_t CUpti_ActivityNvLink::index
Index of the NPU. First index will always be zero.

CUpti_ActivityKind CUpti_ActivityNvLink::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_NVLINK.

uint32_t CUpti_ActivityNvLink::nvlinkVersion
NVLink version.

uint32_t CUpti_ActivityNvLink::physicalNvLinkCount
Number of physical NVLinks present between two devices.
int8_t CUpti_ActivityNvLink::portDev0
Port numbers for maximum 4 NVLinks connected to device 0. If typeDev0 is
CUPTI_DEV_TYPE_NPU, ignore this field. In case of invalid/unknown port number,
this field will be set to value CUPTI_NVLINK_INVALID_PORT. This will be used to
correlate the metric values to individual physical link and attribute traffic to the logical
NVLink in the topology.

int8_t CUpti_ActivityNvLink::portDev1
Port numbers for maximum 4 NVLinks connected to device 1. If typeDev1 is
CUPTI_DEV_TYPE_NPU, ignore this field. In case of invalid/unknown port number,
this field will be set to value CUPTI_NVLINK_INVALID_PORT. This will be used to
correlate the metric values to individual physical link and attribute traffic to the logical
NVLink in the topology.

CUpti_DevType CUpti_ActivityNvLink::typeDev0
Type of device 0 CUpti_DevType

CUpti_DevType CUpti_ActivityNvLink::typeDev1
Type of device 1 CUpti_DevType

3.42. CUpti_ActivityNvLink2 Struct Reference
NVLink information.
This structure gives capabilities of each logical NVLink connection between two devices,
gpu<->gpu or gpu<->CPU which can be used to understand the topology.

uint64_t CUpti_ActivityNvLink2::bandwidth
Banwidth of NVLink in kbytes/sec

uint32_t CUpti_ActivityNvLink2::domainId
Domain ID of NPU. On Linux, this can be queried using lspci.

uint32_t CUpti_ActivityNvLink2::flag
Flag gives capabilities of the link

See also:
CUpti_LinkFlag

CUpti_ActivityNvLink2::@21
CUpti_ActivityNvLink2::idDev0

If typeDev0 is CUPTI_DEV_TYPE_GPU, UUID for device 0. CUpti_ActivityDevice2. If typeDev0 is CUPTI_DEV_TYPE_NPU, struct npu for NPU.

CUpti_ActivityNvLink2::@22
CUpti_ActivityNvLink2::idDev1

If typeDev1 is CUPTI_DEV_TYPE_GPU, UUID for device 1. CUpti_ActivityDevice2. If typeDev1 is CUPTI_DEV_TYPE_NPU, struct npu for NPU.

uint32_t CUpti_ActivityNvLink2::index

Index of the NPU. First index will always be zero.

CUpti_ActivityKind CUpti_ActivityNvLink2::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_NVLINK.

uint32_t CUpti_ActivityNvLink2::nvlinkVersion

NvLink version.

uint32_t CUpti_ActivityNvLink2::physicalNvLinkCount

Number of physical NVLinks present between two devices.

int8_t CUpti_ActivityNvLink2::portDev0

Port numbers for maximum 16 NVLinks connected to device 0. If typeDev0 is CUPTI_DEV_TYPE_NPU, ignore this field. In case of invalid/unknown port number, this field will be set to value CUPTI_NVLINK_INVALID_PORT. This will be used to correlate the metric values to individual physical link and attribute traffic to the logical NVLink in the topology.

int8_t CUpti_ActivityNvLink2::portDev1

Port numbers for maximum 16 NVLinks connected to device 1. If typeDev1 is CUPTI_DEV_TYPE_NPU, ignore this field. In case of invalid/unknown port number, this field will be set to value CUPTI_NVLINK_INVALID_PORT. This will be used to
correlate the metric values to individual physical link and attribute traffic to the logical NVLink in the topology.

**CUpti_DevType CUpti_ActivityNvLink2::typeDev0**

Type of device 0 *CUpti_DevType*

**CUpti_DevType CUpti_ActivityNvLink2::typeDev1**

Type of device 1 *CUpti_DevType*

### 3.43. **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.

### 3.44. **CUpti_ActivityOpenAcc Struct Reference**

The base activity record for OpenAcc records.

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

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

**See also:**
CUpti_ActivityKind

uint32_t CUpti_ActivityOpenAcc::cuContextId
CUDA context id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAcc::cuDeviceId
CUDA device id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAcc::cuProcessId
The ID of the process where the OpenACC activity is executing.

uint32_t CUpti_ActivityOpenAcc::cuStreamId
CUDA stream id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAcc::cuThreadId
The ID of the thread where the OpenACC activity is executing.

uint64_t CUpti_ActivityOpenAcc::end
CUPTI end timestamp

CUpti_OpenAccEventKind
CUpti_ActivityOpenAcc::eventKind
CUPTI OpenACC event kind

See also:
CUpti_OpenAccEventKind)

uint32_t CUpti_ActivityOpenAcc::externalId
The OpenACC correlation ID. Valid only if deviceType is acc_device_nvidia. If not 0, it uniquely identifies this record. It is identical to the externalId in the preceeding external correlation record of type CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC.

CUpti_ActivityKind CUpti_ActivityOpenAcc::kind
The kind of this activity.
CUpti_OpenAccConstructKind
CUpti_ActivityOpenAcc::parentConstruct

CUPTI OpenACC parent construct kind (See also:
CUpti_OpenAccConstructKind)

Note that for applications using PGI OpenACC runtime < 16.1, this will always be
CUPTI_OPENACC_CONSTRUCT_KIND_UNKNOWN.

uint64_t CUpti_ActivityOpenAcc::start

CUPTI start timestamp

uint32_t CUpti_ActivityOpenAcc::threadId

ThreadId

3.45. CUpti_ActivityOpenAccData Struct
Reference

The activity record for OpenACC data.
(CUPTI_ACTIVITY_KIND_OPENACC_DATA).

uint64_t CUpti_ActivityOpenAccData::bytes

Number of bytes

uint32_t CUpti_ActivityOpenAccData::cuContextId

CUDA context id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccData::cuDeviceId

CUDA device id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccData::cuProcessId

The ID of the process where the OpenACC activity is executing.
uint32_t CUpti_ActivityOpenAccData::cuStreamId
CUDA stream id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccData::cuThreadId
The ID of the thread where the OpenACC activity is executing.

uint64_t CUpti_ActivityOpenAccData::devicePtr
Device pointer if available

uint64_t CUpti_ActivityOpenAccData::end
CUPTI end timestamp

CUpti_OpenAccEventKind
CUpti_ActivityOpenAccData::eventKind
CUPTI OpenACC event kind (See also: CUpti_OpenAccEventKind)

uint32_t CUpti_ActivityOpenAccData::externalId
The OpenACC correlation ID. Valid only if deviceType is acc_device_nvidia. If not 0, it uniquely identifies this record. It is identical to the externalId in the preceding external correlation record of type CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC.

uint64_t CUpti_ActivityOpenAccData::hostPtr
Host pointer if available

CUpti_ActivityKind CUpti_ActivityOpenAccData::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_OPENACC_DATA.

uint32_t CUpti_ActivityOpenAccData::pad1
Undefined. Reserved for internal use.
uint64_t CUpti_ActivityOpenAccData::start
CUPTI start timestamp

uint32_t CUpti_ActivityOpenAccData::threadId
ThreadId

3.46. CUpti_ActivityOpenAccLaunch Struct Reference
The activity record for OpenACC launch.
(CUPTI_ACTIVITY_KIND_OPENACC_LAUNCH).

uint32_t CUpti_ActivityOpenAccLaunch::cuContextId
CUDA context id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccLaunch::cuDeviceId
CUDA device id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccLaunch::cuProcessId
The ID of the process where the OpenACC activity is executing.

uint32_t CUpti_ActivityOpenAccLaunch::cuStreamId
CUDA stream id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccLaunch::cuThreadId
The ID of the thread where the OpenACC activity is executing.

uint64_t CUpti_ActivityOpenAccLaunch::end
CUPTI end timestamp

CUpti_OpenAccEventKind
CUpti_ActivityOpenAccLaunch::eventKind
CUPTI OpenACC event kind (
See also:
CUpti_OpenAccEventKind)

uint32_t CUpti_ActivityOpenAccLaunch::externalId
The OpenACC correlation ID. Valid only if deviceType is acc_device_nvidia. If not 0, it uniquely identifies this record. It is identical to the externalId in the preceeding external correlation record of type CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC.

CUpti_ActivityKind CUpti_ActivityOpenAccLaunch::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_OPENACC_LAUNCH.

uint64_t CUpti_ActivityOpenAccLaunch::numGangs
The number of gangs created for this kernel launch

uint64_t CUpti_ActivityOpenAccLaunch::numWorkers
The number of workers created for this kernel launch

uint32_t CUpti_ActivityOpenAccLaunch::pad1
Undefined. Reserved for internal use.

uint64_t CUpti_ActivityOpenAccLaunch::start
CUPTI start timestamp

uint32_t CUpti_ActivityOpenAccLaunch::threadId
ThreadId

uint64_t CUpti_ActivityOpenAccLaunch::vectorLength
The number of vector lanes created for this kernel launch

3.47. CUpti_ActivityOpenAccOther Struct
Reference
The activity record for OpenACC other.
(CUPTI_ACTIVITY_KIND_OPENACC_OTHER).
uint32_t CUpti_ActivityOpenAccOther::cuContextId
CUDA context id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccOther::cuDeviceId
CUDA device id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccOther::cuProcessId
The ID of the process where the OpenACC activity is executing.

uint32_t CUpti_ActivityOpenAccOther::cuStreamId
CUDA stream id Valid only if deviceType is acc_device_nvidia.

uint32_t CUpti_ActivityOpenAccOther::cuThreadId
The ID of the thread where the OpenACC activity is executing.

uint64_t CUpti_ActivityOpenAccOther::end
CUPTI end timestamp

CUpti_OpenAccEventKind
CUpti_ActivityOpenAccOther::eventKind
CUPTI OpenACC event kind (See also:
CUpti_OpenAccEventKind)

uint32_t CUpti_ActivityOpenAccOther::externalId
The OpenACC correlation ID. Valid only if deviceType is acc_device_nvidia. If not 0, it uniquely identifies this record. It is identical to the externalId in the preceeding external correlation record of type CUPTI_EXTERNAL_CORRELATION_KIND_OPENACC.

CUpti_ActivityKind CUpti_ActivityOpenAccOther::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_OPENACC_OTHER.
3.48. 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::start
The CUPTI start timestamp

uint32_t CUpti_ActivityOverhead::threadId
ThreadId

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.

3.49. CUpti_ActivityPcie Struct Reference

PCI devices information required to construct topology.

This structure gives capabilities of GPU and PCI bridge connected to the PCIE bus which can be used to understand the topology.

CUpti_ActivityPcie::@26 CUpti_ActivityPcie::attr

Attributes for more information about GPU (gpuAttr) or PCI Bridge (bridgeAttr)

uint32_t CUpti_ActivityPcie::bridgedId

A unique identifier for Bridge in the Topology

uint16_t CUpti_ActivityPcie::devicId

Device ID of the bridge

CUdevice CUpti_ActivityPcie::devId

GPU device ID

uint32_t CUpti_ActivityPcie::domain

Domain for the GPU or Bridge, required to identify which PCIE bus it belongs to in multiple NUMA systems.

CUpti_ActivityPcie::@25 CUpti_ActivityPcie::id

A unique identifier for GPU or Bridge in Topology

CUpti_ActivityKind CUpti_ActivityPcie::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_PCIE.

uint16_t CUpti_ActivityPcie::linkRate

Link rate of the GPU or bridge in gigatransfers per second (GT/s)
uint16_t CUpti_ActivityPcie::linkWidth
Link width of the GPU or bridge

uint16_t CUpti_ActivityPcie::pad0
Padding for alignment

uint16_t CUpti_ActivityPcie::pcieGeneration
PCIE Generation of GPU or Bridge.

CUdevice CUpti_ActivityPcie::peerDev
CUdevice with which this device has P2P capability. This can also be obtained by
querying cuDeviceCanAccessPeer or cudaDeviceCanAccessPeer APIs

uint16_t CUpti_ActivityPcie::secondaryBus
The downstream bus number, used to search downstream devices/bridges connected to
this bridge.

CUpti_PcieDeviceType CUpti_ActivityPcie::type
Type of device in topology, CUpti_PcieDeviceType. If type is
CUPTI_PCIE_DEVICE_TYPE_GPU use devld for id and gpuAttr and if type is
CUPTI_PCIE_DEVICE_TYPE_BRIDGE use bridgeld for id and bridgeAttr.

uint16_t CUpti_ActivityPcie::upstreamBus
Upstream bus ID for the GPU or PCI bridge. Required to identify which bus it is
connected to in the topology.

CUuuid CUpti_ActivityPcie::uuidDev
UUID for the device, CUpti_ActivityDevice2.

uint16_t CUpti_ActivityPcie::vendorId
Vendor ID of the bridge
3.50. CUpdi_ActivityPCSampling Struct Reference

The activity record for PC sampling. (deprecated in CUDA 8.0).

This activity records information obtained by sampling PC (CUPTI_ACTIVITY_KIND_PC_SAMPLING). PC sampling activities are now reported using the CUpdi_ActivityPCSampling2 activity record.

```cpp
uint32_t CUpdi_ActivityPCSampling::correlationId
```

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

```cpp
CUpdi_ActivityFlag CUpdi_ActivityPCSampling::flags
```

The properties of this instruction.

```cpp
uint32_t CUpdi_ActivityPCSampling::functionId
```

Correlation ID with global/device function name.

```cpp
CUpdi_ActivityKind CUpdi_ActivityPCSampling::kind
```

The activity record kind, must be CUPTI_ACTIVITY_KIND_PC_SAMPLING.

```cpp
uint32_t CUpdi_ActivityPCSampling::pcOffset
```

The pc offset for the instruction.

```cpp
uint32_t CUpdi_ActivityPCSampling::samples
```

Number of times the PC was sampled with the stallReason in the record. The same PC can be sampled with different stall reasons.

```cpp
uint32_t CUpdi_ActivityPCSampling::sourceLocatorId
```

The ID for source locator.

```cpp
CUpdi_ActivityPCSamplingStallReason CUpdi_ActivityPCSampling::stallReason
```

Current stall reason. Includes one of the reasons from CUpdi_ActivityPCSamplingStallReason
3.51. CUpti_ActivityPCSampling2 Struct Reference

The activity record for PC sampling. (deprecated in CUDA 9.0).

This activity records information obtained by sampling PC (CUPTI_ACTIVITY_KIND_PC_SAMPLING). PC sampling activities are now reported using the CUpti_ActivityPCSampling3 activity record.

`uint32_t CUpti_ActivityPCSampling2::correlationId`

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

`CUpti_ActivityFlag CUpti_ActivityPCSampling2::flags`

The properties of this instruction.

`uint32_t CUpti_ActivityPCSampling2::functionId`

Correlation ID with global/device function name

`CUpti_ActivityKind CUpti_ActivityPCSampling2::kind`

The activity record kind, must be CUPTI_ACTIVITY_KIND_PC_SAMPLING.

`uint32_t CUpti_ActivityPCSampling2::latencySamples`

Number of times the PC was sampled with the stallReason in the record. These samples indicate that no instruction was issued in that cycle from the warp scheduler from where the warp was sampled. Field is valid for devices with compute capability 6.0 and higher.

`uint32_t CUpti_ActivityPCSampling2::pcOffset`

The pc offset for the instruction.

`uint32_t CUpti_ActivityPCSampling2::samples`

Number of times the PC was sampled with the stallReason in the record. The same PC can be sampled with different stall reasons. The count includes latencySamples.

`uint32_t CUpti_ActivityPCSampling2::sourceLocatorId`

The ID for source locator.
CUpti_ActivityPCSamplingStallReason
CUpti_ActivityPCSampling2::stallReason

Current stall reason. Includes one of the reasons from
CUpti_ActivityPCSamplingStallReason

3.52. CUpti_ActivityPCSampling3 Struct Reference
The activity record for PC sampling.
This activity records information obtained by sampling PC
(CUPTI_ACTIVITY_KIND_PC_SAMPLING).

uint32_t CUpti_ActivityPCSampling3::correlationId
The correlation ID of the kernel to which this result is associated.

CUpti_ActivityFlag CUpti_ActivityPCSampling3::flags
The properties of this instruction.

uint32_t CUpti_ActivityPCSampling3::functionId
Correlation ID with global/device function name

CUpti_ActivityKind CUpti_ActivityPCSampling3::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_PC_SAMPLING.

uint32_t CUpti_ActivityPCSampling3::latencySamples
Number of times the PC was sampled with the stallReason in the record. These samples
indicate that no instruction was issued in that cycle from the warp scheduler from where
the warp was sampled. Field is valid for devices with compute capability 6.0 and higher

uint64_t CUpti_ActivityPCSampling3::pcOffset
The pc offset for the instruction.

uint32_t CUpti_ActivityPCSampling3::samples
Number of times the PC was sampled with the stallReason in the record. The same PC
can be sampled with different stall reasons. The count includes latencySamples.
uint32_t CUpti_ActivityPCSampling3::sourceLocatorId
The ID for source locator.

CUpti_ActivityPCSamplingStallReason
CUpti_ActivityPCSampling3::stallReason
Current stall reason. Includes one of the reasons from
CUpti_ActivityPCSamplingStallReason

3.53. CUpti_ActivityPCSamplingConfig Struct Reference
PC sampling configuration structure.
This structure defines the pc sampling configuration.
See function /ref cuptiActivityConfigurePCSampling

CUpti_ActivityPCSamplingPeriod
CUpti_ActivityPCSamplingConfig::samplingPeriod
There are 5 level provided for sampling period. The level internally maps to a period
in terms of cycles. Same level can map to different number of cycles on different gpus.
No of cycles will be chosen to minimize information loss. The period chosen will be
given by samplingPeriodInCycles in /ref CUpti_ActivityPCSamplingRecordInfo for each
kernel instance.

uint32_t
CUpti_ActivityPCSamplingConfig::samplingPeriod2
This will override the period set by samplingPeriod. Value 0 in samplingPeriod2 will be
considered as samplingPeriod2 should not be used and samplingPeriod should be used.
Valid values for samplingPeriod2 are between 5 to 31 both inclusive. This will set the
sampling period to \(2^{\text{samplingPeriod2}}\) cycles.

uint32_t CUpti_ActivityPCSamplingConfig::size
Size of configuration structure. CUPTI client should set the size of the structure. It will
be used in CUPTI to check what fields are available in the structure. Used to preserve
backward compatibility.
3.54. `CUpti_ActivityPCSamplingRecordInfo` Struct Reference

The activity record for record status for PC sampling.

This activity records information obtained by sampling PC
(`CUPTI_ACTIVITY_KIND_PC_SAMPLING_RECORD_INFO`).

`uint32_t`  
`CUpti_ActivityPCSamplingRecordInfo::correlationId`

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

`uint64_t`  
`CUpti_ActivityPCSamplingRecordInfo::droppedSamples`

Number of samples that were dropped by hardware due to backpressure/overflow.

`CUpti_ActivityKind`  
`CUpti_ActivityPCSamplingRecordInfo::kind`

The activity record kind, must be
`CUPTI_ACTIVITY_KIND_PC_SAMPLING_RECORD_INFO`.

`uint64_t`  
`CUpti_ActivityPCSamplingRecordInfo::samplingPeriodInCycles`

Sampling period in terms of number of cycles.

`uint64_t`  
`CUpti_ActivityPCSamplingRecordInfo::totalSamples`

Number of times the PC was sampled for this kernel instance including all dropped samples.

3.55. `CUpti_ActivityPreemption` Struct Reference

The activity record for a preemption of a CDP kernel.

This activity record represents a preemption of a CDP kernel.
The X-dimension of the block that is preempted

The Y-dimension of the block that is preempted

The Z-dimension of the block that is preempted

The grid-id of the block that is preempted

The activity record kind, must be CUPTI_ACTIVITY_KIND_PREEMPTION

Undefined. Reserved for internal use.

kind of the preemption

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

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 per warp. It will be incremented when at least one of thread among warp is active with predicate and condition code evaluating to true.

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.

3.57. 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.

3.58. CUpti_ActivityStream Struct Reference

The activity record for CUDA stream.

This activity is used to track created streams. (CUPTI_ACTIVITY_KIND_STREAM).

uint32_t CUpti_ActivityStream::contextId

The ID of the context where the stream was created.

uint32_t CUpti_ActivityStream::correlationId

The correlation ID of the API to which this result is associated.
CUpti_ActivityStreamFlag CUpti_ActivityStream::flag
Flags associated with the stream.

CUpti_ActivityKind CUpti_ActivityStream::kind
The activity record kind, must be CUPTI_ACTIVITY_KIND_STREAM.

uint32_t CUpti_ActivityStream::priority
The clamped priority for the stream.

uint32_t CUpti_ActivityStream::streamId
A unique stream ID to identify the stream.

3.59. CUpti_ActivitySynchronization Struct Reference
The activity record for synchronization management.
This activity is used to track various CUDA synchronization APIs. (CUPTI_ACTIVITY_KIND_SYNCHRONIZATION).

uint32_t CUpti_ActivitySynchronization::contextId
The ID of the context for which the synchronization API is called. In case of context synchronization API it is the context id for which the API is called. In case of stream/event synchronization it is the ID of the context where the stream/event was created.

uint32_t CUpti_ActivitySynchronization::correlationId
The correlation ID of the API to which this result is associated.

uint32_t CUpti_ActivitySynchronization::cudaEventId
The event ID for which the synchronization API is called. A
CUPTI_SYNCHRONIZATION_INVALID_VALUE value indicate the field is not
applicable for this record. Not valid for cuCtxSynchronize, cuStreamSynchronize.

uint64_t CUpti_ActivitySynchronization::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_ActivitySynchronization::kind

The activity record kind, must be CUPTI_ACTIVITY_KIND_SYNCHRONIZATION.

uint64_t CUpti_ActivitySynchronization::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_ActivitySynchronization::streamId

The compute stream for which the synchronization API is called. A
CUPTI_SYNCHRONIZATION_INVALID_VALUE value indicate the field is not
applicable for this record. Not valid for cuCtxSynchronize, cuEventSynchronize.

CUpti_ActivitySynchronization::type

The type of record.

3.60. CUpti_ActivityUnifiedMemoryCounter Struct

Reference

The activity record for Unified Memory counters (deprecated in CUDA 7.0).

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

3.61. CUpti_ActivityUnifiedMemoryCounter2
Struct Reference

The activity record for Unified Memory counters (CUDA 7.0 and beyond).

This activity record represents a Unified Memory counter
(CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER).
uint64_t CUpti_ActivityUnifiedMemoryCounter2::address

This is the virtual base address of the page/s being transferred. For cpu and gpu faults, the virtual address for the page that faulted.

CUpti_ActivityUnifiedMemoryCounterKind

CUpti_ActivityUnifiedMemoryCounter2::counterKind

The Unified Memory counter kind

uint32_t CUpti_ActivityUnifiedMemoryCounter2::dstId

The ID of the destination CPU/device involved in the memory transfer or remote map operation. Ignore this field if counterKind is
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT or
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT
or CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING or
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING

uint64_t CUpti_ActivityUnifiedMemoryCounter2::end

The end timestamp of the counter, in ns. Ignore this field if counterKind is
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT_COUNT or
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING or
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_REMOTE_MAP. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_HTOD and
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH, timestamp is captured when activity finishes on GPU. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT, timestamp is captured when CUDA driver queues the replay of faulting memory accesses on the GPU. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING, timestamp is captured when throttling operation was finished by CUDA driver

uint32_t CUpti_ActivityUnifiedMemoryCounter2::flags

The flags associated with this record. See enums
CUpti_ActivityUnifiedMemoryAccessType if counterKind is
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT and
CUpti_ActivityUnifiedMemoryMigrationCause if counterKind is
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND.GetBytes_TRANSFER_HTOD
CUpti_ActivityKind
CUpti_ActivityUnifiedMemoryCounter2::kind
The activity record kind, must be
CUPTI_ACTIVITY_KIND_UNIFIED_MEMORY_COUNTER

uint32_t CUpti_ActivityUnifiedMemoryCounter2::pad
Undefined. Reserved for internal use.

uint32_t
CUpti_ActivityUnifiedMemoryCounter2::processId
The ID of the process to which this record belongs to.

uint32_t CUpti_ActivityUnifiedMemoryCounter2::srcId
The ID of the source CPU/device involved in the memory transfer, page fault, thrashing, throttling or remote map operation. For counterKind CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING, it is a bitwise ORing of the device IDs fighting for the memory region. Ignore this field if counterKind is CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT

uint64_t CUpti_ActivityUnifiedMemoryCounter2::start
The start timestamp of the counter, in ns. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_HTOD and
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH, timestamp is captured when activity starts on GPU. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT and
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT, timestamp is captured when CUDA driver started processing the fault. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING, timestamp is captured when CUDA driver detected thrashing of memory region. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_REMOTE_MAP and
CUpti_ActivityUnifiedMemoryRemoteMapCause if counterKind is
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THRASHING or
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THROTTLING, timestamp is captured when throttling operation was started by CUDA driver. For counterKind CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_REMOTE_MAP, timestamp is captured when CUDA driver has pushed all required operations to the processor specified by dstId.

```c
uint32_t
CUpti_ActivityUnifiedMemoryCounter2::streamId
```

The ID of the stream causing the transfer. This value of this field is invalid.

```c
uint64_t
CUpti_ActivityUnifiedMemoryCounter2::value
```

Value of the counter For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_HTOD,
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_BYTES_TRANSFER_DTOH,
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_THREASHING and
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_REMOTE_MAP,
it is the size of the memory region in bytes. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_GPU_PAGE_FAULT,
it is the number of page fault groups for the same page. For counterKind
CUPTI_ACTIVITY_UNIFIED_MEMORY_COUNTER_KIND_CPU_PAGE_FAULT_COUNT,
it is the program counter for the instruction that caused fault.

### 3.62. 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

```c
uint32_t
CUpti_ActivityUnifiedMemoryCounterConfig::deviceId
```

Device id of the target device. This is relevant only for single device scopes. (deprecated in CUDA 7.0)
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. (deprecated in CUDA 7.0)

3.63. 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.
3.64. 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.

3.65. 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.

3.66. 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.
3.67. 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_DOMAINRESOURCE`. 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.

```csharp
size_t CUpti_ModuleResourceData::cubinSize
```

The size of the cubin.

```csharp
uint32_t CUpti_ModuleResourceData::moduleId
```

Identifier to associate with the CUDA module.

```csharp
const char *CUpti_ModuleResourceData::pCubin
```

Pointer to the associated cubin.

3.68. 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 noted, 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.

```csharp
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.

```csharp
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.
3.69. 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.

3.70. 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:

A
address
   CUpti_ActivityMemory
   CUpti_ActivityUnifiedMemoryCounter2
allocPC
   CUpti_ActivityMemory
attr
   CUpti_ActivityPcie
attribute
   CUpti_ActivityDeviceAttribute

B
bandwidth
   CUpti_ActivityNvLink
   CUpti_ActivityNvLink2
blockX
   CUpti_ActivityKernel2
   CUpti_ActivityPreemption
   CUpti_ActivityKernel3
   CUpti_ActivityKernel
   CUpti_ActivityKernel4
   CUpti_ActivityCdpKernel
blockY
   CUpti_ActivityKernel
   CUpti_ActivityKernel2
   CUpti_ActivityKernel3
   CUpti_ActivityKernel4
CUpti_ActivityCdpKernel
CUpti_ActivityPreemption
blockZ
CUpti_ActivityKernel2
CUpti_ActivityKernel3
CUpti_ActivityCdpKernel
CUpti_ActivityKernel4
CUpti_ActivityPreemption
CUpti_ActivityKernel
bridgeId
CUpti_ActivityPcie
bytes
CUpti_ActivityOpenAccData
CUpti_ActivityMemory
CUpti_ActivityMemcpy
CUpti_ActivityMemcpy2
CUpti_ActivityMemset
C
cacheConfig
CUpti_ActivityKernel4
cacheConfigExecuted
CUpti_ActivityKernel
cacheConfigRequested
CUpti_ActivityKernel
callbackSite
CUpti_CallbackData
category
CUpti_ActivityMarkerData
cbid
CUpti_ActivityAPI
clocksThrottleReasons
CUpti_ActivityEnvironment
color
CUpti_ActivityMarkerData
completed
CUpti_ActivityKernel2
CUpti_ActivityKernel3
CUpti_ActivityKernel4
CUpti_ActivityCdpKernel
computeApiKind
CUpti_ActivityContext
Data Fields

**computeCapabilityMajor**
- CUpti_ActivityDevice
- CUpti_ActivityDevice2

**computeCapabilityMinor**
- CUpti_ActivityDevice
- CUpti_ActivityDevice2

**constantMemorySize**
- CUpti_ActivityDevice
- CUpti_ActivityDevice2

**context**
- CUpti_CallbackData
- CUpti_ResourceData
- CUpti_SynchronizeData

**contextId**
- CUpti_ActivityContext
- CUpti_ActivityFunction
- CUpti_ActivityModule
- CUpti_ActivityCudaEvent
- CUpti_ActivityStream
- CUpti_ActivitySynchronization
- CUpti_ActivityMemcpy
- CUpti_ActivityMemcpy2
- CUpti_ActivityMemset
- CUpti_ActivityMemory
- CUpti_ActivityKernel
- CUpti_ActivityKernel2
- CUpti_ActivityKernel3
- CUpti_ActivityKernel4
- CUpti_ActivityCdpKernel

**contextUid**
- CUpti_CallbackData

**cooling**
- CUpti_ActivityEnvironment

**copyKind**
- CUpti_ActivityMemcpy
- CUpti_ActivityMemcpy2

**coreClockRate**
- CUpti_ActivityDevice
- CUpti_ActivityDevice2

**correlationData**
- CUpti_CallbackData

**correlationId**
- CUpti_ActivityInstructionExecution
CUpti_ActivityEventInstance
CUpti_ActivityEvent
CUpti_ActivityPCSampling
CUpti_ActivityPCSampling2
CUpti_ActivityPCSampling3
CUpti_ActivitySharedAccess
CUpti_ActivityCudaEvent
CUpti_ActivityStream
CUpti_ActivityMemset
CUpti_ActivityExternalCorrelation
CUpti_CallbackData
CUpti_ActivityBranch2
CUpti_ActivityGlobalAccess2
CUpti_ActivityMemcpy
CUpti_ActivityMemcpy2
CUpti_ActivityKernel
CUpti_ActivityKernel2
CUpti_ActivityKernel3
CUpti_ActivityKernel4
CUpti_ActivityCdpKernel
CUpti_ActivityAPI
CUpti_ActivitySynchronization
CUpti_ActivityMetric
CUpti_ActivityMetricInstance
CUpti_ActivityPCSamplingRecordInfo
CUpti_ActivityGlobalAccess
CUpti_ActivityGlobalAccess3
CUpti_ActivityBranch

counterKind
  CUpti_ActivityUnifiedMemoryCounter
  CUpti_ActivityUnifiedMemoryCounter2

cubin
  CUpti_ActivityModule

cubinSize
  CUpti_ActivityModule
  CUpti_ModuleResourceData

cuContextId
  CUpti_ActivityOpenAcc
  CUpti_ActivityOpenAccLaunch
  CUpti_ActivityOpenAccData
  CUpti_ActivityOpenAccOther

cudaEventId
  CUpti_ActivitySynchronization
Data Fields

**cuDeviceId**
- CUpti_ActivityOpenAccLaunch
- CUpti_ActivityOpenAccData
- CUpti_ActivityOpenAcc
- CUpti_ActivityOpenAccOther

**cuProcessId**
- CUpti_ActivityOpenAccData
- CUpti_ActivityOpenAcc
- CUpti_ActivityOpenAccLaunch
- CUpti_ActivityOpenAccOther

**cuStreamId**
- CUpti_ActivityOpenAcc
- CUpti_ActivityOpenAccOther
- CUpti_ActivityOpenAccData
- CUpti_ActivityOpenAccLaunch

**cuThreadId**
- CUpti_ActivityOpenAccData
- CUpti_ActivityOpenAccLaunch
- CUpti_ActivityOpenAcc
- CUpti_ActivityOpenAccOther

**D**
- **dcs**
  - CUpti_ActivityObjectKindId

**deviceId**
- CUpti_ActivityUnifiedMemoryCounterConfig
- CUpti_ActivityMemcpy2
- CUpti_ActivityKernel2
- CUpti_ActivityPcie
- CUpti_ActivityInstantaneousEvent
- CUpti_ActivityKernel3
- CUpti_ActivityInstantaneousEventInstance
- CUpti_ActivityInstantaneousMetric
- CUpti_ActivityMemset
- CUpti_ActivityKernel4
- CUpti_ActivityInstantaneousMetricInstance
- CUpti_ActivityCdpKernel
- CUpti_ActivityMemcpy
- CUpti_ActivityMemory
- CUpti_ActivityDeviceAttribute
- CUpti_ActivityContext
- CUpti_ActivityKernel
- CUpti_ActivityEnvironment
Data Fields

- `devicePtr`
  - `CUpti_ActivityOpenAccData`

- `devId`
  - `CUpti_ActivityPcie`

- `diverged`
  - `CUpti_ActivityBranch`
  - `CUpti_ActivityBranch2`

- `domain`
  - `CUpti_ActivityMarker2`
  - `CUpti_ActivityPcie`
  - `CUpti_ActivityEventInstance`
  - `CUpti_ActivityEvent`

- `domainId`
  - `CUpti_ActivityNvLink2`
  - `CUpti_ActivityNvLink`

- `droppedSamples`
  - `CUpti_ActivityPCSamplingRecordInfo`

- `dstContextId`
  - `CUpti_ActivityMemcpy2`

- `dstDeviceId`
  - `CUpti_ActivityMemcpy2`

- `dstId`
  - `CUpti_ActivityUnifiedMemoryCounter2`

- `dstKind`
  - `CUpti_ActivityMemcpy2`
  - `CUpti_ActivityMemcpy`

- `dynamicSharedMemory`
  - `CUpti_ActivityKernel`
  - `CUpti_ActivityKernel4`
  - `CUpti_ActivityKernel3`
  - `CUpti_ActivityCdpKernel`
  - `CUpti_ActivityKernel2`

- `eccEnabled`
  - `CUpti_ActivityDevice2`

- `enable`
  - `CUpti_ActivityUnifiedMemoryCounterConfig`

- `enabled`
  - `CUpti_ActivityAutoBoostState`

- `end`
  - `CUpti_ActivityMemcpy`
Data Fields

CUpti_ActivityKernel
CUpti_ActivitySynchronization
CUpti_ActivityOpenAcc
CUpti_ActivityKernel2
CUpti_ActivityOpenAccData
CUpti_ActivityOpenAccLaunch
CUpti_ActivityMemcpy2
CUpti_ActivityKernel3
CUpti_ActivityOpenAccOther
CUpti_ActivityKernel4
CUpti_ActivityMemset
CUpti_ActivityCdpKernel
CUpti_ActivityAPI
CUpti_ActivityMemory
CUpti_ActivityOverhead
CUpti_ActivityUnifiedMemoryCounter2

environmentKind
CUpti_ActivityEnvironment

eventGroups
CUpti_EventGroupSet

eventId
CUpti_ActivityCudaEvent

eventKind
CUpti_ActivityOpenAcc
CUpti_ActivityOpenAccData
CUpti_ActivityOpenAccLaunch
CUpti_ActivityOpenAccOther

executed
CUpti_ActivityInstructionExecution
CUpti_ActivityGlobalAccess
CUpti_ActivityGlobalAccess2
CUpti_ActivityKernel2
CUpti_ActivityGlobalAccess3
CUpti_ActivityBranch2
CUpti_ActivityBranch
CUpti_ActivityKernel4
CUpti_ActivitySharedAccess
CUpti_ActivityCdpKernel
CUpti_ActivityKernel3

externalId
CUpti_ActivityOpenAccOther
CUpti_ActivityExternalCorrelation
CUpti_ActivityOpenAccLaunch
CUpti_ActivityOpenAccData
CUpti_ActivityOpenAcc

externalKind
CUpti_ActivityExternalCorrelation

F

fanSpeed
CUpti_ActivityEnvironment

fileName
CUpti_ActivitySourceLocator

flag
CUpti_ActivityNvLink
CUpti_ActivityNvLink2
CUpti_ActivityStream

flags
CUpti_ActivityMemset
CUpti_ActivityDeviceAttribute
CUpti_ActivityMarker
CUpti_ActivityMetric
CUpti_ActivityMarker2
CUpti_ActivityMarkerData
CUpti_ActivityMetricInstance
CUpti_ActivityInstructionExecution
CUpti_ActivityPCSampling
CUpti_ActivityGlobalAccess
CUpti_ActivityPCSampling2
CUpti_ActivityPCSampling3
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityInstantaneousMetric
CUpti_ActivityInstructionCorrelation
CUpti_ActivitySharedAccess
CUpti_ActivityMemcpy
CUpti_ActivityGlobalAccess2
CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityGlobalAccess3
CUpti_ActivityMemcpy2
CUpti_ActivityDevice
CUpti_ActivityDevice2

freePC
CUpti_ActivityMemory

functionId
CUpti_ActivityPCSampling
CUpti_ActivityBranch2
Data Fields

CUpti_ActivityInstructionExecution
CUpti_ActivitySharedAccess
CUpti_ActivityGlobalAccess2
CUpti_ActivityPCSampling2
CUpti_ActivityGlobalAccess3
CUpti_ActivityPCSampling3
CUpti_ActivityInstructionCorrelation

functionIndex
  CUpti_ActivityFunction

functionName
  CUpti_CallbackData
  CUpti_NvtxData

functionParams
  CUpti_CallbackData
  CUpti_NvtxData

functionReturnValue
  CUpti_CallbackData

G
globalMemoryBandwidth
  CUpti_ActivityDevice
  CUpti_ActivityDevice2

globalMemorySize
  CUpti_ActivityDevice2
  CUpti_ActivityDevice

gpuTemperature
  CUpti_ActivityEnvironment

gridId
  CUpti_ActivityKernel4
  CUpti_ActivityCdpKernel
  CUpti_ActivityPreemption
  CUpti_ActivityKernel2
  CUpti_ActivityKernel3

gridX
  CUpti_ActivityKernel
  CUpti_ActivityCdpKernel
  CUpti_ActivityKernel2
  CUpti_ActivityKernel3
  CUpti_ActivityKernel4

gridY
  CUpti_ActivityKernel
  CUpti_ActivityCdpKernel
  CUpti_ActivityKernel2
Data Fields

CUpti_ActivityKernel3
CUpti_ActivityKernel4

gridZ
CUpti_ActivityKernel2
CUpti_ActivityCdpKernel
CUpti_ActivityKernel4
CUpti_ActivityKernel3
CUpti_ActivityKernel

H
hostPtr
CUpti_ActivityOpenAccData

l
id
CUpti_ActivityEvent
CUpti_ActivityEventInstance
CUpti_ActivityMetricInstance
CUpti_ActivityMarker
CUpti_ActivityInstantaneousMetric
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityMarker2
CUpti_ActivitySourceLocator
CUpti_ActivityMarkerData
CUpti_ActivityFunction
CUpti_ActivityMetric
CUpti_ActivityDevice
CUpti_ActivityModule
CUpti_ActivityPcie
CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityDevice2
CUpti_ActivityInstantaneousEvent

idDev0
CUpti_ActivityNvLink
CUpti_ActivityNvLink2

idDev1
CUpti_ActivityNvLink2
CUpti_ActivityNvLink

index
CUpti_ActivityNvLink2
CUpti_ActivityNvLink

instance
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityEventInstance
CUpti_ActivityMetricInstance
isSharedMemoryCarveoutRequested
CUpti_ActivityKernel4

K
kind
CUpti_ActivityUnifiedMemoryCounterConfig
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityInstantaneousMetric
CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityInstantaneousEvent
CUpti_ActivityPcie
CUpti_ActivityNvLink2
CUpti_ActivityNvLink
CUpti_ActivityExternalCorrelation
CUpti_ActivityOpenAccOther
CUpti_ActivityOpenAccLaunch
CUpti_ActivityOpenAccData
CUpti_ActivityOpenAcc
CUpti_ActivityInstructionCorrelation
CUpti_ActivitySynchronization
CUpti_ActivityStream
CUpti_ActivityCudaEvent
CUpti_ActivitySharedAccess
CUpti_ActivityModule
CUpti_ActivityFunction
CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityUnifiedMemoryCounter
CUpti_ActivityPCSamplingRecordInfo
CUpti_ActivityPCSampling3
CUpti_ActivityPCSampling2
CUpti_ActivityPCSampling
CUpti_ActivityInstructionExecution
CUpti_ActivityEnvironment
CUpti_ActivityOverhead
CUpti_ActivityMarkerData
CUpti_ActivityMarker2
CUpti_ActivityMarker
CUpti_ActivityName
CUpti_ActivityContext
CUpti_ActivityDeviceAttribute
CUpti_ActivityDevice2
CUpti_ActivityDevice
CUpti_ActivityBranch2
CUpti_ActivityBranch
CUpti_ActivityGlobalAccess3
CUpti_ActivityGlobalAccess2
CUpti_ActivityGlobalAccess
CUpti_ActivitySourceLocator
CUpti_ActivityMetricInstance
CUpti_ActivityMetric
CUpti_ActivityEventInstance
CUpti_ActivityEvent
CUpti_ActivityAPI
CUpti_ActivityPreemption
CUpti_ActivityCdpKernel
CUpti_ActivityKernel4
CUpti_ActivityKernel3
CUpti_ActivityKernel2
CUpti_ActivityKernel
CUpti_ActivityMemory
CUpti_ActivityMemset
CUpti_ActivityMemcpy2
CUpti_ActivityMemcpy
CUpti_Activity

L

l2_transactions
    CUpti_ActivityGlobalAccess
    CUpti_ActivityGlobalAccess2
    CUpti_ActivityGlobalAccess3

l2CacheSize
    CUpti_ActivityDevice
    CUpti_ActivityDevice2

latencySamples
    CUpti_ActivityPCSampling2
    CUpti_ActivityPCSampling3

launchType
    CUpti_ActivityKernel4

lineNumber
    CUpti_ActivitySourceLocator

linkRate
    CUpti_ActivityPcie
**Data Fields**

- **linkWidth**
  - CUpti_ActivityPcie
- **localMemoryPerThread**
  - CUpti_ActivityKernel4
  - CUpti_ActivityCdpKernel
  - CUpti_ActivityKernel
  - CUpti_ActivityKernel2
  - CUpti_ActivityKernel3
- **localMemoryTotal**
  - CUpti_ActivityKernel2
  - CUpti_ActivityCdpKernel
  - CUpti_ActivityKernel
  - CUpti_ActivityKernel4
  - CUpti_ActivityKernel3

**M**

- **maxBlockDimX**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
- **maxBlockDimY**
  - CUpti_ActivityDevice2
  - CUpti_ActivityDevice
- **maxBlockDimZ**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
- **maxBlocksPerMultiprocessor**
  - CUpti_ActivityDevice2
  - CUpti_ActivityDevice
- **maxGridDimX**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
- **maxGridDimY**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
- **maxGridDimZ**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
- **maxIPC**
  - CUpti_ActivityDevice2
  - CUpti_ActivityDevice
- **maxRegistersPerBlock**
  - CUpti_ActivityDevice
  - CUpti_ActivityDevice2
maxRegistersPerMultiprocessor
  CUpti_ActivityDevice2
maxSharedMemoryPerBlock
  CUpti_ActivityDevice
  CUpti_ActivityDevice2
maxSharedMemoryPerMultiprocessor
  CUpti_ActivityDevice2
maxThreadsPerBlock
  CUpti_ActivityDevice
  CUpti_ActivityDevice2
maxWarpsPerMultiprocessor
  CUpti_ActivityDevice2
  CUpti_ActivityDevice
memoryClock
  CUpti_ActivityEnvironment
memoryKind
  CUpti_ActivityMemset
  CUpti_ActivityMemory
moduleId
  CUpti_ActivityFunction
  CUpti_ModuleResourceData

N
name
  CUpti_ActivityMemory
  CUpti_ActivityKernel
  CUpti_ActivityKernel3
  CUpti_ActivityDevice2
  CUpti_ActivityName
  CUpti_ActivityKernel4
  CUpti_ActivityMarker
  CUpti_ActivityMarker2
  CUpti_ActivityKernel2
  CUpti_ActivityCdpKernel
  CUpti_ActivityFunction
  CUpti_ActivityDevice
notPredOffThreadsExecuted
  CUpti_ActivityInstructionExecution
nullStreamId
  CUpti_ActivityContext
numEventGroups
  CUpti_EventGroupSet
Data Fields

numGangs
  CUpti_ActivityOpenAccLaunch

numMemcpyEngines
  CUpti_ActivityDevice
  CUpti_ActivityDevice2

numMultiprocessors
  CUpti_ActivityDevice2
  CUpti_ActivityDevice

numSets
  CUpti_EventGroupSets

numThreadsPerWarp
  CUpti_ActivityDevice
  CUpti_ActivityDevice2

numWorkers
  CUpti_ActivityOpenAccLaunch

nvlinkVersion
  CUpti_ActivityNvLink2
  CUpti_ActivityNvLink

O
objectId
  CUpti_ActivityName
  CUpti_ActivityMarker
  CUpti_ActivityOverhead
  CUpti_ActivityMarker2

objectKind
  CUpti_ActivityMarker
  CUpti_ActivityName
  CUpti_ActivityOverhead
  CUpti_ActivityMarker2

overheadKind
  CUpti_ActivityOverhead

P
pad
  CUpti_ActivityMemcpy2
  CUpti_ActivityKernel
  CUpti_ActivityEventInstance
  CUpti_ActivityBranch2
  CUpti_ActivityCudaEvent
  CUpti_ActivityInstructionCorrelation
  CUpti_ActivityDevice2
  CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityInstantaneousMetric
CUpti_ActivityMetric
CUpti_ActivityMarker2
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityInstructionExecution
CUpti_ActivityPreemption
CUpti_ActivityMetricInstance
CUpti_ActivityUnifiedMemoryCounter
CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityGlobalAccess2
CUpti_ActivityModule
CUpti_ActivitySharedAccess
pad0
   CUpti_ActivityPcie
pad1
   CUpti_ActivityOpenAccData
   CUpti_ActivityOpenAccLaunch
padding
   CUpti_ActivityKernel4
parentBlockX
   CUpti_ActivityCdpKernel
parentBlockY
   CUpti_ActivityCdpKernel
parentBlockZ
   CUpti_ActivityCdpKernel
parentConstruct
   CUpti_ActivityOpenAcc
parentGridId
   CUpti_ActivityCdpKernel
partitionedGlobalCacheExecuted
   CUpti_ActivityKernel3
   CUpti_ActivityKernel4
partitionedGlobalCacheRequested
   CUpti_ActivityKernel3
   CUpti_ActivityKernel4
payload
   CUpti_ActivityMarkerData
payloadKind
   CUpti_ActivityMarkerData
pcieGeneration
   CUpti_ActivityPcie
pcieLinkGen
   CUpti_ActivityEnvironment
pcieLinkWidth
  CUpți_ActivityEnvironment
pcOffset
  CUpți_ActivityPCSampling3
  CUpți_ActivityGlobalAccess2
  CUpți_ActivityGlobalAccess3
  CUpți_ActivityBranch
  CUpți_ActivityInstructionExecution
  CUpți_ActivityPCSampling
  CUpți_ActivityPCSampling2
  CUpți_ActivityGlobalAccess
  CUpți_ActivitySharedAccess
  CUpți_ActivityInstructionCorrelation
  CUpți_ActivityBranch2
pCubin
  CUpți_ModuleResourceData
peerDev
  CUpți_ActivityPcie
physicalNvLinkCount
  CUpți_ActivityNvLink2
  CUpți_ActivityNvLink
pid
  CUpți_ActivityAutoBoostState
portDev0
  CUpți_ActivityNvLink2
  CUpți_ActivityNvLink
portDev1
  CUpți_ActivityNvLink2
  CUpți_ActivityNvLink
power
  CUpți_ActivityEnvironment
powerLimit
  CUpți_ActivityEnvironment
preemptionKind
  CUpți_ActivityPreemption
priority
  CUpți_ActivityStream
processId
  CUpți_ActivityAPI
  CUpți_ActivityMemory
  CUpți_ActivityUnifiedMemoryCounter
  CUpți_ActivityUnifiedMemoryCounter2
pt
  CUpti_ActivityObjectKindId

Q
queued
  CUpti_ActivityKernel4
  CUpti_ActivityCdpKernel

R
registersPerThread
  CUpti_ActivityKernel
  CUpti_ActivityKernel2
  CUpti_ActivityKernel4
  CUpti_ActivityCdpKernel
  CUpti_ActivityKernel3
requested
  CUpti_ActivityKernel4
  CUpti_ActivityCdpKernel
  CUpti_ActivityKernel2
  CUpti_ActivityKernel3
reserved
  CUpti_ActivityExternalCorrelation
  CUpti_ActivityInstantaneousEvent
reserved0
  CUpti_ActivityMemcpy2
  CUpti_ActivityKernel2
  CUpti_ActivityKernel3
  CUpti_ActivityKernel4
  CUpti_ActivityKernel
  CUpti_ActivityMemcpy
  CUpti_ActivityMemcpy
resourceDescriptor
  CUpti_ResourceData
returnValue
  CUpti_ActivityAPI
runtimeCorrelationId
  CUpti_ActivityKernel
  CUpti_ActivityMemcpy

S
samples
  CUpti_ActivityPCSampling
  CUpti_ActivityPCSampling2
Data Fields

- `CUpti_ActivityPCSampling3`
- `samplingPeriod`
  - `CUpti_ActivityPCSamplingConfig`
- `samplingPeriod2`
  - `CUpti_ActivityPCSamplingConfig`
- `samplingPeriodInCycles`
  - `CUpti_ActivityPCSamplingRecordInfo`
- `scope`
  - `CUpti_ActivityUnifiedMemoryCounter`
  - `CUpti_ActivityUnifiedMemoryCounterConfig`
- `secondaryBus`
  - `CUpti_ActivityPcie`
- `sets`
  - `CUpti_EventGroupSets`
- `sharedMemoryCarveoutRequested`
  - `CUpti_ActivityKernel4`
- `sharedMemoryConfig`
  - `CUpti_ActivityKernel2`
  - `CUpti_ActivityKernel3`
  - `CUpti_ActivityKernel4`
  - `CUpti_ActivityCdpKernel`
- `sharedTransactions`
  - `CUpti_ActivitySharedAccess`
- `size`
  - `CUpti_ActivityPCSamplingConfig`
- `smClock`
  - `CUpti_ActivityEnvironment`
- `sourceLocatorId`
  - `CUpti_ActivityGlobalAccess`
  - `CUpti_ActivityGlobalAccess2`
  - `CUpti_ActivityGlobalAccess3`
  - `CUpti_ActivityBranch`
  - `CUpti_ActivityBranch2`
  - `CUpti_ActivityInstructionExecution`
  - `CUpti_ActivityPCSampling`
  - `CUpti_ActivityPCSampling2`
  - `CUpti_ActivityPCSampling3`
  - `CUpti_ActivitySharedAccess`
  - `CUpti_ActivityInstructionCorrelation`
- `speed`
  - `CUpti_ActivityEnvironment`
- `srcContextId`
  - `CUpti_ActivityMemcpy2`
srcDeviceId
   CUpti_ActivityMemcpy2
srcId
   CUpti_ActivityUnifiedMemoryCounter2
srcKind
   CUpti_ActivityMemcpy
   CUpti_ActivityMemcpy2
stallReason
   CUpti_ActivityPCSampling
   CUpti_ActivityPCSampling2
   CUpti_ActivityPCSampling3
start
   CUpti_ActivityCdpKernel
   CUpti_ActivityKernel
   CUpti_ActivityMemcpy
   CUpti_ActivityMemcpy2
   CUpti_ActivityMemset
   CUpti_ActivityMemory
   CUpti_ActivityKernel2
   CUpti_ActivityKernel3
   CUpti_ActivityKernel4
   CUpti_ActivityOpenAccData
   CUpti_ActivityAPI
   CUpti_ActivityOverhead
   CUpti_ActivityUnifiedMemoryCounter2
   CUpti_ActivityOpenAccLaunch
   CUpti_ActivityOpenAcc
   CUpti_ActivityOpenAccOther
   CUpti_ActivitySynchronization
staticSharedMemory
   CUpti_ActivityKernel4
   CUpti_ActivityKernel2
   CUpti_ActivityKernel3
   CUpti_ActivityKernel
   CUpti_ActivityCdpKernel
stream
   CUpti_ResourceData
   CUpti_SynchronizeData
streamId
   CUpti_ActivityKernel4
   CUpti_ActivityMemcpy
   CUpti_ActivityMemcpy2
   CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityCdpKernel
CUpti_ActivityCudaEvent
CUpti_ActivityKernel3
CUpti_ActivitySynchronization
CUpti_ActivityStream
CUpti_ActivityMemset
CUpti_ActivityKernel2
CUpti_ActivityKernel
submitted
CUpti_ActivityCdpKernel
CUpti_ActivityKernel4
symbolName
CUpti_CallbackData

T

temperature
CUpti_ActivityEnvironment

theoreticalL2Transactions
CUpti_ActivityGlobalAccess2
CUpti_ActivityGlobalAccess3

theoreticalSharedTransactions
CUpti_ActivitySharedAccess

threadId
CUpti_ActivityOpenAccLaunch
CUpti_ActivityOpenAccOther
CUpti_ActivityAPI
CUpti_ActivityOpenAcc
CUpti_ActivityOpenAccData

threadsExecuted
CUpti_ActivitySharedAccess
CUpti_ActivityGlobalAccess
CUpti_ActivityGlobalAccess2
CUpti_ActivityGlobalAccess3
CUpti_ActivityBranch
CUpti_ActivityBranch2
CUpti_ActivityInstructionExecution

timestamp
CUpti_ActivityPreemption
CUpti_ActivityMarker
CUpti_ActivityMarker2
CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityEnvironment
CUpti_ActivityInstantaneousEvent
CUpti_ActivityUnifiedMemoryCounter
CUpti_ActivityInstantaneousMetric

totalSamples
CUpti_ActivityPCSamplingRecordInfo

type
CUpti_ActivitySynchronization
CUpti_ActivityPcie

typeDev0
CUpti_ActivityNvLink
CUpti_ActivityNvLink2

typeDev1
CUpti_ActivityNvLink2
CUpti_ActivityNvLink

U
upstreamBus
CUpti_ActivityPcie

type
CUpti_ActivityDevice2

typeDev
CUpti_ActivityPcie

V
value
CUpti_ActivityMemset
CUpti_ActivityEvent
CUpti_ActivityMetric
CUpti_ActivityInstantaneousMetricInstance
CUpti_ActivityInstantaneousMetric
CUpti_ActivityInstantaneousEventInstance
CUpti_ActivityUnifiedMemoryCounter2
CUpti_ActivityInstantaneousEvent
CUpti_ActivityUnifiedMemoryCounter
CUpti_ActivityDeviceAttribute
CUpti_ActivityMetricInstance
CUpti_ActivityEventInstance
vectorLength
CUpti_ActivityOpenAccLaunch

vendorId
CUpti_ActivityPcie
Chapter 5.
LIMITATIONS

The following are known issues with the current release.

- The Continuous event collection mode
  `CUPTI_EVENT_COLLECTION_MODE_CONTINUOUS` is supported only on Tesla devices.

- Profiling results might be inconsistent when auto boost is enabled. Profiler tries to disable auto boost by default. But it might fail to do so in some conditions and profiling will continue and results will be inconsistent. API `cuptiGetAutoBoostState()` can be used 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 with compute capability 3.0 and higher.

- CUPTI doesn't populate the activity structures which are deprecated, instead the newer version of the activity structure is filled with the information.

- While collecting events in continuous mode, event reporting may be delayed i.e. event values may be returned by a later call to readEvent(s) API and the event values for the last readEvent(s) API may get lost.

- When profiling events, it is possible that the domain instance that gets profiled gives event value 0 due to absence of workload on the domain instance since CUPTI profiles one instance of the domain by default. To profile all instances of the domain, user can set event group attribute `CUPTI_EVENT_GROUP_ATTR_PROFILE_ALL_DOMAIN_INSTANCES` through API `cuptiEventGroupSetAttribute()`.

- In CUDA Toolkit 9.0, 9.1 and 9.2, CUPTI doesn't support CUDA Dynamic Parallelism (CDP) kernel launch tracing and source level metrics for devices with compute capability 7.0.

- CUPTI doesn't support tracing and profiling on virtualized GPUs.

- Profiling results might be incorrect for CUDA applications compiled with `nvcc` version older than 9.0 for devices with compute capability 6.0 and 6.1. Profiling session will continue and CUPTI will notify it using error code
CUPTI_ERROR_CUDA_COMPILER_NOT_COMPATIBLE. It is advised to recompile the application code with nvcc version 9.0 or later. Ignore this warning if code is already compiled with the recommended nvcc version.
Chapter 6. CHANGEOLOG

CUPTI changes in CUDA 9.2

CUPTI contains below changes as part of the CUDA Toolkit 9.2 release.

‣ Added support to query PCI devices information which can be used to construct the PCIE topology. See activity kind `CUPTI_ACTIVITY_KIND_PCIE` and related activity record `CUpti_ActivityPcie`.

‣ To view and analyze bandwidth of memory transfers over PCIE topologies, new set of metrics to collect total data bytes transmitted and received through PCIE are added. Those give accumulated count for all devices in the system. These metrics are collected at the device level for the entire application. And those are made available for devices with compute capability 5.2 and higher.

‣ CUPTI added support for new metrics:
  ▪ Instruction executed for different types of load and store
  ▪ Total number of cached global/local load requests from SM to texture cache
  ▪ Global atomic/non-atomic/reduction bytes written to L2 cache from texture cache
  ▪ Surface atomic/non-atomic/reduction bytes written to L2 cache from texture cache
  ▪ Hit rate at L2 cache for all requests from texture cache
  ▪ Device memory (DRAM) read and write bytes
  ▪ The utilization level of the multiprocessor function units that execute tensor core instructions for devices with compute capability 7.0

‣ A new attribute `CUPTI_EVENT_ATTR_PROFILING_SCOPE` is added under enum `CUpti_EventAttribute` to query the profiling scope of a event. Profiling scope indicates if the event can be collected at the context level or device level or both. See `Enum CUpti_EventProfilingScope` for available profiling scopes.

‣ A new error code `CUPTI_ERROR_VIRTUALIZED_DEVICE_NOT_SUPPORTED` is added to indicate that tracing and profiling on virtualized GPU is not supported.
CUPTI changes in CUDA 9.1

List of changes done as part of the CUDA Toolkit 9.1 release.

‣ Added a field for correlation ID in the activity record CUpti_ActivityStream.

CUPTI changes in CUDA 9.0

List of changes done as part of the CUDA Toolkit 9.0 release.

‣ CUPTI extends tracing and profiling support for devices with compute capability 7.0.
‣ Usage of compute device memory can be tracked through CUPTI. A new activity record CUpti_ActivityMemory and activity kind CUPTI_ACTIVITY_KIND_MEMORY are added to track the allocation and freeing of memory. This activity record includes fields like virtual base address, size, PC (program counter), timestamps for memory allocation and free calls.
‣ Unified memory profiling adds new events for thrashing, throttling, remote map and device-to-device migration on 64 bit Linux platforms. New events are added under enum CUpti_ActivityUnifiedMemoryCounterKind. Enum CUpti_ActivityUnifiedMemoryRemoteMapCause lists possible causes for remote map events.
‣ PC sampling supports wide range of sampling periods ranging from $2^{5}$ cycles to $2^{31}$ cycles per sample. This can be controlled through new field samplingPeriod2 in the PC sampling configuration struct CUpti_ActivityPCSamplingConfig.
‣ Added API cuptiDeviceSupported() to check support for a compute device.
‣ Activity record CUpti_ActivityKernel3 for kernel execution has been deprecated and replaced by new activity record CUpti_ActivityKernel4. New record gives information about queued and submit timestamps which can help to determine software and hardware latencies associated with the kernel launch. These timestamps are not collected by default. Use API cuptiActivityEnableLatencyTimestamps() to enable collection. New field launchType of type CUpti_ActivityLaunchType can be used to determine if it is a cooperative CUDA kernel launch.
‣ Activity record CUpti_ActivityPCSampling2 for PC sampling has been deprecated and replaced by new activity record CUpti_ActivityPCSampling3. New record accomodates 64-bit PC Offset supported on devices of compute capability 7.0 and higher.
‣ Activity record CUpti_ActivityNvLink for NVLink attributes has been deprecated and replaced by new activity record CUpti_ActivityNvLink2. New record accomodates increased port numbers between two compute devices.
‣ Activity record CUpti_ActivityGlobalAccess2 for source level global accesses has been deprecated and replaced by new activity record
CUpti_ActivityGlobalAccess3. New record accommodates 64-bit PC Offset supported on devices of compute capability 7.0 and higher.

- New attributes `CUPTI_ACTIVITY_ATTR_PROFILING_SEMAPHORE_POOL_SIZE` and `CUPTI_ACTIVITY_ATTR_PROFILING_SEMAPHORE_POOL_LIMIT` are added in the activity attribute enum `CUpti_ActivityAttribute` to set and get the profiling semaphore pool size and the pool limit.

### CUPTI changes in CUDA 8.0

List of changes done as part of the CUDA Toolkit 8.0 release.

- Sampling of the program counter (PC) is enhanced to point out the true latency issues, it indicates if the stall reasons for warps are actually causing stalls in the issue pipeline. Field `latencySamples` of new activity record `CUpti_ActivityPCSampling2` provides true latency samples. This field is valid for devices with compute capability 6.0 and higher. See section PC Sampling for more details.
- Support for NVLink topology information such as the pair of devices connected via NVLink, peak bandwidth, memory access permissions etc is provided through new activity record `CUpti_ActivityNvLink`. NVLink performance metrics for data transmitted/received, transmit/receive throughput and respective header overhead for each physical link. See section NVLink for more details.
- CUPTI supports profiling of OpenACC applications. OpenACC profiling information is provided in the form of new activity records `CUpti_ActivityOpenAccData`, `CUpti_ActivityOpenAccLaunch` and `CUpti_ActivityOpenAccOther`. This aids in correlating OpenACC constructs on the CPU with the corresponding activity taking place on the GPU, and mapping it back to the source code. New API `cuptiOpenACCInitialize` is used to initialize profiling for supported OpenACC runtimes. See section OpenACC for more details.
- Unified memory profiling provides GPU page fault events on devices with compute capability 6.0 and 64 bit Linux platforms. Enum `CUpti_ActivityUnifiedMemoryAccessType` lists memory access types for GPU page fault events and enum `CUpti_ActivityUnifiedMemoryMigrationCause` lists migration causes for data transfer events.
- Unified Memory profiling support is extended to Mac platform.
- Support for 16-bit floating point (FP16) data format profiling. New metrics `inst_fp_16`, `flop_count_hp_add`, `flop_count_hp_mul`, `flop_count_hp_fma`, `flop_count_hp`, `flop_hp_efficiency`, `half_precision_fu_utilization` are supported. Peak FP16 flops per cycle for device can be queried using the enum `CUPTI_DEVICE_ATTR_FLOP_HP_PER_CYCLE` added to `CUpti_DeviceAttribute`.
- Added new activity kinds `CUPTI_ACTIVITY_KIND_SYNCHRONIZATION`, `CUPTI_ACTIVITY_KIND_STREAM` and `CUPTI_ACTIVITY_KIND_CUDA_EVENT`. 

www.nvidia.com

CUPTI
to support the tracing of CUDA synchronization constructs such as context, stream and CUDA event synchronization. Synchronization details are provided in the form of new activity record \texttt{CUpti\_ActivitySynchronization}. Enum \texttt{CUpti\_ActivitySynchronizationType} lists different types of CUDA synchronization constructs.

- APIs \texttt{cuptiSetThreadIdType()}/\texttt{cuptiGetThreadIdType()} to set/get the mechanism used to fetch the thread-id used in CUPTI records. Enum \texttt{CUpti\_ActivityThreadIdType} lists all supported mechanisms.
- Added API \texttt{cuptiComputeCapabilitySupported()} to check the support for a specific compute capability by the CUPTI.
- Added support to establish correlation between an external API (such as OpenACC, OpenMP) and CUPTI API activity records. APIs \texttt{cuptiActivityPushExternalCorrelationId()} and \texttt{cuptiActivityPopExternalCorrelationId()} should be used to push and pop external correlation ids for the calling thread. Generated records of type \texttt{CUpti\_ActivityExternalCorrelation} contain both external and CUPTI assigned correlation ids.
- Added containers to store the information of events and metrics in the form of activity records \texttt{CUpti\_ActivityInstantaneousEvent}, \texttt{CUpti\_ActivityInstantaneousEventInstance}, \texttt{CUpti\_ActivityInstantaneousMetric} and \texttt{CUpti\_ActivityInstantaneousMetricInstance}. These activity records are not produced by the CUPTI, these are included for completeness and ease-of-use. Profilers built on top of CUPTI that sample events may choose to use these records to store the collected event data.
- Support for domains and annotation of synchronization objects added in NVTX v2. New activity record \texttt{CUpti\_ActivityMarker2} and enums to indicate various stages of synchronization object i.e. \texttt{CUPTI\_ACTIVITY\_FLAG\_MARKER\_SYNC\_ACQUIRE}, \texttt{CUPTI\_ACTIVITY\_FLAG\_MARKER\_SYNC\_ACQUIRE\_SUCCESS}, \texttt{CUPTI\_ACTIVITY\_FLAG\_MARKER\_SYNC\_ACQUIRE\_FAILED} and \texttt{CUPTI\_ACTIVITY\_FLAG\_MARKER\_SYNC\_RELEASE} are added.
- Unused field \texttt{runtimeCorrelationId} of the activity record \texttt{CUpti\_ActivityMemset} is broken into two fields \texttt{flags} and \texttt{memoryKind} to indicate the asynchronous behaviour and the kind of the memory used for the memset operation. It is supported by the new flag \texttt{CUPTI\_ACTIVITY\_FLAG\_MEMSET\_ASYNC} added in the enum \texttt{CUpti\_ActivityFlag}.
- Added flag \texttt{CUPTI\_ACTIVITY\_MEMORY\_KIND\_MANAGED} in the enum \texttt{CUpti\_ActivityMemoryKind} to indicate managed memory.
- API \texttt{cuptiGetStreamId} has been deprecated. A new API \texttt{cuptiGetStreamIdEx} is introduced to provide the stream id based on the legacy or per-thread default stream flag.
CUPTI changes in CUDA 7.5

List of changes done as part of the CUDA Toolkit 7.5 release.

- Device-wide sampling of the program counter (PC) is enabled by default. This was a preview feature in the CUDA Toolkit 7.0 release and it was not enabled by default.
- Ability to collect all events and metrics accurately in presence of multiple contexts on the GPU is extended for devices with compute capability 5.x.
- API `cuptiGetLastError` is introduced to return the last error that has been produced by any of the CUPTI API calls or the callbacks in the same host thread.
- Unified memory profiling is supported with MPS (Multi-Process Service)
- Callback is provided to collect replay information after every kernel run during kernel replay. See API `cuptiKernelReplaySubscribeUpdate` and callback type `CUpti_KernelReplayUpdateFunc`.
- Added new attributes in enum `CUpti_DeviceAttribute` to query maximum shared memory size for different cache preferences for a device function.

CUPTI changes in CUDA 7.0

List of changes done as part of the CUDA Toolkit 7.0 release.

- CUPTI supports device-wide sampling of the program counter (PC). Program counters along with the stall reasons from all active warps are sampled at a fixed frequency in the round robin order. Activity record `CUpti_ActivityPCSampling` enabled using activity kind `CUPTI_ACTIVITY_KIND_PC_SAMPLING` outputs stall reason along with PC and other related information. Enum `CUpti_ActivityPCSamplingStallReason` lists all the stall reasons. Sampling period is configurable and can be tuned using API `cuptiActivityConfigurePCSampling`. This feature is available on devices with compute capability 5.2.
- Added new activity record `CUpti_ActivityInstructionCorrelation` which can be used to dump source locator records for all the PCs of the function.
- All events and metrics for devices with compute capability 3.x and 5.0 can be collected accurately in presence of multiple contexts on the GPU. In previous releases only some events and metrics could be collected accurately when multiple contexts were executing on the GPU.
- Unified memory profiling is enhanced by providing fine grain data transfers to and from the GPU, coupled with more accurate timestamps with each transfer. This information is provided through new activity record `CUpti_ActivityUnifiedMemoryCounter2`, deprecating old record `CUpti_ActivityUnifiedMemoryCounter`.
- MPS tracing and profiling support is extended on multi-gpu setups.
- Activity record `CUpti_ActivityDevice` for device information has been deprecated and replaced by new activity record `CUpti_ActivityDevice2`. New
record adds device UUID which can be used to uniquely identify the device across profiler runs.

- Activity record `CUpti_ActivityKernel2` for kernel execution has been deprecated and replaced by new activity record `CUpti_ActivityKernel3`. New record gives information about Global Partitioned Cache Configuration requested and executed. Partitioned global caching has an impact on occupancy calculation. If it is ON, then a CTA can only use a half SM, and thus a half of the registers available per SM. The new fields apply for devices with compute capability 5.2 and higher. Note that this change was done in CUDA 6.5 release with support for compute capability 5.2.

**CUPTI changes in CUDA 6.5**

List of changes done 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 supported only on Tesla devices.
Changelog

- 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.

CUPTI changes in CUDA 6.0

List of changes done as part of the CUDA Toolkit 6.0 release.

- Two new CUPTI activity kinds have been introduced to enable two new types of source-correlated data collection. The `Instruction Execution` kind collects SASS-level instruction execution counts, divergence data, and predication data. The `Shared Access` kind collects source correlated data indication inefficient shared memory accesses.

- CUPTI provides support for CUDA applications using Unified Memory. A new activity record reports Unified Memory activity such as transfers to and from a GPU and the number of Unified Memory related page faults.

- CUPTI recognized and reports the special MPS context that is used by CUDA applications running on a system with MPS enabled.

- The `CUpti_ActivityContext` activity record `CUpti_ActivityContext` has been updated to introduce a new field into the structure in a backwards compatible manner. 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.

- To ensure that metric values are calculated as accurately as possible, a new metric API is introduced. Function `cuptiMetricGetRequiredEventGroupSets` can be used to get the groups of events that should be collected at the same time.

- Execution overheads introduced by CUPTI have been dramatically decreased.

- The new activity buffer API introduced in CUDA Toolkit 5.5 is required. The legacy `cuptiActivityEnqueueBuffer` and `cuptiActivityDequeueBuffer` functions have been removed.
CUPTI changes in CUDA 5.5

List of changes done as part of CUDA Toolkit 5.5 release.

- Applications that use CUDA Dynamic Parallelism can be profiled using CUPTI. Device-side kernel launches are reported using a new activity kind.
- Device attributes such as power usage, clocks, thermals, etc. are reported via a new activity kind.
- A new activity buffer API uses callbacks to request and return buffers of activity records. The existing `cuptiActivityEnqueueBuffer` and `cuptiActivityDequeueBuffer` functions are still supported but are deprecated and will be removed in a future release.
- The Event API supports kernel replay so that any number of events can be collected during a single run of the application.
- A new metric API `cuptiMetricGetValue2` allows metric values to be calculated for any device, even if that device is not available on the system.
- CUDA peer-to-peer memory copies are reported explicitly via the activity API. In previous releases these memory copies were only partially reported.
Notice

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

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

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2017 NVIDIA Corporation. All rights reserved.