CUDA Debugger API

API Reference Manual

TRM-06710-001 _v13.2 | March 2026

Table of Contents

Chapter 1. Release Notes

1.1. CUDA 13.0 Release

Major changes

- Removed support for Maxwell, Pascal and Volta architectures.
- Removed support for CUDA_ENABLE_LIGHTWEIGHT_COREDUMP.
- Reduced CPU call stack collection overhead and disabled it by default.
- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getCudaExceptionString(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get error string for CUDA Exceptions.

setNotifyNewEventCallback(CUDBGNotifyNewEventCallback callback, void* userData)

Provides the API with the function to call to notify the debugger of a new application or device event.

CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES

New capability to collect CPU call stack for kernel launches. The readCPUCallStack cann only be used when this capability is enabled.

Deprecated APIs

- readConstMemory(uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

1.2. CUDA 12.9 Release

Major changes

- Suport late attach on WSL.

- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getCbuWarpState(uint32_t dev, uint32_t sm, uint64_t warpMask, CUDBGCbuWarpState* warpStates, uint32_t numWarpStates)

Gets CBU state of a given warp.

consumeCudaLogs(CUDBGCudaLogMessage* logMessages, uint32_t numMessages, uint32_t* numConsumed)

Get CUDA error log entries. This consumes the log entries, so they will not be available in subsequent calls.

readCPUCallStack(uint32_t dev, uint64_t gridId64, uint64_t *addrs, uint32_t numAddrs, uint32_t* totalNumAddrs)

Read CPU call stack captured at the time of kernel launch.

Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

1.3. CUDA 12.8 Release

Major changes

- Support late attach on Jetson targets.
- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

readWarpResources(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpResources *resources)

Get the resources assigned to a given warp.

1.4. CUDA 12.7 Release

Major changes

- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getClusterExceptionTargetBlock(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx, bool *blockIdxValid)

Retrieves the target block index and validity status for a given device, streaming multiprocessor, and warp for cluster exceptions.

Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getGridInfo(uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)
- getClusterDim(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterDim)
- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

1.5. CUDA 12.6 Release

Major changes

- Reduced overhead for batch breakpoint updates.
- Moved constbank memory dump control to a separate flag.
- Bugfixes and performance improvements.

New APIs

CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY flag for coredump generation (generateCoredump).

In the previous API versions the CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY controlled both the global and the constant memory. Since CUDA 12.6 the constbank memory is controlled by a separate flag (the constbank memory is usually smaller than global memory, so it is feasible to dump it by default even if the global memory is skipped).

1.6. CUDA 12.5 Release

Major changes

- Significant performance improvements for memory accessing operations and stepping.
- Bugfixes and stability improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

readAllVirtualReturnAddresses(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *addrs, uint32_t numAddrs, uint32_t* callDepth, uint32_t* syscallCallDepth)

Reads all the virtual return addresses.

getSupportedDebuggerCapabilities(CUDBGCapabilityFlags* capabilities)

Returns debugger capabilities that are supported by this version of the API.

readSmException(uint32_t dev, uint32_t sm, CUDBGException_t *exception, uint64_t *errorPC, bool *errorPCValid)

Get the SM exception status if it exists.

Deprecated APIs

CUDBG_COREDUMP_SKIP_ABORT

The generateCoredump API no longer accepts the CUDBG_COREDUMP_SKIP_ABORT flag. Note that this flag was ignored in the previous versions of the API.

1.7. CUDA 12.4 Release

Major changes

- Made CUDA ELF file handling more robust and performant.
- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getDeviceInfoSizes(uint32_t dev, CUDBGDeviceInfoSizes* sizes)

Returns sizes for device info structs and defined attributes.

getDeviceInfo(uint32_t dev, CUDBGDeviceInfoQueryType_t type, void *buffer, uint32_t length, uint32_t *dataLength)

Returns full or changed device info.

Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getConstBankAddress(uint32_t dev, uint64_t gridId64, uint32_t bank, uint64_t* address, uint32_t* size)
- singleStepWarp(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t laneHint, uint32_t nsteps, uint32_t flags, uint64_t *warpMask)

1.8. CUDA 12.3 Release

Major changes

- Support generating coredumps after the debugger has attached.
- Bugfixes and performance improvements.

New APIs

getConstBankAddress(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t bank, uint32_t offset, uint64_t* address)

Returns sizes for device info structs and defined attributes.

generateCoredump(const char* filename, CUDBGCoredumpGenerationFlags flags)

Generates a coredump for the current GPU state.

Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getLoadedFunctionInfo(uint32_t devId, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t startIndex, uint32_t numEntries)

Deprecated APIs

disassemble() deprecation notice

The disassemble() API function is deprecated. It will be dropped in an upcoming release. API consumers should use the nvdisasm utility instead.

1.9. CUDA 12.2 Release

Major changes

- Switch to the new debugger back-end (Unified Debugger) on WSL.
- Switch to the new debugger back-end (Unified Debugger) on Jetson.
- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getErrorStringEx(char *buf, uint32_t bufSz, uint32_t *msgSz)

Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string. The error message is specific to the last failed API call and is invalidated after every API call.

1.10. CUDA 12.1 Release

Major changes

- Improved support for single stepping.
- Bugfixes and performance improvements.

1.11. CUDA 12.0 Release

Major changes

- Debugging support for application using CUDA Dynamic Parallelism V2.
- Improved support for latest GPU architectures.
- Bugfixes and performance improvements.

New APIs

The following APIs were added in this release. Please refer to the Modules documentation for more details about the new methods.

getClusterDim(uint32_t dev, uint64_t gridId64, CuDim3 *clusterDim)
Get the number of blocks in the given cluster.

readClusterIdx(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterIdx)
Get the number of blocks in the given cluster.

Updated APIs

The following APIs were updated in this release. Please refer to the method documentation for the details. The old code, compiled for the older versions of the API will still work.

- getGridInfo(uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)
- readWarpState(uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

1.12. CUDA 11.8 Release

New Unified Debugger backend
A new debugger backend named the Unified Debugger (UD) has been introduced on Linux platforms with this release. UD is supported across multiple platforms including both Windows and Linux. The UD should mostly be transparent to existing clients of the API. The previous debugger backend, known as the classic debugger backend, can still be used by setting the environment variable CUDBG_USE_LEGACY_DEBUGGER to 1. UD is not supported on Maxwell GPUs. The clients of the API shall switch to the classic backend if Maxwell support is required.

Device side cudaDeviceSynchronize() undefined behavior
The clients of the API shall prevent the use of SingleStepWarp in the deprecated cudaDeviceSynchronize() function. Instead, revert to stepping over the call with a BP set and resume.

CUDBG_EVENT_KERNEL_READY events are no longer delivered for GPU-launched grids
CUDBG_EVENT_KERNEL_READY events for GPU-launched grids that were delivered over the ASYNC event pipe will no longer be sent. GPU-launched here refers to codes making use of CUDA Dynamic Parallelism. The existing implementation for this use case was imprecise. The callback did not report all GPU-launched grids before execution has begun, only those found on the deivce currently executing that were not previously reported during their launch. This functionality may be reintroduced in a future release. If this functionality is strictly required, the classic debugger backend can be used.

getLoadedFunctionInfo
Added a new getLoadedFunctionInfo call to obtain the section number and address of loaded functions for a given module.

1.13. CUDA 7.0 Release

Stability improvements. No API additions or changes.

1.14. CUDA 6.5 Release

Predicate registers
The per-thread predicate registers can be accessed and modified via the readPredicates() and writePredicates() calls. Each of these calls expects a buffer of sufficient size to cover all predicates for the current GPU architecture. The number of current predicate registers can be read back via the getNumPredicates() API call.

Condition code register
The per-thread condition code register can be accessed and modified via the readCCRegister() and writeCCRegister() calls. The condition code register is a unsigned 32-bit register, whose format may vary by GPU architecture.

Device Name
The getDeviceName() API returns a string containing the publically exposed product name of the GPU.

API Error Reporting Improvement
The symbol CUDBG_REPORT_DRIVER_API_ERROR_FLAGS points to an unsigned 32-bit integer in the application's process space that controls API error reporting. The values that can be written into this flag are specified in the CUDBGReportDriverApiErrorFlags enum. In 6.5, setting the bit corresponding to CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_SUPPRESS_NOT_READY in the variable CUDBG_REPORT_DRIVER_API_ERROR_FLAGS is supported. This will prevent CUDA API calls that return the runtime API error code cudaErrorNotReady or the driver API error code cuErrorNotReady from executing the CUDA API error reporting function.

Chapter 2. Introduction

This document describes the API for the set routines and data structures available in the CUDA library to any debugger.

Starting with 3.0, the CUDA debugger API includes several major changes, of which only few are directly visible to end-users:

- Performance is greatly improved, both with respect to interactions with the debugger and the performance of applications being debugged.
- The format of cubins has changed to ELF and, as a consequence, most restrictions on debug compilations have been lifted. More information about the new object format is included below.

The debugger API has significantly changed, reflected in the CUDA-GDB sources.

2.1. Debugger API

The CUDA Debugger API was developed with the goal of adhering to the following principles:

- Policy free
- Explicit
- Axiomatic
- Extensible
- Machine oriented

Being explicit is another way of saying that we minimize the assumptions we make. As much as possible the API reflects machine state, not internal state.

There are two major "modes" of the devices: stopped or running. We switch between these modes explicitly with suspendDevice and resumeDevice, though the machine may suspend on its own accord, for example when hitting a breakpoint.

Only when stopped, can we query the machine's state. Warp state includes which function is it runnning, which block, which lanes are valid, etc.

As of CUDA 6.0, state collection functions in the debug API will return CUDBG_ERROR_RUNNING_DEVICE if called without first calling the suspendDevice entry point to ensure the device is stopped.

Clients of the debug API should suspend all devices before servicing a CUDBGEvent. A valid CUDBGEvent is only guaranteed to be returned after the notification callback set using CUDBGAPI_st::setNotifyNewEventCallback() is executed. Any debug API entry point will return CUDBG_ERROR_RECURSIVE_API_CALL when the call is made from within the notification callback set using CUDBGAPI_st::setNotifyNewEventCallback().

2.2. ELF and DWARF

CUDA applications are compiled in ELF binary format.

Starting with CUDA 6.0, DWARF device information is obtained through an API call of CUDBGAPI_st::getElfImageByHandle using the handle exposed from CUDBGEvent of type CUDBG_EVENT_ELF_IMAGE_LOADED. This means that the information is not available until runtime, after the CUDA driver has loaded. The DWARF device information lifetime is valid until it is unloaded, which presents a CUDBGEvent of type CUDBG_EVENT_ELF_IMAGE_UNLOADED.

In CUDA 5.5 and earlier, the DWARF device information was returned as part of the `CUDBGEvent` of type CUDBG_EVENT_ELF_IMAGE_LOADED. The pointers presented in CUDBGEvent55 were read-only pointers to memory managed by the debug API. The memory pointed to was implicitly scoped to the lifetime of the loading CUDA context. Accessing the returned pointers after the context was destroyed resulted in undefined behavior.

DWARF device information contains physical addresses for all device memory regions except for code memory. The address class field (DW_AT_address_class) is set for all device variables, and is used to indicate the memory segment type (ptxStorageKind). The physical addresses must be accessed using several segment-specific API calls.

For memory reads, see:

- CUDBGAPI_st::readCodeMemory()
- CUDBGAPI_st::readConstMemory()
- CUDBGAPI_st::readGlobalMemory()
- CUDBGAPI_st::readParamMemory()
- CUDBGAPI_st::readSharedMemory()
- CUDBGAPI_st::readLocalMemory()
- CUDBGAPI_st::readTextureMemory()

For memory writes, see:

- CUDBGAPI_st::writeGlobalMemory()
- CUDBGAPI_st::writeParamMemory()
- CUDBGAPI_st::writeSharedMemory()
- CUDBGAPI_st::writeLocalMemory()

Access to code memory requires a virtual address. This virtual address is embedded for all device code sections in the device ELF image. See the API call:

- CUDBGAPI_st::readVirtualPC()

Here is a typical DWARF entry for a device variable located in memory:

```
<2><321>: Abbrev Number: 18 (DW_TAG_formal_parameter)
     DW_AT_decl_file   : 27
     DW_AT_decl_line   : 5
     DW_AT_name        : res
     DW_AT_type        : <2c6>
     DW_AT_location    : 9 byte block: 3 18 0 0 0 0 0 0 0       (DW_OP_addr: 18)
     DW_AT_address_class: 7
```

The above shows that variable 'res' has an address class of 7 (ptxParamStorage). Its location information shows it is located at address 18 within the parameter memory segment.

Local variables are no longer spilled to local memory by default. The DWARF now contains variable-to-register mapping and liveness information for all variables. It can be the case that variables are spilled to local memory, and this is all contained in the DWARF information which is ULEB128 encoded (as a DW_OP_regx stack operation in the DW_AT_location attribute).

Here is a typical DWARF entry for a variable located in a local register:

```
<3><359>: Abbrev Number: 20 (DW_TAG_variable)
     DW_AT_decl_file   : 27
     DW_AT_decl_line   : 7
     DW_AT_name        : c
     DW_AT_type        : <1aa>
     DW_AT_location    : 7 byte block: 90 b9 e2 90 b3 d6 4      (DW_OP_regx:
 160631632185)
     DW_AT_address_class: 2
```

This shows variable 'c' has address class 2 (ptxRegStorage) and its location can be found by decoding the ULEB128 value, DW_OP_regx: 160631632185. See cuda-tdep.c in the cuda-gdb source drop for information on decoding this value and how to obtain which physical register holds this variable during a specific device PC range.

Access to physical registers liveness information requires a 0-based physical PC. See the API call:

- CUDBGAPI_st::readPC()

2.3. ABI Support

ABI support is handled through the following thread API calls:

- CUDBGAPI_st::readCallDepth()
- CUDBGAPI_st::readReturnAddress()
- CUDBGAPI_st::readVirtualReturnAddress()

The return address is not accessible on the local stack and the API call must be used to access its value.

For more information, please refer to the ABI documentation titled "Fermi ABI: Application Binary Interface".

2.4. Exception Reporting

Some kernel exceptions are reported as device events and accessible via the API call:

- CUDBGAPI_st::readLaneException()

The reported exceptions are listed in the CUDBGException_t enum type. Each prefix, (Device, Warp, Lane), refers to the precision of the exception. That is, the lowest known execution unit that is responsible for the origin of the exception. All lane errors are precise; the exact instruction and lane that caused the error are known. Warp errors are typically within a few instructions of where the actual error occurred, but the exact lane within the warp is not known. On device errors, we *may* know the *kernel* that caused it. Explanations about each exception type can be found in the documentation of the struct.

Exception reporting is only supported on Fermi (sm_20 or greater).

2.5. Attaching and Detaching

The debug client must take the following steps to attach to a running CUDA application:

1. Attach to the CPU process corresponding to the CUDA application. The CPU part of the application will be frozen at this point.
2. Check to see if the `CUDBG_DEBUGGER_INITIALIZED` variable is accessible from the memory space of the application. If not, it implies that the application has not loaded the CUDA driver, and the attaching to the application is complete.
3. If the `CUDBG_DEBUGGER_INITIALIZED` variable is accessible and non-zero, then the attach procedure is finished.
4. Check the existence and value of the global variable `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD`:
   a). If it doesn't exist, then proceed with Legacy debugger initialization described below.
   b). On Linux, if it's less than zero, then the CUDA driver has not fully initialized yet. The attach process should be retried later, once this variable is greater than or equal to zero.
   c). If it's value is greater than or equal to zero, then proceed with debugger initialization as described below.
5. Legacy debugger initialization: If the `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable is not present or is less than zero:
   a). Make a dynamic (inferior) function call to the function `cudbgApiAttach()`, e.g. by using `ptrace(2)` on Linux. This causes a helper process to be forked off from the application, which assists in attaching to the CUDA process.
   b). Ensure that the initialization of the CUDA debug API is complete, or wait till API initialization is successful (i.e. call the `initialize()` API method until it succeeds).

c). Make the `initializeAttachStub()` API call to initialize the helper process that was forked off from the application earlier.

6. If the `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable is greater than or equal to zero:

   a). Insert a breakpoint in the `CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED` function.

   b). Open the application's file descriptor whose number is given in `CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD` variable and write a single byte to it.

   On Linux, this can be achieved by using the `/proc` filesystem and the fd file `/proc/PID/FD`.

   c). Resume all the threads of the attached program.

   d). Wait until the `CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED` breakpoint is hit, remove the breakpoint and then stop/freeze all of the threads again.

   e). Call the `initialize()` API method. It should now succeed.

7. Read the value of the `CUDBG_RESUME_FOR_ATTACH_DETACH` variable from the memory space of the application:

   - If the value is non-zero, resume the CUDA application so that more data can be collected about the application and sent to the debugger. When the application is resumed, the debug client can expect to receive various CUDA events from the CUDA application. Once all state has been collected, the debug client will receive the event `CUDBG_EVENT_ATTACH_COMPLETE`.
   - If the value is zero, there is no more attach data to collect. Set the `CUDBG_IPC_FLAG_NAME` variable to 1 in the application's process space, which enables further events from the CUDA application.

8. At this point, attaching to the CUDA application is complete and all GPUs belonging to the CUDA application will be suspended.

The debug client must take the following steps to detach from a running CUDA application:

1. Check to see if the `CUDBG_IPC_FLAG_NAME` variable is accessible from the memory space of the application, and that the CUDA debug API is initialized. If either of these conditions is not met, treat the application as CPU-only and detach from the application.
2. Next, make the `clearAttachState` API call to prepare the CUDA debug API for detach.
3. Make a dynamic (inferior) function call to the function `cudbgApiDetach()` in the memory space of the application, e.g. by using ptrace(2) on Linux. This causes CUDA driver to setup state for detach.
4. Read the value of the `CUDBG_RESUME_FOR_ATTACH_DETACH` variable from the memory space of the application. If the value is non-zero, make the `requestCleanupOnDetach` API call.
5. Set the `CUDBG_DEBUGGER_INITIALIZED` variable to 0 in the memory space of the application. This makes sure the debugger is reinitialized from scratch if the debug client re-attaches to the application in the future.

6. If the value of the `CUDBG_RESUME_FOR_ATTACH_DETACH` variable was found to be non-zero in step 4, delete all breakpoints and resume the CUDA application. This allows the CUDA driver to perform cleanups before the debug client detaches from it. Once the cleanup is complete, the debug client will receive the event `CUDBG_EVENT_DETACH_COMPLETE`.
7. Set the `CUDBG_IPC_FLAG_NAME` variable to zero in the memory space of the application. This prevents any more callbacks from the CUDA application to the debugger.
8. The client must then finalize the CUDA debug API.
9. Finally, detach from the CPU part of the CUDA application. At this point all GPUs belonging to the CUDA application will be resumed.

Chapter 3. Modules

Here is a list of all modules:

3.1. General

struct CuDim2

2-dimensional coordinates for threads, blocks, etc.

struct CuDim3

3-dimensional coordinates for threads, blocks, etc.

enum CUDBGCapabilityFlags

Debug engine capability flags. Clients should request the capabilities they want by setting the CUDBG_DEBUGGER_CAPABILITIES global variable and then checking the supported capabilities by calling the getSupportedDebuggerCapabilities() API method and adjusting their behavior accordingly. Capabilities requested but not supported by the debug engine will be ignored and should not be relied upon.

Values

CUDBG_DEBUGGER_CAPABILITY_NONE = 0

No capabilities.

CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING = (1<<0)

Lazy function loading. Static flag: cannot be changed after initialization. Requesting this capability will enable CUDBG_EVENT_FUNCTIONS_LOADED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

CUDBG_DEBUGGER_CAPABILITY_SUSPEND_EVENTS = (1<<1)

Suspend events. Static flag: cannot be changed after initialization. Requesting this capability will enable CUDBG_EVENT_ALL_DEVICES_SUSPENDED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS = (1<<2)

Report exceptions in exited warps. Static flag: cannot be changed after initialization. Requesting this capability will enable reporting of exceptions in exited warps. This capability should not be requested until the API client is prepared to handle such situations.

CUDBG_DEBUGGER_CAPABILITY_NO_CONTEXT_PUSH_POP_EVENTS = (1<<3)

No context push/pop events. Static flag: cannot be changed after initialization.
Requesting this capability will disable the CUDBG_EVENT_CONTEXT_PUSH and CUDBG_EVENT_CONTEXT_POP events. This capability should be requested if the push/pop events are not used by the API client.

CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS = (1<<4)

Enable CUDA logs. Dynamic flag: can be changed after initialization.
Requesting this capability will enable CUDA log capture by the debug engine and cause CUDBG_EVENT_CUDA_LOGS_AVAILABLE and CUDBG_EVENT_CUDA_LOGS_THRESHOLD_REACHED events to be sent. This capability should not be requested until the API client is prepared to handle these events.

CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES = (1<<5)

Collect CPU call stack for kernel launches. Dynamic flag: can be changed after initialization.
Requesting this capability will enable collection of CPU call stack for kernel launches. This capability should not be requested if the API client does not plan on calling the readCPUCallStack() API.

CUDBG_DEBUGGER_CAPABILITY_FLUSH_PRINTF_ON_SUSPEND = (1<<6)

Flush printf on suspend. Static flag: cannot be changed after initialization. Requesting this capability will enable flushing of the CUDA printf output on suspend. This capability should generally be requested.

CUDBG_DEBUGGER_CAPABILITY_BREAK_ON_LAUNCH = (1<<7)

Enable break on launch feature. Static flag: cannot be changed after initialization. Requesting this capability will enable the break on launch feature. It may impact launch performance for light workloads. This capability should be requested if break on launch is used.

enum CUDBGReportDriverApiErrorFlags

API error reporting flags.

Values

CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_NONE = 0x0000
No flags set (default behavior).

CUDBG_REPORT_DRIVER_API_ERROR_FLAGS_SUPPRESS_NOT_READY = (1U<<0)
When set, cudaErrorNotReady/cuErrorNotReady will not be reported.

enum CUDBGReportedDriverApiErrorSource

Driver API error source.

Values

CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_NONE = 0x000
No error/source.

CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_DRIVER = 0x001
The error originates from the CUDA Driver API.

CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE_RUNTIME = 0x002
The error originates from the CUDA Runtime API.

enum CUDBGResult

Result values of all the API routines.

Values

CUDBG_SUCCESS = 0x0000
The API call executed successfully.

CUDBG_ERROR_UNKNOWN = 0x0001
Error type not listed below.

CUDBG_ERROR_BUFFER_TOO_SMALL = 0x0002
Cannot copy all the queried data into the buffer argument.

CUDBG_ERROR_UNKNOWN_FUNCTION = 0x0003
Function cannot be found in the CUDA kernel.

CUDBG_ERROR_INVALID_ARGS = 0x0004
Wrong use of arguments (NULL pointer, illegal value,....).

CUDBG_ERROR_UNINITIALIZED = 0x0005
The API has not yet been properly initialized.

CUDBG_ERROR_INVALID_COORDINATES = 0x0006
Invalid block or thread coordinates were provided.

CUDBG_ERROR_INVALID_MEMORY_SEGMENT = 0x0007
Invalid memory segment requested.

CUDBG_ERROR_INVALID_MEMORY_ACCESS = 0x0008
Requested address (+size) is not within proper segment boundaries.
CUDBG_ERROR_MEMORY_MAPPING_FAILED = 0x0009
Memory is not mapped and cannot be mapped.
CUDBG_ERROR_INTERNAL = 0x000a
A debug engine internal error occurred.
CUDBG_ERROR_INVALID_DEVICE = 0x000b
Specified device cannot be found.
CUDBG_ERROR_INVALID_SM = 0x000c
Specified sm cannot be found.
CUDBG_ERROR_INVALID_WARP = 0x000d
Specified warp cannot be found.
CUDBG_ERROR_INVALID_LANE = 0x000e
Specified lane cannot be found.
CUDBG_ERROR_SUSPENDED_DEVICE = 0x000f
The requested operation is not allowed when the device is suspended.
CUDBG_ERROR_RUNNING_DEVICE = 0x0010
Device is running and not suspended.
CUDBG_ERROR_RESERVED_0 = 0x0011
Reserved error code.
CUDBG_ERROR_INVALID_ADDRESS = 0x0012
Address is out-of-range.
CUDBG_ERROR_INCOMPATIBLE_API = 0x0013
The requested API is not available.
CUDBG_ERROR_INITIALIZATION_FAILURE = 0x0014
The API could not be initialized.
CUDBG_ERROR_INVALID_GRID = 0x0015
The specified grid is not valid.
CUDBG_ERROR_NO_EVENT_AVAILABLE = 0x0016
The event queue is empty and there is no event left to be processed.
CUDBG_ERROR_SOME_DEVICES_WATCHDOGGED = 0x0017
Some devices were excluded because they have a watchdog associated with them.
CUDBG_ERROR_ALL_DEVICES_WATCHDOGGED = 0x0018
All devices were exclude because they have a watchdog associated with them.
CUDBG_ERROR_INVALID_ATTRIBUTE = 0x0019
Specified attribute does not exist or is incorrect.
CUDBG_ERROR_ZERO_CALL_DEPTH = 0x001a
No function calls have been made on the device.
CUDBG_ERROR_INVALID_CALL_LEVEL = 0x001b
Specified call level is invalid.
CUDBG_ERROR_COMMUNICATION_FAILURE = 0x001c
Communication error between the debug engine and the application.

CUDBG_ERROR_INVALID_CONTEXT = 0x001d
Specified context cannot be found.

CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM = 0x001e
Requested address was not originally allocated from device memory (most likely visible in system memory).

CUDBG_ERROR_MEMORY_UNMAPPING_FAILED = 0x001f
Requested address is not mapped and cannot be unmapped.

CUDBG_ERROR_INCOMPATIBLE_DISPLAY_DRIVER = 0x0020
The display driver is incompatible with the API.

CUDBG_ERROR_INVALID_MODULE = 0x0021
The specified module is not valid.

CUDBG_ERROR_LANE_NOT_IN_SYSCALL = 0x0022
The specified lane is not inside a device syscall.

CUDBG_ERROR_RESERVED_1 = 0x0023
Reserved error code.

CUDBG_ERROR_INVALID_ENVVAR_ARGS = 0x0024
Some environment variable's value is invalid.

CUDBG_ERROR_OS_RESOURCES = 0x0025
Error while allocating resources from the OS.

CUDBG_ERROR_FORK_FAILED = 0x0026
Error while forking the debug engine process.

CUDBG_ERROR_NO_DEVICE_AVAILABLE = 0x0027
No CUDA capable device was found.

CUDBG_ERROR_ATTACH_NOT_POSSIBLE = 0x0028
Attaching to the CUDA program is not possible.

CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE = 0x0029
The resumeWarpsUntilPC() API is not possible, use resumeDevice() or singleStepWarp() instead.

CUDBG_ERROR_INVALID_WARP_MASK = 0x002a
Specified warp mask is zero, or contains invalid warps.

CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS = 0x002b
Specified device pointer cannot be resolved to a GPU unambiguously because it is valid on more than one GPU.

CUDBG_ERROR_RECURSIVE_API_CALL = 0x002c
Debug API entry point called from within a debug API callback.

CUDBG_ERROR_MISSING_DATA = 0x002d
The requested data is missing.

CUDBG_ERROR_NOT_SUPPORTED = 0x002e
Attempted operation is not supported.

CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT = 0x002f
The current breakpoint state conflicts with the requested operation.

CUDBG_PRE_INIT ()

Empty global function that gets called before CUDA driver initialization. The API client can set a breakpoint on this function to be notified before the CUDA driver starts its initialization.

CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED (void)

Report that the attach procedure has finished. The API client can set a breakpoint on this function to be notified that the attach procedure has finished.

CUDBG_REPORT_DRIVER_API_ERROR (void)

Report a driver API error. The API client can set a breakpoint on this function to be notified about driver API errors.

CUDBG_REPORT_DRIVER_INTERNAL_ERROR (void)

Report a driver internal error. The API client can set a breakpoint on this function to be notified about driver internal errors.

cudbgApiAttach (void)

Attach to a CUDA application. Remotely called by the API client during the attach procedure to attach to the CUDA application.

cudbgApiDetach (void)

Detach from a CUDA application. Remotely called by the API client during the detach procedure to detach from the CUDA application.

cudbgApiInit (uint32_t arg)

Initialize the CUDA Debugger API. Remotely called by the API client during the attach procedure to initialize the debugger API.

CUDBGResult cudbgGetAPIVersion (uint32_t *major, uint32_t *minor, uint32_t *rev)

Get the API version supported by the CUDA driver.

Parameters

major

- the major version number

minor
- the minor version number

rev
- the revision version number

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_SUCCESS

Description

See also:

cudbgGetAPI

const _CUDBG_INLINE char *cudbgGetErrorString (CUDBGResult error)

Returns the string representation of a result value. It is preferred to use this function instead of indexing CUDBGResultNames directly as future versions of the old API methods might start returning new result values.

Parameters

error
The result value to get the string representation of.

Returns

The string representation of the result value or "*UNDEFINED*" if the result value is not recognized.

#define CUDBG_API_VERSION_MAJOR 13

Major release version number. This matches the major version of the CUDA driver exposing this API.

#define CUDBG_API_VERSION_MINOR 2

Minor release version number. This matches the minor version of the CUDA driver exposing this API.

#define CUDBG_API_VERSION_REVISION 172

API revision number. This number is incremented every time changes are made to the API.

#define CUDBG_APICLIENT_PID cudbgApiClientPid

Name of the global variable containing the API client PID. This variable is set by the API client to identify itself.

#define CUDBG_APICLIENT_REVISION cudbgApiClientRevision

Name of the global variable containing the API client revision. This variable is set by the API client to identify the API revision it wants to use.

#define CUDBG_ATTACH_HANDLER_AVAILABLE cudbgAttachHandlerAvailable

Name of the global variable containing the attach handler available flag. This variable is set by the debug engine to indicate that the attach handler is available.

#define CUDBG_DEBUGGER_CAPABILITIES cudbgDebuggerCapabilities

Name of the global variable containing the debug engine capabilities bitmask. This variable is set by the API client to indicate the requested capabilities. See the CUDBGCapabilityFlags enum for the list of available capabilities.

#define CUDBG_DEBUGGER_INITIALIZED cudbgDebuggerInitialized

Name of the global variable containing the debug engine initialized flag. This variable is set by the debug engine to indicate that it has been initialized.

#define CUDBG_ENABLE_LAUNCH_BLOCKING cudbgEnableLaunchBlocking

Name of the global variable containing the enable launch blocking flag. This variable is set by the API client to enable blocking launches of CUDA kernels.

#define CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD cudbgInitiateDebuggerAttachProcedureFd

Name of the global variable containing the debug engine attach procedure initiation FD.
If this FD is not -1, write a single byte (any value) to it to initiate the CUDA debug engine attach procedure. After the request was received and the attach procedure has finished, the CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED function is called.

#define CUDBG_IPC_FLAG_NAME cudbgIpcFlag

Name of the global variable containing the IPC flag. This variable is set by the API client to indicate that it is ready to attach to a running CUDA application.

#define CUDBG_MAX_DEVICES 64

Maximum number of supported devices.

#define CUDBG_MAX_LANES 32

Maximum number of lanes per warp.

#define CUDBG_MAX_LOG_LEN 256

Maximum length of a single CUDA log message.

#define CUDBG_MAX_SMS 256

Maximum number of SMs per device.

#define CUDBG_MAX_WARP_BARRIERS 16

Maximum number of convergence barriers per warp.

#define CUDBG_MAX_WARPS 64

Maximum number of warps per SM.

#define CUDBG_PRE_INIT cudbgPreInit

Name of the pre-init global function. The API client can set a breakpoint on this function to be notified before the CUDA driver starts its initialization.

#define CUDBG_REPORT_ATTACH_PROCEDURE_FINISHED cudbgReportAttachProcedureFinished

Name of the global function that's called to report the completion of the attach procedure. The API client can set a breakpoint on this function to be notified about the completion of the attach procedure. This function is called after the attach is done after signalling CUDBG_INITIATE_DEBUGGER_ATTACH_PROCEDURE_FD.

#define CUDBG_REPORT_DRIVER_API_ERROR cudbgReportDriverApiError

Name of the global function that's called to report a driver API error. The API client can set a breakpoint on this function to be notified about driver API errors.

#define CUDBG_REPORT_DRIVER_API_ERROR_FLAGS cudbgReportDriverApiErrorFlags

Name of the global variable containing the driver API error flag. This variable is set by the API client to indicate which driver API errors should be reported. See the CUDBGReportDriverApiErrorFlags enum for the list of available flags.

#define CUDBG_REPORT_DRIVER_INTERNAL_ERROR cudbgReportDriverInternalError

Name of the global function that's called to report an internal driver error. The API client can set a breakpoint on this function to be notified about internal driver errors.

#define CUDBG_REPORTED_DRIVER_API_ERROR_CODE cudbgReportedDriverApiErrorCode

Name of the global variable containing the code of the driver API error being reported. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_AD cudbgReportedDriverApiErrorFuncNameAddr

Name of the global variable containing the address of the name of the function in which the driver API error occurred. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_FUNC_NAME_SIZ cudbgReportedDriverApiErrorFuncNameSize

Name of the global variable containing the size of the name of the function in which the driver API error occurred. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_NAME_ADDR cudbgReportedDriverApiErrorNameAddr

Name of the global variable containing the address where the driver API error name is stored. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_NAME_SIZE cudbgReportedDriverApiErrorNameSize

Name of the global variable containing the size of the driver API error name being reported. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_SOURCE cudbgReportedDriverApiErrorSource

Name of the global variable containing the driver API error source. This variable is set by the debug engine when a driver API error is reported. See the CUDBGReportedDriverApiErrorSource enum for the list of available sources.

#define CUDBG_REPORTED_DRIVER_API_ERROR_STRING_ADDR cudbgReportedDriverApiErrorStringAddr

Name of the global variable containing the address where the driver API error string is stored. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_API_ERROR_STRING_SIZE cudbgReportedDriverApiErrorStringSize

Name of the global variable containing the size of the driver API error string being reported. This variable is set by the debug engine when a driver API error is reported.

#define CUDBG_REPORTED_DRIVER_INTERNAL_ERROR_CODE cudbgReportedDriverInternalErrorCode

Name of the global variable containing the driver internal error code. This variable is set by the debug engine when an internal driver error is reported.

#define CUDBG_RESUME_FOR_ATTACH_DETACH cudbgResumeForAttachDetach

Name of the global variable containing the resume for attach detach flag. This variable is set by the debug engine to indicate that the API client should resume the CUDA application (including the CPU) to complete the attach or detach procedure.

#define CUDBG_RPC_ENABLED cudbgRpcEnabled

Name of the global variable containing the RPC enabled flag. This variable is only used by debuggers that use the RPCD interface (e.g. CUDA-GDB).

#define CUDBG_SESSION_ID cudbgSessionId

Name of the global variable containing the session ID. This variable is set by the API client to identify the session.

#define CUDBG_USE_EXTERNAL_DEBUGGER cudbgUseExternalDebugger

Name of the global variable containing the external debug engine in use flag. Can be read to detect whether the external debug engine implementation (libcudadebugger.so) is used or not.

Note:

Since CUDA 13.1, the external debug engine implementation is always used.

3.2. Initialization

CUDBGResult (*CUDBGAPI_st::clearAttachState) ()

Clear attach-specific state prior to detach.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This call prepares the API for detaching. See the "Attaching and Detaching" section for more information.

Since CUDA 5.0.

CUDBGResult (*CUDBGAPI_st::finalize) ()

Finalize the API, shutting down the debugging session.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

initialize

CUDBGResult (*CUDBGAPI_st::getSupportedDebuggerCapabilities) (CUDBGCapabilityFlags *capabilities)

Returns debug agent capabilities that are supported by this version of the API.

Parameters

capabilities
- returned debug engine capabilities

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This API method can be called without initializing the API.

Since CUDA 12.5.

CUDBGResult (*CUDBGAPI_st::initialize) ()

Initialize the API.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_DEVICE_AVAILABLE

setNotifyNewEventCallback() and getSupportedDebuggerCapabilities() can be called before initialize(). If no CUDA devices are detected on the system, CUDBG_ERROR_NO_DEVICE_AVAILABLE is returned.

Since CUDA 3.0.

See also:

finalize

CUDBGResult (*CUDBGAPI_st::initializeAttachStub) ()

Initialize the attach stub.

Returns

CUDBG_SUCCESS

This is no longer necessary starting with driver version r590.

Since CUDA 5.0.

CUDBGResult (*CUDBGAPI_st::requestCleanupOnDetach) (uint32_t appResumeFlag)

Request for cleanup of driver state when detaching.

Parameters

appResumeFlag

- value of CUDBG_RESUME_FOR_ATTACH_DETACH as read from the application's process space.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Needs to be conditionally called by the client depending on the state of the debugged application. See the "Attaching and Detaching" section for more information.

Since CUDA 6.0.

CUDBGResult (*CUDBGAPI_st::requestCleanupOnDetach55) ()

Request for cleanup of driver state when detaching.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Needs to be conditionally called by the client depending on the state of the debugged application. See the "Attaching and Detaching" section for more information.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 6.0: Use requestCleanupOnDetach instead.

See also:

requestCleanupOnDetach

3.3. Device Execution Control

enum CUDBGSingleStepFlags

Single step flags.

Values

CUDBG_SINGLE_STEP_FLAGS_NONE = 0

Default behavior.

CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS = (1U<<0)

Disable optimized warp barrier stepping. Do not step over warp-wide barriers using a breakpoint and resume, instead perform a single step and return. Passing this flag in means that the API client plans to repeat the singleStepWarp() call until the warp barrier is stepped over. This gives a more precise exception information if an exception is encountered by the diverged threads while stepping.

This flag is only valid for the singleStepWarp() API method, resumeWarpsUntilPC() always steps over warp-wide barriers.

CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING = (1U<<1)

Don't block on the stepping operations, instead return early and send an event once stepping is done.

enum CUDBGSingleStepType

Single step operation type (API method that was used to start the single step operation).

Values

CUDBG_SINGLE_STEP_TYPE_INVALID = 0

Invalid single step operation type.

CUDBG_SINGLE_STEP_TYPE_SINGLE_STEP_WARP = 1

The singleStepWarp() API method was used to start the single step operation.

CUDBG_SINGLE_STEP_TYPE_RESUME_WARPS_UNTIL_PC = 2

The resumeWarpsUntilPC() API method was used to start the single step operation.

CUDBGResult (*CUDBGAPI_st::executeInternalCommand) (const char *command, char *resultBuffer, uint32_t sizeInBytes)

Execute an internal command (not available in public driver builds).

Parameters

command

- the command name and arguments

resultBuffer

- the destination buffer

sizeInBytes

- buffer size in bytes

Returns

CUDBG_ERROR_NOT_SUPPORTED

Always returns CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 12.6.

CUDBGResult (*CUDBGAPI_st::resumeAllDevices) ()

Resume all running CUDA devices.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

suspendAllDevices

CUDBGResult (*CUDBGAPI_st::resumeDevice) (uint32_t dev)

Resume a suspended CUDA device.

Parameters

dev
- device index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Using this method is discouraged, use resumeAllDevices() instead to avoid race conditions. This method has no effect if the device is already running.

Since CUDA 3.0.

See also:

resumeAllDevices

CUDBGResult (*CUDBGAPI_st::resumeWarpsUntilPC)(uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t pc, uint32_t flags)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Parameters

dev
- device index

sm
- the SM index

warpMask
- the bitmask of warps to resume (1 = resume, 0 = do not resume)

pc

flags
- flags of type CUDBGSingleStepFlags to change the stepping behavior

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Compared to resumeDevice(), this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using singleStepWarp() or resumeDevice().

Since CUDA 13.2.

See also:

resumeAllDevices

singleStepWarp

CUDBGResult (*CUDBGAPI_st::resumeWarpsUntilPC60) (uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t virtPC)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Parameters

dev
- device index

sm
- SM index

warpMask
- the bitmask of warps to resume (1 = resume, 0 = do not resume)

virtPC
- the virtual PC where the temporary breakpoint will be inserted

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Compared to resumeDevice(), this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using singleStepWarp() or resumeDevice().

Since CUDA 6.0.

Note:

DEPRECATED in CUDA 13.2: Use resumeWarpsUntilPC instead.

See also:

resumeWarpsUntilPC

CUDBGResult (*CUDBGAPI_st::setKernelLaunchNotificationMode) (CUDBGKernelLaunchNotifyMode mode)

Set the launch notification policy.

Parameters

mode
- mode to deliver kernel launch notifications in

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If mode is CUDBG_KNL_LAUNCH_NOTIFY_EVENT, enable synchronous launch notification reporting (via events). This can noticeably slow down the execution of the application. If mode is CUDBG_KNL_LAUNCH_NOTIFY_DEFER, the launch notifications are not reported at all.

Since CUDA 5.5.

CUDBGResult (*CUDBGAPI_st::singleStepWarp) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t laneHint, uint32_t nsteps, uint32_t flags, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

laneHint
- focused lane (~0 to let the API decide)

nsteps
- number of single steps

flags
- flags of type CUDBGSingleStepFlags to change the stepping behavior

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

By default, if the warp is on a convergence barrier, resumeWarpsUntilPC is called internally to quickly advance the warp past that barrier. If the CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag is passed in, this optimization is not performed (which would likely lead to diverged threads becoming focused and starting to advance towards the convergence barrier). If a warp is on a block-wide barrier (or wider), other warps required to advance past the barrier are automatically resumed. The output parameter warpMask will have the warps resumed in the current SM. Warps can also be resumed in other SMs, but are not reported via the API. This method is synchronous and will not return until the step is complete.

Since CUDA 12.4.

See also:

resumeAllDevices

resumeWarpsUntilPC

suspendAllDevices

CUDBGResult (*CUDBGAPI_st::singleStepWarp40)(uint32_t dev, uint32_t sm, uint32_t wp)

Single step an individual warp on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp41 without the output warpMask parameter.

Since CUDA 3.0.

See also:

singleStepWarp

CUDBGResult (*CUDBGAPI_st::singleStepWarp41)(uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *warpMask)

Single step an individual warp on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT,

CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,
CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp65 with nsteps set to 1.

Since CUDA 4.1.

See also:

singleStepWarp

CUDBGResult (*CUDBGAPI_st::singleStepWarp65) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t nsteps, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

nsteps
- number of single steps

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION,
CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED,
CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP,
CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS,
CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT,
CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE,
CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp with no lane hint and the
CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag set.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 12.4: Use singleStepWarp instead.

See also:

singleStepWarp

CUDBGResult (*CUDBGAPI_st::suspendAllDevices) (uint32_t nonBlocking)

Suspend all running CUDA devices.

Parameters

nonBlocking
- whether or not asynchronous operation is desired

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_SUSPENDED_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the nonBlocking flag is non-zero, the function returns immediately and sends CUDBG_EVENT_ALL_DEVICES_SUSPENDED when the operation finishes in the background. Otherwise, if the function returns with CUDBG_SUCCESS, that guarantees that all devices have been suspended.

Since CUDA 13.2.

See also:

resumeAllDevices

CUDBGResult (*CUDBGAPI_st::suspendDevice) (uint32_t dev)

Suspends a running CUDA device.

Parameters

dev
- device index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_SUSPENDED_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Using this method is discouraged, use suspendAllDevices() instead to avoid race conditions. The device has to be suspended in order to execute most operations on it. CUDBG_ERROR_SUSPENDED_DEVICE is returned if the device is already suspended.

Since CUDA 3.0.

See also:

suspendAllDevices

3.4. Breakpoints

enum CUDBGAdjAddrAction

Describes which adjusted code address is to be returned.

Values

CUDBG_ADJ_PREVIOUS_ADDRESS = 0x000
Get the adjusted previous code address.

CUDBG_ADJ_CURRENT_ADDRESS = 0x001
Get the adjusted next code address.

CUDBG_ADJ_NEXT_ADDRESS = 0x002
Get the adjusted current code address.

typedef uint64_t CUDBGBreakpointHandle

GPU debugger breakpoint handle. Valid breakpoint handles will grow from 1. Handle values of removed breakpoints can be reused in some cases. It is not guaranteed that new (non-reused) breakpoint handles will always be increasing by 1, there can be gaps. A valid handle always corresponds to a single inserted breakpoint, and each breakpoint can only have one handle. Handles with the highest bit set are reserved for special purposes and can correspond to multiple breakpoints.

#define CUDBG_BREAKPOINT_HANDLE_BREAK_ON_LAUNCH (0xFFFFFFFFFFFFFFFCULL)

Break On Launch breakpoint handle - this breakpoint can be enabled/disabled with enableBreakpoint/ disableBreakpoint. It is hit once every time a kernel is launched. It can be hit by one or more warps, but only once. Only supports CUDA Kernel launches (both host-side and device-side launches).

#define CUDBG_BREAKPOINT_HANDLE_INTERNAL (0xFFFFFFFFFFFFFFFDULL)

Special breakpoint handle - internal breakpoint (e.g. used internally for stepping).

#define CUDBG_BREAKPOINT_HANDLE_INVALID (0ULL)

Invalid breakpoint handle.

#define CUDBG_BREAKPOINT_HANDLE_LEGACY (0xFFFFFFFFFFFFFFFEULL)

Special breakpoint handle - breakpoint inserted with setBreakpoint (handle-less).

#define CUDBG_BREAKPOINT_HANDLE_TRAP (0xFFFFFFFFFFFFFFFFULL)

Special breakpoint handle - hard-coded trap.

CUDBGResult (*CUDBGAPI_st::disableBreakpoint) (CUDBGBreakpointHandle handle)

Disable a breakpoint specified by its handle.

Parameters

handle

- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*CUDBGAPI_st::enableBreakpoint)(CUDBGBreakpointHandle handle)

Enable a breakpoint specified by its handle.

Parameters

handle

- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

disableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*CUDBGAPI_st::getAdjustedCodeAddress) (uint32_t dev, uint64_t address, uint64_t *adjustedAddress, CUDBGAdjAddrAction adjAction)

Get the adjusted code address for a given code address for a given device.

Parameters

dev
- device index

address

adjustedAddress
- adjusted address

adjAction
- whether the adjusted next, previous or current address is needed

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The client must call this function before inserting a breakpoint, or when the previous or next code address is needed for breakpoint inserting purposes.

Since CUDA 5.5.

See also:

setBreakpoint

CUDBGResult (*CUDBGAPI_st::getWarpHitBreakpoint) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGBreakpointHandle *handle)

Get the handle of the breakpoint that the given warp hit.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

handle
- the returned breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

An error is returned if the warp did not hit a breakpoint. Use readBrokenWarps() to check if the warp is broken before calling this method. Some breakpoint handles are special, see the documentation of CUDBGBreakpointHandle for more details.

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

insertBreakpoint

isBreakpointEnabled

readBrokenWarps

removeBreakpoint

CUDBGResult (*CUDBGAPI_st::insertBreakpoint)(uint32_t dev, uint64_t addr, CUDBGBreakpointHandle *handle)

Set a breakpoint at the given instruction address for the given device.

Parameters

dev
- the device index

addr
- instruction address

handle
- the returned breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Before setting a breakpoint, getAdjustedCodeAddress() should be called to get the adjusted breakpoint address. The returned handle can be used to enable/disable/remove the breakpoint.

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*CUDBGAPI_st::isBreakpointEnabled) (CUDBGBreakpointHandle handle, uint32_t *enabled)

Check if a breakpoint specified by its handle is enabled.

Parameters

handle
- the breakpoint handle

enabled
- whether the breakpoint is enabled

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

removeBreakpoint

CUDBGResult (*CUDBGAPI_st::removeBreakpoint) (CUDBGBreakpointHandle handle)

Remove a breakpoint specified by its handle.

Parameters

handle
- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

CUDBGResult (*CUDBGAPI_st::setBreakpoint) (uint32_t dev, uint64_t addr)

Set a breakpoint at the given instruction address for the given device.

Parameters

dev
- device index

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Before setting a breakpoint, getAdjustedCodeAddress() should be called to get the adjusted breakpoint address.

Since CUDA 3.2.

See also:

unsetBreakpoint

CUDBGResult (*CUDBGAPI_st::setBreakpoint31)(uint64_t addr)

Set a breakpoint at the given instruction address.

Parameters

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use setBreakpoint instead.

See also:

setBreakpoint

CUDBGResult (*CUDBGAPI_st::unsetBreakpoint) (uint32_t dev, uint64_t addr)

Unset a breakpoint at the given instruction address for the given device.

Parameters

dev
- device index

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.2.

See also:

setBreakpoint

CUDBGResult (*CUDBGAPI_st::unsetBreakpoint31) (uint64_t addr)

Unset a breakpoint at the given instruction address.

Parameters

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like unsetBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use unsetBreakpoint instead.

See also:

unsetBreakpoint

3.5. Device State Inspection

struct CUDBGCbuWarpState

Warp state in the CBU (Convergence Barrier Unit).

struct CUDBGCudaLogMessage

CUDA Log Message.

struct CUDBGDeviceInfo

Device-level information. This is the first element in the deviceInfoBuffer, and is always present. getDeviceInfo() takes a deviceId as input, so no need to explicitly pass it back here. Only "valid & updated" SMs/Warps/Lanes are included in the buffer, which allows us to determine indexes without having to encode an explicit ID field in the following buffer datastructures.

struct CUDBGDeviceInfoSizes

Sizes of the various structs returned by the batched device update APIs. No explicit version field - implied by debugAPI major.minor.revision.

struct CUDBGLaneState

Lane state (state of a single thread).

struct CUDBGLoadedFunctionInfo

Information about a lazily loaded function.

struct CUDBGMemoryInfo

Memory information.

struct CUDBGSMInfo

SM-level information.

struct CUDBGWarpInfo

Warp-level information.

struct CUDBGWarpResources

Warp resources. These resources can change at runtime between suspends.

struct CUDBGWarpState

Warp state information.

struct CUDBGWarpState120

Warp state for API version 12.0.

struct CUDBGWarpState127

Warp state for API version 12.7.

struct CUDBGWarpState60

Warp state for API version 6.0.

enum CUDBGBarrierScope

CUDA barrier scope.

Values

CUDBG_BARRIER_SCOPE_INVALID = 0xFFFFFFFFU
Invalid barrier scope.
CUDBG_BARRIER_SCOPE_NONE = 0
No barrier.
CUDBG_BARRIER_SCOPE_WARP = 1
Warp-wide barrier.
CUDBG_BARRIER_SCOPE_WARP_GROUP = 2
Warp group-wide barrier.
CUDBG_BARRIER_SCOPE_BLOCK = 3
Block-wide barrier.
CUDBG_BARRIER_SCOPE_CLUSTER = 4
Cluster-wide barrier.
CUDBG_BARRIER_SCOPE_KERNEL = 5
Kernel-wide barrier.

enum CUDBGCbuThreadState

Thread state in the CBU (Convergence Barrier Unit).

Values

CUDBG_CBU_THREAD_STATE_INVALID = 0xFFFFFFFFU
CUDBG_CBU_THREAD_STATE_EXITED = 0
CUDBG_CBU_THREAD_STATE_READY
CUDBG_CBU_THREAD_STATE_YIELDED
CUDBG_CBU_THREAD_STATE_SLEEP
CUDBG_CBU_THREAD_STATE_SLEEPYIELD
CUDBG_CBU_THREAD_STATE_READYATNEXT
CUDBG_CBU_THREAD_STATE_BLOCKEDPLUS
CUDBG_CBU_THREAD_STATE_BLOCKEDALL
CUDBG_CBU_THREAD_STATE_BLOCKEDCOLLECTIVE
CUDBG_CBU_THREAD_STATE_BLOCKEDB0
CUDBG_CBU_THREAD_STATE_BLOCKEDB1
CUDBG_CBU_THREAD_STATE_BLOCKEDB2
CUDBG_CBU_THREAD_STATE_BLOCKEDB3
CUDBG_CBU_THREAD_STATE_BLOCKEDB4
CUDBG_CBU_THREAD_STATE_BLOCKEDB5
CUDBG_CBU_THREAD_STATE_BLOCKEDB6
CUDBG_CBU_THREAD_STATE_BLOCKEDB7
CUDBG_CBU_THREAD_STATE_BLOCKEDB8
CUDBG_CBU_THREAD_STATE_BLOCKEDB9
CUDBG_CBU_THREAD_STATE_BLOCKEDB10
CUDBG_CBU_THREAD_STATE_BLOCKEDB11

CUDBG_CBU_THREAD_STATE_BLOCKEDB12
CUDBG_CBU_THREAD_STATE_BLOCKEDB13
CUDBG_CBU_THREAD_STATE_BLOCKEDB14
CUDBG_CBU_THREAD_STATE_BLOCKEDB15

enum CUDBGCoredumpGenerationFlags

Coredump generation flags.

Values

CUDBG_COREDUMP_DEFAULT_FLAGS = 0
Default flags.
CUDBG_COREDUMP_SKIP_NONRELOCATED_ELF_IMAGES = (1<<0)
Skip dumping non-relocated ELF images.
CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY = (1<<1)
Skip dumping global memory.
CUDBG_COREDUMP_SKIP_SHARED_MEMORY = (1<<2)
Skip dumping shared memory.
CUDBG_COREDUMP_SKIP_LOCAL_MEMORY = (1<<3)
Skip dumping local memory.
CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY = (1<<5)
Skip dumping constant bank memory.
CUDBG_COREDUMP_GZIP_COMPRESS = (1<<6)
Compress the coredump with gzip.
CUDBG_COREDUMP_LIGHTWEIGHT_FLAGS =
CUDBG_COREDUMP_SKIP_NONRELOCATED_ELF_IMAGES
|CUDBG_COREDUMP_SKIP_GLOBAL_MEMORY |
CUDBG_COREDUMP_SKIP_SHARED_MEMORY |
CUDBG_COREDUMP_SKIP_LOCAL_MEMORY |
CUDBG_COREDUMP_SKIP_CONSTBANK_MEMORY
Lightweight flags.

enum CUDBGCudaLogLevel

CUDA Log severity level.

Values

CUDBG_CUDA_LOG_LEVEL_INVALID = 0xFFFFFFFFU
Invalid log level.
CUDBG_CUDA_LOG_LEVEL_ERROR = 0
Error log level, matches CU_LOG_LEVEL_ERROR.
CUDBG_CUDA_LOG_LEVEL_WARNING = 1
Warning log level, matches CU_LOG_LEVEL_WARNING.

enum CUDBGDeviceInfoAttribute_t

Device-level attributes.

Values

CUDBG_DEVICE_ATTRIBUTE_SM_UPDATE_MASK = 0
Mask of updated SMs reported by this response. Optional: Yes, assume all 1's if absent. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

CUDBG_DEVICE_ATTRIBUTE_SM_ACTIVE_MASK = 1
Mask of SMs with any valid warp. Optional: No, always returned by the API. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

CUDBG_DEVICE_ATTRIBUTE_SM_EXCEPTION_MASK = 2
Mask of SMs with any warps with exceptions. Optional: Yes, assume all 0's if absent. Size: Number of SMs-sized bitmask, rounded up to be divisible by 8.

CUDBG_DEVICE_ATTRIBUTE_COUNT
Device attributes count.

enum CUDBGDeviceInfoQueryType_t

Device info query type.

Values

CUDBG_RESPONSE_TYPE_FULL
Request state information for all valid SMs/Warps/Lanes.

CUDBG_RESPONSE_TYPE_UPDATE
Request state information for all changed SMs/Warps/Lanes since the last call. It's safe to always use this type, the API will respond with the full information when necessary.

CUDBG_RESPONSE_TYPE_UNKNOWN = 0xFFFFFFFFU
Unknown device info query type.

enum CUDBGLaneInfoAttribute_t

Lane-level (thread-level) attributes.

Values

CUDBG_LANE_ATTRIBUTE_COUNT
Lane (thread) attributes count.

enum CUDBGSMInfoAttribute_t

SM-level attributes.

Values

CUDBG_SM_ATTRIBUTE_WARP_UPDATE_MASK = 0
Mask of updated warps reported by this response. Optional: Yes, assume all 1's if absent. Size: uint64_t.

CUDBG_SM_ATTRIBUTE_COUNT
SM attributes count.

enum CUDBGWarpInfoAttribute_t

Warp-level attributes.

Values

CUDBG_WARP_ATTRIBUTE_LANE_UPDATE_MASK = 0
Mask of updated lanes reported by this response. Optional: Yes, assume all 1's if absent. Size: uint32_t.

CUDBG_WARP_ATTRIBUTE_LANE_ATTRIBUTES = 1
Signals whether the attribute flags field is present on the lane level for this warp. Optional: Yes, assume no lane attributes for this warp if absent. Size: 0 (doesn't have an associated warp-level field).

CUDBG_WARP_ATTRIBUTE_EXCEPTION = 2
CUDBGException_t for this warp. Optional: Yes, assume CUDBG_EXCEPTION_NONE if absent. Size: uint32_t.

CUDBG_WARP_ATTRIBUTE_ERRORPC = 3
Error PC for this warp. Optional: Yes, assume no error PC is available if absent. Size: uint64_t.

CUDBG_WARP_ATTRIBUTE_CLUSTERIDX = 4
Cluster index for this warp. Optional: Yes if warp is not in a cluster. Size: CuDim3.

CUDBG_WARP_ATTRIBUTE_CLUSTERDIM = 5
Cluster dimensions for this warp. Optional: Yes if warp is not in a cluster. Size: CuDim3.

CUDBG_WARP_ATTRIBUTE_CLUSTER_EXCEPTION_TARGET_BLOCK_IDX = 6
For cluster exceptions, this represents the target block index handling cluster requests. Optional: Yes, assume no block index is available if absent. Size: CuDim3.

CUDBG_WARP_ATTRIBUTE_IN_SYSCALL_LANES = 7
Lane mask showing threads that are in a syscall. Optional: Yes, use readSyscallCallDepth() if this attribute is not present. Size: uint32_t.

CUDBG_WARP_ATTRIBUTE_HIT_BREAKPOINT_HANDLE = 8
Breakpoint handle of a broken warp (of type CUDBGBreakpointHandle) Optional: Yes, only present for broken warps Size: CUDBGBreakpointHandle.

CUDBG_WARP_ATTRIBUTE_COUNT
Warp attributes count.

enum ptxStorageKind

Memory segments for DWARF.

Note:

DEPRECATED: This enum is no longer used since the API methods that use it have been deprecated.

Values

ptxUNSPECIFIEDStorage
ptxCodeStorage
ptxRegStorage
ptxSregStorage
ptxConstStorage
ptxGlobalStorage
ptxLocalStorage
ptxParamStorage
ptxSharedStorage
ptxSurfStorage
ptxTexStorage
ptxTexSamplerStorage
ptxGenericStorage
ptxIParamStorage
ptxOParamStorage
ptxFrameStorage
ptxURegStorage
ptxMAXStorage

CUDBGResult (*CUDBGAPI_st::consumeCudaLogs) (CUDBGCudaLogMessage *logMessages, uint32_t numMessages, uint32_t *numConsumed)

Get CUDA error log entries.

Parameters

logMessages
- client-allocated array to store log entries

numMessages
- capacity of the logMessages array, in number of elements

numConsumed
- returned number of entries written to logMessages

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_RECURSIVE_API_CALL

This consumes the log entries, so they will not be available in subsequent calls. This functionality is only available if the CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS capability is enabled.

Since CUDA 12.9.

CUDBGResult (*CUDBGAPI_st::generateCoredump) (const char *filename, CUDBGCoredumpGenerationFlags flags)

Generate a coredump for the current GPU state.

Parameters

filename
- target coredump file name

flags
- coredump generation flags/options

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.3.

CUDBGResult (*CUDBGAPI_st::getCbuWarpState) (uint32_t dev, uint32_t sm, uint64_t warpMask, CUDBGCbuWarpState *warpStates, uint32_t numWarpStates)

Get the CBU state of a given warp.

Parameters

dev
- device index

sm
- SM index

warpMask
- bitmask of the warps which states should be returned in warpStates

warpStates
- pointer to the array of CUDBGCbuWarpState structures

numWarpStates
- number of elements in warpStates array

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.9.

CUDBGResult (*CUDBGAPI_st::getClusterExceptionTargetBlock)(uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx, bool *blockIdxValid)

Get the target block index and validity status for cluster exceptions.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- pointer to a `CuDim3` structure that will be populated with the target block index

blockIdxValid
- pointer to a boolean variable that will be set to `true` if the target block index is valid, and `false` otherwise. Value will be set to false if the warp is not stopped on a cluster exception

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Since CUDA 12.7.

CUDBGResult (*CUDBGAPI_st::getConstBankAddress) (uint32_t dev, uint64_t gridId64, uint32_t bank, uint64_t *address, uint32_t *size)

Get constant bank GPU VA and size.

Parameters

dev
- device index

gridId64
- grid ID of the grid containing the constant bank

bank
- constant bank number

address
- GPU VA of the bank memory

size
- bank size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_MISSING_DATA

Since CUDA 12.4.

CUDBGResult (*CUDBGAPI_st::getConstBankAddress123) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t bank, uint32_t offset, uint64_t *address)

Convert constant bank number and offset into GPU VA.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

bank
- constant bank number

offset
- offset within the bank

address
- GPU VA

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_MISSING_DATA

It's more convenient to get the constbank address and then calculate the VA for any const address using that.

Since CUDA 12.3.

Note:

DEPRECATED in CUDA 12.4: Use getConstBankAddress instead.

See also:

getConstBankAddress

CUDBGResult (*CUDBGAPI_st::getCudaExceptionString) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get the error string for CUDA Exceptions.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

buf
- buffer for the error string

bufSz
- buffer size

msgSz
- error message size with null character, can be null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.0.

CUDBGResult (*CUDBGAPI_st::getDeviceInfo) (uint32_t dev, CUDBGDeviceInfoQueryType_t type, void *buffer, uint32_t length, uint32_t *dataLength)

Read full device info for the device.

Parameters

dev
- device index

type
- query type (full or delta)

buffer
- output buffer

length
- output buffer length

dataLength
- number of bytes written to the buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Information returned by this method is cheap to calculate, so it can be used after every suspend to quickly get the updated device state. For convenience, the caller can always request partial

updates, the API will return a full response when returning a partial one is not possible. If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.4.

See also:

getDeviceInfoSizes

CUDBGResult (*CUDBGAPI_st::getDeviceInfoSizes) (uint32_t dev, CUDBGDeviceInfoSizes *sizes)

Return sizes for device info structs and defined attributes.

Parameters

dev
- device index

sizes
- device info sizes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.4.

See also:

getDeviceInfo

CUDBGResult (*CUDBGAPI_st::getDevicePCIBusInfo) (uint32_t dev, uint32_t *pciBusId, uint32_t *pciDevId)

Get PCI bus and device ids associated with device index.

Parameters

dev
- device index

pciBusId
- pointer where corresponding PCI BUS ID would be stored

pciDevId
- pointer where corresponding PCI DEVICE ID would be stored

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 5.5.

CUDBGResult (*CUDBGAPI_st::getHardwareBarrierInfo) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CUDBGBarrierScope *scope, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get hardware barrier information.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

scope
- barrier scope

buf
- buffer for the barrier information

bufSz
- buffer size

msgSz
- error message size with null character, can be null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.1.

CUDBGResult (*CUDBGAPI_st::getLoadedFunctionInfo) (uint32_t dev, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t startIndex, uint32_t numEntries)

Get the section number and address of loaded functions for a given module.

Parameters

dev
- device index

handle
- ELF/cubin image handle

info
- information about loaded functions

startIndex
- start index of the entries to get

numEntries
- number of function load entries to read

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING capability is enabled, CUDA loads functions lazily after the module has been reported. This method could be used to get the lazily loaded functions.

Since CUDA 12.3.

CUDBGResult (*CUDBGAPI_st::getLoadedFunctionInfo118) (uint32_t dev, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t numEntries)

Get the section number and address of loaded functions for a given module.

Parameters

dev
- device index

handle
- ELF/cubin image handle

info
- information about loaded functions

numEntries
- number of function load entries to read

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getLoadedFunctionInfo but doesn't allow querying a subset of all lazily loaded functions.

Since CUDA 11.8.

Note:

DEPRECATED in CUDA 12.3: Use getLoadedFunctionInfo instead.

See also:

getLoadedFunctionInfo

CUDBGResult (*CUDBGAPI_st::getManagedMemoryRegionInfo) (uint64_t startAddress, CUDBGMemoryInfo *memoryInfo, uint32_t memoryInfo_size, uint32_t *numEntries)

Get a sorted list of managed memory regions.

Parameters

startAddress
- the address that the first region in the list must contain

memoryInfo
- client-allocated array of memory region records of type CUDBGMemoryInfo

memoryInfo_size
- number of records of type CUDBGMemoryInfo that memoryInfo can hold

numEntries
- pointer to a client-allocated variable holding the number of valid entries returned in memoryInfo

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The sorted list of memory regions starts from a region containing the specified starting address. If the starting address is set to 0, a sorted list of managed memory regions is returned which starts from the managed memory region with the lowest start address.

Since CUDA 6.0.

CUDBGResult (*CUDBGAPI_st::memcheckReadErrorAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *address, ptxStorageKind *storage)

Get the address that memcheck detected an error on.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

address
- returned address detected by memcheck

storage
- returned address class of address

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*CUDBGAPI_st::readActiveLanes) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *activeLanesMask)

Read the lane bitmask of active threads on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

activeLanesMask
- the returned bitmask of active threads

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readAllVirtualReturnAddresses) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *addrs, uint32_t numAddrs, uint32_t *callDepth, uint32_t *syscallCallDepth)

Read all the virtual return addresses for a thread (the full backtrace).

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addrs
- the returned addresses array

numAddrs
- number of elements in addrs array

callDepth
- the returned call depth

syscallCallDepth
- the returned syscall call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that syscallCallDepth is always set to 0.

Since CUDA 12.5.

See also:

readCallDepth

readReturnAddress

CUDBGResult (*CUDBGAPI_st::readBlockIdx) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx)

Read the CUDA block index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- the returned CUDA block index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readActiveLanes

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readBlockIdx32) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *blockIdx)

Read the two-dimensional CUDA block index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- the returned CUDA block index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readBlockIdx but doesn't return the z dimension.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use readBlockIdx instead.

See also:

readBlockIdx

CUDBGResult (*CUDBGAPI_st::readBrokenWarps) (uint32_t dev, uint32_t sm, uint64_t *brokenWarpsMask)

Read the bitmask of warps that are at a breakpoint on a given SM.

Parameters

dev
- device index

sm
- SM index

brokenWarpsMask
- the returned bitmask of broken warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readCallDepth) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth (number of calls) for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP,

CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readReturnAddress

readVirtualReturnAddress

CUDBGResult (*CUDBGAPI_st::readCallDepth32)(uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *depth)

Read the call depth (number of calls) for a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readCallDepth() for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readCallDepth instead.

See also:

readCallDepth

CUDBGResult (*CUDBGAPI_st::readCCRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *val)

Read the hardware CC register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

val
- the returned value of the CC register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 13.1: Do not use.

CUDBGResult (*CUDBGAPI_st::readClusterIdx) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterIdx)

Read the CUDA cluster index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

clusterIdx
- the returned CUDA cluster index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readCodeMemory) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the code memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS,

CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

It is generally not necessary to call this function - instead, the same memory could be read from the ELF module images received from the API.

Since CUDA 3.0.

See also:

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readConstMemory129) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the constant memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves exactly like readGlobalMemory.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 13.0: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*CUDBGAPI_st::readCPUCallStack) (uint32_t dev, uint64_t gridId64, uint64_t *addrs, uint32_t numAddrs, uint32_t *totalNumAddrs)

Read the CPU call stack captured at the time of kernel launch.

Parameters

dev
- device index

gridId64
- 64-bit grid ID

addrs
- the returned addresses array, can be NULL

numAddrs
- capacity of addrs (possibly 0)

totalNumAddrs
- the actual size of the stack (number of frames) is written here; the value written can be greater than numAddrs

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method only works if the CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES capability is enabled.

Since CUDA 12.9.

CUDBGResult (*CUDBGAPI_st::readDeviceExceptionState) (uint32_t dev, uint64_t *mask, uint32_t numWords)

Get the exception state of the SMs on the device.

Parameters

dev
- device index

mask
- Arbitrarily sized bit field containing a 1 at (1 << i) if SM i hit an exception

numWords
- Number of uint64_t elements in `mask` (must be large enough to hold a bit for each sm on the device)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 9.0.

See also:

getNumSMs

CUDBGResult (*CUDBGAPI_st::readDeviceExceptionState80) (uint32_t dev, uint64_t *exceptionSMMask)

Get the exception state of the SMs on the device.

Parameters

dev
- device index

exceptionSMMask
- Bit field containing a 1 at (1 << i) if SM i hit an exception

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readDeviceExceptionState but only supports up to 64 SMs.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 9.0: Use readDeviceExceptionState instead.

See also:

readDeviceExceptionState

CUDBGResult (*CUDBGAPI_st::readErrorPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *errorPC, bool *errorPCValid)

Get the hardware reported error PC if it exists.

Parameters

dev
- device index

sm
- SM index

wp

errorPC
- PC ofthe exception

errorPCValid
- boolean to indicate that the returned error PC is valid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The error PC, if available, shows the PC where an error happened (the thread can progress past that so its PC could be beyond that).

Since CUDA 6.0.

CUDBGResult (*CUDBGAPI_st::readGenericMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at an address in any memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

The address will be used to determine whether the read is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

readCodeMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readGlobalMemory) (uint64_t addr, void *buf, uint32_t sz)

Read content at an address in the global address space.

Parameters

addr
- memory address

buf
- buffer

sz

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

If the address is valid on more than one device and one of those devices does not support UVA, an error is returned.

Since CUDA 6.0.

See also:

readCodeMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readGlobalMemory31) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves like readGenericMemory() with sm, wp, ln == 0. This makes this method not at all useful.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*CUDBGAPI_st::readGlobalMemory55) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves exactly like readGenericMemory().

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 6.0: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*CUDBGAPI_st::readGridId) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *gridId64)

Read the 64-bit CUDA grid index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridId64

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The grid ID is guaranteed to be unique within a device, but not globally.

Since CUDA 5.5.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readGridId50) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *gridId)

Read the CUDA grid index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridId
- the returned CUDA grid index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readGridId but truncates the grid ID to 32bit. This is incompatible with some grid IDs like those used by the OptiX applications.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 5.5: Use readGridId instead.

See also:

readGridId

CUDBGResult (*CUDBGAPI_st::readLaneException) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CUDBGException_t *exception)

Read the exception type for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

exception
- the returned exception type

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.1.

CUDBGResult (*CUDBGAPI_st::readLaneStatus) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, bool *error)

Read the status of the given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

error
- true if there is an error

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

For specific error values, use readLaneException.

Since CUDA 3.0.

CUDBGResult (*CUDBGAPI_st::readLocalMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the local memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readParamMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the param memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC offset on the given active thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

pc
- the returned PC

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The returned PC offset is from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readParamMemory

readRegister

readSharedMemory

readTextureMemory

readVirtualPC

CUDBGResult (*CUDBGAPI_st::readPinnedMemory) (uint64_t addr, void *buf, uint32_t sz)

Read content at pinned address in system memory.

Parameters

addr

- system memory address

buf

- buffer

sz

- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_MEMORY_MAPPING_FAILED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL

Depending on the platform, this method may fail and a platform-specific CPU RAM way of reading memory from the debuggee must be used (e.g. ptrace).

Since CUDA 3.2.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, uint32_t *predicates)

Read content of hardware predicate registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

predicates_size
- number of predicate registers to read

predicates
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 6.5.

See also:

readCodeMemory

readGenericMemory

readGlobalMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t *val)

Read content of a hardware register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

regno
- register index

val
- the returned value of the register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that warps can dynamically change the number of used registers at runtime, readWarpResources() could be used to query that.

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readRegisterRange) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t numRegisters, uint32_t *registers, uint32_t *numRegistersRead)

Read content of a hardware range of hardware registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

index
- index of the first register to read

numRegisters
- number of registers to read

registers
- buffer

numRegistersRead
- number of registers actually read, ignored if null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readRegisterRange60) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t registers_size, uint32_t *registers)

Read content of a range of hardware registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

index
- index of the first register to read

registers_size
- number of registers to read

registers
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 6.0.

Note:

DEPRECATED in CUDA 13.2: Use readRegisterRange instead.

See also:

readRegisterRange

CUDBGResult (*CUDBGAPI_st::readReturnAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

level
- the specified call level

ra
- the returned return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The returned return address is an offset from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 4.0.

See also:

readCallDepth

readVirtualReturnAddress

CUDBGResult (*CUDBGAPI_st::readReturnAddress32) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

level
- the specified call level

ra
- the returned return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readReturnAddress() for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readReturnAddress instead.

See also:

readReturnAddress

CUDBGResult (*CUDBGAPI_st::readSharedMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the shared memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readTextureMemory

CUDBGResult (*CUDBGAPI_st::readSmException) (uint32_t dev, uint32_t sm, CUDBGException_t *exception, uint64_t *errorPC, bool *errorPCValid)

Get the SM exception status if it exists.

Parameters

dev
- the device index

sm
- the SM index

exception
- returned exception

errorPC
- returned PC of the exception

errorPCValid
- boolean to indicate that the returned error PC is valid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.5.

CUDBGResult (*CUDBGAPI_st::readSyscallCallDepth) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth of syscalls for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Will always return 0.

Since CUDA 4.1.

Note:

DEPRECATED in CUDA 12.9: Do not use.

CUDBGResult (*CUDBGAPI_st::readTextureMemory) (uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t id, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Parameters

dev
- device index

vsm

wp
- warp index

id
- texture id (the value of DW_AT_location attribute in the relocated ELF image)

dim
- texture dimension (1 to 4)

coords
- array of coordinates of size dim

buf
- result buffer

sz
- buffer size in bytes

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.0.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*CUDBGAPI_st::readTextureMemoryBindless) (uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t texSymtabIndex, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Parameters

dev
- device index

vsm

wp
- warp index

texSymtabIndex
- global symbol table index of the texture symbol

dim
- texture dimension (1 to 4)

coords
- array of coordinates of size dim

buf
- result buffer

sz
- buffer size in bytes

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.2.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*CUDBGAPI_st::readThreadIdx) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CuDim3 *threadIdx)

Read the CUDA thread index running on valid thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

threadIdx
- the returned CUDA thread index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readValidLanes

readValidWarps

CUDBGResult (*CUDBGAPI_st::readUniformPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, uint32_t *predicates)

Read contents of uniform predicate registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

predicates_size
- number of predicate registers to read

predicates
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

readPredicates

CUDBGResult (*CUDBGAPI_st::readUniformRegisterRange) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t registers_size, uint32_t *registers)

Read a range of uniform registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

regno
- starting index into uniform register file

registers_size
- number of bytes to read

registers
- pointer to buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

readRegister

CUDBGResult (*CUDBGAPI_st::readValidLanes) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *validLanesMask)

Read the lane bitmask of valid threads on a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

validLanesMask
- the returned bitmask of valid threads

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidWarps

CUDBGResult (*CUDBGAPI_st::readValidWarps) (uint32_t dev, uint32_t sm, uint64_t *validWarpsMask)

Read the bitmask of valid warps on a given SM.

Parameters

dev
- device index

sm
- SM index

validWarpsMask
- the returned bitmask of valid warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

CUDBGResult (*CUDBGAPI_st::readVirtualPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC (virtual address) on the given active thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

pc
- the returned PC

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP,

CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readPC

CUDBGResult (*CUDBGAPI_st::readVirtualReturnAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

level
- the specified call level

ra
- the returned virtual return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readCallDepth

readReturnAddress

CUDBGResult (*CUDBGAPI_st::readVirtualReturnAddress32) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

level
- the specified call level

ra
- the returned virtual return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readVirtualReturnAddress for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readVirtualReturnAddress instead.

See also:

readVirtualReturnAddress

CUDBGResult (*CUDBGAPI_st::readWarpResources) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpResources *resources)

Get the resources assigned to a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

resources
- pointer to structure that contains warp resources

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that these resources can change between suspends, which makes this method useful for avoiding warp data access errors.

Since CUDA 12.8.

CUDBGResult (*CUDBGAPI_st::readWarpState) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.9.

CUDBGResult (*CUDBGAPI_st::readWarpState120) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState120 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use readWarpState instead.

See also:

readWarpState

CUDBGResult (*CUDBGAPI_st::readWarpState127) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState127 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.7.

Note:

DEPRECATED in CUDA 12.9: Use readWarpState instead.

See also:

readWarpState

CUDBGResult (*CUDBGAPI_st::readWarpState60) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState60 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 6.0.

Note:

DEPRECATED in CUDA 12.0: Use readWarpState instead.

See also:

readWarpState

3.6. Device State Alteration

CUDBGResult (*CUDBGAPI_st::writeCCRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t val)

Write to the hardware CC register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

val
- the new value of the CC register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 13.1: Do not use.

CUDBGResult (*CUDBGAPI_st::writeGenericMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in any memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

The address will be used to determine whether the write is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

writeGlobalMemory

writeLocalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*CUDBGAPI_st::writeGlobalMemory) (uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

It's not possible to access a shared memory page or an ambiguous address allocated on several devices that don't support UVA.

Since CUDA 6.0.

See also:

writeGenericMemory

writeLocalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*CUDBGAPI_st::writeGlobalMemory31) (uint32_t dev, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

dev
- device index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

This method is unsupported on Hopper and later architectures. Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use writeGlobalMemory instead.

See also:

writeGlobalMemory

CUDBGResult (*CUDBGAPI_st::writeGlobalMemory55) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 6.0: Use writeGlobalMemory instead.

See also:

writeGlobalMemory

CUDBGResult (*CUDBGAPI_st::writeLocalMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in local memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within local memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*CUDBGAPI_st::writeParamMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in param memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within param memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeLocalMemory

writeSharedMemory

CUDBGResult (*CUDBGAPI_st::writePinnedMemory) (uint64_t addr, const void *buf, uint32_t sz)

Write to a pinned memory address.

Parameters

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_MEMORY_MAPPING_FAILED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL

It's not possible to access an ambiguous access allocated on several devices that don't support UVA.

Since CUDA 3.2.

See also:

readPinnedMemory

CUDBGResult (*CUDBGAPI_st::writePredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware predicates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

predicates_size
- predicates count

predicates
- predicate values

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method writes to predicates_size predicates, starting from P0. Each predicate value must be either 0 or 1.

Since CUDA 6.5.

See also:

writeRegister

writeUniformPredicates

writeUniformRegister

CUDBGResult (*CUDBGAPI_st::writeRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t val)

Write to a hardware register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

regno
- register number

val
- value

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

writePredicates

writeUniformPredicates

writeUniformRegister

CUDBGResult (*CUDBGAPI_st::writeSharedMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in shared memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within shared memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeLocalMemory

writeParamMemory

CUDBGResult (*CUDBGAPI_st::writeUniformPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware uniform predicates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

predicates_size
- predicates count

predicates
- predicate values

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method writes to predicates_size uniform predicates, starting from UP0. Each predicate value must be either 0 or 1.

Since CUDA 10.0.

See also:

writePredicates

writeRegister

writeUniformRegister

CUDBGResult (*CUDBGAPI_st::writeUniformRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t val)

Write to a hardware uniform register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

regno
- register number

val
- value

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

writePredicates

writeRegister

writeUniformPredicates

3.7. Grid Properties

struct CUDBGAttributeValuePair

Grid attribute-value pair.

struct CUDBGGridInfo

Information about a CUDA grid.

struct CUDBGGridInfo120

Grid info.

struct CUDBGGridInfo55

Grid info.

enum CUDBGAttribute

Queryable grid attributes.

Values

CUDBG_ATTR_GRID_LAUNCH_BLOCKING = 0x000
Whether the launch is synchronous (blocking) or not.

CUDBG_ATTR_GRID_TID = 0x001
The id of the host thread that launched the grid.

enum CUDBGGridStatus

Grid status.

Values

CUDBG_GRID_STATUS_INVALID
An invalid grid ID was passed, or an error occurred during status lookup.

CUDBG_GRID_STATUS_PENDING
The grid was launched but is not running on the HW yet.

CUDBG_GRID_STATUS_ACTIVE
The grid is currently running on the HW.

CUDBG_GRID_STATUS_SLEEPING
The grid is on the device, doing a join.

CUDBG_GRID_STATUS_TERMINATED
The grid has finished executing.

CUDBG_GRID_STATUS_UNDETERMINED
The grid is either QUEUED or TERMINATED.

enum CUDBGKernelLaunchNotifyMode

Kernel launch notification mode.

Values

CUDBG_KNL_LAUNCH_NOTIFY_EVENT = 0x000
Kernel launches generate launch notification events.

CUDBG_KNL_LAUNCH_NOTIFY_DEFER = 0x001
Kernel launches do not generate any notification.

enum CUDBGKernelOrigin

Kernel origin.

Values

CUDBG_KNL_ORIGIN_CPU = 0x000
The kernel was launched from the CPU.

CUDBG_KNL_ORIGIN_GPU = 0x001
The kernel was launched from the GPU.

enum CUDBGKernelType

Kernel types.

Values

CUDBG_KNL_TYPE_UNKNOWN = 0x000
Unknown kernel type. Fall-back value.

CUDBG_KNL_TYPE_SYSTEM = 0x001
System kernel, launched by the CUDA driver (cudaMemset, ...).

CUDBG_KNL_TYPE_APPLICATION = 0x002
Application kernel, launched by the application (user-defined or libraries).

CUDBGResult (*CUDBGAPI_st::getBlockDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockDim)

Get the dimensions of the given block.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockDim
- the returned number of threads in the block

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

getClusterDim

getGridDim

CUDBGResult (*CUDBGAPI_st::getClusterDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterDim)

Get the number of blocks in the given cluster.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

clusterDim
- the returned number of blocks in the cluster

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.7.

See also:

getBlockDim

getGridDim

CUDBGResult (*CUDBGAPI_st::getClusterDim120) (uint32_t dev, uint64_t gridId64, CuDim3 *clusterDim)

Get the number of blocks in the given cluster.

Parameters

dev
- device index

gridId64
- grid ID

clusterDim
- the returned number of blocks in the cluster

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getClusterDim, but takes a grid ID instead of warp coordinates. In newer GPU architectures, it's possible to have different warps belong to blocks of clusters of different size.

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use getClusterDim instead.

See also:

getClusterDim

CUDBGResult (*CUDBGAPI_st::getElfImage) (uint32_t dev, uint32_t sm, uint32_t wp, bool relocated, void **elfImage, uint64_t *size)

Get the relocated or non-relocated ELF image and size for the grid on the given device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

relocated
- set to true to specify the relocated ELF image, false otherwise

***elfImage**

size
- size of the ELF image (64 bits)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

getElfImageByHandle

getLoadedFunctionInfo

CUDBGResult (*CUDBGAPI_st::getElfImage32) (uint32_t dev, uint32_t sm, uint32_t wp, bool relocated, void **elfImage, uint32_t *size)

Get the relocated or non-relocated ELF image and size for the grid on the given device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

relocated
- set to true to specify the relocated ELF image, false otherwise

***elfImage**

size
- size of the ELF image (32 bits)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getElfImage but will truncate the image size for cubins larger than 4GiB.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use getElfImage instead.

See also:

getElfImage

getElfImageByHandle

CUDBGResult (*CUDBGAPI_st::getGridAttribute) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGAttribute attr, uint64_t *value)

Get the value of a grid attribute.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

attr
- the attribute

value
- the returned value of the attribute

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_ATTRIBUTE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttributes

CUDBGResult (*CUDBGAPI_st::getGridAttributes) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGAttributeValuePair *pairs, uint32_t numPairs)

Get several grid attribute values in a single API call.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

pairs
- array of attribute/value pairs

numPairs
- the number of attribute/values pairs in the array

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_ATTRIBUTE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttribute

CUDBGResult (*CUDBGAPI_st::getGridDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *gridDim)

Get the dimensions in blocks of the given grid.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridDim
- the dimensions of the grid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

getBlockDim

getClusterDim

CUDBGResult (*CUDBGAPI_st::getGridDim32) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *gridDim)

Get the dimensions of the given grid.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridDim
- the returned number of blocks in the grid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridDim but doesn't return the z dimension.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use getGridDim instead.

See also:

getGridDim

CUDBGResult (*CUDBGAPI_st::getGridInfo) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.7.

CUDBGResult (*CUDBGAPI_st::getGridInfo120) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo120 *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use getGridInfo instead.

See also:

getGridInfo

CUDBGResult (*CUDBGAPI_st::getGridInfo55) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo55 *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 12.0: Use getGridInfo instead.

See also:

getGridInfo

CUDBGResult (*CUDBGAPI_st::getGridStatus) (uint32_t dev, uint64_t gridId64, CUDBGGridStatus *status)

Check whether the grid corresponding to the ID is still present on the device.

Parameters

dev
- device index

gridId64
- 64-bit grid ID

status
- enum indicating the grid status

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 5.5.

CUDBGResult (*CUDBGAPI_st::getGridStatus50) (uint32_t dev, uint32_t gridId, CUDBGGridStatus *status)

Check whether the grid corresponding to the ID is still present on the device.

Parameters

dev
- device index

gridId
- grid ID

status
- enum indicating the grid status

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridStatus, but takes a 32-bit grid ID instead of a 64-bit one.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getGridStatus instead.

See also:

getGridStatus

CUDBGResult (*CUDBGAPI_st::getTID) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *tid)

Get the ID of the Linux thread hosting the CUDA context active at the given coordinates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

tid
- the returned thread id

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This returns a Linux thread ID.

Since CUDA 3.0.

See also:

getGridAttributes

3.8. Device Properties

CUDBGResult (*CUDBGAPI_st::getDeviceName) (uint32_t dev, char *buf, uint32_t sz)

Get the device name string.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getDeviceType

getSMType

CUDBGResult (*CUDBGAPI_st::getDeviceType) (uint32_t dev, char *buf, uint32_t sz)

Get the string description of the device.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

getSMType

CUDBGResult (*CUDBGAPI_st::getNumDevices) (uint32_t *numDev)

Get the number of installed CUDA devices.

Parameters

numDev
- the returned number of devices

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session.

Since CUDA 3.0.

See also:

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumLanes) (uint32_t dev, uint32_t *numLanes)

Get the number of lanes per warp on the device.

Parameters

dev
- device index

numLanes
- the returned number of lanes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumPredicates) (uint32_t dev, uint32_t *numPredicates)

Get the number of predicate registers per lane on the device.

Parameters

dev
- device index

numPredicates
- the returned number of predicate registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getNumDevices

getNumLanes

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumRegisters) (uint32_t dev, uint32_t *numRegs)

Get the maximum number of registers per lane on the device.

Parameters

dev
- device index

numRegs
- the returned number of registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device. Note that the actual number of registers can change per warp, use readWarpResources() to query that number dynamically.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

readWarpResources

CUDBGResult (*CUDBGAPI_st::getNumSMs) (uint32_t dev, uint32_t *numSMs)

Get the total number of SMs on the device.

Parameters

dev
- device index

numSMs
- the returned number of SMs

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumUniformPredicates) (uint32_t dev, uint32_t *numPredicates)

Get the number of uniform predicate registers per warp on the device.

Parameters

dev
- device index

numPredicates
- the returned number of uniform predicate registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformRegisters

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumUniformRegisters) (uint32_t dev, uint32_t *numRegs)

Get the number of uniform registers per warp on the device.

Parameters

dev
- device index

numRegs
- the returned number of uniform registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumWarps

CUDBGResult (*CUDBGAPI_st::getNumWarps) (uint32_t dev, uint32_t *numWarps)

Get the number of warps per SM on the device.

Parameters

dev
- device index

numWarps
- the returned number of warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

CUDBGResult (*CUDBGAPI_st::getSmType) (uint32_t dev, char *buf, uint32_t sz)

Get the SM type of the device.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

getDeviceType

3.9. DWARF Utilities

enum CUDBGElfImageType

CUDA ELF image type.

Values

CUDBG_ELF_IMAGE_TYPE_NONRELOCATED = 0
Non-relocated ELF image.

CUDBG_ELF_IMAGE_TYPE_RELOCATED = 1
Relocated ELF image.

enum CUDBGRegClass

Physical register types.

Values

REG_CLASS_INVALID = 0x000
The physical register is invalid.

REG_CLASS_REG_CC = 0x001
The physical register is a condition code register. Unused.

REG_CLASS_REG_PRED = 0x002
The physical register is a predicate register. Unused.

REG_CLASS_REG_ADDR = 0x003
The physical register is an address register. Unused.

REG_CLASS_REG_HALF = 0x004
The physical register is a 16-bit register. Unused.

REG_CLASS_REG_FULL = 0x005
The physical register is a 32-bit register.

REG_CLASS_MEM_LOCAL = 0x006
The content of the physical register has been spilled to memory.

REG_CLASS_LMEM_REG_OFFSET = 0x007
The content of the physical register has been spilled to the local stack (ABI only).

REG_CLASS_UREG_PRED = 0x009
The physical register is a uniform predicate register.

REG_CLASS_UREG_HALF = 0x00a
The physical register is a 16-bit uniform register.

REG_CLASS_UREG_FULL = 0x00b
The physical register is a 32-bit uniform register.

CUDBGResult (*CUDBGAPI_st::disassemble) (uint32_t dev, uint64_t addr, uint32_t *instSize, char *buf, uint32_t sz)

Disassemble instruction at instruction address.

Parameters

dev
- device index

addr
- instruction address

instSize
- instruction size (32 or 64 bits)

buf
- disassembled instruction buffer

sz
- disassembled instruction buffer size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method does not guarantee any specific output format and its result should be treated as plain text.

Since CUDA 3.0.

CUDBGResult (*CUDBGAPI_st::getElfImageByHandle) (uint32_t dev, uint64_t handle, CUDBGElfImageType type, void *elfImage, uint64_t size)

Get the relocated or non-relocated ELF image for the given handle on the given device.

Parameters

dev
- device index

handle
- elf image handle

type
- type of the requested ELF image

elfImage
- pointer to the ELF image

size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The handle is provided in the ELF Image Loaded notification event.

Since CUDA 6.0.

See also:

getElfImage

getLoadedFunctionInfo

CUDBGResult (*CUDBGAPI_st::getHostAddrFromDeviceAddr) (uint32_t dev, uint64_t device_addr, uint64_t *host_addr)

Given a device virtual address, return a corresponding system memory virtual address.

Parameters

dev
- device index

device_addr
- device memory address

host_addr
- returned system memory address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_SEGMENT, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.1.

See also:

readGenericMemory

writeGenericMemory

CUDBGResult (*CUDBGAPI_st::getPhysicalRegister30) (uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass)

Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

Parameters

pc
- Program counter

reg
- virtual register index

buf
- physical register name(s)

sz
- the physical register name buffer size

numPhysRegs
- number of physical register names returned

regClass
- the class of the physical registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Do not use.

CUDBGResult (*CUDBGAPI_st::getPhysicalRegister40) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass)

Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

pc
- Program counter

reg
- virtual register index

buf
- physical register name(s)

sz
- the physical register name buffer size

numPhysRegs
- number of physical register names returned

regClass
- the class of the physical registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Instead, the PTX to SASS mappings can be read from the cubin directly.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.1: Do not use.

CUDBGResult (*CUDBGAPI_st::isDeviceCodeAddress) (uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Parameters

addr
- virtual address

isDeviceAddress
- true if address resides within device code

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

CUDBGResult (*CUDBGAPI_st::isDeviceCodeAddress55) (uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Parameters

addr
- virtual address

isDeviceAddress
- true if address resides within device code

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves exactly like isDeviceCodeAddress.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 6.0: Use isDeviceCodeAddress instead.

See also:

isDeviceCodeAddress

CUDBGResult (*CUDBGAPI_st::lookupDeviceCodeSymbol) (char *symName, bool *symFound, uintptr_t *symAddr)

Determines whether a symbol represents a function in device code and returns its virtual address.

Parameters

symName
- symbol name

symFound
- set to true if the symbol is found

symAddr
- the symbol virtual address if found

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

3.10. Events

One of those events will create a CUDBGEvent:

- the elf image of the current kernel has been loaded and the addresses within its DWARF sections have been relocated (and can now be used to set breakpoints),
- a device breakpoint has been hit,
- a CUDA kernel is ready to be launched,
- a CUDA kernel has terminated.

When a CUDBGEvent is created, the debugger is notified by calling the callback functions registered with setNotifyNewEventCallback() after the API struct initialization. It is up to the debugger to decide what method is best to be notified. The debugger API routines cannot be called from within the callback function or the routine will return an error.

Upon notification the debugger is responsible for handling the CUDBGEvents in the event queue by using CUDBGAPI_st::getNextEvent(), and for acknowledging the debugger API that the event has been handled by calling CUDBGAPI_st::acknowledgeEvent(). In the case of an event raised by the device itself, such as a breakpoint being hit, the event queue will be empty. It is the responsibility of the debugger to inspect the hardware any time a CUDBGEvent is received.

Example:

```
CUDBGEvent event;
      CUDBGResult res;
      for (res = cudbgAPI->getNextEvent(&event);
           res == CUDBG_SUCCESS && event.kind != CUDBG_EVENT_INVALID;
           res = cudbgAPI->getNextEvent(&event)) {
          switch (event.kind)
              {
              case CUDBG_EVENT_ELF_IMAGE_LOADED:
                  //...
                  break;
              case CUDBG_EVENT_KERNEL_READY:
                  //...
                  break;
              case CUDBG_EVENT_KERNEL_FINISHED:
                  //...
                  break;
              default:
                  error(...);
              }
          }
```

See cuda-tdep.c and cuda-linux-nat.c files in the cuda-gdb source code for a more detailed example on how to use CUDBGEvent.

struct CUDBGEvent

Event information container.

struct CUDBGEvent30

Event information container for API version 3.0.

struct CUDBGEvent32

Event information container for API version 3.2.

struct CUDBGEvent42

Event information container for API version 4.2.

struct CUDBGEvent50

Event information container for API version 5.0.

struct CUDBGEvent55

Event information container for API version 5.5.

struct CUDBGEventCallbackData

Callback data passed to callback set with setNotifyNewEventCallback function.

struct CUDBGEventCallbackData40

Callback data passed to callback set with setNotifyNewEventCallback40 function.

struct CUDBGEventCallbackData41

Callback data passed to callback set with setNotifyNewEventCallback41 function.

enum CUDBGEventKind

Kinds of events sent by the debug engine to the API client.

Values

CUDBG_EVENT_INVALID = 0x000
Invalid event.

CUDBG_EVENT_ELF_IMAGE_LOADED = 0x001
The ELF image for a CUDA source module is available.

CUDBG_EVENT_KERNEL_READY = 0x002
A CUDA kernel is about to be launched.

CUDBG_EVENT_KERNEL_FINISHED = 0x003
A CUDA kernel has terminated.

CUDBG_EVENT_INTERNAL_ERROR = 0x004
An internal error occurred. The API may be unstable.

CUDBG_EVENT_CTX_PUSH = 0x005
A CUDA context has been pushed.

CUDBG_EVENT_CTX_POP = 0x006
A CUDA context has been popped.

CUDBG_EVENT_CTX_CREATE = 0x007
A CUDA context has been created.

CUDBG_EVENT_CTX_DESTROY = 0x008
A CUDA context has been, popped if pushed, then destroyed.

CUDBG_EVENT_TIMEOUT = 0x009
A timeout event is sent at regular interval. This event can safely be ignored. Only sent by the classic backend.

CUDBG_EVENT_ATTACH_COMPLETE = 0x00a
The attach process has completed and debugging of device code may start.

CUDBG_EVENT_DETACH_COMPLETE = 0x00b
The detach process has completed.

CUDBG_EVENT_ELF_IMAGE_UNLOADED = 0x00c
The ELF image for CUDA kernel(s) no longer available.

CUDBG_EVENT_FUNCTIONS_LOADED = 0x00d
A group of functions/kernels have been loaded Will only be sent if the debug engine capability CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING is set.

CUDBG_EVENT_ALL_DEVICES_SUSPENDED = 0x00e
All CUDA devices have been suspended due to a breakpoint hit or an exception. Does not get sent for GPU events that result in synchronous API method calls, such as singleStepWarp or resumeWarpsUntilPC. Will only be sent if the debug engine capability CUDBG_DEBUGGER_CAPABILITY_SUSPEND_EVENTS is set.

CUDBG_EVENT_CUDA_LOGS_AVAILABLE = 0x00f
(Async) CUDA Logs are available for the debugger to consume. After receiving this asynchronous event, debuggers should drain all available log entries by repeatedly calling consumeCudaLogs until no more logs are available. This event is only sent for the first log message that's generated after the client has read all logs with consumeCudaLogs. It is not sent by default, and can be enabled via the capability CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS.

CUDBG_EVENT_CUDA_LOGS_THRESHOLD_REACHED = 0x010
(Sync) CUDA Logs buffer has reached an implementation-defined threshold. The client should call consumeCudaLogs to avoid excessive log buildup. New logs will still be collected even if consumeCudaLogs is not called.

CUDBG_EVENT_SINGLE_STEP_COMPLETE = 0x011
(Sync) Asynchronous single step operation has been completed. Call singleStepWarp or resumeWarpsUntilPC with CUDBG_SINGLE_STEP_FLAGS_NON_BLOCKING to enable this event. This event is only sent if the single step operation was completed. If it's interrupted by a

breakpoint or an exception, a CUDBG_EVENT_ALL_DEVICES_SUSPENDED event will be sent instead.

enum CUDBGEventQueueType

Application event queue type.

Values

CUDBG_EVENT_QUEUE_TYPE_SYNC = 0
Synchronous event queue.

CUDBG_EVENT_QUEUE_TYPE_ASYNC = 1
Asynchronous event queue.

typedef (*CUDBGNotifyNewEventCallback) (CUDBGEventCallbackData* data)

Function type of the function called to notify debugger of the presence of a new event in the event queue.

typedef (*CUDBGNotifyNewEventCallback31) (void* data)

Function type of the function called to notify debugger of the presence of a new event in the event queue.

Note:

DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

typedef (*CUDBGNotifyNewEventCallback40) (CUDBGEventCallbackData40* data)

Function type of the function called to notify debugger of the presence of a new event in the event queue.

Note:

DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

typedef (*CUDBGNotifyNewEventCallback41) (CUDBGEventCallbackData41* data)

Function type of the function called to notify debugger of the presence of a new event in the event queue.

Note:

DEPRECATED: Use CUDBGNotifyNewEventCallback instead.

CUDBGResult (*CUDBGAPI_st::acknowledgeEvent30) (CUDBGEvent30 *event)

Inform the debugger API that synchronous events have been processed.

Parameters

event

- pointer to the event that has been processed

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

Behaves exactly like acknowledgeSyncEvents (the event parameter is ignored).

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Use acknowledgeSyncEvents instead.

See also:

acknowledgeSyncEvents

CUDBGResult (*CUDBGAPI_st::acknowledgeEvents42) ()

Inform the debugger API that synchronous events have been processed.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

Behaves exactly like acknowledgeSyncEvents.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 5.0: Use acknowledgeSyncEvents instead.

See also:

acknowledgeSyncEvents

CUDBGResult (*CUDBGAPI_st::acknowledgeSyncEvents) ()

Inform the debugger API that synchronous events have been processed.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

This resumes any process that was interrupted by the synchronous event (e.g. a context creation, a module load, etc.). This method always acknowledges only those SYNC events that have been read with getNextEvent (or its deprecated variants). SYNC events that haven't been read are not acknowledged and will continue to prevent their corresponding processes from proceeding. ASYNC events do not require acknowledgement.

Since CUDA 5.0.

See also:

getNextEvent

setNotifyNewEventCallback

CUDBGResult (*CUDBGAPI_st::getErrorStringEx) (char *buf, uint32_t bufSz, uint32_t *msgSz)

Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string.

Parameters

buf
- the destination buffer

bufSz
- the size of the destination buffer in bytes

msgSz
- the size of the written error message including the terminating null character.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The error message is specific to the last failed API call and is invalidated after every API method call except this one. It's possible to query the size of the error message without reading it by passing 0 as `buf` and `bufSz` parameters. The `msgSz` parameter is optional unless 0 as passed in as `buf` and `bufSz`. CUDBG_ERROR_BUFFER_TOO_SMALL is returned when the passed in buffer is too small to contain the message.

Since CUDA 12.2.

CUDBGResult (*CUDBGAPI_st::getNextAsyncEvent50) (CUDBGEvent50 *event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

Parameters

event
- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextAsyncEvent55) (CUDBGEvent55 *event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 6.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextEvent) (CUDBGEventQueueType type, CUDBGEvent *event)

Copies the next available event into 'event' and removes it from the queue.

Parameters

type

- application event queue type

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE

CUDBG_ERROR_NO_EVENT_AVAILABLE is returned if the queue is empty. ASYNC and SYNC queues are separate and each one is ordered separately, but it's impossible to find out the relative order of ASYNC and SYNC events.

Since CUDA 6.0.

See also:

acknowledgeSyncEvents

setNotifyNewEventCallback

CUDBGResult (*CUDBGAPI_st::getNextEvent30) (CUDBGEvent30 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextEvent32) (CUDBGEvent32 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextEvent42) (CUDBGEvent42 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 4.0.

Note:

DEPRECATED in CUDA 5.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextSyncEvent50) (CUDBGEvent50 *event)

Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::getNextSyncEvent55) (CUDBGEvent55 *event)

Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 6.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*CUDBGAPI_st::setNotifyNewEventCallback) (CUDBGNotifyNewEventCallback callback, void *userData)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback

- the callback function

userData

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The callback function is called for every ASYNC and SYNC event. The callback function is always called on the same thread. No API methods can be called from that thread except getNextEvent() and acknowledgeSyncEvents() (and their deprecated variants), otherwise CUDBG_ERROR_RECURSIVE_API_CALL will be returned.

Since CUDA 13.0.

See also:

acknowledgeSyncEvents

getNextEvent

CUDBGResult (*CUDBGAPI_st::setNotifyNewEventCallback31) (CUDBGNotifyNewEventCallback31 callback, void *data)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

data
- a pointer to be passed to the callback when called

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't return the host thread ID from which the event originates.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

CUDBGResult (*CUDBGAPI_st::setNotifyNewEventCallback40) (CUDBGNotifyNewEventCallback40 callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer.

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 4.1: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

CUDBGResult (*CUDBGAPI_st::setNotifyNewEventCallback41) (CUDBGNotifyNewEventCallback41 callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer. The timeout field is always 0.

Since CUDA 4.1.

Note:

DEPRECATED in CUDA 13.0: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

Chapter 4. Data Structures

Here are the data structures with brief descriptions:

cudbgGetAPI
CUDA debugger API methods

CUDBGAttributeValuePair
Grid attribute-value pair

CUDBGCbuWarpState
Warp state in the CBU (Convergence Barrier Unit)

CUDBGCudaLogMessage
CUDA Log Message

CUDBGDeviceInfo
Device-level information. This is the first element in the deviceInfoBuffer, and is always present. getDeviceInfo() takes a deviceId as input, so no need to explicitly pass it back here. Only "valid & updated" SMs/Warps/Lanes are included in the buffer, which allows us to determine indexes without having to encode an explicit ID field in the following buffer datastructures

CUDBGDeviceInfoSizes
Sizes of the various structs returned by the batched device update APIs. No explicit version field - implied by debugAPI major.minor.revision

CUDBGEvent
Event information container

CUDBGEvent30
Event information container for API version 3.0

CUDBGEvent32
Event information container for API version 3.2

CUDBGEvent42
Event information container for API version 4.2

CUDBGEvent50
Event information container for API version 5.0

CUDBGEvent55
Event information container for API version 5.5

CUDBGEvent::CUDBGEvent::cases_st
Information for each type of event

CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::allDevicesSuspended_st
Information about an event which forced all devices to be suspended

CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st
Information about the context being created
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st
Information about the context being destroyed
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st
Information about the context being popped
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st
Information about the context being pushed
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st
Information about the loaded ELF image
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageUnloaded_st
Information about the ELF image about to be unloaded
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::functionsLoaded_st
Information about the functions being lazily loaded
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::internalError_st
Information about internal erros
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st
Information about the kernel that just terminated
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
Information about the kernel ready to be launched
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st
Information about a single step operation that has completed
CUDBGEventCallbackData
Callback data passed to callback set with setNotifyNewEventCallback function
CUDBGEventCallbackData40
Callback data passed to callback set with setNotifyNewEventCallback40 function
CUDBGEventCallbackData41
Callback data passed to callback set with setNotifyNewEventCallback41 function
CUDBGGridInfo
Information about a CUDA grid
CUDBGGridInfo120
Grid info
CUDBGGridInfo55
Grid info
CUDBGLaneState
Lane state (state of a single thread)
CUDBGLoadedFunctionInfo
Information about a lazily loaded function
CUDBGMemoryInfo
Memory information
CUDBGSMInfo
SM-level information

CUDBGWarpInfo
Warp-level information
CUDBGWarpResources
Warp resources. These resources can change at runtime between suspends
CUDBGWarpState
Warp state information
CUDBGWarpState120
Warp state for API version 12.0
CUDBGWarpState127
Warp state for API version 12.7
CUDBGWarpState60
Warp state for API version 6.0
CuDim2
2-dimensional coordinates for threads, blocks, etc
CuDim3
3-dimensional coordinates for threads, blocks, etc

4.1. CUDBGAPI_st Struct Reference

CUDA debugger API methods.

CUDA Debugger API methods.

Get the API associated with the major/minor/revision version numbers.

See also:

cudbgGetAPIVersion

CUDBGResult (*acknowledgeEvent30) (CUDBGEvent30 *event)

Inform the debugger API that synchronous events have been processed.

Parameters

event
- pointer to the event that has been processed

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

Behaves exactly like acknowledgeSyncEvents (the event parameter is ignored).

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Use acknowledgeSyncEvents instead.

See also:

acknowledgeSyncEvents

CUDBGResult (*acknowledgeEvents42) ()

Inform the debugger API that synchronous events have been processed.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

Behaves exactly like acknowledgeSyncEvents.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 5.0: Use acknowledgeSyncEvents instead.

See also:

acknowledgeSyncEvents

CUDBGResult (*acknowledgeSyncEvents) ()

Inform the debugger API that synchronous events have been processed.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE

This resumes any process that was interrupted by the synchronous event (e.g. a context creation, a module load, etc.). This method always acknowledges only those SYNC events that have been read with getNextEvent (or its deprecated variants). SYNC events that haven't been read are not acknowledged and will continue to prevent their corresponding processes from proceeding. ASYNC events do not require acknowledgement.

Since CUDA 5.0.

See also:

getNextEvent

setNotifyNewEventCallback

CUDBGResult (*clearAttachState) ()

Clear attach-specific state prior to detach.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This call prepares the API for detaching. See the "Attaching and Detaching" section for more information.

Since CUDA 5.0.

CUDBGResult (*consumeCudaLogs) (CUDBGCudaLogMessage *logMessages, uint32_t numMessages, uint32_t *numConsumed)

Get CUDA error log entries.

Parameters

logMessages
- client-allocated array to store log entries

numMessages
- capacity of the logMessages array, in number of elements

numConsumed
- returned number of entries written to logMessages

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_RECURSIVE_API_CALL

This consumes the log entries, so they will not be available in subsequent calls. This functionality is only available if the CUDBG_DEBUGGER_CAPABILITY_ENABLE_CUDA_LOGS capability is enabled.

Since CUDA 12.9.

CUDBGResult (*disableBreakpoint) (CUDBGBreakpointHandle handle)

Disable a breakpoint specified by its handle.

Parameters

handle
- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*disassemble) (uint32_t dev, uint64_t addr, uint32_t *instSize, char *buf, uint32_t sz)

Disassemble instruction at instruction address.

Parameters

dev
- device index

addr
- instruction address

instSize
- instruction size (32 or 64 bits)

buf
- disassembled instruction buffer

sz
- disassembled instruction buffer size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method does not guarantee any specific output format and its result should be treated as plain text.

Since CUDA 3.0.

CUDBGResult (*enableBreakpoint)(CUDBGBreakpointHandle handle)

Enable a breakpoint specified by its handle.

Parameters

handle
- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Disabling/enabling a breakpoint might be faster than removing and inserting it again.

Since CUDA 13.2.

See also:

disableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*executeInternalCommand) (const char *command, char *resultBuffer, uint32_t sizeInBytes)

Execute an internal command (not available in public driver builds).

Parameters

command
- the command name and arguments

resultBuffer
- the destination buffer

sizeInBytes
- buffer size in bytes

Returns

CUDBG_ERROR_NOT_SUPPORTED

Always returns CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 12.6.

CUDBGResult (*finalize) ()

Finalize the API, shutting down the debugging session.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

initialize

CUDBGResult (*generateCoredump) (const char *filename, CUDBGCoredumpGenerationFlags flags)

Generate a coredump for the current GPU state.

Parameters

filename
- target coredump file name

flags
- coredump generation flags/options

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.3.

CUDBGResult (*getAdjustedCodeAddress) (uint32_t dev, uint64_t address, uint64_t *adjustedAddress, CUDBGAdjAddrAction adjAction)

Get the adjusted code address for a given code address for a given device.

Parameters

dev
- device index

address

adjustedAddress
- adjusted address

adjAction
- whether the adjusted next, previous or current address is needed

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The client must call this function before inserting a breakpoint, or when the previous or next code address is needed for breakpoint inserting purposes.

Since CUDA 5.5.

See also:

setBreakpoint

CUDBGResult (*getBlockDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockDim)

Get the dimensions of the given block.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockDim
- the returned number of threads in the block

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

getClusterDim

getGridDim

CUDBGResult (*getCbuWarpState) (uint32_t dev, uint32_t sm, uint64_t warpMask, CUDBGCbuWarpState *warpStates, uint32_t numWarpStates)

Get the CBU state of a given warp.

Parameters

dev
- device index

sm
- SM index

warpMask
- bitmask of the warps which states should be returned in warpStates

warpStates
- pointer to the array of CUDBGCbuWarpState structures

numWarpStates
- number of elements in warpStates array

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.9.

CUDBGResult (*getClusterDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterDim)

Get the number of blocks in the given cluster.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

clusterDim
- the returned number of blocks in the cluster

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.7.

See also:

getBlockDim

getGridDim

CUDBGResult (*getClusterDim120) (uint32_t dev, uint64_t gridId64, CuDim3 *clusterDim)

Get the number of blocks in the given cluster.

Parameters

dev
- device index

gridId64
- grid ID

clusterDim
- the returned number of blocks in the cluster

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getClusterDim, but takes a grid ID instead of warp coordinates. In newer GPU architectures, it's possible to have different warps belong to blocks of clusters of different size.

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use getClusterDim instead.

See also:

getClusterDim

CUDBGResult (*getClusterExceptionTargetBlock) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx, bool *blockIdxValid)

Get the target block index and validity status for cluster exceptions.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- pointer to a `CuDim3` structure that will be populated with the target block index

blockIdxValid
- pointer to a boolean variable that will be set to `true` if the target block index is valid, and `false` otherwise. Value will be set to false if the warp is not stopped on a cluster exception

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Since CUDA 12.7.

CUDBGResult (*getConstBankAddress) (uint32_t dev, uint64_t gridId64, uint32_t bank, uint64_t *address, uint32_t *size)

Get constant bank GPU VA and size.

Parameters

dev
- device index

gridId64
- grid ID of the grid containing the constant bank

bank
- constant bank number

address
- GPU VA of the bank memory

size
- bank size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_MISSING_DATA

Since CUDA 12.4.

CUDBGResult (*getConstBankAddress123) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t bank, uint32_t offset, uint64_t *address)

Convert constant bank number and offset into GPU VA.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

bank
- constant bank number

offset
- offset within the bank

address
- GPU VA

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_MISSING_DATA

It's more convenient to get the constbank address and then calculate the VA for any const address using that.

Since CUDA 12.3.

Note:

DEPRECATED in CUDA 12.4: Use getConstBankAddress instead.

See also:

getConstBankAddress

CUDBGResult (*getCudaExceptionString) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get the error string for CUDA Exceptions.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

buf
- buffer for the error string

bufSz
- buffer size

msgSz
- error message size with null character, can be null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.0.

CUDBGResult (*getDeviceInfo) (uint32_t dev, CUDBGDeviceInfoQueryType_t type, void *buffer, uint32_t length, uint32_t *dataLength)

Read full device info for the device.

Parameters

dev
- device index

type
- query type (full or delta)

buffer
- output buffer

length
- output buffer length

dataLength
- number of bytes written to the buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Information returned by this method is cheap to calculate, so it can be used after every suspend to quickly get the updated device state. For convenience, the caller can always request partial updates, the API will return a full response when returning a partial one is not possible. If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.4.

See also:

getDeviceInfoSizes

CUDBGResult (*getDeviceInfoSizes) (uint32_t dev, CUDBGDeviceInfoSizes *sizes)

Return sizes for device info structs and defined attributes.

Parameters

dev
- device index

sizes
- device info sizes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.4.

See also:

getDeviceInfo

CUDBGResult (*getDeviceName) (uint32_t dev, char *buf, uint32_t sz)

Get the device name string.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getDeviceType

getSMType

CUDBGResult (*getDevicePCIBusInfo) (uint32_t dev, uint32_t *pciBusId, uint32_t *pciDevId)

Get PCI bus and device ids associated with device index.

Parameters

dev
- device index

pciBusId
- pointer where corresponding PCI BUS ID would be stored

pciDevId
- pointer where corresponding PCI DEVICE ID would be stored

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 5.5.

CUDBGResult (*getDeviceType) (uint32_t dev, char *buf, uint32_t sz)

Get the string description of the device.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

getSMType

CUDBGResult (*getElfImage) (uint32_t dev, uint32_t sm, uint32_t wp, bool relocated, void **elfImage, uint64_t *size)

Get the relocated or non-relocated ELF image and size for the grid on the given device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

relocated
- set to true to specify the relocated ELF image, false otherwise

***elfImage**

size
- size of the ELF image (64 bits)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

getElfImageByHandle

getLoadedFunctionInfo

CUDBGResult (*getElfImage32) (uint32_t dev, uint32_t sm, uint32_t wp, bool relocated, void **elfImage, uint32_t *size)

Get the relocated or non-relocated ELF image and size for the grid on the given device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

relocated
- set to true to specify the relocated ELF image, false otherwise

***elfImage**

size
- size of the ELF image (32 bits)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getElfImage but will truncate the image size for cubins larger than 4GiB.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use getElfImage instead.

See also:

getElfImage

getElfImageByHandle

CUDBGResult (*getElfImageByHandle) (uint32_t dev, uint64_t handle, CUDBGElfImageType type, void *elfImage, uint64_t size)

Get the relocated or non-relocated ELF image for the given handle on the given device.

Parameters

dev
- device index

handle
- elf image handle

type
- type of the requested ELF image

elfImage
- pointer to the ELF image

size

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The handle is provided in the ELF Image Loaded notification event.

Since CUDA 6.0.

See also:

getElfImage

getLoadedFunctionInfo

CUDBGResult (*getErrorStringEx) (char *buf, uint32_t bufSz, uint32_t *msgSz)

Fills a user-provided buffer with an error message encoded as a null-terminated ASCII string.

Parameters

buf
- the destination buffer

bufSz
- the size of the destination buffer in bytes

msgSz
- the size of the written error message including the terminating null character.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The error message is specific to the last failed API call and is invalidated after every API method call except this one. It's possible to query the size of the error message without reading it by passing 0 as `buf` and `bufSz` parameters. The `msgSz` parameter is optional unless 0 as passed in as `buf` and `bufSz`. CUDBG_ERROR_BUFFER_TOO_SMALL is returned when the passed in buffer is too small to contain the message.

Since CUDA 12.2.

CUDBGResult (*getGridAttribute) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGAttribute attr, uint64_t *value)

Get the value of a grid attribute.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

attr
- the attribute

value
- the returned value of the attribute

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_ATTRIBUTE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttributes

CUDBGResult (*getGridAttributes) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGAttributeValuePair *pairs, uint32_t numPairs)

Get several grid attribute values in a single API call.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

pairs
- array of attribute/value pairs

numPairs
- the number of attribute/values pairs in the array

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_ATTRIBUTE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

See CUDBGAttribute for the list of available attributes.

Since CUDA 3.1.

See also:

getGridAttribute

CUDBGResult (*getGridDim) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *gridDim)

Get the dimensions in blocks of the given grid.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridDim
- the dimensions of the grid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

getBlockDim

getClusterDim

CUDBGResult (*getGridDim32) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *gridDim)

Get the dimensions of the given grid.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridDim
- the returned number of blocks in the grid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridDim but doesn't return the z dimension.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use getGridDim instead.

See also:

getGridDim

CUDBGResult (*getGridInfo) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.7.

CUDBGResult (*getGridInfo120) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo120 *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use getGridInfo instead.

See also:

getGridInfo

CUDBGResult (*getGridInfo55) (uint32_t dev, uint64_t gridId64, CUDBGGridInfo55 *gridInfo)

Get information about the specified grid.

Parameters

dev
- device index

gridId64

gridInfo
- pointer to a client allocated structure in which grid info will be returned

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridInfo, but returns less information. Returns CUDBG_ERROR_INVALID_GRID if the context of the grid has already been destroyed (even if grid ID itself is correct).

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 12.0: Use getGridInfo instead.

See also:

getGridInfo

CUDBGResult (*getGridStatus) (uint32_t dev, uint64_t gridId64, CUDBGGridStatus *status)

Check whether the grid corresponding to the ID is still present on the device.

Parameters

dev
- device index

gridId64
- 64-bit grid ID

status
- enum indicating the grid status

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 5.5.

CUDBGResult (*getGridStatus50) (uint32_t dev, uint32_t gridId, CUDBGGridStatus *status)

Check whether the grid corresponding to the ID is still present on the device.

Parameters

dev
- device index

gridId
- grid ID

status
- enum indicating the grid status

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getGridStatus, but takes a 32-bit grid ID instead of a 64-bit one.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getGridStatus instead.

See also:

getGridStatus

CUDBGResult (*getHardwareBarrierInfo) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CUDBGBarrierScope *scope, char *buf, uint32_t bufSz, uint32_t *msgSz)

Get hardware barrier information.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

scope
- barrier scope

buf
- buffer for the barrier information

bufSz
- buffer size

msgSz
- error message size with null character, can be null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.1.

CUDBGResult (*getHostAddrFromDeviceAddr) (uint32_t dev, uint64_t device_addr, uint64_t *host_addr)

Given a device virtual address, return a corresponding system memory virtual address.

Parameters

dev
- device index

device_addr
- device memory address

host_addr
- returned system memory address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_SEGMENT, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.1.

See also:

readGenericMemory

writeGenericMemory

CUDBGResult (*getLoadedFunctionInfo) (uint32_t dev, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t startIndex, uint32_t numEntries)

Get the section number and address of loaded functions for a given module.

Parameters

dev
- device index

handle
- ELF/cubin image handle

info
- information about loaded functions

startIndex
- start index of the entries to get

numEntries
- number of function load entries to read

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_LAZY_FUNCTION_LOADING capability is enabled, CUDA loads functions lazily after the module has been reported. This method could be used to get the lazily loaded functions.

Since CUDA 12.3.

CUDBGResult (*getLoadedFunctionInfo118) (uint32_t dev, uint64_t handle, CUDBGLoadedFunctionInfo *info, uint32_t numEntries)

Get the section number and address of loaded functions for a given module.

Parameters

dev
- device index

handle
- ELF/cubin image handle

info
- information about loaded functions

numEntries
- number of function load entries to read

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like getLoadedFunctionInfo but doesn't allow querying a subset of all lazily loaded functions.

Since CUDA 11.8.

Note:

DEPRECATED in CUDA 12.3: Use getLoadedFunctionInfo instead.

See also:

getLoadedFunctionInfo

CUDBGResult (*getManagedMemoryRegionInfo) (uint64_t startAddress, CUDBGMemoryInfo *memoryInfo, uint32_t memoryInfo_size, uint32_t *numEntries)

Get a sorted list of managed memory regions.

Parameters

startAddress
- the address that the first region in the list must contain

memoryInfo
- client-allocated array of memory region records of type CUDBGMemoryInfo

memoryInfo_size
- number of records of type CUDBGMemoryInfo that memoryInfo can hold

numEntries
- pointer to a client-allocated variable holding the number of valid entries returned in memoryInfo

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The sorted list of memory regions starts from a region containing the specified starting address. If the starting address is set to 0, a sorted list of managed memory regions is returned which starts from the managed memory region with the lowest start address.

Since CUDA 6.0.

CUDBGResult (*getNextAsyncEvent50) (CUDBGEvent50 *event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

Parameters

event
- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextAsyncEvent55) (CUDBGEvent55 *event)

Copies the next available event in the asynchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for ASYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 6.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextEvent) (CUDBGEventQueueType type, CUDBGEvent *event)

Copies the next available event into 'event' and removes it from the queue.

Parameters

type
- application event queue type

event
- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE

CUDBG_ERROR_NO_EVENT_AVAILABLE is returned if the queue is empty. ASYNC and SYNC queues are separate and each one is ordered separately, but it's impossible to find out the relative order of ASYNC and SYNC events.

Since CUDA 6.0.

See also:

acknowledgeSyncEvents

setNotifyNewEventCallback

CUDBGResult (*getNextEvent30) (CUDBGEvent30 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event
- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextEvent32) (CUDBGEvent32 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextEvent42) (CUDBGEvent42 *event)

Copies the next available event in the event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 4.0.

Note:

DEPRECATED in CUDA 5.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextSyncEvent50) (CUDBGEvent50 *event)

Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 5.5: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNextSyncEvent55) (CUDBGEvent55 *event)

Copies the next available event in the synchronous event queue into 'event' and removes it from the queue.

Parameters

event

- pointer to an event container where to copy the event parameters

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_EVENT_AVAILABLE, CUDBG_ERROR_INVALID_CONTEXT

Behaves like getNextEvent but only for SYNC events and doesn't support the latest event struct format so some fields won't be available.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 6.0: Use getNextEvent instead.

See also:

getNextEvent

CUDBGResult (*getNumDevices) (uint32_t *numDev)

Get the number of installed CUDA devices.

Parameters

numDev

- the returned number of devices

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session.

Since CUDA 3.0.

See also:

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*getNumLanes) (uint32_t dev, uint32_t *numLanes)

Get the number of lanes per warp on the device.

Parameters

dev
- device index

numLanes
- the returned number of lanes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*getNumPredicates) (uint32_t dev, uint32_t *numPredicates)

Get the number of predicate registers per lane on the device.

Parameters

dev
- device index

numPredicates
- the returned number of predicate registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 6.5.

See also:

getNumDevices

getNumLanes

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*getNumRegisters) (uint32_t dev, uint32_t *numRegs)

Get the maximum number of registers per lane on the device.

Parameters

dev
- device index

numRegs
- the returned number of registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device. Note that the actual number of registers can change per warp, use readWarpResources() to query that number dynamically.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

readWarpResources

CUDBGResult (*getNumSMs) (uint32_t dev, uint32_t *numSMs)

Get the total number of SMs on the device.

Parameters

dev
- device index

numSMs
- the returned number of SMs

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumUniformPredicates

getNumUniformRegisters

getNumWarps

CUDBGResult (*getNumUniformPredicates) (uint32_t dev, uint32_t *numPredicates)

Get the number of uniform predicate registers per warp on the device.

Parameters

dev
- device index

numPredicates
- the returned number of uniform predicate registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformRegisters

getNumWarps

CUDBGResult (*getNumUniformRegisters) (uint32_t dev, uint32_t *numRegs)

Get the number of uniform registers per warp on the device.

Parameters

dev
- device index

numRegs
- the returned number of uniform registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 10.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumWarps

CUDBGResult (*getNumWarps) (uint32_t dev, uint32_t *numWarps)

Get the number of warps per SM on the device.

Parameters

dev
- device index

numWarps
- the returned number of warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getNumDevices

getNumLanes

getNumPredicates

getNumRegisters

getNumSMs

getNumUniformPredicates

getNumUniformRegisters

CUDBGResult (*getPhysicalRegister30) (uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass)

Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

Parameters

pc
- Program counter

reg
- virtual register index

buf
- physical register name(s)

sz
- the physical register name buffer size

numPhysRegs
- number of physical register names returned

regClass
- the class of the physical registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.1: Do not use.

CUDBGResult (*getPhysicalRegister40) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t pc, char *reg, uint32_t *buf, uint32_t sz, uint32_t *numPhysRegs, CUDBGRegClass *regClass)

Get the physical register number(s) assigned to a virtual register name at a given PC, if it's live at that PC.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

pc
- Program counter

reg
- virtual register index

buf
- physical register name(s)

sz
- the physical register name buffer size

numPhysRegs
- number of physical register names returned

regClass
- the class of the physical registers

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Instead, the PTX to SASS mappings can be read from the cubin directly.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.1: Do not use.

CUDBGResult (*getSmType) (uint32_t dev, char *buf, uint32_t sz)

Get the SM type of the device.

Parameters

dev
- device index

buf
- the destination buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_BUFFER_TOO_SMALL, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Returns CUDBG_ERROR_BUFFER_TOO_SMALL if the provided buffer is not large enough. This value is constant within a single session for a given device.

Since CUDA 3.0.

See also:

getDeviceName

getDeviceType

CUDBGResult (*getSupportedDebuggerCapabilities) (CUDBGCapabilityFlags *capabilities)

Returns debug agent capabilities that are supported by this version of the API.

Parameters

capabilities
- returned debug engine capabilities

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

This API method can be called without initializing the API.

Since CUDA 12.5.

CUDBGResult (*getTID) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *tid)

Get the ID of the Linux thread hosting the CUDA context active at the given coordinates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

tid
- the returned thread id

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This returns a Linux thread ID.

Since CUDA 3.0.

See also:

getGridAttributes

CUDBGResult (*getWarpHitBreakpoint) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGBreakpointHandle *handle)

Get the handle of the breakpoint that the given warp hit.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

handle
- the returned breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

An error is returned if the warp did not hit a breakpoint. Use readBrokenWarps() to check if the warp is broken before calling this method. Some breakpoint handles are special, see the documentation of CUDBGBreakpointHandle for more details.

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

insertBreakpoint

isBreakpointEnabled

readBrokenWarps

removeBreakpoint

CUDBGResult (*initialize) ()

Initialize the API.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_NO_DEVICE_AVAILABLE

setNotifyNewEventCallback() and getSupportedDebuggerCapabilities() can be called before initialize(). If no CUDA devices are detected on the system, CUDBG_ERROR_NO_DEVICE_AVAILABLE is returned.

Since CUDA 3.0.

See also:

finalize

CUDBGResult (*initializeAttachStub) ()

Initialize the attach stub.

Returns

CUDBG_SUCCESS

This is no longer necessary starting with driver version r590.

Since CUDA 5.0.

CUDBGResult (*insertBreakpoint) (uint32_t dev, uint64_t addr, CUDBGBreakpointHandle *handle)

Set a breakpoint at the given instruction address for the given device.

Parameters

dev
- the device index

addr
- instruction address

handle
- the returned breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_BREAKPOINT_STATE_CONFLICT

Before setting a breakpoint, getAdjustedCodeAddress() should be called to get the adjusted breakpoint address. The returned handle can be used to enable/disable/remove the breakpoint.

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

isBreakpointEnabled

removeBreakpoint

CUDBGResult (*isBreakpointEnabled) (CUDBGBreakpointHandle handle, uint32_t *enabled)

Check if a breakpoint specified by its handle is enabled.

Parameters

handle
- the breakpoint handle

enabled
- whether the breakpoint is enabled

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

removeBreakpoint

CUDBGResult (*isDeviceCodeAddress) (uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Parameters

addr
- virtual address

isDeviceAddress
- true if address resides within device code

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

CUDBGResult (*isDeviceCodeAddress55) (uintptr_t addr, bool *isDeviceAddress)

Determine whether a virtual address resides within device code.

Parameters

addr
- virtual address

isDeviceAddress
- true if address resides within device code

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves exactly like isDeviceCodeAddress.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 6.0: Use isDeviceCodeAddress instead.

See also:

isDeviceCodeAddress

CUDBGResult (*lookupDeviceCodeSymbol) (char *symName, bool *symFound, uintptr_t *symAddr)

Determines whether a symbol represents a function in device code and returns its virtual address.

Parameters

symName
- symbol name

symFound
- set to true if the symbol is found

symAddr
- the symbol virtual address if found

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

CUDBGResult (*memcheckReadErrorAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *address, ptxStorageKind *storage)

Get the address that memcheck detected an error on.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

address
- returned address detected by memcheck

storage
- returned address class of address

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*readActiveLanes) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *activeLanesMask)

Read the lane bitmask of active threads on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

activeLanesMask
- the returned bitmask of active threads

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*readAllVirtualReturnAddresses) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *addrs, uint32_t numAddrs, uint32_t *callDepth, uint32_t *syscallCallDepth)

Read all the virtual return addresses for a thread (the full backtrace).

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addrs
- the returned addresses array

numAddrs
- number of elements in addrs array

callDepth
- the returned call depth

syscallCallDepth
- the returned syscall call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that syscallCallDepth is always set to 0.

Since CUDA 12.5.

See also:

readCallDepth

readReturnAddress

CUDBGResult (*readBlockIdx) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *blockIdx)

Read the CUDA block index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- the returned CUDA block index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readActiveLanes

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*readBlockIdx32) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim2 *blockIdx)

Read the two-dimensional CUDA block index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

blockIdx
- the returned CUDA block index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readBlockIdx but doesn't return the z dimension.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 4.0: Use readBlockIdx instead.

See also:

readBlockIdx

CUDBGResult (*readBrokenWarps) (uint32_t dev, uint32_t sm, uint64_t *brokenWarpsMask)

Read the bitmask of warps that are at a breakpoint on a given SM.

Parameters

dev
- device index

sm
- SM index

brokenWarpsMask
- the returned bitmask of broken warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*readCallDepth) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth (number of calls) for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readReturnAddress

readVirtualReturnAddress

CUDBGResult (*readCallDepth32) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *depth)

Read the call depth (number of calls) for a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readCallDepth() for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readCallDepth instead.

See also:

readCallDepth

CUDBGResult (*readCCRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *val)

Read the hardware CC register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

val
- the returned value of the CC register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 13.1: Do not use.

CUDBGResult (*readClusterIdx) (uint32_t dev, uint32_t sm, uint32_t wp, CuDim3 *clusterIdx)

Read the CUDA cluster index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

clusterIdx
- the returned CUDA cluster index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*readCodeMemory) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the code memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

It is generally not necessary to call this function - instead, the same memory could be read from the ELF module images received from the API.

Since CUDA 3.0.

See also:

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readConstMemory129) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the constant memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves exactly like readGlobalMemory.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 13.0: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*readCPUCallStack) (uint32_t dev, uint64_t gridId64, uint64_t *addrs, uint32_t numAddrs, uint32_t *totalNumAddrs)

Read the CPU call stack captured at the time of kernel launch.

Parameters

dev
- device index

gridId64
- 64-bit grid ID

addrs
- the returned addresses array, can be NULL

numAddrs
- capacity of addrs (possibly 0)

totalNumAddrs
- the actual size of the stack (number of frames) is written here; the value written can be greater than numAddrs

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method only works if the CUDBG_DEBUGGER_CAPABILITY_COLLECT_CPU_CALL_STACK_FOR_KERNEL_LAUNCHES capability is enabled.

Since CUDA 12.9.

CUDBGResult (*readDeviceExceptionState) (uint32_t dev, uint64_t *mask, uint32_t numWords)

Get the exception state of the SMs on the device.

Parameters

dev
- device index

mask
- Arbitrarily sized bit field containing a 1 at (1 << i) if SM i hit an exception

numWords
- Number of uint64_t elements in `mask` (must be large enough to hold a bit for each sm on the device)

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 9.0.

See also:

getNumSMs

CUDBGResult (*readDeviceExceptionState80) (uint32_t dev, uint64_t *exceptionSMMask)

Get the exception state of the SMs on the device.

Parameters

dev
- device index

exceptionSMMask
- Bit field containing a 1 at (1 << i) if SM i hit an exception

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readDeviceExceptionState but only supports up to 64 SMs.

Since CUDA 5.5.

Note:

DEPRECATED in CUDA 9.0: Use readDeviceExceptionState instead.

See also:

readDeviceExceptionState

CUDBGResult (*readErrorPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *errorPC, bool *errorPCValid)

Get the hardware reported error PC if it exists.

Parameters

dev
- device index

sm
- SM index

wp

errorPC
- PC ofthe exception

errorPCValid
- boolean to indicate that the returned error PC is valid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The error PC, if available, shows the PC where an error happened (the thread can progress past that so its PC could be beyond that).

Since CUDA 6.0.

CUDBGResult (*readGenericMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at an address in any memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

The address will be used to determine whether the read is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

readCodeMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readGlobalMemory) (uint64_t addr, void *buf, uint32_t sz)

Read content at an address in the global address space.

Parameters

addr
- memory address

buf
- buffer

sz

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

If the address is valid on more than one device and one of those devices does not support UVA, an error is returned.

Since CUDA 6.0.

See also:

readCodeMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readGlobalMemory31) (uint32_t dev, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Parameters

dev
- device index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves like readGenericMemory() with sm, wp, ln == 0. This makes this method not at all useful.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*readGlobalMemory55) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the global memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Behaves exactly like readGenericMemory().

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 6.0: Use readGlobalMemory instead.

See also:

readGlobalMemory

CUDBGResult (*readGridId) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *gridId64)

Read the 64-bit CUDA grid index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridId64

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The grid ID is guaranteed to be unique within a device, but not globally.

Since CUDA 5.5.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readThreadIdx

readValidLanes

readValidWarps

CUDBGResult (*readGridId50) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *gridId)

Read the CUDA grid index running on a valid warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

gridId
- the returned CUDA grid index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readGridId but truncates the grid ID to 32bit. This is incompatible with some grid IDs like those used by the OptiX applications.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 5.5: Use readGridId instead.

See also:

readGridId

CUDBGResult (*readLaneException) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CUDBGException_t *exception)

Read the exception type for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

exception
- the returned exception type

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.1.

CUDBGResult (*readLaneStatus) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, bool *error)

Read the status of the given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

error
- true if there is an error

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

For specific error values, use readLaneException.

Since CUDA 3.0.

CUDBGResult (*readLocalMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the local memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readParamMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the param memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC offset on the given active thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

pc
- the returned PC

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The returned PC offset is from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readParamMemory

readRegister

readSharedMemory

readTextureMemory

readVirtualPC

CUDBGResult (*readPinnedMemory) (uint64_t addr, void *buf, uint32_t sz)

Read content at pinned address in system memory.

Parameters

addr

- system memory address

buf

- buffer

sz

- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_MEMORY_MAPPING_FAILED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL

Depending on the platform, this method may fail and a platform-specific CPU RAM way of reading memory from the debuggee must be used (e.g. ptrace).

Since CUDA 3.2.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, uint32_t *predicates)

Read content of hardware predicate registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

predicates_size
- number of predicate registers to read

predicates
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 6.5.

See also:

readCodeMemory

readGenericMemory

readGlobalMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t *val)

Read content of a hardware register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

regno
- register index

val
- the returned value of the register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that warps can dynamically change the number of used registers at runtime, readWarpResources() could be used to query that.

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readSharedMemory

readTextureMemory

CUDBGResult (*readRegisterRange) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t numRegisters, uint32_t *registers, uint32_t *numRegistersRead)

Read content of a hardware range of hardware registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

index
- index of the first register to read

numRegisters
- number of registers to read

registers
- buffer

numRegistersRead
- number of registers actually read, ignored if null

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readSharedMemory

readTextureMemory

CUDBGResult (*readRegisterRange60) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t index, uint32_t registers_size, uint32_t *registers)

Read content of a range of hardware registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

index
- index of the first register to read

registers_size
- number of registers to read

registers
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 6.0.

Note:

DEPRECATED in CUDA 13.2: Use readRegisterRange instead.

See also:

readRegisterRange

CUDBGResult (*readReturnAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

level
- the specified call level

ra
- the returned return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The returned return address is an offset from the start of the current function. If a function can't be found, the full virtual address is returned.

Since CUDA 4.0.

See also:

readCallDepth

readVirtualReturnAddress

CUDBGResult (*readReturnAddress32) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the return address (offset) for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

level
- the specified call level

ra
- the returned return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readReturnAddress() for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readReturnAddress instead.

See also:

readReturnAddress

CUDBGResult (*readSharedMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, void *buf, uint32_t sz)

Read content at address in the shared memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- memory address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readCodeMemory

readGenericMemory

readLocalMemory

readPC

readParamMemory

readRegister

readTextureMemory

CUDBGResult (*readSmException) (uint32_t dev, uint32_t sm, CUDBGException_t *exception, uint64_t *errorPC, bool *errorPCValid)

Get the SM exception status if it exists.

Parameters

dev
- the device index

sm
- the SM index

exception
- returned exception

errorPC
- returned PC of the exception

errorPCValid
- boolean to indicate that the returned error PC is valid

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

If the CUDBG_DEBUGGER_CAPABILITY_REPORT_EXCEPTIONS_IN_EXITED_WARPS capability is enabled, exceptions in exited warps will be reported.

Since CUDA 12.5.

CUDBGResult (*readSyscallCallDepth) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t *depth)

Read the call depth of syscalls for a given thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

depth
- the returned call depth

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Will always return 0.

Since CUDA 4.1.

Note:

DEPRECATED in CUDA 12.9: Do not use.

CUDBGResult (*readTextureMemory) (uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t id, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Parameters

dev
- device index

vsm

wp
- warp index

id
- texture id (the value of DW_AT_location attribute in the relocated ELF image)

dim
- texture dimension (1 to 4)

coords
- array of coordinates of size dim

buf
- result buffer

sz
- buffer size in bytes

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.0.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*readTextureMemoryBindless) (uint32_t dev, uint32_t vsm, uint32_t wp, uint32_t texSymtabIndex, uint32_t dim, uint32_t *coords, void *buf, uint32_t sz)

This method is no longer supported since CUDA 12.0.

Parameters

dev
- device index

vsm

wp
- warp index

texSymtabIndex
- global symbol table index of the texture symbol

dim
- texture dimension (1 to 4)

coords
- array of coordinates of size dim

buf
- result buffer

sz
- buffer size in bytes

Returns

CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Will always return CUDBG_ERROR_NOT_SUPPORTED.

Since CUDA 4.2.

Note:

DEPRECATED in CUDA 12.0: Do not use.

CUDBGResult (*readThreadIdx) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, CuDim3 *threadIdx)

Read the CUDA thread index running on valid thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

threadIdx
- the returned CUDA thread index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readValidLanes

readValidWarps

CUDBGResult (*readUniformPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, uint32_t *predicates)

Read contents of uniform predicate registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

predicates_size
- number of predicate registers to read

predicates
- buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

readPredicates

CUDBGResult (*readUniformRegisterRange) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t registers_size, uint32_t *registers)

Read a range of uniform registers.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

regno
- starting index into uniform register file

registers_size
- number of bytes to read

registers
- pointer to buffer

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

readRegister

CUDBGResult (*readValidLanes) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t *validLanesMask)

Read the lane bitmask of valid threads on a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

validLanesMask
- the returned bitmask of valid threads

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidWarps

CUDBGResult (*readValidWarps) (uint32_t dev, uint32_t sm, uint64_t *validWarpsMask)

Read the bitmask of valid warps on a given SM.

Parameters

dev
- device index

sm
- SM index

validWarpsMask
- the returned bitmask of valid warps

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readActiveLanes

readBlockIdx

readBrokenWarps

readGridId

readThreadIdx

readValidLanes

CUDBGResult (*readVirtualPC) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t *pc)

Read the PC (virtual address) on the given active thread.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

pc
- the returned PC

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

readPC

CUDBGResult (*readVirtualReturnAddress) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

level
- the specified call level

ra
- the returned virtual return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 4.0.

See also:

readCallDepth

readReturnAddress

CUDBGResult (*readVirtualReturnAddress32) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t level, uint64_t *ra)

Read the virtual return address for a call level.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

level
- the specified call level

ra
- the returned virtual return address for level

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_LANE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CALL_LEVEL, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readVirtualReturnAddress for the active thread group.

Since CUDA 3.1.

Note:

DEPRECATED in CUDA 4.0: Use readVirtualReturnAddress instead.

See also:

readVirtualReturnAddress

CUDBGResult (*readWarpResources) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpResources *resources)

Get the resources assigned to a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

resources
- pointer to structure that contains warp resources

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Note that these resources can change between suspends, which makes this method useful for avoiding warp data access errors.

Since CUDA 12.8.

CUDBGResult (*readWarpState) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 12.9.

CUDBGResult (*readWarpState120) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState120 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP,

CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.0.

Note:

DEPRECATED in CUDA 12.7: Use readWarpState instead.

See also:

readWarpState

CUDBGResult (*readWarpState127) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState127 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 12.7.

Note:

DEPRECATED in CUDA 12.9: Use readWarpState instead.

See also:

readWarpState

CUDBGResult (*readWarpState60) (uint32_t dev, uint32_t sm, uint32_t wp, CUDBGWarpState60 *state)

Read the state of a given warp.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

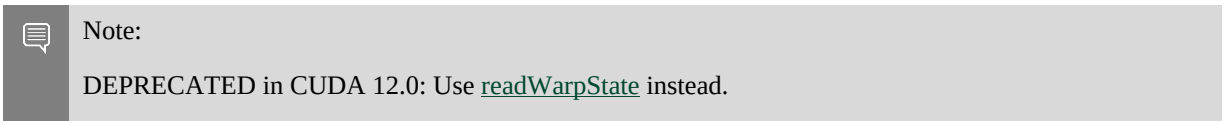
state
- pointer to structure that contains warp state

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_GRID, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like readWarpState but returns fewer fields.

Since CUDA 6.0.



See also:

readWarpState

CUDBGResult (*removeBreakpoint) (CUDBGBreakpointHandle handle)

Remove a breakpoint specified by its handle.

Parameters

handle
- the breakpoint handle

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

disableBreakpoint

enableBreakpoint

getWarpHitBreakpoint

insertBreakpoint

isBreakpointEnabled

CUDBGResult (*requestCleanupOnDetach) (uint32_t appResumeFlag)

Request for cleanup of driver state when detaching.

Parameters

appResumeFlag

- value of CUDBG_RESUME_FOR_ATTACH_DETACH as read from the application's process space.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Needs to be conditionally called by the client depending on the state of the debugged application. See the "Attaching and Detaching" section for more information.

Since CUDA 6.0.

CUDBGResult (*requestCleanupOnDetach55) ()

Request for cleanup of driver state when detaching.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Needs to be conditionally called by the client depending on the state of the debugged application. See the "Attaching and Detaching" section for more information.

Since CUDA 5.0.

Note:

DEPRECATED in CUDA 6.0: Use requestCleanupOnDetach instead.

See also:

requestCleanupOnDetach

CUDBGResult (*resumeAllDevices) ()

Resume all running CUDA devices.

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 13.2.

See also:

suspendAllDevices

CUDBGResult (*resumeDevice) (uint32_t dev)

Resume a suspended CUDA device.

Parameters

dev

- device index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Using this method is discouraged, use resumeAllDevices() instead to avoid race conditions. This method has no effect if the device is already running.

Since CUDA 3.0.

See also:

resumeAllDevices

CUDBGResult (*resumeWarpsUntilPC) (uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t pc, uint32_t flags)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Parameters

dev
- device index

sm
- the SM index

warpMask
- the bitmask of warps to resume (1 = resume, 0 = do not resume)

pc

flags
- flags of type CUDBGSingleStepFlags to change the stepping behavior

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Compared to resumeDevice(), this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is

hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using singleStepWarp() or resumeDevice().

Since CUDA 13.2.

See also:

resumeAllDevices

singleStepWarp

CUDBGResult (*resumeWarpsUntilPC60) (uint32_t dev, uint32_t sm, uint64_t warpMask, uint64_t virtPC)

Insert a temporary breakpoint at the specified virtual PC and resume all warps in the specified bitmask on a given SM.

Parameters

dev
- device index

sm
- SM index

warpMask
- the bitmask of warps to resume (1 = resume, 0 = do not resume)

virtPC
- the virtual PC where the temporary breakpoint will be inserted

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Compared to resumeDevice(), this method provides finer-grain control by resuming a selected set of warps on the same SM. The main intended usage is to accelerate the single-stepping process when the target PC is known in advance. Instead of single-stepping each warp individually until the target PC is hit, the client can use this method. If an unsteppable barrier is hit by the resumed warps, this method returns early (before reaching the target PC). When this method is used, errors within CUDA kernels will no longer be reported precisely. In the situation where resuming warps is not possible, this method

will return CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE. The client should then fall back to using singleStepWarp() or resumeDevice().

Since CUDA 6.0.

Note:

DEPRECATED in CUDA 13.2: Use resumeWarpsUntilPC instead.

See also:

resumeWarpsUntilPC

CUDBGResult (*setBreakpoint) (uint32_t dev, uint64_t addr)

Set a breakpoint at the given instruction address for the given device.

Parameters

dev
- device index

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Before setting a breakpoint, getAdjustedCodeAddress() should be called to get the adjusted breakpoint address.

Since CUDA 3.2.

See also:

unsetBreakpoint

CUDBGResult (*setBreakpoint31) (uint64_t addr)

Set a breakpoint at the given instruction address.

Parameters

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use setBreakpoint instead.

See also:

setBreakpoint

CUDBGResult (*setKernelLaunchNotificationMode)(CUDBGKernelLaunchNotifyMode mode)

Set the launch notification policy.

Parameters

mode

- mode to deliver kernel launch notifications in

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If mode is CUDBG_KNL_LAUNCH_NOTIFY_EVENT, enable synchronous launch notification reporting (via events). This can noticeably slow down the execution of the application. If mode is CUDBG_KNL_LAUNCH_NOTIFY_DEFER, the launch notifications are not reported at all.

Since CUDA 5.5.

CUDBGResult (*setNotifyNewEventCallback) (CUDBGNotifyNewEventCallback callback, void *userData)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

userData

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The callback function is called for every ASYNC and SYNC event. The callback function is always called on the same thread. No API methods can be called from that thread except getNextEvent() and acknowledgeSyncEvents() (and their deprecated variants), otherwise CUDBG_ERROR_RECURSIVE_API_CALL will be returned.

Since CUDA 13.0.

See also:

acknowledgeSyncEvents

getNextEvent

CUDBGResult (*setNotifyNewEventCallback31) (CUDBGNotifyNewEventCallback31 callback, void *data)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

data
- a pointer to be passed to the callback when called

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't return the host thread ID from which the event originates.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

CUDBGResult (*setNotifyNewEventCallback40) (CUDBGNotifyNewEventCallback40 callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback

- the callback function

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer.

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 4.1: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

CUDBGResult (*setNotifyNewEventCallback41) (CUDBGNotifyNewEventCallback41 callback)

Provides the API with the function to call to notify the debugger of a new application or device event.

Parameters

callback
- the callback function

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like setNotifyNewEventCallback but doesn't allow passing in the user data pointer. The timeout field is always 0.

Since CUDA 4.1.

Note:

DEPRECATED in CUDA 13.0: Use setNotifyNewEventCallback instead.

See also:

setNotifyNewEventCallback

CUDBGResult (*singleStepWarp) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t laneHint, uint32_t nsteps, uint32_t flags, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

laneHint
- focused lane (~0 to let the API decide)

nsteps
- number of single steps

flags
- flags of type CUDBGSingleStepFlags to change the stepping behavior

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

By default, if the warp is on a convergence barrier, resumeWarpsUntilPC is called internally to quickly advance the warp past that barrier. If the CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag is passed in, this optimization is not performed (which would likely lead to diverged threads becoming focused and starting to advance towards the convergence barrier). If a warp is on a block-wide barrier (or wider), other warps required to advance past the barrier are automatically resumed. The output parameter warpMask will have the warps resumed in the current SM. Warps can also be resumed in other SMs, but are not reported via the API. This method is synchronous and will not return until the step is complete.

Since CUDA 12.4.

See also:

resumeAllDevices

resumeWarpsUntilPC

suspendAllDevices

CUDBGResult (*singleStepWarp40) (uint32_t dev, uint32_t sm, uint32_t wp)

Single step an individual warp on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp41 without the output warpMask parameter.

Since CUDA 3.0.

See also:

singleStepWarp

CUDBGResult (*singleStepWarp41) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t *warpMask)

Single step an individual warp on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS,

CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp65 with nsteps set to 1.

Since CUDA 4.1.

Note:

DEPRECATED in CUDA 6.5: Use singleStepWarp instead.

See also:

singleStepWarp

CUDBGResult (*singleStepWarp65) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t nsteps, uint64_t *warpMask)

Single step an individual warp nsteps times on a suspended CUDA device.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

nsteps
- number of single steps

warpMask
- the warps that have been single-stepped

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN_FUNCTION, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_RUNNING_DEVICE, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_WARP_RESUME_NOT_POSSIBLE, CUDBG_ERROR_INVALID_WARP_MASK, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like singleStepWarp with no lane hint and the CUDBG_SINGLE_STEP_FLAGS_NO_STEP_OVER_WARP_BARRIERS flag set.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 12.4: Use singleStepWarp instead.

See also:

singleStepWarp

CUDBGResult (*suspendAllDevices) (uint32_t nonBlocking)

Suspend all running CUDA devices.

Parameters

nonBlocking

- whether or not asynchronous operation is desired

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_SUSPENDED_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

If the nonBlocking flag is non-zero, the function returns immediately and sends CUDBG_EVENT_ALL_DEVICES_SUSPENDED when the operation finishes in the background. Otherwise, if the function returns with CUDBG_SUCCESS, that guarantees that all devices have been suspended.

Since CUDA 13.2.

See also:

resumeAllDevices

CUDBGResult (*suspendDevice) (uint32_t dev)

Suspends a running CUDA device.

Parameters

dev

- device index

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_SUSPENDED_DEVICE, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Using this method is discouraged, use suspendAllDevices() instead to avoid race conditions. The device has to be suspended in order to execute most operations on it. CUDBG_ERROR_SUSPENDED_DEVICE is returned if the device is already suspended.

Since CUDA 3.0.

See also:

suspendAllDevices

CUDBGResult (*unsetBreakpoint) (uint32_t dev, uint64_t addr)

Unset a breakpoint at the given instruction address for the given device.

Parameters

dev
- device index

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.2.

See also:

setBreakpoint

CUDBGResult (*unsetBreakpoint31) (uint64_t addr)

Unset a breakpoint at the given instruction address.

Parameters

addr
- instruction address

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

Behaves like unsetBreakpoint but tries to automatically find a device for the given address.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use unsetBreakpoint instead.

See also:

unsetBreakpoint

CUDBGResult (*writeCCRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t val)

Write to the hardware CC register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

val
- the new value of the CC register

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_RECURSIVE_API_CALL

The CC register is no longer available in the supported hardware.

Since CUDA 6.5.

Note:

DEPRECATED in CUDA 13.1: Do not use.

CUDBGResult (*writeGenericMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in any memory segment.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

The address will be used to determine whether the write is to local, shared or global memory. The target address range should entirely reside within a single memory segment. Coordinate arguments are only used when relevant. They should be provided for the following segments:

- Shared memory: SM and Warp
- Local memory: SM, Warp and Lane

Since CUDA 6.0.

See also:

writeGlobalMemory

writeLocalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*writeGlobalMemory) (uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

It's not possible to access a shared memory page or an ambiguous address allocated on several devices that don't support UVA.

Since CUDA 6.0.

See also:

writeGenericMemory

writeLocalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*writeGlobalMemory31) (uint32_t dev, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

dev
- device index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

This method is unsupported on Hopper and later architectures. Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.0.

Note:

DEPRECATED in CUDA 3.2: Use writeGlobalMemory instead.

See also:

writeGlobalMemory

CUDBGResult (*writeGlobalMemory55) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in global memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL, CUDBG_ERROR_NOT_SUPPORTED

Use newer methods: writeGlobalMemory or writeGenericMemory.

Since CUDA 3.2.

Note:

DEPRECATED in CUDA 6.0: Use writeGlobalMemory instead.

See also:

writeGlobalMemory

CUDBGResult (*writeLocalMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in local memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INVALID_ADDRESS, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within local memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeParamMemory

writeSharedMemory

CUDBGResult (*writeParamMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in param memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within param memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeLocalMemory

writeSharedMemory

CUDBGResult (*writePinnedMemory) (uint64_t addr, const void *buf, uint32_t sz)

Write to a pinned memory address.

Parameters

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_UNKNOWN, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_MEMORY_MAPPING_FAILED, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_ADDRESS_NOT_IN_DEVICE_MEM, CUDBG_ERROR_AMBIGUOUS_MEMORY_ADDRESS, CUDBG_ERROR_RECURSIVE_API_CALL

It's not possible to access an ambiguous access allocated on several devices that don't support UVA.

Since CUDA 3.2.

See also:

readPinnedMemory

CUDBGResult (*writePredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware predicates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

predicates_size
- predicates count

predicates
- predicate values

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method writes to predicates_size predicates, starting from P0. Each predicate value must be either 0 or 1.

Since CUDA 6.5.

See also:

writeRegister

writeUniformPredicates

writeUniformRegister

CUDBGResult (*writeRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t ln, uint32_t regno, uint32_t val)

Write to a hardware register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

ln
- lane index

regno
- register number

val
- value

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 3.0.

See also:

writePredicates

writeUniformPredicates

writeUniformRegister

CUDBGResult (*writeSharedMemory) (uint32_t dev, uint32_t sm, uint32_t wp, uint64_t addr, const void *buf, uint32_t sz)

Write to an address in shared memory.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

addr
- address

buf
- buffer

sz
- buffer size in bytes

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INVALID_MEMORY_ACCESS, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

The destination address range must be within shared memory.

Since CUDA 3.0.

See also:

writeGenericMemory

writeGlobalMemory

writeLocalMemory

writeParamMemory

CUDBGResult (*writeUniformPredicates) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t predicates_size, const uint32_t *predicates)

Write to hardware uniform predicates.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

predicates_size
- predicates count

predicates
- predicate values

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

This method writes to predicates_size uniform predicates, starting from UP0. Each predicate value must be either 0 or 1.

Since CUDA 10.0.

See also:

writePredicates

writeRegister

writeUniformRegister

CUDBGResult (*writeUniformRegister) (uint32_t dev, uint32_t sm, uint32_t wp, uint32_t regno, uint32_t val)

Write to a hardware uniform register.

Parameters

dev
- device index

sm
- SM index

wp
- warp index

regno
- register number

val
- value

Returns

CUDBG_SUCCESS, CUDBG_ERROR_INVALID_ARGS, CUDBG_ERROR_UNINITIALIZED, CUDBG_ERROR_INTERNAL, CUDBG_ERROR_INVALID_WARP, CUDBG_ERROR_INITIALIZATION_FAILURE, CUDBG_ERROR_INVALID_CONTEXT, CUDBG_ERROR_RECURSIVE_API_CALL

Since CUDA 10.0.

See also:

writePredicates

writeRegister

writeUniformPredicates

4.2. CUDBGAttributeValuePair Struct Reference

Grid attribute-value pair.

CUDBGAttribute CUDBGAttributeValuePair::attribute

The attribute to query.

uint64_t CUDBGAttributeValuePair::value

The value of the attribute.

4.3. CUDBGCbuWarpState Struct Reference

Warp state in the CBU (Convergence Barrier Unit).

uint32_t CUDBGCbuWarpState::activeMask

Active threads mask.

uint32_t CUDBGCbuWarpState::barrierMasks

Convergence barrier participation masks.

uint32_t CUDBGCbuWarpState::collectiveMask

Mask of threads that are part of a collective operation.

uint32_t CUDBGCbuWarpState::exitedMask

Exited threads mask.

CUDBGCbuThreadState CUDBGCbuWarpState::threadState

Thread state for each warp lane.

4.4. CUDBGCudaLogMessage Struct Reference

CUDA Log Message.

CUDBGCudaLogLevel CUDBGCudaLogMessage::logLevel

The log severity level of the message.

char CUDBGCudaLogMessage::message

The log message string.

uint32_t CUDBGCudaLogMessage::osThreadId

The OS thread ID of the thread that generated the log message.

uint64_t CUDBGCudaLogMessage::unixTimestampNs

The timestamp the log message was received at, in nanoseconds since the Unix epoch.

4.5. CUDBGDeviceInfo Struct Reference

Device-level information. This is the first element in the deviceInfoBuffer, and is always present. getDeviceInfo() takes a deviceId as input, so no need to explicitly pass it back here. Only "valid & updated" SMs/Warps/Lanes are included in the buffer, which allows us to determine indexes without having to encode an explicit ID field in the following buffer datastructures.

uint32_t CUDBGDeviceInfo::deviceAttributeFlags

Bitmask of CUDBGDeviceInfoAttribute_t enums for a device.

CUDBGDeviceInfoQueryType_t CUDBGDeviceInfo::responseType

Response type.

4.6. CUDBGDeviceInfoSizes Struct Reference

Sizes of the various structs returned by the batched device update APIs. No explicit version field - implied by debugAPI major.minor.revision.

uint32_t CUDBGDeviceInfoSizes::deviceInfoAttributeSizes

Device info attribute sizes.

uint32_t CUDBGDeviceInfoSizes::deviceInfoSize

Device info size.

uint32_t CUDBGDeviceInfoSizes::laneInfoAttributeSizes

Lane (thread) info attribute sizes.

uint32_t CUDBGDeviceInfoSizes::laneInfoSize

Lane (thread) info size.

uint32_t CUDBGDeviceInfoSizes::requiredBufferSize

Required buffer size.

uint32_t CUDBGDeviceInfoSizes::smInfoAttributeSizes

SM info attribute sizes.

uint32_t CUDBGDeviceInfoSizes::smInfoSize

SM info size.

uint32_t CUDBGDeviceInfoSizes::warpInfoAttributeSizes

Warp info attribute sizes.

uint32_t CUDBGDeviceInfoSizes::warpInfoSize

Warp info size.

4.7. CUDBGEvent Struct Reference

Event information container.

union cases_st

Information for each type of event.

CUDBGEvent::cases

Information for each type of event.

4.8. CUDBGEvent30 Struct Reference

Event information container for API version 3.0.

Note:

DEPRECATED: Use CUDBGEvent instead. For documentation of individual fields, see the documentation of the CUDBGEvent struct instead.

See also:

CUDBGEvent

4.9. CUDBGEvent32 Struct Reference

Event information container for API version 3.2.

Note:

DEPRECATED: Use CUDBGEvent instead. For documentation of individual fields, see the documentation of the CUDBGEvent struct instead.

See also:

CUDBGEvent

4.10. CUDBGEvent42 Struct Reference

Event information container for API version 4.2.

Note:

DEPRECATED: Use CUDBGEvent instead. For documentation of individual fields, see the documentation of the CUDBGEvent struct instead.

See also:

CUDBGEvent

4.11. CUDBGEvent50 Struct Reference

Event information container for API version 5.0.

Note:

DEPRECATED: Use CUDBGEvent instead. For documentation of individual fields, see the documentation of the CUDBGEvent struct instead.

See also:

CUDBGEvent

4.12. CUDBGEvent55 Struct Reference

Event information container for API version 5.5.

Note:

DEPRECATED: Use CUDBGEvent instead. For documentation of individual fields, see the documentation of the CUDBGEvent struct instead.

See also:

CUDBGEvent

4.13. CUDBGEvent::cases_st Union Reference

Information for each type of event.

struct allDevicesSuspended_st

Information about an event which forced all devices to be suspended.

struct contextCreate_st

Information about the context being created.

struct contextDestroy_st

Information about the context being destroyed.

struct contextPop_st

Information about the context being popped.

struct contextPush_st

Information about the context being pushed.

struct elfImageLoaded_st

Information about the loaded ELF image.

struct elfImageUnloaded_st

Information about the ELF image about to be unloaded.

struct functionsLoaded_st

Information about the functions being lazily loaded.

struct internalError_st

Information about internal erros.

struct kernelFinished_st

Information about the kernel that just terminated.

struct kernelReady_st

Information about the kernel ready to be launched.

struct singleStepComplete_st

Information about a single step operation that has completed.

struct CUDBGEvent::cases_st::allDevicesSuspended_st
CUDBGEvent::cases_st::allDevicesSuspended

Information about an event which forced all devices to be suspended.

struct CUDBGEvent::cases_st::contextCreate_st
CUDBGEvent::cases_st::contextCreate

Information about the context being created.

struct CUDBGEvent::cases_st::contextDestroy_st
CUDBGEvent::cases_st::contextDestroy

Information about the context being destroyed.

struct CUDBGEvent::cases_st::contextPop_st
CUDBGEvent::cases_st::contextPop

Information about the context being popped.

struct CUDBGEvent::cases_st::contextPush_st
CUDBGEvent::cases_st::contextPush

Information about the context being pushed.

struct CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::cases_st::elfImageLoaded

Information about the loaded ELF image.

struct CUDBGEvent::cases_st::elfImageUnloaded_st
CUDBGEvent::cases_st::elfImageUnloaded

Information about the ELF image about to be unloaded.

struct CUDBGEvent::cases_st::functionsLoaded_st
CUDBGEvent::cases_st::functionsLoaded

Information about the functions being lazily loaded.

struct CUDBGEvent::cases_st::internalError_st
CUDBGEvent::cases_st::internalError

Information about internal erros.

struct CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::cases_st::kernelFinished

Information about the kernel that just terminated.

struct CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::cases_st::kernelReady

Information about the kernel ready to be launched.

struct CUDBGEvent::cases_st::singleStepComplete_st
CUDBGEvent::cases_st::singleStepComplete

Information about a single step operation that has completed.

4.14. CUDBGEvent::cases_st::allDevicesSuspended_st Struct Reference

Information about an event which forced all devices to be suspended.

uint64_t
CUDBGEvent::cases_st::allDevicesSuspended_st::brokenDevicesMa

Device bitmask. This mask has bits set for devices with any warps that hit a breakpoint.

uint64_t
CUDBGEvent::cases_st::allDevicesSuspended_st::faultedDevicesMa

Device bitmask. This mask has bits set for devices with any warps that hit an exception.

4.15. CUDBGEvent::cases_st::contextCreate_st Struct Reference

Information about the context being created.

uint64_t CUDBGEvent::cases_st::contextCreate_st::context

Context handle of the context being created.

uint32_t CUDBGEvent::cases_st::contextCreate_st::dev

Device index of the context.

uint32_t CUDBGEvent::cases_st::contextCreate_st::tid

Host thread id (or LWP id) of the thread hosting the context (Linux only).

4.16. CUDBGEvent::cases_st::contextDestroy_st Struct Reference

Information about the context being destroyed.

uint64_t CUDBGEvent::cases_st::contextDestroy_st::context

Context handle of the context being destroyed.

uint32_t CUDBGEvent::cases_st::contextDestroy_st::dev

Device index of the context.

uint32_t CUDBGEvent::cases_st::contextDestroy_st::tid

Host thread id (or LWP id) of the thread hosting the context (Linux only).

4.17. CUDBGEvent::cases_st::contextPop_st Struct Reference

Information about the context being popped.

uint64_t CUDBGEvent::cases_st::contextPop_st::context

Context handle of the context being popped.

uint32_t CUDBGEvent::cases_st::contextPop_st::dev

Device index of the context.

uint32_t CUDBGEvent::cases_st::contextPop_st::tid

Host thread id (or LWP id) of the thread hosting the context (Linux only).

4.18. CUDBGEvent::cases_st::contextPush_st Struct Reference

Information about the context being pushed.

uint64_t CUDBGEvent::cases_st::contextPush_st::context

Context handle of the context being pushed.

uint32_t CUDBGEvent::cases_st::contextPush_st::dev

Device index of the context.

uint32_t CUDBGEvent::cases_st::contextPush_st::tid

Host thread id (or LWP id) of the thread hosting the context (Linux only).

4.19. CUDBGEvent::cases_st::elfImageLoaded_st Struct Reference

Information about the loaded ELF image.

uint64_t CUDBGEvent::cases_st::elfImageLoaded_st::context

Context handle of the loaded module.

uint32_t CUDBGEvent::cases_st::elfImageLoaded_st::dev

Device index of the loaded module.

uint64_t CUDBGEvent::cases_st::elfImageLoaded_st::handle

ELF image handle.

uint64_t CUDBGEvent::cases_st::elfImageLoaded_st::module

Loaded module handle.

uint32_t CUDBGEvent::cases_st::elfImageLoaded_st::properties

ELF image properties.

uint64_t CUDBGEvent::cases_st::elfImageLoaded_st::size

Size of the ELF image (64-bit).

4.20. CUDBGEvent::cases_st::elfImageUnloaded_st Struct Reference

Information about the ELF image about to be unloaded.

uint64_t CUDBGEvent::cases_st::elfImageUnloaded_st::context

Context handle of the module being unloaded.

uint32_t CUDBGEvent::cases_st::elfImageUnloaded_st::dev

Device index of the module being unloaded.

uint64_t CUDBGEvent::cases_st::elfImageUnloaded_st::handle

ELF image handle.

uint64_t CUDBGEvent::cases_st::elfImageUnloaded_st::module

Module handle of the module being unloaded.

uint64_t CUDBGEvent::cases_st::elfImageUnloaded_st::size

Size of the ELF image (64-bit).

4.21. CUDBGEvent::cases_st::functionsLoaded_st Struct Reference

Information about the functions being lazily loaded.

uint64_t CUDBGEvent::cases_st::functionsLoaded_st::context

Context handle of the module containing the functions.

uint32_t CUDBGEvent::cases_st::functionsLoaded_st::count

Functions count.

uint32_t CUDBGEvent::cases_st::functionsLoaded_st::dev

Device index of the module containing the functions.

uint64_t CUDBGEvent::cases_st::functionsLoaded_st::module

Module handle of the module containing the functions.

4.22. CUDBGEvent::cases_st::internalError_st Struct Reference

Information about internal erros.

CUDBGResult CUDBGEvent::cases_st::internalError_st::errorType

Type of the internal error.

4.23. CUDBGEvent::cases_st::kernelFinished_st Struct Reference

Information about the kernel that just terminated.

uint64_t CUDBGEvent::cases_st::kernelFinished_st::context

Context handle of the kernel.

uint32_t CUDBGEvent::cases_st::kernelFinished_st::dev

Device index of the kernel.

uint64_t CUDBGEvent::cases_st::kernelFinished_st::function

Function handle of the kernel.

uint64_t CUDBGEvent::cases_st::kernelFinished_st::functionEntry

Entry address of the kernel.

uint64_t CUDBGEvent::cases_st::kernelFinished_st::gridId

Grid ID of the kernel.

uint64_t CUDBGEvent::cases_st::kernelFinished_st::module

Module handle of the kernel.

uint32_t CUDBGEvent::cases_st::kernelFinished_st::tid

Host thread id (or LWP id) of the thread hosting the kernel (Linux only).

4.24. CUDBGEvent::cases_st::kernelReady_st Struct Reference

Information about the kernel ready to be launched.

struct CuDim3 CUDBGEvent::cases_st::kernelReady_st::blockDim

Block dimensions of the kernel.

uint64_t CUDBGEvent::cases_st::kernelReady_st::context

Context handle of the kernel.

uint32_t CUDBGEvent::cases_st::kernelReady_st::dev

Device index of the kernel.

uint64_t CUDBGEvent::cases_st::kernelReady_st::function

Function handle of the kernel.

uint64_t CUDBGEvent::cases_st::kernelReady_st::functionEntry

Entry address of the kernel.

struct CuDim3 CUDBGEvent::cases_st::kernelReady_st::gridDim

Grid dimensions of the kernel.

uint64_t CUDBGEvent::cases_st::kernelReady_st::gridId

Grid ID of the kernel.

uint64_t CUDBGEvent::cases_st::kernelReady_st::module

Module handleof the kernel.

CUDBGKernelOrigin CUDBGEvent::cases_st::kernelReady_st::origin

Origin of the kernel (CPU or GPU).

uint64_t CUDBGEvent::cases_st::kernelReady_st::parentGridId

Grid ID of the parent grid.

uint32_t CUDBGEvent::cases_st::kernelReady_st::tid

Host thread id (or LWP id) of the thread hosting the kernel (Linux only).

CUDBGKernelType CUDBGEvent::cases_st::kernelReady_st::type

Type of the kernel: system or application.

4.25. CUDBGEvent::cases_st::singleStepComplete_st Struct Reference

Information about a single step operation that has completed.

uint64_t CUDBGEvent::cases_st::singleStepComplete_st::brokenDevicesMask

Device bitmask. This mask has bits set for devices with any warps that hit a breakpoint during the step.

uint64_t CUDBGEvent::cases_st::singleStepComplete_st::context

Context that was stepping.

uint32_t CUDBGEvent::cases_st::singleStepComplete_st::dev

Device that was stepping.

uint64_t CUDBGEvent::cases_st::singleStepComplete_st::faultedDevicesMask

Device bitmask. This mask has bits set for devices with any warps that hit an exception during the step.

uint64_t CUDBGEvent::cases_st::singleStepComplete_st::finalWarpMask

Warps that ended up being stepped (only contains warps from the original SM, other SMs could be stepped and are not reported).

uint64_t CUDBGEvent::cases_st::singleStepComplete_st::originalWarpMask

Warps that were requested to be stepped.

uint32_t CUDBGEvent::cases_st::singleStepComplete_st::sm

SM that was stepping.

CUDBGSingleStepType CUDBGEvent::cases_st::singleStepComplete_st::type

Which single step method has initiated stepping.

4.26. CUDBGEventCallbackData Struct Reference

Callback data passed to callback set with setNotifyNewEventCallback function.

uint32_t CUDBGEventCallbackData::tid

Host thread id of the context generating the event. Zero if not available.

void *CUDBGEventCallbackData::userData

User data passed to the callback.

4.27. CUDBGEventCallbackData40 Struct Reference

Callback data passed to callback set with setNotifyNewEventCallback40 function.

Note:

DEPRECATED: Use CUDBGEventCallbackData instead.

uint32_t CUDBGEventCallbackData40::tid

Host thread id of the context generating the event. Zero if not available.

4.28. CUDBGEventCallbackData41 Struct Reference

Callback data passed to callback set with setNotifyNewEventCallback41 function.

Note:

DEPRECATED: Use CUDBGEventCallbackData instead.

uint32_t CUDBGEventCallbackData41::tid

Host thread id of the context generating the event. Zero if not available.

uint32_t CUDBGEventCallbackData41::timeout

A boolean notifying the debugger that the debug API timed while waiting for a reponse from the debugger to a previous event. It is up to the debugger to decide what to do in response to a timeout.

4.29. CUDBGGridInfo Struct Reference

Information about a CUDA grid.

struct CuDim3 CUDBGGridInfo::blockDim

Block dimensions.

struct CuDim3 CUDBGGridInfo::clusterDim

Number of blocks in the cluster.

uint64_t CUDBGGridInfo::context

Context handle of the context this grid belongs to.

uint32_t CUDBGGridInfo::dev

Index of the device this grid is running on.

uint64_t CUDBGGridInfo::function

Function handle of the function corresponding to this grid.

uint64_t CUDBGGridInfo::functionEntry

Entry address of the function corresponding to this grid.

struct CuDim3 CUDBGGridInfo::gridDim

Grid dimensions.

uint64_t CUDBGGridInfo::gridId64

Grid ID of this grid.

uint64_t CUDBGGridInfo::module

Module handle of the module this grid belongs to.

CUDBGKernelOrigin CUDBGGridInfo::origin

Origin of this grid: CPU or GPU.

uint64_t CUDBGGridInfo::parentGridId

Grid ID of the parent grid that launched this grid.

struct CuDim3 CUDBGGridInfo::preferredClusterDim

Preferred number of blocks in the cluster.

uint32_t CUDBGGridInfo::tid

Thread ID of the host thread that launched this grid.

CUDBGKernelType CUDBGGridInfo::type

Grid type: system or application.

4.30. CUDBGGridInfo120 Struct Reference

Grid info.

Note:

DEPRECATED: Use CUDBGGridInfo instead. For documentation of individual fields, see the documentation of the CUDBGGridInfo struct instead.

See also:

CUDBGGridInfo

4.31. CUDBGGridInfo55 Struct Reference

Grid info.

Note:

DEPRECATED: Use CUDBGGridInfo instead. For documentation of individual fields, see the documentation of the CUDBGGridInfo struct instead.

See also:

CUDBGGridInfo

4.32. CUDBGLaneState Struct Reference

Lane state (state of a single thread).

CUDBGException_t CUDBGLaneState::exception

Exception of the thread (if any).

struct CuDim3 CUDBGLaneState::threadIdx

Thread index of the thread.

uint64_t CUDBGLaneState::virtualPC

(VA) PC of the thread.

4.33. CUDBGLoadedFunctionInfo Struct Reference

Information about a lazily loaded function.

uint64_t CUDBGLoadedFunctionInfo::address

Address of a loaded function.

uint64_t CUDBGLoadedFunctionInfo::sectionIndex

Section index of a loaded function.

4.34. CUDBGMemoryInfo Struct Reference

Memory information.

uint64_t CUDBGMemoryInfo::size

Size of the memory region.

uint64_t CUDBGMemoryInfo::startAddress

Start address of the memory region.

4.35. CUDBGSMInfo Struct Reference

SM-level information.

uint32_t CUDBGSMInfo::smAttributeFlags

Bitmask of CUDBGSmInfoAttribute_t enums for a SM.

uint64_t CUDBGSMInfo::warpBrokenMask

Broken warps mask.

uint64_t CUDBGSMInfo::warpValidMask

Valid warps mask.

4.36. CUDBGWarpInfo Struct Reference

Warp-level information.

uint32_t CUDBGWarpInfo::activeLanes

Active lanes (threads) mask.

struct CuDim3 CUDBGWarpInfo::baseThreadIdx

Base thread index (index of the first thread in the warp). Indices of all other threads in the warp can be calculated by monotonically increasing the coordinates of the base thread index and wrapping around the block dimensions.

struct CuDim3 CUDBGWarpInfo::blockIdx

Block index.

uint64_t CUDBGWarpInfo::gridId

Grid ID.

uint32_t CUDBGWarpInfo::validLanes

Valid lanes (threads) mask.

uint32_t CUDBGWarpInfo::warpAttributeFlags

Bitmask of CUDBGWarpInfoAttribute_t enums for warps and their lanes.

4.37. CUDBGWarpResources Struct Reference

Warp resources. These resources can change at runtime between suspends.

uint32_t CUDBGWarpResources::numRegisters

Number of registers used by the warp.

uint32_t CUDBGWarpResources::sharedMemSize

Shared memory size used by the warp.

4.38. CUDBGWarpState Struct Reference

Warp state information.

uint32_t CUDBGWarpState::activeLanes

Lane mask of active threads in the warp.

struct CuDim3 CUDBGWarpState::blockIdx

Block index of the block containing the warp.

struct CuDim3 CUDBGWarpState::clusterDim

Cluster dimensions of the cluster containing the warp. Can be different for different warps in the same grid.

struct CuDim3 CUDBGWarpState::clusterExceptionTargetBlockIdx

Cluster exception target block index of the the warp.

uint32_t CUDBGWarpState::clusterExceptionTargetBlockIdxValid

Whether the cluster exception target block index is valid.

struct CuDim3 CUDBGWarpState::clusterIdx

Cluster index of the cluster containing the warp.

uint64_t CUDBGWarpState::errorPC

Error PC of the warp (if any).

uint32_t CUDBGWarpState::errorPCValid

Whether the error PC is valid.

uint64_t CUDBGWarpState::gridId

Grid ID of the grid running in the warp.

uint32_t CUDBGWarpState::inSyscallLanes

Lane mask of threads in a syscall.

struct CUDBGLaneState CUDBGWarpState::lane

State of the lanes (threads) in the warp.

uint32_t CUDBGWarpState::validLanes

Lane mask of valid threads in the warp.

4.39. CUDBGWarpState120 Struct Reference

Warp state for API version 12.0.

> Note:
>
> DEPRECATED: Use CUDBGWarpState instead. For documentation of individual fields, see the documentation of the CUDBGWarpState struct instead.

See also:

CUDBGWarpState

4.40. CUDBGWarpState127 Struct Reference

Warp state for API version 12.7.

> Note:
>
> DEPRECATED: Use CUDBGWarpState instead. For documentation of individual fields, see the documentation of the CUDBGWarpState struct instead.

See also:

CUDBGWarpState

4.41. CUDBGWarpState60 Struct Reference

Warp state for API version 6.0.

> Note:
>
> DEPRECATED: Use CUDBGWarpState instead. For documentation of individual fields, see the documentation of the CUDBGWarpState struct instead.

See also:

CUDBGWarpState

4.42. CuDim2 Struct Reference

2-dimensional coordinates for threads, blocks, etc.

Note:

DEPRECATED: Use CuDim3 instead.

uint32_t CuDim2::x

X coordinate.

uint32_t CuDim2::y

Y coordinate.

4.43. CuDim3 Struct Reference

3-dimensional coordinates for threads, blocks, etc.

uint32_t CuDim3::x

X coordinate.

uint32_t CuDim3::y

Y coordinate.

uint32_t CuDim3::z

Z coordinate.

Chapter 5. Data Fields

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

A

acknowledgeEvent30
cudbgGetAPI
acknowledgeEvents42
cudbgGetAPI
acknowledgeSyncEvents
cudbgGetAPI
activeLanes
CUDBGWarpState
CUDBGWarpInfo
activeMask
CUDBGCbuWarpState
address
CUDBGLoadedFunctionInfo
allDevicesSuspended
CUDBGEvent::CUDBGEvent::cases_st
attribute
CUDBGAttributeValuePair

B

barrierMasks
CUDBGCbuWarpState
baseThreadIdx
CUDBGWarpInfo
blockDim
CUDBGGridInfo
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
blockIdx
CUDBGWarpState
CUDBGWarpInfo

brokenDevicesMask
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::allDevicesSuspended_st

C

cases
- CUDBGEvent

clearAttachState
- cudbgGetAPI

clusterDim
- CUDBGWarpState
- CUDBGGridInfo

clusterExceptionTargetBlockIdx
- CUDBGWarpState

clusterExceptionTargetBlockIdxValid
- CUDBGWarpState

clusterIdx
- CUDBGWarpState

collectiveMask
- CUDBGCbuWarpState

consumeCudaLogs
- cudbgGetAPI

context
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::functionsLoaded_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st
- CUDBGGridInfo
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageUnloaded_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st
- CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st

contextCreate
- CUDBGEvent::CUDBGEvent::cases_st

contextDestroy
- CUDBGEvent::CUDBGEvent::cases_st

contextPop
- CUDBGEvent::CUDBGEvent::cases_st

contextPush
- CUDBGEvent::CUDBGEvent::cases_st

count
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::functionsLoaded_st

D

dev
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageUnloaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::functionsLoaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st
CUDBGGridInfo
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st

deviceAttributeFlags
CUDBGDeviceInfo

deviceInfoAttributeSizes
CUDBGDeviceInfoSizes

deviceInfoSize
CUDBGDeviceInfoSizes

disableBreakpoint
cudbgGetAPI

disassemble
cudbgGetAPI

E

elfImageLoaded
CUDBGEvent::CUDBGEvent::cases_st

elfImageUnloaded
CUDBGEvent::CUDBGEvent::cases_st

enableBreakpoint
cudbgGetAPI

errorPC
CUDBGWarpState

errorPCValid
CUDBGWarpState

errorType
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::internalError_st

exception
CUDBGLaneState

F

G

getCudaExceptionString
cudbgGetAPI
getDeviceInfo
cudbgGetAPI
getDeviceInfoSizes
cudbgGetAPI
getDeviceName
cudbgGetAPI
getDevicePCIBusInfo
cudbgGetAPI
getDeviceType
cudbgGetAPI
getElfImage
cudbgGetAPI
getElfImage32
cudbgGetAPI
getElfImageByHandle
cudbgGetAPI
getErrorStringEx
cudbgGetAPI
getGridAttribute
cudbgGetAPI
getGridAttributes
cudbgGetAPI
getGridDim
cudbgGetAPI
getGridDim32
cudbgGetAPI
getGridInfo
cudbgGetAPI
getGridInfo120
cudbgGetAPI
getGridInfo55
cudbgGetAPI
getGridStatus
cudbgGetAPI
getGridStatus50
cudbgGetAPI
getHardwareBarrierInfo
cudbgGetAPI
getHostAddrFromDeviceAddr
cudbgGetAPI

getLoadedFunctionInfo
cudbgGetAPI
getLoadedFunctionInfo118
cudbgGetAPI
getManagedMemoryRegionInfo
cudbgGetAPI
getNextAsyncEvent50
cudbgGetAPI
getNextAsyncEvent55
cudbgGetAPI
getNextEvent
cudbgGetAPI
getNextEvent30
cudbgGetAPI
getNextEvent32
cudbgGetAPI
getNextEvent42
cudbgGetAPI
getNextSyncEvent50
cudbgGetAPI
getNextSyncEvent55
cudbgGetAPI
getNumDevices
cudbgGetAPI
getNumLanes
cudbgGetAPI
getNumPredicates
cudbgGetAPI
getNumRegisters
cudbgGetAPI
getNumSMs
cudbgGetAPI
getNumUniformPredicates
cudbgGetAPI
getNumUniformRegisters
cudbgGetAPI
getNumWarps
cudbgGetAPI
getPhysicalRegister30
cudbgGetAPI
getPhysicalRegister40
cudbgGetAPI

H

I

K

kernelFinished
CUDBGEvent::CUDBGEvent::cases_st
kernelReady
CUDBGEvent::CUDBGEvent::cases_st

L

lane
CUDBGWarpState
laneInfoAttributeSizes
CUDBGDeviceInfoSizes
laneInfoSize
CUDBGDeviceInfoSizes
logLevel
CUDBGCudaLogMessage
lookupDeviceCodeSymbol
cudbgGetAPI

M

memcheckReadErrorAddress
cudbgGetAPI
message
CUDBGCudaLogMessage
module
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::functionsLoaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageUnloaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGGridInfo

N

numRegisters
CUDBGWarpResources

O

origin
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
originalWarpMask
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st

osThreadId
CUDBGCudaLogMessage

P

parentGridId
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
preferredClusterDim
CUDBGGridInfo
properties
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st

R

readActiveLanes
cudbgGetAPI
readAllVirtualReturnAddresses
cudbgGetAPI
readBlockIdx
cudbgGetAPI
readBlockIdx32
cudbgGetAPI
readBrokenWarps
cudbgGetAPI
readCallDepth
cudbgGetAPI
readCallDepth32
cudbgGetAPI
readCCRegister
cudbgGetAPI
readClusterIdx
cudbgGetAPI
readCodeMemory
cudbgGetAPI
readConstMemory129
cudbgGetAPI
readCPUCallStack
cudbgGetAPI
readDeviceExceptionState
cudbgGetAPI
readDeviceExceptionState80
cudbgGetAPI
readErrorPC
cudbgGetAPI

readGenericMemory
cudbgGetAPI
readGlobalMemory
cudbgGetAPI
readGlobalMemory31
cudbgGetAPI
readGlobalMemory55
cudbgGetAPI
readGridId
cudbgGetAPI
readGridId50
cudbgGetAPI
readLaneException
cudbgGetAPI
readLaneStatus
cudbgGetAPI
readLocalMemory
cudbgGetAPI
readParamMemory
cudbgGetAPI
readPC
cudbgGetAPI
readPinnedMemory
cudbgGetAPI
readPredicates
cudbgGetAPI
readRegister
cudbgGetAPI
readRegisterRange
cudbgGetAPI
readRegisterRange60
cudbgGetAPI
readReturnAddress
cudbgGetAPI
readReturnAddress32
cudbgGetAPI
readSharedMemory
cudbgGetAPI
readSmException
cudbgGetAPI
readSyscallCallDepth
cudbgGetAPI

readTextureMemory
cudbgGetAPI
readTextureMemoryBindless
cudbgGetAPI
readThreadIdx
cudbgGetAPI
readUniformPredicates
cudbgGetAPI
readUniformRegisterRange
cudbgGetAPI
readValidLanes
cudbgGetAPI
readValidWarps
cudbgGetAPI
readVirtualPC
cudbgGetAPI
readVirtualReturnAddress
cudbgGetAPI
readVirtualReturnAddress32
cudbgGetAPI
readWarpResources
cudbgGetAPI
readWarpState
cudbgGetAPI
readWarpState120
cudbgGetAPI
readWarpState127
cudbgGetAPI
readWarpState60
cudbgGetAPI
removeBreakpoint
cudbgGetAPI
requestCleanupOnDetach
cudbgGetAPI
requestCleanupOnDetach55
cudbgGetAPI
requiredBufferSize
CUDBGDeviceInfoSizes
responseType
CUDBGDeviceInfo
resumeAllDevices
cudbgGetAPI

resumeDevice
cudbgGetAPI
resumeWarpsUntilPC
cudbgGetAPI
resumeWarpsUntilPC60
cudbgGetAPI

S

sectionIndex
CUDBGLoadedFunctionInfo
setBreakpoint
cudbgGetAPI
setBreakpoint31
cudbgGetAPI
setKernelLaunchNotificationMode
cudbgGetAPI
setNotifyNewEventCallback
cudbgGetAPI
setNotifyNewEventCallback31
cudbgGetAPI
setNotifyNewEventCallback40
cudbgGetAPI
setNotifyNewEventCallback41
cudbgGetAPI
sharedMemSize
CUDBGWarpResources
singleStepComplete
CUDBGEvent::CUDBGEvent::cases_st
singleStepWarp
cudbgGetAPI
singleStepWarp40
cudbgGetAPI
singleStepWarp41
cudbgGetAPI
singleStepWarp65
cudbgGetAPI
size
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageLoaded_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::elfImageUnloaded_st
CUDBGMemoryInfo
sm
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st

smAttributeFlags
CUDBGSMInfo
smInfoAttributeSizes
CUDBGDeviceInfoSizes
smInfoSize
CUDBGDeviceInfoSizes
startAddress
CUDBGMemoryInfo
suspendAllDevices
cudbgGetAPI
suspendDevice
cudbgGetAPI

T

threadIdx
CUDBGLaneState
threadState
CUDBGCbuWarpState
tid
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelFinished_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextDestroy_st
CUDBGEventCallbackData40
CUDBGGridInfo
CUDBGEventCallbackData
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPush_st
CUDBGEventCallbackData41
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextCreate_st
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::contextPop_st
timeout
CUDBGEventCallbackData41
type
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::kernelReady_st
CUDBGGridInfo
CUDBGEvent::CUDBGEvent::cases_st::CUDBGEvent::cases_st::singleStepComplete_st

U

unixTimestampNs
CUDBGCudaLogMessage
unsetBreakpoint
cudbgGetAPI
unsetBreakpoint31
cudbgGetAPI

writeSharedMemory
cudbgGetAPI
writeUniformPredicates
cudbgGetAPI
writeUniformRegister
cudbgGetAPI

X

x
CuDim2
CuDim3

Y

y
CuDim2
CuDim3

Z

z
CuDim3

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

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

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
https://www.nvidia.com