CHIP-SPV

Generated by Doxygen 1.9.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 allocation_info Struct Reference

Public Attributes

- void * **base_ptr**
- size_t **size**

The documentation for this struct was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh

4.2 CHIPAllocationTracker Class Reference

Class for keeping track of device allocations.

`#include <CHIPBackend.hh>`

Public Member Functions

- allocation_info * getByHostPtr (const void *)
  Get allocation_info based on host pointer.
- allocation_info * getByDevPtr (const void *)
  Get allocation_info based on device pointer.

4.2.1 Detailed Description

Class for keeping track of device allocations.

4.2.2 Member Function Documentation

4.2.2.1 getByDevPtr()

```
allocation_info * CHIPAllocationTracker::getByDevPtr (
            const void * dev_ptr )
```

Get allocation_info based on device pointer.

Returns

allocation_info contains the base pointer and allocation size;

4.2.2.2 getByHostPtr()

```
allocation_info * CHIPAllocationTracker::getByHostPtr (
            const void * host_ptr )
```

Get allocation_info based on host pointer.

Returns

allocation_info contains the base pointer and allocation size;

The documentation for this class was generated from the following files:

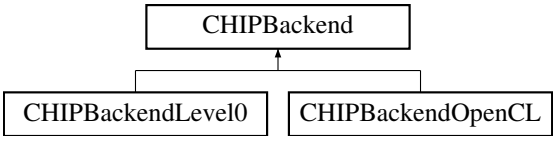
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.3 CHIPBackend Class Reference

Primary object to interact with the backend.

```
#include <CHIPBackend.hh>
```

Inheritance diagram for CHIPBackend:



Public Member Functions

- **CHIPBackend** ()

  Construct a new CHIPBackend object.
- ~**CHIPBackend** ()

  Destroy the CHIPBackend objectk.
- virtual void initialize (std::string platform_str, std::string device_type_str, std::string device_ids_str)

  Initialize this backend with given environment flags.
- virtual void **initialize** ()=0
- virtual void **uninitialize** ()=0
- std::vector< CHIPQueue * > & getQueues ()

  Get the Queues object.
- CHIPQueue * getActiveQueue ()

  Get the Active Queue object.
- CHIPContext * getActiveContext ()

  Get the Active Context object. Returns the context of the active queue.
- CHIPDevice * getActiveDevice ()

  Get the Active Device object. Returns the device of the active queue.
- void setActiveDevice (CHIPDevice *chip_dev)

  Set the active device. Sets the active queue to this device's first/default/primary queue.
- std::vector< CHIPDevice * > & **getDevices** ()
- size_t getNumDevices ()

  Get the Num Devices object.
- std::vector< std::string * > & getModulesStr ()

  Get the vector of registered modules (in string/binary format)
- void addContext (CHIPContext *ctx_in)

  Add a context to this backend.
- void addQueue (CHIPQueue *q_in)

  Add a context to this backend.
- void addDevice (CHIPDevice *dev_in)

  Add a device to this backend.
- void registerModuleStr (std::string *mod_str)
- void unregisterModuleStr (std::string *mod_str)
- hipError_t configureCall (dim3 grid, dim3 block, size_t shared, hipStream_t q)

  Configure an upcoming kernel call.
- hipError_t setArg (const void *arg, size_t size, size_t offset)

  Set the Arg object.
- virtual bool registerFunctionAsKernel (std::string *module_str, const void *host_f_ptr, const char *host_f_↩ name)

  Register this function as a kernel for all devices initialized in this backend.
- CHIPDevice * findDeviceMatchingProps (const hipDeviceProp_t *props)

  Return a device which meets or exceeds the requirements.
- hipError_t addModule (CHIPModule *chip_module)

  Add a CHIPModule to every initialized device.
- hipError_t removeModule (CHIPModule *chip_module)

  Remove this module from every device.

Public Attributes

- CHIPAllocationTracker **AllocationTracker**

  Keep track of pointers allocated on the device. Used to get info about allocaitons based on device poitner in case that findPointerInfo() is not overriden.
- std::stack< CHIPExecItem * > **chip_execstack**
- std::vector< CHIPContext * > **chip_contexts**
- std::vector< CHIPQueue * > **chip_queues**
- std::vector< CHIPDevice * > **chip_devices**

Static Public Attributes

- static thread_local hipError_t **tls_last_error** = hipSuccess
- static thread_local CHIPContext * **tls_active_ctx**

Protected Attributes

- std::vector< std::string * > **modules_str**

  chip_modules stored in binary representation. During compilation each translation unit is parsed for functions that are marked for execution on the device. These functions are then compiled to device code and stored in binary representation.
- std::mutex **mtx**
- CHIPContext * **active_ctx**
- CHIPDevice * **active_dev**
- CHIPQueue * **active_q**

4.3.1 Detailed Description

Primary object to interact with the backend.

4.3.2 Member Function Documentation

4.3.2.1 addContext()

```
void CHIPBackend::addContext (
            CHIPContext * ctx_in )
```

Add a context to this backend.

Parameters

ctx_in	

4.3.2.2 addDevice()

```
void CHIPBackend::addDevice (
            CHIPDevice * dev_in )
```

Add a device to this backend.

Parameters

dev↩ _in	

4.3.2.3 addModule()

```
hipError_t CHIPBackend::addModule (
            CHIPModule * chip_module )
```

Add a CHIPModule to every initialized device.

Parameters

chip_module	pointer to CHIPModule object

Returns

hipError_t

4.3.2.4 addQueue()

```
void CHIPBackend::addQueue (
            CHIPQueue * q_in )
```

Add a context to this backend.

Parameters

q↩ _in	

4.3.2.5 configureCall()

```
hipError_t CHIPBackend::configureCall (
            dim3 grid,
```

```
            dim3 block,
            size_t shared,
            hipStream_t q )
```

Configure an upcoming kernel call.

Parameters

grid	
block	
shared	
q	

Returns

hipError_t

4.3.2.6 findDeviceMatchingProps()

```
CHIPDevice * CHIPBackend::findDeviceMatchingProps (
            const hipDeviceProp_t * props )
```

Return a device which meets or exceeds the requirements.

Parameters

props	

Returns

CHIPDevice∗

4.3.2.7 getActiveContext()

```
CHIPContext * CHIPBackend::getActiveContext ( )
```

Get the Active Context object. Returns the context of the active queue.

Returns

CHIPContext∗

4.3.2.8 getActiveDevice()

```
CHIPDevice * CHIPBackend::getActiveDevice ( )
```

Get the Active Device object. Returns the device of the active queue.

Returns

CHIPDevice*

4.3.2.9 getActiveQueue()

```
CHIPQueue * CHIPBackend::getActiveQueue ( )
```

Get the Active Queue object.

Returns

CHIPQueue*

4.3.2.10 getModulesStr()

```
std::vector< std::string * > & CHIPBackend::getModulesStr ( )
```

Get the vector of registered modules (in string/binary format)

Returns

std::vector<std::string*>&

4.3.2.11 getNumDevices()

```
size_t CHIPBackend::getNumDevices ( )
```

Get the Num Devices object.

Returns

size_t

4.3.2.12 getQueues()

```
std::vector< CHIPQueue * > & CHIPBackend::getQueues ( )
```

Get the Queues object.

Returns

std::vector<CHIPQueue*>&

4.3.2.13 initialize()

```
void CHIPBackend::initialize (
            std::string platform_str,
            std::string device_type_str,
            std::string device_ids_str )  [virtual]
```

Initialize this backend with given environment flags.

Parameters

platform_str	
device_type_str	
device_ids_str	

Reimplemented in CHIPBackendLevel0, and CHIPBackendOpenCL.

4.3.2.14 registerFunctionAsKernel()

```
bool CHIPBackend::registerFunctionAsKernel (
            std::string * module_str,
            const void * host_f_ptr,
            const char * host_f_name )  [virtual]
```

Register this function as a kernel for all devices initialized in this backend.

Parameters

module_str	
host_f_ptr	
host_f_name	

Returns

true

false

Parameters

module_str	
HostFunctionPtr	
FunctionName	

Returns

true

false

4.3.2.15 registerModuleStr()

```
void CHIPBackend::registerModuleStr (
            std::string * mod_str )
```

Parameters

mod_str	

4.3.2.16 removeModule()

```
hipError_t CHIPBackend::removeModule (
            CHIPModule * chip_module )
```

Remove this module from every device.

Parameters

chip_module	pointer to the module which is to be removed

Returns

hipError_t

4.3.2.17 setActiveDevice()

```
void CHIPBackend::setActiveDevice (
            CHIPDevice * chip_dev )
```

Set the active device. Sets the active queue to this device's first/default/primary queue.

Parameters

chip_dev	

4.3.2.18 setArg()

```
hipError_t CHIPBackend::setArg (
            const void * arg,
            size_t size,
            size_t offset )
```

Set the Arg object.

Parameters

arg	
size	
offset	

Returns

hipError_t

4.3.2.19 unregisterModuleStr()

```
void CHIPBackend::unregisterModuleStr (
            std::string * mod_str )
```

Parameters

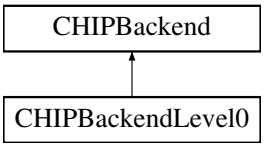
mod_str	

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.4 CHIPBackendLevel0 Class Reference

Inheritance diagram for CHIPBackendLevel0:



Public Member Functions

- virtual void initialize (std::string CHIPPlatformStr, std::string CHIPDeviceTypeStr, std::string CHIPDeviceStr) override

  Initialize this backend with given environment flags.
- virtual void initialize () override
- void uninitialize () override

Additional Inherited Members

4.4.1 Member Function Documentation

4.4.1.1 initialize() [1/2]

```
virtual void CHIPBackendLevel0::initialize ( )  [inline], [override], [virtual]
```

Implements CHIPBackend.

4.4.1.2 initialize() [2/2]

```
virtual void CHIPBackendLevel0::initialize (
            std::string platform_str,
            std::string device_type_str,
            std::string device_ids_str )  [inline], [override], [virtual]
```

Initialize this backend with given environment flags.

Parameters

platform_str	
device_type_str	
device_ids_str	

Reimplemented from CHIPBackend.

4.4.1.3 uninitialize()

```
void CHIPBackendLevel0::uninitialize ( )  [inline], [override], [virtual]
```

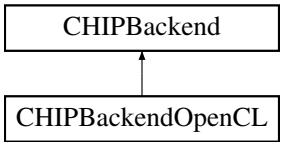
Implements CHIPBackend.

The documentation for this class was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.hh

4.5 CHIPBackendOpenCL Class Reference

Inheritance diagram for CHIPBackendOpenCL:



Public Member Functions

- void initialize (std::string CHIPPlatformStr, std::string CHIPDeviceTypeStr, std::string CHIPDeviceStr) override
  Initialize this backend with given environment flags.
- virtual void initialize () override
- void uninitialize () override

Additional Inherited Members

4.5.1 Member Function Documentation

4.5.1.1 initialize() [1/2]

```
virtual void CHIPBackendOpenCL::initialize ( )  [inline], [override], [virtual]
```

Implements CHIPBackend.

4.5.1.2 initialize() [2/2]

```
void CHIPBackendOpenCL::initialize (
            std::string platform_str,
            std::string device_type_str,
            std::string device_ids_str )  [inline], [override], [virtual]
```

Initialize this backend with given environment flags.

Parameters

platform_str	
device_type_str	
device_ids_str	

Reimplemented from CHIPBackend.

4.5.1.3 uninitialize()

```
void CHIPBackendOpenCL::uninitialize ( )  [inline], [override], [virtual]
```

Implements CHIPBackend.

The documentation for this class was generated from the following file:

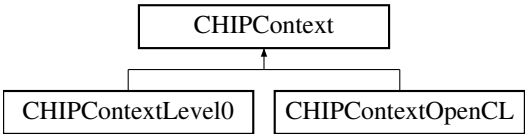
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh

4.6 CHIPContext Class Reference

Context class Contexts contain execution queues and are created on top of a single or multiple devices. Provides for creation of additional queues, events, and interaction with devices.

```
#include <CHIPBackend.hh>
```

Inheritance diagram for CHIPContext:



Public Member Functions

- **CHIPContext** ()

  Construct a new CHIPContext object.
- ~**CHIPContext** ()

  Destroy the CHIPContext object.
- bool addDevice (CHIPDevice *dev)

  Add a device to this context.
- void addQueue (CHIPQueue *q)

  Add a queue to this context.
- std::vector< CHIPDevice * > & getDevices ()

  Get this context's CHIPDevices.
- std::vector< CHIPQueue * > & getQueues ()

Get the this contexts CHIPQueues.

- hipStream_t findQueue (hipStream_t stream)

  Find a queue. If a null pointer is passed, return the Active Queue (active devices's primary queue). If this queue is not found in this context then return nullptr.
- void * allocate (size_t size)

  Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type) with allignment = 0 and allocation type = Shared.
- void * allocate (size_t size, CHIPMemoryType mem_type)

  Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type) with allignment = 0.
- void * allocate (size_t size, size_t alignment, CHIPMemoryType mem_type)

  Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type)
- virtual void * allocate_ (size_t size, size_t alignment, CHIPMemoryType mem_type)=0

  Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.
- hipError_t free (void *ptr)

  Free memory.
- virtual void free_ (void *ptr)=0

  Free memory To be overriden by the backend.
- virtual hipError_t memCopy (void *dst, const void *src, size_t size, hipStream_t stream)=0

  Copy memory.
- void **finishAll** ()

  Finish all the queues in this context.
- virtual hipError_t findPointerInfo (hipDeviceptr_t *pbase, size_t *psize, hipDeviceptr_t dptr)

  For a given device pointer, return the base address of the allocation to which it belongs to along with the allocation size.
- unsigned int getFlags ()

  Get the flags set on this context.
- void setFlags (unsigned int flags)

  Set the flags for this context.
- void **reset** ()

  Reset this context. TODO: what does it mean to reset a context?
- CHIPContext * retain ()

  Retain this context. TODO: What does it mean to retain a context?
- void recordEvent (CHIPQueue *q, CHIPEvent *event)

  Record an event in a given queue.
- virtual CHIPTexture * createImage (hipResourceDesc *resDesc, hipTextureDesc *texDesc)

  Create a Image objct.

Protected Attributes

- std::vector< CHIPDevice * > **chip_devices**
- std::vector< CHIPQueue * > **chip_queues**
- std::mutex **mtx**

4.6.1 Detailed Description

Context class Contexts contain execution queues and are created on top of a single or multiple devices. Provides for creation of additional queues, events, and interaction with devices.

4.6.2 Member Function Documentation

4.6.2.1 addDevice()

```
bool CHIPContext::addDevice (
            CHIPDevice * dev )
```

Add a device to this context.

Parameters

dev	pointer to CHIPDevice object

Returns

true if device was added successfully

false upon failure

4.6.2.2 addQueue()

```
void CHIPContext::addQueue (
            CHIPQueue * q )
```

Add a queue to this context.

Parameters

q	CHIPQueue to be added

4.6.2.3 allocate() [1/3]

```
void * CHIPContext::allocate (
            size_t size )
```

Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type) with allignment = 0 and allocation type = Shared.

Parameters

size	size of the allocation

Returns

void* pointer to allocated memory

4.6.2.4 allocate() [2/3]

```
void * CHIPContext::allocate (
            size_t size,
            CHIPMemoryType mem_type )
```

Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type) with allignment = 0.

Parameters

size	size of the allocation
mem_type	type of the allocation: Host, Device, Shared

Returns

void* pointer to allocated memory

4.6.2.5 allocate() [3/3]

```
void * CHIPContext::allocate (
            size_t size,
            size_t alignment,
            CHIPMemoryType mem_type )
```

Allocate data. Calls reserveMem() to keep track memory used on the device. Calls CHIPContext::allocate_(size_t size, size_t alignment, CHIPMemoryType mem_type)

Parameters

size	size of the allocation
alignment	allocation alignment in bytes
mem_type	type of the allocation: Host, Device, Shared

Returns

void* pointer to allocated memory

4.6.2.6 allocate_()

```
virtual void * CHIPContext::allocate_ (
            size_t size,
            size_t alignment,
            CHIPMemoryType mem_type )  [pure virtual]
```

Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.

Parameters

size	size of the allocation.
alignment	allocation alignment in bytes
mem_type	type of the allocation: Host, Device, Shared

Returns

void∗

Implemented in CHIPContextOpenCL, and CHIPContextLevel0.

4.6.2.7 createImage()

```
CHIPTexture * CHIPContext::createImage (
            hipResourceDesc * resDesc,
            hipTextureDesc * texDesc )  [virtual]
```

Create a Image objct.

Parameters

resDesc	
texDesc	

Returns

CHIPTexture∗

4.6.2.8 findPointerInfo()

```
hipError_t CHIPContext::findPointerInfo (
            hipDeviceptr_t * pbase,
            size_t * psize,
            hipDeviceptr_t dptr )  [virtual]
```

For a given device pointer, return the base address of the allocation to which it belongs to along with the allocation size.

Parameters

pbase	device base pointer to which dptr belongs to
psize	size of the allocation with which pbase was created
dptr	device pointer

Returns

hipError_t

4.6.2.9 findQueue()

```
hipStream_t CHIPContext::findQueue (
            hipStream_t stream )
```

Find a queue. If a null pointer is passed, return the Active Queue (active devices's primary queue). If this queue is not found in this context then return nullptr.

Parameters

stream	CHIPQueue to find

Returns

hipStream_t

4.6.2.10 free()

```
hipError_t CHIPContext::free (
            void * ptr )
```

Free memory.

Parameters

ptr	pointer to the memory location to be deallocated. Internally calls CHIPContext::free_()

Returns

true Success

false Failure

4.6.2.11 free_()

```
virtual void CHIPContext::free_ (
            void * ptr )  [pure virtual]
```

Free memory To be overriden by the backend.

Parameters

ptr	

Returns

true

false

Implemented in CHIPContextLevel0, and CHIPContextOpenCL.

4.6.2.12 getDevices()

```
std::vector< CHIPDevice * > & CHIPContext::getDevices ( )
```

Get this context's CHIPDevices.

Returns

std::vector<CHIPDevice*>&

4.6.2.13 getFlags()

```
unsigned int CHIPContext::getFlags ( )
```

Get the flags set on this context.

Returns

unsigned int context flags

4.6.2.14 getQueues()

```
std::vector< CHIPQueue * > & CHIPContext::getQueues ( )
```

Get the this contexts CHIPQueues.

Returns

std::vector<CHIPQueue*>&

4.6.2.15 memCopy()

```
virtual hipError_t CHIPContext::memCopy (
            void * dst,
            const void * src,
            size_t size,
            hipStream_t stream )  [pure virtual]
```

Copy memory.

Parameters

dst	destination
src	source
size	size of the copy
stream	queue to which this copy should be submitted to

Returns

hipError_t

Implemented in CHIPContextLevel0, and CHIPContextOpenCL.

4.6.2.16 recordEvent()

```
void CHIPContext::recordEvent (
            CHIPQueue * q,
            CHIPEvent * event )
```

Record an event in a given queue.

Parameters

q	queue into which to insert the event
event	event to be inserted

4.6.2.17 retain()

```
CHIPContext * CHIPContext::retain ( )
```

Retain this context. TODO: What does it mean to retain a context?

Returns

CHIPContext*

4.6.2.18 setFlags()

```
void CHIPContext::setFlags (
            unsigned int flags )
```

Set the flags for this context.

Parameters

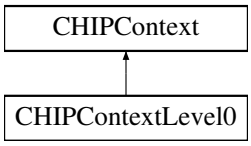
flags	flags to set on this context

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.7 CHIPContextLevel0 Class Reference

Inheritance diagram for CHIPContextLevel0:



Public Member Functions

- ze_command_list_handle_t **get_cmd_list** ()
- **CHIPContextLevel0** (ze_context_handle_t &&_ze_ctx)
- void * allocate_ (size_t size, size_t alignment, CHIPMemoryType memTy) override

Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.

- void free_ (void *ptr) override

  Free memory To be overriden by the backend.
- ze_context_handle_t & **get** ()
- virtual hipError_t memCopy (void *dst, const void *src, size_t size, hipStream_t stream) override

  Copy memory.

Public Attributes

- ze_command_list_handle_t **ze_cmd_list**

Additional Inherited Members

4.7.1 Member Function Documentation

4.7.1.1 allocate_()

```
void * CHIPContextLevel0::allocate_ (
            size_t size,
            size_t alignment,
            CHIPMemoryType mem_type )  [inline], [override], [virtual]
```

Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.

Parameters

size	size of the allocation.
alignment	allocation alignment in bytes
mem_type	type of the allocation: Host, Device, Shared

Returns

void*

Implements CHIPContext.

4.7.1.2 free_()

```
void CHIPContextLevel0::free_ (
            void * ptr )  [inline], [override], [virtual]
```

Free memory To be overriden by the backend.

Parameters

ptr	

Returns

true

false

Implements CHIPContext.

4.7.1.3 memCopy()

```
hipError_t CHIPContextLevel0::memCopy (
            void * dst,
            const void * src,
            size_t size,
            hipStream_t stream )  [override], [virtual]
```

Copy memory.

Parameters

dst	destination
src	source
size	size of the copy
stream	queue to which this copy should be submitted to

Returns

hipError_t

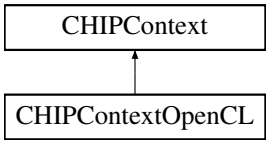
Implements CHIPContext.

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.hh
- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.cc

4.8 CHIPContextOpenCL Class Reference

Inheritance diagram for CHIPContextOpenCL:



Public Member Functions

- **CHIPContextOpenCL** (cl::Context *ctx_in)
- void * allocate_ (size_t size, size_t alignment, CHIPMemoryType mem_type) override

  Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.
- void free_ (void *ptr) override

  Free memory To be overriden by the backend.
- virtual hipError_t memCopy (void *dst, const void *src, size_t size, hipStream_t stream) override

  Copy memory.
- cl::Context * **get** ()

Public Attributes

- SVMemoryRegion **svm_memory**
- cl::Context * **cl_ctx**

Additional Inherited Members

4.8.1 Member Function Documentation

4.8.1.1 allocate_()

```
void * CHIPContextOpenCL::allocate_ (
            size_t size,
            size_t alignment,
            CHIPMemoryType mem_type )  [override], [virtual]
```

Allocate data. Pure virtual function - to be overriden by each backend. This member function is the one that's called by all the publically visible CHIPContext::allocate() variants.

Parameters

size	size of the allocation.
alignment	allocation alignment in bytes
mem_type	type of the allocation: Host, Device, Shared

Returns

void*

Implements CHIPContext.

4.8.1.2 free_()

```
void CHIPContextOpenCL::free_ (
            void * ptr )  [inline], [override], [virtual]
```

Free memory To be overriden by the backend.

Parameters

ptr	

Returns

true

false

Implements CHIPContext.

4.8.1.3 memCopy()

```
hipError_t CHIPContextOpenCL::memCopy (
            void * dst,
            const void * src,
            size_t size,
            hipStream_t stream )  [override], [virtual]
```

Copy memory.

Parameters

dst	destination
src	source
size	size of the copy
stream	queue to which this copy should be submitted to

Returns

hipError_t

Implements CHIPContext.

The documentation for this class was generated from the following files:

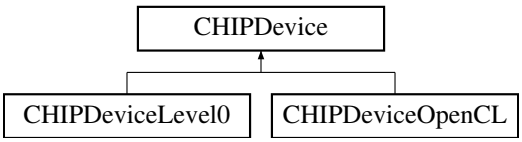
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.cc

4.9 CHIPDevice Class Reference

Compute device class.

`#include <CHIPBackend.hh>`

Inheritance diagram for CHIPDevice:



Public Member Functions

- **CHIPDevice** ()

  Construct a new CHIPDevice object.
- ∼**CHIPDevice** ()

  Destroy the CHIPDevice object.
- std::vector< CHIPKernel * > & getKernels ()

  Get the Kernels object.
- virtual void populateDeviceProperties ()=0

  Use a backend to populate device properties such as memory available, frequencies, etc.
- void copyDeviceProperties (hipDeviceProp_t *prop)

  Query the device for properties.
- CHIPKernel * findKernelByHostPtr (const void *hostPtr)

  Use the host function pointer to retrieve the kernel.
- CHIPContext * getContext ()

  Get the context object.
- void addQueue (CHIPQueue *chip_queue_)

  Construct an additional queue for this device.
- std::vector< CHIPQueue * > getQueues ()

  Get the Queues object.
- CHIPQueue * getActiveQueue ()

  HIP API allows for setting the active device, not the active queue so active device's active queue is always it's 0th/default/primary queue.
- bool removeQueue (CHIPQueue *q)

  Remove a queue from this device's queue vector.
- int getDeviceId ()

  Get the integer ID of this device as it appears in the Backend's chip_devices list.
- virtual std::string getName ()=0

  Get the device name.
- bool reserveMem (size_t bytes)

  Reserve memory for an allocation. This method is run prior to allocations to keep track of how much memory is available on the device TODO: Move to AllocationTracker.
- bool releaseMemReservation (size_t bytes)

  Release some of the reserved memory. Called by free() TODO: Move to AllocationTracker.
- virtual void reset ()=0

  Destroy all allocations and reset all state on the current device in the current process.
- int getAttr (hipDeviceAttribute_t attr)

Query for a specific device attribute. Implementation copied from HIPAMD.

- size_t getGlobalMemSize ()

  Get the total global memory available for this device.
- virtual void setCacheConfig (hipFuncCache_t cfg)

  Set the Cache Config object.
- virtual hipFuncCache_t getCacheConfig ()

  Get the cache configuration for this device.
- virtual void setSharedMemConfig (hipSharedMemConfig config)

  Configure shared memory for this device.
- virtual hipSharedMemConfig getSharedMemConfig ()

  Get the shared memory configuration for this device.
- virtual void setFuncCacheConfig (const void *func, hipFuncCache_t config)

  Setup the cache configuration for the device to use when executing this function.
- bool hasPCIBusId (int pciDomainID, int pciBusID, int pciDeviceID)

  Check if the current device has same PCI bus ID as the one given by input.
- int getPeerAccess (CHIPDevice *peerDevice)

  Get peer-accesability between this and another device.
- hipError_t setPeerAccess (CHIPDevice *peer, int flags, bool canAccessPeer)

  Set access between this and another device.
- size_t getUsedGlobalMem ()

  Get the total used global memory.
- CHIPDeviceVar * getDynGlobalVar (const void *host_var_ptr)

  Get the global variable that came from a FatBinary module.
- CHIPDeviceVar * getStatGlobalVar (const void *host_var_ptr)

  Get the global variable that from from a module loaded at runtime.
- CHIPDeviceVar * getGlobalVar (const void *host_var_ptr)

  Get the global variable.
- void registerFunctionAsKernel (std::string *module_str, const void *host_f_ptr, const char *host_f_name)

  Take the module source, compile the kernels and associate the host function pointer with a kernel whose name matches host function name.

Public Attributes

- std::vector< std::string * > **modules_str**

  chip_modules in binary representation
- std::vector< CHIPModule * > **chip_modules**

  chip_modules in parsed representation
- std::unordered_map< const void *, std::string * > **host_f_ptr_to_module_str_map**

  Map host pointer to module in binary representation.
- std::unordered_map< const void *, CHIPModule * > **host_f_ptr_to_chipmodule_map**

  Map host pointer to module in parsed representation.
- std::unordered_map< const void *, std::string > **host_f_ptr_to_host_f_name_map**

  Map host pointer to a function name.
- std::unordered_map< const void *, CHIPKernel * > **host_ptr_to_chipkernel_map**

  Map host pointer to CHIPKernel.
- std::unordered_map< const void *, CHIPDeviceVar * > host_var_ptr_to_chipdevicevar_stat
- std::unordered_map< const void *, CHIPDeviceVar * > host_var_ptr_to_chipdevicevar_dyn
- int **idx**
- size_t **total_used_mem**
- size_t **max_used_mem**

Protected Attributes

- std::string **device_name**
- std::mutex **mtx**
- std::vector< CHIPKernel * > **chip_kernels**
- CHIPContext * **ctx**
- std::vector< CHIPQueue * > **chip_queues**
- int **active_queue_id** = 0
- hipDeviceAttribute_t **attrs**
- hipDeviceProp_t **hip_device_props**

4.9.1 Detailed Description

Compute device class.

4.9.2 Member Function Documentation

4.9.2.1 addQueue()

```
void CHIPDevice::addQueue (
            CHIPQueue * chip_queue_ )
```

Construct an additional queue for this device.

Parameters

flags	
priority	

Returns

CHIPQueue* pointer to the newly created queue (can also be found in chip_queues vector)

4.9.2.2 copyDeviceProperties()

```
void CHIPDevice::copyDeviceProperties (
            hipDeviceProp_t * prop )
```

Query the device for properties.

Parameters

prop	

4.9.2.3 findKernelByHostPtr()

```
CHIPKernel * CHIPDevice::findKernelByHostPtr (
            const void * hostPtr )
```

Use the host function pointer to retrieve the kernel.

Parameters

hostPtr	

Returns

CHIPKernel* CHIPKernel associated with this host pointer

4.9.2.4 getActiveQueue()

```
CHIPQueue * CHIPDevice::getActiveQueue ( )
```

HIP API allows for setting the active device, not the active queue so active device's active queue is always it's 0th/default/primary queue.

Returns

CHIPQueue*

4.9.2.5 getAttr()

```
int CHIPDevice::getAttr (
            hipDeviceAttribute_t attr )
```

Query for a specific device attribute. Implementation copied from HIPAMD.

Parameters

attr	attribute to query

Returns

int attribute value. In case invalid query returns -1;

4.9.2.6 getCacheConfig()

```
hipFuncCache_t CHIPDevice::getCacheConfig ( )  [virtual]
```

Get the cache configuration for this device.

Returns

hipFuncCache_t

4.9.2.7 getContext()

```
CHIPContext * CHIPDevice::getContext ( )
```

Get the context object.

Returns

CHIPContext* pointer to the CHIPContext object this CHIPDevice was created with

4.9.2.8 getDeviceId()

```
int CHIPDevice::getDeviceId ( )
```

Get the integer ID of this device as it appears in the Backend's chip_devices list.

Returns

int

4.9.2.9 getDynGlobalVar()

```
CHIPDeviceVar * CHIPDevice::getDynGlobalVar (
            const void * host_var_ptr )
```

Get the global variable that came from a FatBinary module.

Parameters

host_var_ptr	host pointer to the variable

Returns

CHIPDeviceVar*

4.9.2.10 getGlobalMemSize()

```
size_t CHIPDevice::getGlobalMemSize ( )
```

Get the total global memory available for this device.

Returns

size_t

4.9.2.11 getGlobalVar()

```
CHIPDeviceVar * CHIPDevice::getGlobalVar (
            const void * host_var_ptr )
```

Get the global variable.

Parameters

host_var_ptr	host pointer to the variable

Returns

CHIPDeviceVar* if not found returns nullptr

4.9.2.12 getKernels()

```
std::vector< CHIPKernel * > & CHIPDevice::getKernels ( )
```

Get the Kernels object.

Returns

std::vector<CHIPKernel*>&

4.9.2.13 getName()

```
virtual std::string CHIPDevice::getName ( )  [pure virtual]
```

Get the device name.

Returns

std::string

Implemented in CHIPDeviceLevel0, and CHIPDeviceOpenCL.

4.9.2.14 getPeerAccess()

```
int CHIPDevice::getPeerAccess (
            CHIPDevice * peerDevice )
```

Get peer-accesability between this and another device.

Parameters

peerDevice	

Returns

int

4.9.2.15 getQueues()

```
std::vector< CHIPQueue * > CHIPDevice::getQueues ( )
```

Get the Queues object.

Returns

std::vector<CHIPQueue*>

4.9.2.16 getSharedMemConfig()

```
hipSharedMemConfig CHIPDevice::getSharedMemConfig ( )  [virtual]
```

Get the shared memory configuration for this device.

Returns

hipSharedMemConfig

4.9.2.17 getStatGlobalVar()

```
CHIPDeviceVar * CHIPDevice::getStatGlobalVar (
            const void * host_var_ptr )
```

Get the global variable that from from a module loaded at runtime.

Parameters

host_var_ptr	host pointer to the variable

Returns

CHIPDeviceVar∗

4.9.2.18 getUsedGlobalMem()

```
size_t CHIPDevice::getUsedGlobalMem ( )
```

Get the total used global memory.

Returns

size_t

4.9.2.19 hasPCIBusId()

```
bool CHIPDevice::hasPCIBusId (
            int pciDomainID,
            int pciBusID,
            int pciDeviceID )
```

Check if the current device has same PCI bus ID as the one given by input.

Parameters

pciDomainID	
pciBusID	
pciDeviceID	

Returns

true

false

4.9.2.20 populateDeviceProperties()

```
virtual void CHIPDevice::populateDeviceProperties ( )  [pure virtual]
```

Use a backend to populate device properties such as memory available, frequencies, etc.

Implemented in CHIPDeviceLevel0, and CHIPDeviceOpenCL.

4.9.2.21 registerFunctionAsKernel()

```
void CHIPDevice::registerFunctionAsKernel (
            std::string * module_str,
            const void * host_f_ptr,
            const char * host_f_name )
```

Take the module source, compile the kernels and associate the host function pointer with a kernel whose name matches host function name.

Parameters

module_str	Binary representation of the SPIR-V module
host_f_ptr	host function pointer
host_f_name	host function name

4.9.2.22 releaseMemReservation()

```
bool CHIPDevice::releaseMemReservation (
            size_t bytes )
```

Release some of the reserved memory. Called by free() TODO: Move to AllocationTracker.

Parameters

bytes	

Returns

true

false

4.9.2.23 removeQueue()

```
bool CHIPDevice::removeQueue (
            CHIPQueue * q )
```

Remove a queue from this device's queue vector.

Parameters

q	

Returns

true

false

4.9.2.24 reserveMem()

```
bool CHIPDevice::reserveMem (
            size_t bytes )
```

Reserve memory for an allocation. This method is run prior to allocations to keep track of how much memory is available on the device TODO: Move to AllocationTracker.

Parameters

bytes	

Returns

true Reservation successful

false Not enough available memory for reservation of this size.

4.9.2.25 reset()

```
virtual void CHIPDevice::reset ( )  [pure virtual]
```

Destroy all allocations and reset all state on the current device in the current process.

Implemented in CHIPDeviceLevel0, and CHIPDeviceOpenCL.

4.9.2.26 setCacheConfig()

```
void CHIPDevice::setCacheConfig (
            hipFuncCache_t cfg )  [virtual]
```

Set the Cache Config object.

Parameters

cfg	configuration

4.9.2.27 setFuncCacheConfig()

```
void CHIPDevice::setFuncCacheConfig (
            const void * func,
            hipFuncCache_t config )  [virtual]
```

Setup the cache configuration for the device to use when executing this function.

Parameters

func	
config	

4.9.2.28 setPeerAccess()

```
hipError_t CHIPDevice::setPeerAccess (
            CHIPDevice * peer,
            int flags,
            bool canAccessPeer )
```

Set access between this and another device.

Parameters

peer	
flags	
canAccessPeer	

Returns

hipError_t

4.9.2.29 setSharedMemConfig()

```
void CHIPDevice::setSharedMemConfig (
            hipSharedMemConfig config )  [virtual]
```

Configure shared memory for this device.

Parameters

config	

4.9.3 Member Data Documentation

4.9.3.1 host_var_ptr_to_chipdevicevar_dyn

```
std::unordered_map<const void*, CHIPDeviceVar*> CHIPDevice::host_var_ptr_to_chipdevicevar_dyn
```

Map host variable address to device pointer and size for dynamically loaded global vars

4.9.3.2 host_var_ptr_to_chipdevicevar_stat

```
std::unordered_map<const void*, CHIPDeviceVar*> CHIPDevice::host_var_ptr_to_chipdevicevar_stat
```

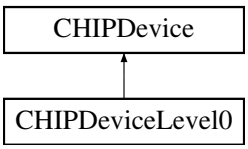
Map host variable address to device pointer and size for statically loaded global vars

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.10 CHIPDeviceLevel0 Class Reference

Inheritance diagram for CHIPDeviceLevel0:



Public Member Functions

- **CHIPDeviceLevel0** (ze_device_handle_t &&ze_dev_, ze_context_handle_t ze_ctx_)
- virtual void populateDeviceProperties () override

  Use a backend to populate device properties such as memory available, frequencies, etc.
- virtual std::string getName () override

  Get the device name.
- ze_device_handle_t & **get** ()
- virtual void reset () override

  Destroy all allocations and reset all state on the current device in the current process.

Additional Inherited Members

4.10.1 Member Function Documentation

4.10.1.1 getName()

```
virtual std::string CHIPDeviceLevel0::getName ( )  [inline], [override], [virtual]
```

Get the device name.

Returns

std::string

Implements CHIPDevice.

4.10.1.2 populateDeviceProperties()

```
virtual void CHIPDeviceLevel0::populateDeviceProperties ( )  [inline], [override], [virtual]
```

Use a backend to populate device properties such as memory available, frequencies, etc.

Implements CHIPDevice.

4.10.1.3 reset()

```
void CHIPDeviceLevel0::reset ( )  [override], [virtual]
```

Destroy all allocations and reset all state on the current device in the current process.

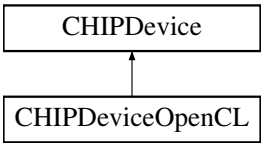
Implements CHIPDevice.

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.hh
- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.cc

4.11 CHIPDeviceOpenCL Class Reference

Inheritance diagram for CHIPDeviceOpenCL:



Public Member Functions

- **CHIPDeviceOpenCL** (CHIPContextOpenCL *chip_ctx, cl::Device *dev_in, int idx)
- cl::Device * **get** ()
- virtual void populateDeviceProperties () override

  Use a backend to populate device properties such as memory available, frequencies, etc.
- virtual std::string getName () override

  Get the device name.
- virtual void reset () override

  Destroy all allocations and reset all state on the current device in the current process.

Public Attributes

- cl::Device * **cl_dev**
- cl::Context * **cl_ctx**

Additional Inherited Members

4.11.1 Member Function Documentation

4.11.1.1 getName()

```
std::string CHIPDeviceOpenCL::getName ( )  [override], [virtual]
```

Get the device name.

Returns

std::string

Implements CHIPDevice.

4.11.1.2 populateDeviceProperties()

```
void CHIPDeviceOpenCL::populateDeviceProperties ( )  [override], [virtual]
```

Use a backend to populate device properties such as memory available, frequencies, etc.

Implements CHIPDevice.

4.11.1.3 reset()

```
void CHIPDeviceOpenCL::reset ( )  [override], [virtual]
```

Destroy all allocations and reset all state on the current device in the current process.

Implements CHIPDevice.

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.cc

4.12 CHIPDeviceVar Class Reference

Public Member Functions

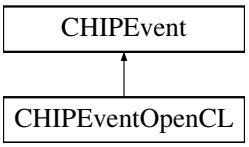
- **CHIPDeviceVar** (std::string host_var_name_, void *dev_ptr_, size_t size)
- void * **getDevAddr** ()
- std::string **getName** ()
- size_t **getSize** ()

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.13 CHIPEvent Class Reference

Inheritance diagram for CHIPEvent:



Public Member Functions

- **CHIPEvent** (CHIPContext *ctx_, CHIPEventType flags_=CHIPEventType::Default)

  CHIPEvent constructor. Must always be created with some context.
- ~**CHIPEvent** ()

  Destroy the CHIPEvent object.
- virtual bool recordStream (CHIPQueue *chip_queue_)

  Enqueue this event in a given CHIPQueue.
- virtual bool wait ()

  Wait for this event to complete.
- virtual bool isFinished ()

  Query the event to see if it completed.
- virtual float getElapsedTime (CHIPEvent *other)

  Calculate absolute difference between completion timestamps of this event and other.

Protected Member Functions

- **CHIPEvent** ()=default

  hidden default constructor for CHIPEvent. Only derived class constructor should be called.

Protected Attributes

- std::mutex **mutex**
- event_status_e **status**
- CHIPEventType **flags**

  event bahavior modifier - valid values are hipEventDefault, hipEventBlockingSync, hipEventDisableTiming, hipEventInterprocess
- CHIPContext * **chip_context**

  Events are always created with a context.

4.13.1 Member Function Documentation

4.13.1.1 getElapsedTime()

```
float CHIPEvent::getElapsedTime (
            CHIPEvent * other )  [virtual]
```

Calculate absolute difference between completion timestamps of this event and other.

Parameters

other	

Returns

float

4.13.1.2 isFinished()

```
bool CHIPEvent::isFinished ( )  [virtual]
```

Query the event to see if it completed.

Returns

true

false

4.13.1.3 recordStream()

```
bool CHIPEvent::recordStream (
            CHIPQueue * chip_queue_ )  [virtual]
```

Enqueue this event in a given CHIPQueue.

Parameters

*chip_↩ queue_*	CHIPQueue in which to enque this event

Returns

true

false

4.13.1.4 wait()

```
bool CHIPEvent::wait ( )  [virtual]
```

Wait for this event to complete.

Returns

true

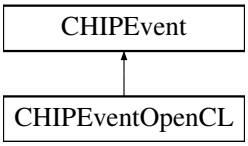
false

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.14 CHIPEventOpenCL Class Reference

Inheritance diagram for CHIPEventOpenCL:



Protected Attributes

- cl::Event * **cl_event**

Additional Inherited Members

The documentation for this class was generated from the following file:

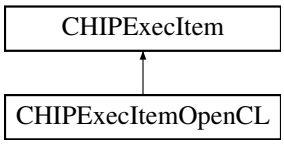
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh

4.15 CHIPExecItem Class Reference

Contains kernel arguments and a queue on which to execute. Prior to kernel launch, the arguments are setup via CHIPBackend::configureCall(). Because of this, we get the kernel last so the kernel so the launch() takes a kernel argument as opposed to queue receiving a CHIPExecItem containing the kernel and arguments.

`#include <CHIPBackend.hh>`

Inheritance diagram for CHIPExecItem:



Public Member Functions

- **CHIPExecItem** ()=delete

  Deleted default constructor Doesn't make sense for CHIPExecItem to exist without arguments.
- CHIPExecItem (dim3 grid_dim_, dim3 block_dim_, size_t shared_mem_, hipStream_t chip_queue_)

  Construct a new CHIPExecItem object.
- ~**CHIPExecItem** ()

  Destroy the CHIPExecItem object.
- CHIPKernel * getKernel ()

  Get the Kernel object.
- CHIPQueue * getQueue ()

  Get the Queue object.
- dim3 getGrid ()

  Get the Grid object.
- dim3 getBlock ()

  Get the Block object.
- void setArg (const void *arg, size_t size, size_t offset)

  Setup a single argument. gets called by hipSetupArgument calls to which are emitted by hip-clang.
- virtual hipError_t launch (CHIPKernel *Kernel)

  Submit a kernel to the associated queue for execution. chip_queue must be set prior to this call.
- hipError_t launchByHostPtr (const void *hostPtr)

  Launch a kernel associated with a host function pointer. Looks up the CHIPKernel associated with this pointer and calls launch()

Protected Attributes

- size_t **shared_mem**
- std::vector< uint8_t > **arg_data**
- std::vector< std::tuple< size_t, size_t > > **offset_sizes**
- dim3 **grid_dim**
- dim3 **block_dim**
- CHIPQueue * **stream**
- CHIPKernel * **chip_kernel**
- CHIPQueue * **chip_queue**

4.15.1 Detailed Description

Contains kernel arguments and a queue on which to execute. Prior to kernel launch, the arguments are setup via CHIPBackend::configureCall(). Because of this, we get the kernel last so the kernel so the launch() takes a kernel argument as opposed to queue receiving a CHIPExecItem containing the kernel and arguments.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 CHIPExecItem()

```
CHIPExecItem::CHIPExecItem (
            dim3 grid_dim_,
            dim3 block_dim_,
            size_t shared_mem_,
            hipStream_t chip_queue_ )
```

Construct a new CHIPExecItem object.

Parameters

*grid_dim_*	
*block_dim_*	
*shared_mem_*	
*chip_queue_*	

4.15.3 Member Function Documentation

4.15.3.1 getBlock()

```
dim3 CHIPExecItem::getBlock ( )
```

Get the Block object.

Returns

dim3

4.15.3.2 getGrid()

```
dim3 CHIPExecItem::getGrid ( )
```

Get the Grid object.

Returns

dim3

4.15.3.3 getKernel()

```
CHIPKernel * CHIPExecItem::getKernel ( )
```

Get the Kernel object.

Returns

CHIPKernel* Kernel to be executed

4.15.3.4 getQueue()

```
CHIPQueue * CHIPExecItem::getQueue ( )
```

Get the Queue object.

Returns

CHIPQueue*

4.15.3.5 launch()

```
hipError_t CHIPExecItem::launch (
            CHIPKernel * Kernel )  [virtual]
```

Submit a kernel to the associated queue for execution. chip_queue must be set prior to this call.

Parameters

Kernel	kernel which is to be launched

Returns

hipError_t possible values: hipSuccess, hipErrorLaunchFailure

Reimplemented in CHIPExecItemOpenCL.

4.15.3.6 launchByHostPtr()

```
hipError_t CHIPExecItem::launchByHostPtr (
            const void * hostPtr )
```

Launch a kernel associated with a host function pointer. Looks up the CHIPKernel associated with this pointer and calls launch()

Parameters

hostPtr	pointer to the host function

Returns

hipError_t possible values: hipSuccess, hipErrorLaunchFailure

4.15.3.7 setArg()

```
void CHIPExecItem::setArg (
            const void * arg,
            size_t size,
            size_t offset )
```

Setup a single argument. gets called by hipSetupArgument calls to which are emitted by hip-clang.

Parameters

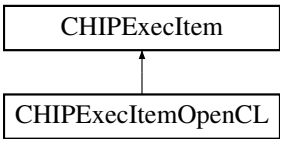
arg	
size	
offset	

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.16 CHIPExecItemOpenCL Class Reference

Inheritance diagram for CHIPExecItemOpenCL:



Public Member Functions

- virtual hipError_t launch (CHIPKernel *chip_kernel) override

  Submit a kernel to the associated queue for execution. chip_queue must be set prior to this call.
- int **setup_all_args** (CHIPKernelOpenCL *kernel)
- cl::Kernel * **get** ()

Public Attributes

- OCLFuncInfo **FuncInfo**

Additional Inherited Members

4.16.1 Member Function Documentation

4.16.1.1 launch()

```
hipError_t CHIPExecItemOpenCL::launch (
            CHIPKernel * Kernel )  [override], [virtual]
```

Submit a kernel to the associated queue for execution. chip_queue must be set prior to this call.

Parameters

Kernel	kernel which is to be launched

Returns

hipError_t possible values: hipSuccess, hipErrorLaunchFailure

Reimplemented from CHIPExecItem.

The documentation for this class was generated from the following files:

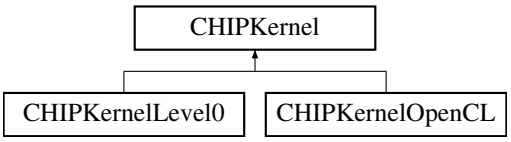
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.cc

4.17 CHIPKernel Class Reference

Contains information about the function on the host and device.

`#include <CHIPBackend.hh>`

Inheritance diagram for CHIPKernel:



Public Member Functions

- std::string getName ()
  Get the Name object.
- const void * getHostPtr ()
  Get the associated host pointer to a host function.
- const void * getDevPtr ()
  Get the associated funciton pointer on the device.
- void setName (std::string host_f_name_)
  Get the Name object.
- void setHostPtr (const void *host_f_ptr_)
  Get the associated host pointer to a host function.
- void setDevPtr (const void *hev_f_ptr_)
  Get the associated funciton pointer on the device.

Protected Member Functions

- **CHIPKernel** ()=default
  hidden default constructor. Only derived type constructor should be called.

Protected Attributes

- std::string **host_f_name**
  Name of the function.
- const void * **host_f_ptr**
  Pointer to the host function.
- const void * **dev_f_ptr**
  Pointer to the device function.

4.17.1 Detailed Description

Contains information about the function on the host and device.

4.17.2 Member Function Documentation

4.17.2.1 getDevPtr()

```
const void * CHIPKernel::getDevPtr ( )
```

Get the associated funciton pointer on the device.

Returns

const void*

4.17.2.2 getHostPtr()

```
const void * CHIPKernel::getHostPtr ( )
```

Get the associated host pointer to a host function.

Returns

const void*

4.17.2.3 getName()

```
std::string CHIPKernel::getName ( )
```

Get the Name object.

Returns

std::string

4.17.2.4 setDevPtr()

```
void CHIPKernel::setDevPtr (
            const void * hev_f_ptr_ )
```

Get the associated funciton pointer on the device.

Returns

const void*

4.17.2.5 setHostPtr()

```
void CHIPKernel::setHostPtr (
            const void * host_f_ptr_ )
```

Get the associated host pointer to a host function.

Returns

const void*

4.17.2.6 setName()

```
void CHIPKernel::setName (
            std::string host_f_name_ )
```

Get the Name object.

Returns

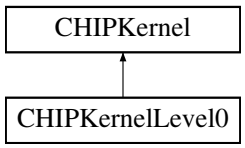
std::string

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.18 CHIPKernelLevel0 Class Reference

Inheritance diagram for CHIPKernelLevel0:



Public Member Functions

- **CHIPKernelLevel0** (ze_kernel_handle_t _ze_kernel, std::string _funcName, const void *_host_ptr)
- ze_kernel_handle_t **get** ()

Protected Attributes

- ze_kernel_handle_t **ze_kernel**

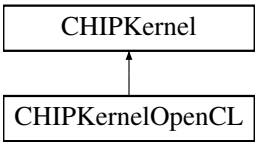
Additional Inherited Members

The documentation for this class was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.hh

4.19 CHIPKernelOpenCL Class Reference

Inheritance diagram for CHIPKernelOpenCL:



Public Member Functions

- **CHIPKernelOpenCL** (const cl::Kernel &&cl_kernel, OpenCLFunctionInfoMap &func_info_map)
- OCLFuncInfo * **get_func_info** () const
- std::string **get_name** ()
- cl::Kernel **get** () const
- size_t **getTotalArgSize** () const

Additional Inherited Members

The documentation for this class was generated from the following file:

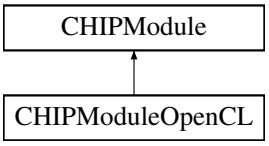
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh

4.20 CHIPModule Class Reference

Module abstraction. Contains global variables and kernels. Can be extracted from FatBinary or loaded at runtime. OpenCL - ClProgram Level Zero - zeModule ROCclr - amd::Program CUDA - CUmodule.

`#include <CHIPBackend.hh>`

Inheritance diagram for CHIPModule:



Public Member Functions

- ~**CHIPModule** ()

  Destroy the CHIPModule object.
- CHIPModule (std::string *module_str)

  Construct a new CHIPModule object. This constructor should be implemented by the derived class (specific backend implementation). Call to this constructor should result in a populated chip_kernels vector.
- CHIPModule (std::string &&module_str)

  Construct a new CHIPModule object using move semantics.
- void addKernel (CHIPKernel *kernel)

  Add a CHIPKernel to this module. During initialization when the FatBinary is consumed, a CHIPModule is constructed for every device. SPIR-V kernels reside in this module. This method is called called via the constructor during this initialization phase. Modules can also be loaded from a file during runtime, however.
- void compileOnce (CHIPDevice *chip_dev)

  Wrapper around compile() called via std::call_once.
- virtual void compile (CHIPDevice *chip_dev)

  Kernel JIT compilation can be lazy. This is configured via Cmake LAZY_JIT option. If LAZY_JIT is set to true then this module won't be compiled until the first call to one of its kernels. If LAZY_JIT is set to false(default) then this method should be called in the constructor;.
- CHIPDeviceVar * getGlobalVar (std::string name)

  Get the Global Var object A module, along with device kernels, can also contain global variables.
- CHIPKernel * getKernel (std::string name)

  Get the Kernel object.
- std::vector< CHIPKernel * > & getKernels ()

  Get the Kernels object.
- CHIPKernel * getKernel (const void *host_f_ptr)

  Get the Kernel object.

Protected Member Functions

- **CHIPModule** ()=default

  hidden default constuctor. Only derived type constructor should be called.

Protected Attributes

- std::mutex **mtx**
- std::vector< CHIPDeviceVar * > **chip_vars**
- std::vector< CHIPKernel * > **chip_kernels**
- std::string **src**

  Binary representation extracted from FatBinary.
- std::once_flag **compiled**

4.20.1 Detailed Description

Module abstraction. Contains global variables and kernels. Can be extracted from FatBinary or loaded at runtime. OpenCL - ClProgram Level Zero - zeModule ROCclr - amd::Program CUDA - CUmodule.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 CHIPModule() [1/2]

```
CHIPModule::CHIPModule (
            std::string * module_str )
```

Construct a new CHIPModule object. This constructor should be implemented by the derived class (specific backend implementation). Call to this constructor should result in a populated chip_kernels vector.

Parameters

module_str	string prepresenting the binary extracted from FatBinary

4.20.2.2 CHIPModule() [2/2]

```
CHIPModule::CHIPModule (
            std::string && module_str )
```

Construct a new CHIPModule object using move semantics.

Parameters

module_str	string from which to move resources

4.20.3 Member Function Documentation

4.20.3.1 addKernel()

```
void CHIPModule::addKernel (
            CHIPKernel * kernel )
```

Add a CHIPKernel to this module. During initialization when the FatBinary is consumed, a CHIPModule is constructed for every device. SPIR-V kernels reside in this module. This method is called called via the constructor during this initialization phase. Modules can also be loaded from a file during runtime, however.

Parameters

kernel	CHIPKernel to be added to this module.

4.20.3.2 compile()

```
void CHIPModule::compile (
```

```
            CHIPDevice * chip_dev )  [virtual]
```

Kernel JIT compilation can be lazy. This is configured via Cmake LAZY_JIT option. If LAZY_JIT is set to true then this module won't be compiled until the first call to one of its kernels. If LAZY_JIT is set to false(default) then this method should be called in the constructor;.

This method should populate this modules chip_kernels vector. These kernels would have a name extracted from the kernel but no associated host function pointers.

Reimplemented in CHIPModuleOpenCL.

4.20.3.3 compileOnce()

```
void CHIPModule::compileOnce (
            CHIPDevice * chip_dev )
```

Wrapper around compile() called via std::call_once.

Parameters

chip_dev	device for which to compile the kernels

4.20.3.4 getGlobalVar()

```
CHIPDeviceVar * CHIPModule::getGlobalVar (
            std::string name )
```

Get the Global Var object A module, along with device kernels, can also contain global variables.

Parameters

name	global variable name

Returns

CHIPDeviceVar*

4.20.3.5 getKernel() [1/2]

```
CHIPKernel * CHIPModule::getKernel (
            const void * host_f_ptr )
```

Get the Kernel object.

Parameters

host_f_ptr	host-side function pointer

Returns

CHIPKernel*

4.20.3.6 getKernel() [2/2]

```
CHIPKernel * CHIPModule::getKernel (
            std::string name )
```

Get the Kernel object.

Parameters

name	name of the corresponding host function

Returns

CHIPKernel*

4.20.3.7 getKernels()

```
std::vector< CHIPKernel * > & CHIPModule::getKernels ( )
```

Get the Kernels object.

Returns

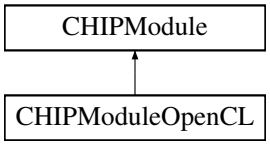
std::vector<CHIPKernel*>&

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.21 CHIPModuleOpenCL Class Reference

Inheritance diagram for CHIPModuleOpenCL:



Public Member Functions

- virtual void compile (CHIPDevice *chip_dev) override

  Kernel JIT compilation can be lazy. This is configured via Cmake LAZY_JIT option. If LAZY_JIT is set to true then this module won't be compiled until the first call to one of its kernels. If LAZY_JIT is set to false(default) then this method should be called in the constructor;.
- cl::Program & **get** ()

Protected Attributes

- cl::Program **program**

Additional Inherited Members

4.21.1 Member Function Documentation

4.21.1.1 compile()

```
void CHIPModuleOpenCL::compile (
            CHIPDevice * chip_dev )  [override], [virtual]
```

Kernel JIT compilation can be lazy. This is configured via Cmake LAZY_JIT option. If LAZY_JIT is set to true then this module won't be compiled until the first call to one of its kernels. If LAZY_JIT is set to false(default) then this method should be called in the constructor;.

This method should populate this modules chip_kernels vector. These kernels would have a name extracted from the kernel but no associated host function pointers.

Reimplemented from CHIPModule.

The documentation for this class was generated from the following files:

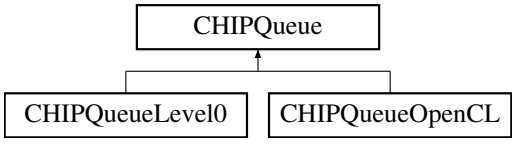
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.cc

4.22 CHIPQueue Class Reference

Queue class for submitting kernels to for execution.

```
#include <CHIPBackend.hh>
```

Inheritance diagram for CHIPQueue:



Public Member Functions

- CHIPQueue (CHIPDevice *chip_dev)

  Construct a new CHIPQueue object.
- CHIPQueue (CHIPDevice *chip_dev, unsigned int flags)

  Construct a new CHIPQueue object.
- CHIPQueue (CHIPDevice *chip_dev, unsigned int flags, int priority)

  Construct a new CHIPQueue object.
- ~**CHIPQueue** ()

  Destroy the CHIPQueue object.
- virtual hipError_t memCopy (void *dst, const void *src, size_t size)

  Blocking memory copy.
- virtual hipError_t memCopyAsync (void *dst, const void *src, size_t size)

  Non-blocking memory copy.
- virtual void memFill (void *dst, size_t size, const void *pattern, size_t pattern_size)

  Blocking memset.
- virtual void memFillAsync (void *dst, size_t size, const void *pattern, size_t pattern_size)

  Non-blocking mem set.
- virtual hipError_t launch (CHIPExecItem *exec_item)

  Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.
- CHIPDevice * getDevice ()

  Get the Device obj.
- virtual void finish ()

  Wait for this queue to finish.
- bool query ()

  Check if the queue is still actively executing.
- int getPriorityRange (int lower_or_upper)

  Get the Priority Range object defining the bounds for hipStreamCreateWithPriority.
- bool enqueueBarrierForEvent (CHIPEvent *e)

  Insert an event into this queue.
- unsigned int getFlags ()

  Get the Flags object with which this queue was created.
- int getPriority ()

  Get the Priority object with which this queue was created.
- bool addCallback (hipStreamCallback_t callback, void *userData)

  Add a callback funciton to be called on the host after the specified stream is done.
- bool memPrefetch (const void *ptr, size_t count)

  Insert a memory prefetch.
- bool launchHostFunc (const void *hostFunction, dim3 numBlocks, dim3 dimBlocks, void **args, size_t sharedMemBytes)

  Launch a kernel on this queue given a host pointer and arguments.
- hipError_t launchWithKernelParams (dim3 grid, dim3 block, unsigned int sharedMemBytes, void **args, CHIPKernel *kernel)
- hipError_t launchWithExtraParams (dim3 grid, dim3 block, unsigned int sharedMemBytes, void **extra, CHIPKernel *kernel)

Protected Attributes

- std::mutex **mtx**
- int **priority**
- unsigned int **flags**
- CHIPDevice * **chip_device**

  Device on which this queue will execute.
- CHIPContext * **chip_context**

  Context to which device belongs to.

4.22.1 Detailed Description

Queue class for submitting kernels to for execution.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 CHIPQueue() [1/3]

```
CHIPQueue::CHIPQueue (
            CHIPDevice * chip_dev )
```

Construct a new CHIPQueue object.

Parameters

chip_dev	

4.22.2.2 CHIPQueue() [2/3]

```
CHIPQueue::CHIPQueue (
            CHIPDevice * chip_dev,
            unsigned int flags )
```

Construct a new CHIPQueue object.

Parameters

chip_dev	
flags	

4.22.2.3 CHIPQueue() [3/3]

```
CHIPQueue::CHIPQueue (
            CHIPDevice * chip_dev,
            unsigned int flags,
            int priority )
```

Construct a new CHIPQueue object.

Parameters

chip_dev	
flags	
priority	

4.22.3 Member Function Documentation

4.22.3.1 addCallback()

```
bool CHIPQueue::addCallback (
            hipStreamCallback_t callback,
            void * userData )
```

Add a callback funciton to be called on the host after the specified stream is done.

Parameters

callback	function pointer for a ballback function
userData	

Returns

true

false

4.22.3.2 enqueueBarrierForEvent()

```
bool CHIPQueue::enqueueBarrierForEvent (
            CHIPEvent * e )
```

Insert an event into this queue.

Parameters

e	

Returns

true
false

4.22.3.3 finish()

```
void CHIPQueue::finish ( )  [virtual]
```

Wait for this queue to finish.

Reimplemented in CHIPQueueOpenCL.

4.22.3.4 getDevice()

```
CHIPDevice * CHIPQueue::getDevice ( )
```

Get the Device obj.

Returns

CHIPDevice*

4.22.3.5 getFlags()

```
unsigned int CHIPQueue::getFlags ( )
```

Get the Flags object with which this queue was created.

Returns

unsigned int

4.22.3.6 getPriority()

```
int CHIPQueue::getPriority ( )
```

Get the Priority object with which this queue was created.

Returns

int

4.22.3.7 getPriorityRange()

```
int CHIPQueue::getPriorityRange (
            int lower_or_upper )
```

Get the Priority Range object defining the bounds for hipStreamCreateWithPriority.

Parameters

lower_or_upper	0 to get lower bound, 1 to get upper bound

Returns

int bound

4.22.3.8 launch()

```
hipError_t CHIPQueue::launch (
            CHIPExecItem * exec_item )  [virtual]
```

Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.

Parameters

exec_item	

Returns

hipError_t

Reimplemented in CHIPQueueLevel0, and CHIPQueueOpenCL.

4.22.3.9 launchHostFunc()

```
bool CHIPQueue::launchHostFunc (
            const void * hostFunction,
            dim3 numBlocks,
            dim3 dimBlocks,
            void ** args,
            size_t sharedMemBytes )
```

Launch a kernel on this queue given a host pointer and arguments.

Parameters

hostFunction	
numBlocks	
dimBlocks	
args	
sharedMemBytes	

Returns

true

false

4.22.3.10 launchWithExtraParams()

```
hipError_t CHIPQueue::launchWithExtraParams (
             dim3 grid,
             dim3 block,
             unsigned int sharedMemBytes,
             void ** extra,
             CHIPKernel * kernel )
```

Parameters

grid	
block	
sharedMemBytes	
extra	
kernel	

Returns

hipError_t

4.22.3.11 launchWithKernelParams()

```
hipError_t CHIPQueue::launchWithKernelParams (
             dim3 grid,
             dim3 block,
             unsigned int sharedMemBytes,
             void ** args,
             CHIPKernel * kernel )
```

Parameters

grid	
block	
sharedMemBytes	
args	
kernel	

Returns

hipError_t

4.22.3.12 memCopy()

```
hipError_t CHIPQueue::memCopy (
            void * dst,
            const void * src,
            size_t size )  [virtual]
```

Blocking memory copy.

Parameters

dst	Destination
src	Source
size	Transfer size

Returns

hipError_t

Reimplemented in CHIPQueueLevel0, and CHIPQueueOpenCL.

4.22.3.13 memCopyAsync()

```
hipError_t CHIPQueue::memCopyAsync (
            void * dst,
            const void * src,
            size_t size )  [virtual]
```

Non-blocking memory copy.

Parameters

dst	Destination
src	Source
size	Transfer size

Returns

hipError_t

4.22.3.14 memFill()

```
void CHIPQueue::memFill (
            void * dst,
```

```
            size_t size,
            const void * pattern,
            size_t pattern_size )  [virtual]
```

Blocking memset.

Parameters

dst	
size	
pattern	
pattern_size	

4.22.3.15 memFillAsync()

```
void CHIPQueue::memFillAsync (
            void * dst,
            size_t size,
            const void * pattern,
            size_t pattern_size )  [virtual]
```

Non-blocking mem set.

Parameters

dst	
size	
pattern	
pattern_size	

4.22.3.16 memPrefetch()

```
bool CHIPQueue::memPrefetch (
            const void * ptr,
            size_t count )
```

Insert a memory prefetch.

Parameters

ptr	
count	

Returns

true

false

4.22.3.17 query()

```
bool CHIPQueue::query ( )
```

Check if the queue is still actively executing.

Returns

true

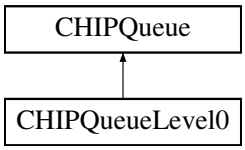
false

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh
- /Users/pvelesko/local/HIPxx/src/CHIPBackend.cc

4.23 CHIPQueueLevel0 Class Reference

Inheritance diagram for CHIPQueueLevel0:



Public Member Functions

- **CHIPQueueLevel0** (CHIPDeviceLevel0 *hixx_dev_)
- virtual hipError_t launch (CHIPExecItem *exec_item) override

  Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.
- ze_command_queue_handle_t **get** ()
- virtual hipError_t memCopy (void *dst, const void *src, size_t size) override

  Blocking memory copy.

Protected Attributes

- ze_command_queue_handle_t **ze_q**
- ze_context_handle_t **ze_ctx**
- ze_device_handle_t **ze_dev**

4.23.1 Member Function Documentation

4.23.1.1 launch()

```
virtual hipError_t CHIPQueueLevel0::launch (
            CHIPExecItem * exec_item )  [inline], [override], [virtual]
```

Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.

Parameters

exec_item	

Returns

hipError_t

Reimplemented from CHIPQueue.

4.23.1.2 memCopy()

```
hipError_t CHIPQueueLevel0::memCopy (
             void * dst,
             const void * src,
             size_t size )  [override], [virtual]
```

Blocking memory copy.

Parameters

dst	Destination
src	Source
size	Transfer size

Returns

hipError_t

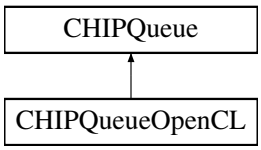
Reimplemented from CHIPQueue.

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.hh
- /Users/pvelesko/local/HIPxx/src/backend/Level0/Level0Backend.cc

4.24 CHIPQueueOpenCL Class Reference

Inheritance diagram for CHIPQueueOpenCL:



Public Member Functions

- **CHIPQueueOpenCL** (const CHIPQueueOpenCL &)=delete
- **CHIPQueueOpenCL** (CHIPDevice *chip_device)
- virtual hipError_t launch (CHIPExecItem *exec_item) override

  Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.
- virtual void finish () override

  Wait for this queue to finish.
- virtual hipError_t memCopy (void *dst, const void *src, size_t size) override

  Blocking memory copy.
- cl::CommandQueue * **get** ()

Protected Attributes

- cl::Context * **cl_ctx**
- cl::Device * **cl_dev**
- cl::CommandQueue * **cl_q**

4.24.1 Member Function Documentation

4.24.1.1 finish()

```
void CHIPQueueOpenCL::finish ( )  [override], [virtual]
```

Wait for this queue to finish.

Reimplemented from CHIPQueue.

4.24.1.2 launch()

```
hipError_t CHIPQueueOpenCL::launch (
            CHIPExecItem * exec_item )  [override], [virtual]
```

Submit a CHIPExecItem to this queue for execution. CHIPExecItem needs to be complete - contain the kernel and arguments.

Parameters

exec_item	

Returns

hipError_t

Reimplemented from CHIPQueue.

4.24.1.3 memCopy()

```
hipError_t CHIPQueueOpenCL::memCopy (
            void * dst,
            const void * src,
            size_t size )  [override], [virtual]
```

Blocking memory copy.

Parameters

dst	Destination
src	Source
size	Transfer size

Returns

hipError_t

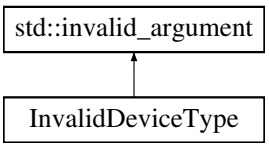
Reimplemented from CHIPQueue.

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.cc

4.25 InvalidDeviceType Class Reference

Inheritance diagram for InvalidDeviceType:

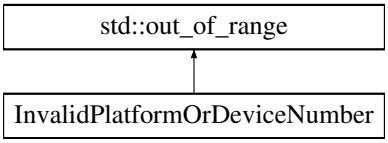


The documentation for this class was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/exceptions.hh

4.26 InvalidPlatformOrDeviceNumber Class Reference

Inheritance diagram for InvalidPlatformOrDeviceNumber:



The documentation for this class was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/exceptions.hh

4.27 OCLArgTypeInfo Struct Reference

Public Attributes

- OCLType **type**
- OCLSpace **space**
- size_t **size**

The documentation for this struct was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/common.hh

4.28 OCLFuncInfo Struct Reference

Public Attributes

- std::vector< OCLArgTypeInfo > **ArgTypeInfo**
- OCLArgTypeInfo **retTypeInfo**

The documentation for this struct was generated from the following file:

- /Users/pvelesko/local/HIPxx/src/common.hh

4.29 SVMemoryRegion Class Reference

Public Member Functions

- void **init** (cl::Context &C)
- SVMemoryRegion & **operator=** (SVMemoryRegion &&rhs)
- void * **allocate** (cl::Context ctx, size_t size)
- bool **free** (void *p, size_t *size)
- bool **hasPointer** (const void *p)
- bool **pointerSize** (void *ptr, size_t *size)
- bool **pointerInfo** (void *ptr, void **pbase, size_t *psize)
- int **memCopy** (void *dst, const void *src, size_t size, cl::CommandQueue &queue)
- int **memFill** (void *dst, size_t size, const void *pattern, size_t patt_size, cl::CommandQueue &queue)
- void **clear** ()

The documentation for this class was generated from the following files:

- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/CHIPBackendOpenCL.hh
- /Users/pvelesko/local/HIPxx/src/backend/OpenCL/SVMemoryRegion.cc

Chapter 5

File Documentation

5.1 backends.hh

```
1 #ifndef CHIP_BACKENDS_H
2 #define CHIP_BACKENDS_H
3
4 #include "Level0/Level0Backend.hh"
5 #include "OpenCL/CHIPBackendOpenCL.hh"
6
7 #endif
```

5.2 Level0Backend.hh

```
1 #ifndef CHIP_BACKEND_LEVEL0_H
2 #define CHIP_BACKEND_LEVEL0_H
3
4 #include "../src/common.hh"
5 #include "../../CHIPBackend.hh"
6 #include "../include/ze_api.h"
7 enum class LZMemoryType : unsigned { Host = 0, Device = 1, Shared = 2 };
8
9 const char* lzResultToString(ze_result_t status);
10
11 #define LZ_LOG_ERROR(msg, status)                                          \
12   logError("{} ({}) in {}:{}:{}\n", msg, lzResultToString(status), __FILE__, \
13            __LINE__, __func__)
14
15 #define LZ_PROCESS_ERROR_MSG(msg, status)                                  \
16   do {                                                                     \
17     if (status != ZE_RESULT_SUCCESS && status != ZE_RESULT_NOT_READY) {    \
18       LZ_LOG_ERROR(msg, status);                                           \
19       throw status;                                                        \
20     }                                                                      \
21   } while (0)
22
23 #define LZ_PROCESS_ERROR(status) \
24   LZ_PROCESS_ERROR_MSG("Level Zero Error", status)
25
26 #define LZ_RETURN_ERROR_MSG(msg, status)                                   \
27   do {                                                                     \
28     if (status != ZE_RESULT_SUCCESS && status != ZE_RESULT_NOT_READY) {    \
29       LZ_LOG_ERROR(msg, status);                                           \
30       return lzConvertResult(status);                                      \
31     }                                                                      \
32   } while (0)
33
34 #define HIP_LOG_ERROR(msg, status)                                         \
35   logError("{} ({}) in {}:{}:{}\n", msg, hipGetErrorName(status), __FILE__, \
36            __LINE__, __func__)
37
38 #define HIP_PROCESS_ERROR_MSG(msg, status)                   \
39   do {                                                       \
40     if (status != hipSuccess && status != hipErrorNotReady) { \
41       HIP_LOG_ERROR(msg, status);                            \
42       throw status;                                          \
43     }                                                        \
44   } while (0)
```

```
45
46 #define HIP_PROCESS_ERROR(status) HIP_PROCESS_ERROR_MSG("HIP Error", status)
47
48 #define HIP_RETURN_ERROR(status)                                \
49   HIP_RETURN_ERROR_MSG("HIP Error", status)                     \
50   if (status != hipSuccess && status != hipErrorNotReady) {     \
51     HIP_LOG_ERROR(msg, status);                                 \
52     return status;                                              \
53   }                                                             \
54   }                                                             \
55   while (0)
56 class CHIPContextLevel0;
57 class CHIPDeviceLevel0;
58 class CHIPKernelLevel0 : public CHIPKernel {
59  protected:
60   ze_kernel_handle_t ze_kernel;
61
62  public:
63   CHIPKernelLevel0(){};
64   CHIPKernelLevel0(ze_kernel_handle_t _ze_kernel, std::string _funcName,
65                    const void* _host_ptr) {
66     ze_kernel = _ze_kernel;
67     host_f_name = _funcName;
68     host_f_ptr = _host_ptr;
69     logTrace("CHIPKernelLevel0 constructor via ze_kernel_handle");
70   }
71
72   ze_kernel_handle_t get() { return ze_kernel; }
73 };
74
75 class CHIPQueueLevel0 : public CHIPQueue {
76  protected:
77   ze_command_queue_handle_t ze_q;
78   ze_context_handle_t ze_ctx;
79   ze_device_handle_t ze_dev;
80
81  public:
82   CHIPQueueLevel0(CHIPDeviceLevel0* hixx_dev_);
83
84   virtual hipError_t launch(CHIPExecItem* exec_item) override {
85     logWarn("CHIPQueueLevel0.launch() not yet implemented");
86     return hipSuccess;
87   };
88
89   ze_command_queue_handle_t get() { return ze_q; }
90
91   virtual hipError_t memCopy(void* dst, const void* src, size_t size) override;
92 };
93
94 class CHIPDeviceLevel0 : public CHIPDevice {
95   ze_device_handle_t ze_dev;
96   ze_context_handle_t ze_ctx;
97
98  public:
99   CHIPDeviceLevel0(ze_device_handle_t&& ze_dev_, ze_context_handle_t ze_ctx_)
100       : ze_dev(ze_dev_), ze_ctx(ze_ctx_) {}
101   virtual void populateDeviceProperties() override {
102     logWarn(
103         "CHIPDeviceLevel0.populate_device_properties not yet "
104         "implemented");
105   }
106   virtual std::string getName() override { return device_name; }
107   ze_device_handle_t& get() { return ze_dev; }
108
109   virtual void reset() override;
110 };
111
112 class CHIPContextLevel0 : public CHIPContext {
113   ze_context_handle_t ze_ctx;
114   OpenCLFunctionInfoMap FuncInfos;
115
116  public:
117   ze_command_list_handle_t ze_cmd_list;
118   ze_command_list_handle_t get_cmd_list() { return ze_cmd_list; }
119   CHIPContextLevel0(ze_context_handle_t&& _ze_ctx) : ze_ctx(_ze_ctx) {}
120
121   void* allocate_(size_t size, size_t alignment,
122                   CHIPMemoryType memTy) override {
123     alignment = 0x1000;  // TODO Where/why
124     void* ptr = 0;
125     if (memTy == CHIPMemoryType::Shared) {
126       ze_device_mem_alloc_desc_t dmaDesc;
127       dmaDesc.stype = ZE_STRUCTURE_TYPE_DEVICE_MEM_ALLOC_DESC;
128       dmaDesc.pNext = NULL;
129       dmaDesc.flags = 0;
130       dmaDesc.ordinal = 0;
131       ze_host_mem_alloc_desc_t hmaDesc;
```

```
132       hmaDesc.stype = ZE_STRUCTURE_TYPE_HOST_MEM_ALLOC_DESC;
133       hmaDesc.pNext = NULL;
134       hmaDesc.flags = 0;
135
136       // TODO Check if devices support cross-device sharing?
137       ze_device_handle_t ze_dev = ((CHIPDeviceLevel0*)getDevices()[0])->get();
138       ze_dev = nullptr;  // Do not associate allocation
139
140       ze_result_t status = zeMemAllocShared(ze_ctx, &dmaDesc, &hmaDesc, size,
141                                             alignment, ze_dev, &ptr);
142
143       // LZ_PROCESS_ERROR_MSG(
144       //     "HipLZ could not allocate shared memory with error code:
145       //     ", status);
146       logDebug("LZ MEMORY ALLOCATE via calling zeMemAllocShared {} ", status);
147
148       return ptr;
149     } else if (memTy == CHIPMemoryType::Device) {
150       ze_device_mem_alloc_desc_t dmaDesc;
151       dmaDesc.stype = ZE_STRUCTURE_TYPE_DEVICE_MEM_ALLOC_DESC;
152       dmaDesc.pNext = NULL;
153       dmaDesc.flags = 0;
154       dmaDesc.ordinal = 0;
155
156       // TODO Select proper device
157       ze_device_handle_t ze_dev = ((CHIPDeviceLevel0*)getDevices()[0])->get();
158
159       ze_result_t status =
160           zeMemAllocDevice(ze_ctx, &dmaDesc, size, alignment, ze_dev, &ptr);
161       LZ_PROCESS_ERROR_MSG(
162           "HipLZ could not allocate device memory with error code: ", status);
163       logDebug("LZ MEMORY ALLOCATE via calling zeMemAllocDevice {} ", status);
164
165       return ptr;
166     }
167
168     // HIP_PROCESS_ERROR_MSG("HipLZ could not recognize allocation
169     // options",
170     //                       hipErrorNotSupported);
171     return nullptr;
172   }
173
174   void free_(void* ptr) override{};  // TODO
175   ze_context_handle_t& get() { return ze_ctx; }
176   virtual hipError_t memCopy(void* dst, const void* src, size_t size,
177                              hipStream_t stream) override;
178
179   // virtual bool registerFunctionAsKernel(std::string* module_str,
180   //                                       const void* HostFunctionPtr,
181   //                                       const char* FunctionName) override {
182   //   logWarn(
183   //       "CHIPContextLevel0.register_function_as_kernel not "
184   //       "implemented");
185   //   logDebug("CHIPContextLevel0.register_function_as_kernel {} ",
186   //            FunctionName);
187   //   uint8_t* funcIL = (uint8_t*)module_str->data();
188   //   size_t ilSize = module_str->length();
189   //   std::string funcName = FunctionName;
190
191   //   // Parse the SPIR-V fat binary to retrieve kernel function
192   //   size_t numWords = ilSize / 4;
193   //   int32_t* binarydata = new int32_t[numWords + 1];
194   //   std::memcpy(binarydata, funcIL, ilSize);
195   //   // Extract kernel function information
196   //   bool res = parseSPIR(binarydata, numWords, FuncInfos);
197   //   delete[] binarydata;
198   //   if (!res) {
199   //     logError("SPIR-V parsing failed\n");
200   //     return false;
201   //   }
202
203   //   logDebug("LZ PARSE SPIR {} ", funcName);
204   //   ze_module_handle_t ze_module;
205   //   // Create module with global address aware
206   //   std::string compilerOptions =
207   //       " -cl-std=CL2.0 -cl-take-global-address -cl-match-sincospi";
208   //   ze_module_desc_t moduleDesc = {ZE_STRUCTURE_TYPE_MODULE_DESC,
209   //                                  nullptr,
210   //                                  ZE_MODULE_FORMAT_IL_SPIRV,
211   //                                  ilSize,
212   //                                  funcIL,
213   //                                  compilerOptions.c_str(),
214   //                                  nullptr};
215   //   for (CHIPDevice* chip_dev : getDevices()) {
216   //     ze_device_handle_t ze_dev = ((CHIPDeviceLevel0*)chip_dev)->get();
217   //     ze_result_t status =
218   //         zeModuleCreate(ze_ctx, ze_dev, &moduleDesc, &ze_module, nullptr);
```

```
219   //     logDebug("LZ CREATE MODULE via calling zeModuleCreate {} ", status);
220
221   //     // Create kernel
222   //     ze_kernel_handle_t ze_kernel;
223   //     ze_kernel_desc_t kernelDesc = {ZE_STRUCTURE_TYPE_KERNEL_DESC, nullptr,
224   //                                    0,  // flags
225   //                                    funcName.c_str()};
226   //     status = zeKernelCreate(ze_module, &kernelDesc, &ze_kernel);
227
228   //     // LZ_PROCESS_ERROR_MSG("HipLZ zeKernelCreate FAILED with return
229   //     // code ", status);
230
231   //     logDebug("LZ KERNEL CREATION via calling zeKernelCreate {} ", status);
232   //     CHIPKernelLevel0* chip_ze_kernel =
233   //         new CHIPKernelLevel0(ze_kernel, FunctionName, HostFunctionPtr);
234
235   //     chip_dev->addKernel(chip_ze_kernel);
236   //   }
237
238   //   return true;
239   // }
240 };  // CHIPContextLevel0
241
242 class CHIPBackendLevel0 : public CHIPBackend {
243  public:
244   virtual void initialize(std::string CHIPPlatformStr,
245                           std::string CHIPDeviceTypeStr,
246                           std::string CHIPDeviceStr) override {
247     logDebug("CHIPBackendLevel0 Initialize");
248     ze_result_t status;
249     status = zeInit(0);
250     logDebug("INITIALIZE LEVEL-0 (via calling zeInit) {}\n", status);
251
252     ze_device_type_t ze_device_type;
253     if (!CHIPDeviceTypeStr.compare("GPU")) {
254       ze_device_type = ZE_DEVICE_TYPE_GPU;
255     } else if (!CHIPDeviceTypeStr.compare("FPGA")) {
256       ze_device_type = ZE_DEVICE_TYPE_FPGA;
257     } else {
258       logCritical("CHIP_DEVICE_TYPE must be either GPU or FPGA");
259     }
260     int platform_idx = std::atoi(CHIPPlatformStr.c_str());
261     std::vector<ze_driver_handle_t> ze_drivers;
262     std::vector<ze_device_handle_t> ze_devices;
263
264     // Get number of drivers
265     uint32_t driverCount = 0, deviceCount = 0;
266     status = zeDriverGet(&driverCount, nullptr);
267     logDebug("Found Level0 Drivers: {}", driverCount);
268     // Resize and fill ze_driver vector with drivers
269     ze_drivers.resize(driverCount);
270     status = zeDriverGet(&driverCount, ze_drivers.data());
271
272     // TODO Allow for multilpe platforms(drivers)
273     // TODO Check platform ID is not the same as OpenCL. You can have
274     // two OCL platforms but only one level0 driver
275     ze_driver_handle_t ze_driver = ze_drivers[platform_idx];
276
277     assert(ze_driver != nullptr);
278     // Load devices to device vector
279     zeDeviceGet(ze_driver, &deviceCount, nullptr);
280     ze_devices.resize(deviceCount);
281     zeDeviceGet(ze_driver, &deviceCount, ze_devices.data());
282
283     const ze_context_desc_t ctxDesc = {ZE_STRUCTURE_TYPE_CONTEXT_DESC, nullptr,
284                                        0};
285
286     ze_context_handle_t ze_ctx;
287     zeContextCreateEx(ze_driver, &ctxDesc, deviceCount, ze_devices.data(),
288                       &ze_ctx);
289     CHIPContextLevel0* chip_l0_ctx = new CHIPContextLevel0(std::move(ze_ctx));
290
291     // Filter in only devices of selected type and add them to the
292     // backend as derivates of CHIPDevice
293     for (int i = 0; i < deviceCount; i++) {
294       auto dev = ze_devices[i];
295       ze_device_properties_t device_properties;
296       zeDeviceGetProperties(dev, &device_properties);
297       if (ze_device_type == device_properties.type) {
298         CHIPDeviceLevel0* chip_l0_dev =
299             new CHIPDeviceLevel0(std::move(dev), ze_ctx);
300         Backend->addDevice(chip_l0_dev);
301         // TODO
302         break;  // For now don't add more than one device
303       }
304     }  // End adding CHIPDevices
305
```

```
306     Backend->addContext(chip_l0_ctx);
307   }
308
309   virtual void initialize() override {
310     std::string empty;
311     initialize(empty, empty, empty);
312   }
313
314   void uninitialize() override {
315     logTrace("CHIPBackendLevel0 uninitializing");
316     logWarn("CHIPBackendLevel0->uninitialize() not implemented");
317   }
318 };  // CHIPBackendLevel0
319
320 #endif
```

5.3 CHIPBackendOpenCL.hh

```
1
12 #ifndef CHIP_BACKEND_OPENCL_H
13 #define CHIP_BACKEND_OPENCL_H
14
15 #define CL_TARGET_OPENCL_VERSION 210
16 #define CL_MINIMUM_OPENCL_VERSION 200
17 #define CL_HPP_TARGET_OPENCL_VERSION 210
18 #define CL_HPP_MINIMUM_OPENCL_VERSION 200
19
20 #include <CL/cl_ext_intel.h>
21
22 #include <CL/opencl.hpp>
23
24 #include "../../CHIPBackend.hh"
25 #include "exceptions.hh"
26 #include "spirv.hh"
27
28 class CHIPContextOpenCL;
29 class CHIPDeviceOpenCL;
30 class CHIPExecItemOpenCL;
31 class CHIPKernelOpenCL;
32 class CHIPQueueOpenCL;
33 class CHIPEventOpenCL;
34 class CHIPBackendOpenCL;
35 class CHIPModuleOpenCL;
36
37 class CHIPModuleOpenCL : public CHIPModule {
38  protected:
39   cl::Program program;
40
41  public:
42   virtual void compile(CHIPDevice *chip_dev) override;
43   cl::Program &get() { return program; }
44 };
45
46 class SVMemoryRegion {
47   // ContextMutex should be enough
48
49   std::map<void *, size_t> SvmAllocations;
50   cl::Context Context;
51
52  public:
53   void init(cl::Context &C) { Context = C; }
54   SVMemoryRegion &operator=(SVMemoryRegion &&rhs) {
55     SvmAllocations = std::move(rhs.SvmAllocations);
56     Context = std::move(rhs.Context);
57     return *this;
58   }
59
60   void *allocate(cl::Context ctx, size_t size);
61   bool free(void *p, size_t *size);
62   bool hasPointer(const void *p);
63   bool pointerSize(void *ptr, size_t *size);
64   bool pointerInfo(void *ptr, void **pbase, size_t *psize);
65   int memCopy(void *dst, const void *src, size_t size, cl::CommandQueue &queue);
66   int memFill(void *dst, size_t size, const void *pattern, size_t patt_size,
67               cl::CommandQueue &queue);
68   void clear();
69 };
70
71 class CHIPContextOpenCL : public CHIPContext {
72  public:
73   SVMemoryRegion svm_memory;
74   cl::Context *cl_ctx;
75   CHIPContextOpenCL(cl::Context *ctx_in);
```

```
76
77   void *allocate_(size_t size, size_t alignment,
78                   CHIPMemoryType mem_type) override;
79
80   void free_(void *ptr) override{};
81   virtual hipError_t memCopy(void *dst, const void *src, size_t size,
82                              hipStream_t stream) override;
83   cl::Context *get() { return cl_ctx; }
84 };
85
86 class CHIPDeviceOpenCL : public CHIPDevice {
87  public:
88   cl::Device *cl_dev;
89   cl::Context *cl_ctx;
90   CHIPDeviceOpenCL(CHIPContextOpenCL *chip_ctx, cl::Device *dev_in, int idx);
91
92   cl::Device *get() { return cl_dev; }
93
94   virtual void populateDeviceProperties() override;
95   virtual std::string getName() override;
96
97   virtual void reset() override;
98 };
99
100 class CHIPQueueOpenCL : public CHIPQueue {
101  protected:
102   // Any reason to make these private/protected?
103   cl::Context *cl_ctx;
104   cl::Device *cl_dev;
105   cl::CommandQueue *cl_q;
106
107  public:
108   CHIPQueueOpenCL() = delete;  // delete default constructor
109   CHIPQueueOpenCL(const CHIPQueueOpenCL &) = delete;
110   CHIPQueueOpenCL(CHIPDevice *chip_device);
111   ~CHIPQueueOpenCL();
112
113   virtual hipError_t launch(CHIPExecItem *exec_item) override;
114   virtual void finish() override;
115
116   virtual hipError_t memCopy(void *dst, const void *src, size_t size) override;
117   cl::CommandQueue *get() { return cl_q; }
118 };
119
120 class CHIPKernelOpenCL : public CHIPKernel {
121  private:
122   std::string name;
123   size_t TotalArgSize;
124   OCLFuncInfo *func_info;
125   cl::Kernel ocl_kernel;
126
127  public:
128   CHIPKernelOpenCL(const cl::Kernel &&cl_kernel,
129                    OpenCLFunctionInfoMap &func_info_map) {
130     ocl_kernel = cl_kernel;
131
132     int err = 0;
133     name = ocl_kernel.getInfo<CL_KERNEL_FUNCTION_NAME>(&err);
134     if (err != CL_SUCCESS) {
135       logError("clGetKernelInfo(CL_KERNEL_FUNCTION_NAME) failed: {}\n", err);
136     }
137
138     auto it = func_info_map.find(name);
139     assert(it != func_info_map.end());
140     func_info = it->second;
141
142     // TODO attributes
143     cl_uint NumArgs = ocl_kernel.getInfo<CL_KERNEL_NUM_ARGS>(&err);
144     if (err != CL_SUCCESS) {
145       logError("clGetKernelInfo(CL_KERNEL_NUM_ARGS) failed: {}\n", err);
146     }
147
148     assert(func_info->ArgTypeInfo.size() == NumArgs);
149
150     if (NumArgs > 0) {
151       logDebug("Kernel {} numArgs: {} \n", name, NumArgs);
152       logDebug("  RET_TYPE: {} {} {}\n", func_info->retTypeInfo.size,
153                (unsigned)func_info->retTypeInfo.space,
154                (unsigned)func_info->retTypeInfo.type);
155       for (auto &argty : func_info->ArgTypeInfo) {
156         logDebug("  ARG: SIZE {} SPACE {} TYPE {}\n", argty.size,
157                  (unsigned)argty.space, (unsigned)argty.type);
158         TotalArgSize += argty.size;
159       }
160     }
161   }
162
```

```
163   OCLFuncInfo *get_func_info() const { return func_info; }
164   std::string get_name() { return name; }
165   cl::Kernel get() const { return ocl_kernel; }
166   size_t getTotalArgSize() const { return TotalArgSize; };
167 };
168
169 class CHIPExecItemOpenCL : public CHIPExecItem {
170  private:
171   cl::Kernel *cl_kernel;
172
173  public:
174   OCLFuncInfo FuncInfo;
175   virtual hipError_t launch(CHIPKernel *chip_kernel) override;
176   int setup_all_args(CHIPKernelOpenCL *kernel);
177   cl::Kernel *get() { return cl_kernel; }
178 };
179
180 class CHIPBackendOpenCL : public CHIPBackend {
181  public:
182   void initialize(std::string CHIPPlatformStr, std::string CHIPDeviceTypeStr,
183                   std::string CHIPDeviceStr) override {
184     logDebug("CHIPBackendOpenCL Initialize");
185     std::vector<cl::Platform> Platforms;
186     cl_int err = cl::Platform::get(&Platforms);
187     if (err != CL_SUCCESS) {
188       logCritical("Failed to get OpenCL platforms! {}", err);
189       std::abort();
190     }
191     std::cout « "\nFound " « Platforms.size() « " OpenCL platforms:\n";
192     for (int i = 0; i < Platforms.size(); i++) {
193       std::cout « i « ". " « Platforms[i].getInfo<CL_PLATFORM_NAME>()
194                 « "\n";
195     }
196
197     std::vector<cl::Device> enabled_devices;
198     std::vector<cl::Device> Devices;
199     int selected_platform;
200     int selected_device;
201     cl_bitfield selected_dev_type = 0;
202
203     try {
204       if (!CHIPDeviceStr.compare("all")) {  // Use all devices that match type
205         selected_device = -1;
206       } else {
207         selected_device = std::stoi(CHIPDeviceStr);
208       }
209
210       // Platform index in range?
211       selected_platform = std::stoi(CHIPPlatformStr);
212       if ((selected_platform < 0) || (selected_platform >= Platforms.size()))
213         throw InvalidPlatformOrDeviceNumber(
214             "CHIP_PLATFORM: platform number out of range");
215       std::cout « "Selected Platform: " « selected_platform « ". "
216                 « Platforms[selected_platform].getInfo<CL_PLATFORM_NAME>()
217                 « "\n";
218
219       // Device  index in range?
220       err =  // Get All devices and print
221           Platforms[selected_platform].getDevices(CL_DEVICE_TYPE_ALL, &Devices);
222       for (int i = 0; i < Devices.size(); i++) {
223         std::cout « i « ". " « Devices[i].getInfo<CL_DEVICE_NAME>() « "\n";
224       }
225       if (selected_device >= Devices.size())
226         throw InvalidPlatformOrDeviceNumber(
227             "CHIP_DEVICE: device number out of range");
228       if (selected_device == -1) {  // All devices enabled
229         enabled_devices = Devices;
230         logDebug("All Devices enabled\n", "");
231       } else {
232         enabled_devices.push_back(Devices[selected_device]);
233         std::cout « "\nEnabled Devices:\n";
234         std::cout « selected_device « ". "
235                   « enabled_devices[0].getInfo<CL_DEVICE_NAME>() « "\n";
236       }
237
238       if (err != CL_SUCCESS)
239         throw InvalidPlatformOrDeviceNumber(
240             "CHIP_DEVICE: can't get devices for platform");
241
242       std::transform(CHIPDeviceTypeStr.begin(), CHIPDeviceTypeStr.end(),
243                      CHIPDeviceTypeStr.begin(), ::tolower);
244       if (CHIPDeviceTypeStr == "all")
245         selected_dev_type = CL_DEVICE_TYPE_ALL;
246       else if (CHIPDeviceTypeStr == "cpu")
247         selected_dev_type = CL_DEVICE_TYPE_CPU;
248       else if (CHIPDeviceTypeStr == "gpu")
249         selected_dev_type = CL_DEVICE_TYPE_GPU;
```

```
250       else if (CHIPDeviceTypeStr == "default")
251         selected_dev_type = CL_DEVICE_TYPE_DEFAULT;
252       else if (CHIPDeviceTypeStr == "accel")
253         selected_dev_type = CL_DEVICE_TYPE_ACCELERATOR;
254       else
255         throw InvalidDeviceType(
256             "Unknown value provided for CHIP_DEVICE_TYPE\n");
257       std::cout « "Using Devices of type " « CHIPDeviceTypeStr « "\n";
258
259     } catch (const InvalidDeviceType &e) {
260       logCritical("{}\n", e.what());
261       return;
262     } catch (const InvalidPlatformOrDeviceNumber &e) {
263       logCritical("{}\n", e.what());
264       return;
265     } catch (const std::invalid_argument &e) {
266       logCritical(
267           "Could not convert CHIP_PLATFORM or CHIP_DEVICES to a number");
268       return;
269     } catch (const std::out_of_range &e) {
270       logCritical("CHIP_PLATFORM or CHIP_DEVICES is out of range", "");
271       return;
272     }
273
274     std::vector<cl::Device> spirv_enabled_devices;
275     for (cl::Device dev : enabled_devices) {
276       std::string ver = dev.getInfo<CL_DEVICE_IL_VERSION>(&err);
277       if ((err == CL_SUCCESS) && (ver.rfind("SPIR-V_1.", 0) == 0)) {
278         spirv_enabled_devices.push_back(dev);
279       }
280     }
281
282     // TODO uncomment this once testing on SPIR-V Enabled OpenCL HW
283     // std::cout « "SPIR-V Enabled Devices: " « spirv_enabled_devices.size()
284     //          « "\n";
285     // for (int i = 0; i < spirv_enabled_devices.size(); i++) {
286     //  std::cout « i « ". "
287     //            « spirv_enabled_devices[i].getInfo<CL_DEVICE_NAME>() «
288     //            "\n";
289     //}
290
291     // Create context which has devices
292     // Create queues that have devices each of which has an associated context
293     // TODO Change this to spirv_enabled_devices
294     cl::Context *ctx = new cl::Context(enabled_devices);
295     CHIPContextOpenCL *chip_context = new CHIPContextOpenCL(ctx);
296     Backend->addContext(chip_context);
297     for (int i = 0; i < enabled_devices.size(); i++) {
298       cl::Device *dev = new cl::Device(enabled_devices[i]);
299       CHIPDeviceOpenCL *chip_dev = new CHIPDeviceOpenCL(chip_context, dev, i);
300       logDebug("CHIPDeviceOpenCL {}",
301                chip_dev->cl_dev->getInfo<CL_DEVICE_NAME>());
302       chip_dev->populateDeviceProperties();
303       Backend->addDevice(chip_dev);
304       CHIPQueueOpenCL *queue = new CHIPQueueOpenCL(chip_dev);
305       Backend->addQueue(queue);
306     }
307     std::cout « "OpenCL Context Initialized.\n";
308   };
309
310   virtual void initialize() override {
311     std::string empty;
312     initialize(empty, empty, empty);
313   }
314   void uninitialize() override {
315     logTrace("CHIPBackendOpenCL uninitializing");
316     logWarn("CHIPBackendOpenCL->uninitialize() not implemented");
317   }
318 };
319
320 class CHIPEventOpenCL : public CHIPEvent {
321  protected:
322   cl::Event *cl_event;
323 };
324
325 #endif
```

5.4 exceptions.hh

```
1 #ifndef OPENCL_EXCEPTIONS_H
2 #define OPENCL_EXCEPTIONS_H
3 class InvalidDeviceType : public std::invalid_argument {
4   using std::invalid_argument::invalid_argument;
```

```
5 };
6
7 class InvalidPlatformOrDeviceNumber : public std::out_of_range {
8   using std::out_of_range::out_of_range;
9 };
10 #endif
```

5.5 /Users/pvelesko/local/HIPxx/src/CHIPBackend.hh File Reference

CHIPBackend class definition. CHIP backends are to inherit from this base class and override desired virtual functions. Overrides for this class are expected to be minimal with primary overrides being done on lower-level classes such as CHIPContext consturctors, etc.

```
#include <algorithm>
#include <iostream>
#include <map>
#include <mutex>
#include <string>
#include <vector>
#include <stack>
#include "spirv.hh"
#include "include/hip/hip.hh"
#include "CHIPDriver.hh"
#include "logging.hh"
#include "macros.hh"
```

Classes

- struct allocation_info
- class CHIPAllocationTracker

  Class for keeping track of device allocations.
- class CHIPDeviceVar
- class CHIPEvent
- class CHIPModule

  Module abstraction. Contains global variables and kernels. Can be extracted from FatBinary or loaded at runtime. OpenCL - ClProgram Level Zero - zeModule ROCclr - amd::Program CUDA - CUmodule.
- class CHIPKernel

  Contains information about the function on the host and device.
- class CHIPExecItem

  Contains kernel arguments and a queue on which to execute. Prior to kernel launch, the arguments are setup via CHIPBackend::configureCall(). Because of this, we get the kernel last so the kernel so the launch() takes a kernel argument as opposed to queue receiving a CHIPExecItem containing the kernel and arguments.
- class CHIPDevice

  Compute device class.
- class CHIPContext

  Context class Contexts contain execution queues and are created on top of a single or multiple devices. Provides for creation of additional queues, events, and interaction with devices.
- class CHIPBackend

  Primary object to interact with the backend.
- class CHIPQueue

  Queue class for submitting kernels to for execution.

Enumerations

- enum class **CHIPMemoryType** : unsigned { **Host** = 0 , **Device** = 1 , **Shared** = 2 }
- enum class **CHIPEventType** : unsigned { **Default** = hipEventDefault , **BlockingSync** = hipEventBlocking↩ Sync , **DisableTiming** = hipEventDisableTiming , **Interprocess** = hipEventInterprocess }

5.5.1 Detailed Description

CHIPBackend class definition. CHIP backends are to inherit from this base class and override desired virtual functions. Overrides for this class are expected to be minimal with primary overrides being done on lower-level classes such as CHIPContext consturctors, etc.

Author

Paulius Velesko (pvelesko@gmail.com)

Version

0.1

Date

2021-08-19

Copyright

Copyright (c) 2021

5.6 CHIPBackend.hh

Go to the documentation of this file.

```
1
14 #ifndef CHIP_BACKEND_H
15 #define CHIP_BACKEND_H
16
17 #include <algorithm>
18 #include <iostream>
19 #include <map>
20 #include <mutex>
21 #include <string>
22 #include <vector>
23 #include <stack>
24
25 #include "spirv.hh"
26 #include "include/hip/hip.hh"
27
28 #include "CHIPDriver.hh"
29 #include "logging.hh"
30 #include "macros.hh"
31
32 enum class CHIPMemoryType : unsigned { Host = 0, Device = 1, Shared = 2 };
33 enum class CHIPEventType : unsigned {
34   Default = hipEventDefault,
35   BlockingSync = hipEventBlockingSync,
36   DisableTiming = hipEventDisableTiming,
37   Interprocess = hipEventInterprocess
38 };
39
40 struct allocation_info {
41   void* base_ptr;
42   size_t size;
43 };
```

```
44
49 class CHIPAllocationTracker {
50  private:
51   std::unordered_map<void*, void*> host_to_dev;
52   std::unordered_map<void*, void*> dev_to_host;
53
54   std::unordered_map<void*, allocation_info> dev_to_allocation_info;
55
56  public:
57   CHIPAllocationTracker();
58   ~CHIPAllocationTracker();
59
65   allocation_info* getByHostPtr(const void*);
71   allocation_info* getByDevPtr(const void*);
72 };
73
74 class CHIPDeviceVar {
75  private:
76   std::string host_var_name;
77   void* dev_ptr;
78   size_t size;
79
80  public:
81   CHIPDeviceVar(std::string host_var_name_, void* dev_ptr_, size_t size);
82   ~CHIPDeviceVar();
83
84   void* getDevAddr();
85   std::string getName();
86   size_t getSize();
87 };
88
89 // fw declares
90 class CHIPExecItem;
91 class CHIPQueue;
92 class CHIPContext;
93 class CHIPDevice;
94
95 class CHIPEvent {
96  protected:
97   std::mutex mutex;
98   event_status_e status;
104   CHIPEventType flags;
109   CHIPContext* chip_context;
110
116   CHIPEvent() = default;
117
118  public:
123   CHIPEvent(CHIPContext* ctx_, CHIPEventType flags_ = CHIPEventType::Default);
128   ~CHIPEvent();
136   virtual bool recordStream(CHIPQueue* chip_queue_);
143   virtual bool wait();
150   virtual bool isFinished();
158   virtual float getElapsedTime(CHIPEvent* other);
159 };
160
169 class CHIPModule {
170  protected:
171   std::mutex mtx;
172   // Global variables
173   std::vector<CHIPDeviceVar*> chip_vars;
174   // Kernels
175   std::vector<CHIPKernel*> chip_kernels;
177   std::string src;
178   // Kernel JIT compilation can be lazy
179   std::once_flag compiled;
180
186   CHIPModule() = default;
187
188  public:
193   ~CHIPModule();
202   CHIPModule(std::string* module_str);
208   CHIPModule(std::string&& module_str);
209
219   void addKernel(CHIPKernel* kernel);
220
226   void compileOnce(CHIPDevice* chip_dev);
238   virtual void compile(CHIPDevice* chip_dev);
246   CHIPDeviceVar* getGlobalVar(std::string name);
247
254   CHIPKernel* getKernel(std::string name);
255
261   std::vector<CHIPKernel*>& getKernels();
262
269   CHIPKernel* getKernel(const void* host_f_ptr);
270 };
271
275 class CHIPKernel {
```

```
276  protected:
282   CHIPKernel() = default;
284   std::string host_f_name;
286   const void* host_f_ptr;
288   const void* dev_f_ptr;
289
290  public:
291   ~CHIPKernel();
292
298   std::string getName();
304   const void* getHostPtr();
310   const void* getDevPtr();
311
317   void setName(std::string host_f_name_);
323   void setHostPtr(const void* host_f_ptr_);
329   void setDevPtr(const void* hev_f_ptr_);
330 };
331
340 class CHIPExecItem {
341  protected:
342   size_t shared_mem;
343   std::vector<uint8_t> arg_data;
344   std::vector<std::tuple<size_t, size_t» offset_sizes;
345
346   dim3 grid_dim;
347   dim3 block_dim;
348
349   CHIPQueue* stream;
350   CHIPKernel* chip_kernel;
351   CHIPQueue* chip_queue;
352
353  public:
359   CHIPExecItem() = delete;
368   CHIPExecItem(dim3 grid_dim_, dim3 block_dim_, size_t shared_mem_,
369                hipStream_t chip_queue_);
370
375   ~CHIPExecItem();
376
382   CHIPKernel* getKernel();
388   CHIPQueue* getQueue();
389
395   dim3 getGrid();
396
402   dim3 getBlock();
403
412   void setArg(const void* arg, size_t size, size_t offset);
413
421   virtual hipError_t launch(CHIPKernel* Kernel);
422
430   hipError_t launchByHostPtr(const void* hostPtr);
431 };
432
436 class CHIPDevice {
437  protected:
438   std::string device_name;
439   std::mutex mtx;
440   std::vector<CHIPKernel*> chip_kernels;
441   CHIPContext* ctx;
442   std::vector<CHIPQueue*> chip_queues;
443   int active_queue_id = 0;
444
445   hipDeviceAttribute_t attrs;
446   hipDeviceProp_t hip_device_props;
447
448  public:
450   std::vector<std::string*> modules_str;
452   std::vector<CHIPModule*> chip_modules;
453
455   std::unordered_map<const void*, std::string*> host_f_ptr_to_module_str_map;
457   std::unordered_map<const void*, CHIPModule*> host_f_ptr_to_chipmodule_map;
459   std::unordered_map<const void*, std::string> host_f_ptr_to_host_f_name_map;
461   std::unordered_map<const void*, CHIPKernel*> host_ptr_to_chipkernel_map;
464   std::unordered_map<const void*, CHIPDeviceVar*>
465       host_var_ptr_to_chipdevicevar_stat;
468   std::unordered_map<const void*, CHIPDeviceVar*>
469       host_var_ptr_to_chipdevicevar_dyn;
470
471   int idx;
472   size_t total_used_mem, max_used_mem;
477   CHIPDevice();
482   ~CHIPDevice();
483
489   std::vector<CHIPKernel*>& getKernels();
490
495   virtual void populateDeviceProperties() = 0;
501   void copyDeviceProperties(hipDeviceProp_t* prop);
502
```

```
509   CHIPKernel* findKernelByHostPtr(const void* hostPtr);
510
517   CHIPContext* getContext();
526   void addQueue(CHIPQueue* chip_queue_);
527
533   std::vector<CHIPQueue*> getQueues();  // TODO CHIP
540   CHIPQueue* getActiveQueue();  // TODO CHIP
548   bool removeQueue(CHIPQueue* q);  // TODO CHIP
549
556   int getDeviceId();
562   virtual std::string getName() = 0;
563
574   bool reserveMem(size_t bytes);
575
584   bool releaseMemReservation(size_t bytes);
585
591   virtual void reset() = 0;
592
600   int getAttr(hipDeviceAttribute_t attr);
601
607   size_t getGlobalMemSize();
608
614   virtual void setCacheConfig(hipFuncCache_t cfg);
615
621   virtual hipFuncCache_t getCacheConfig();
622
628   virtual void setSharedMemConfig(hipSharedMemConfig config);
629
635   virtual hipSharedMemConfig getSharedMemConfig();
636
644   virtual void setFuncCacheConfig(const void* func, hipFuncCache_t config);
645
656   bool hasPCIBusId(int pciDomainID, int pciBusID, int pciDeviceID);
657
664   int getPeerAccess(CHIPDevice* peerDevice);
665
674   hipError_t setPeerAccess(CHIPDevice* peer, int flags, bool canAccessPeer);
675
681   size_t getUsedGlobalMem();
682
689   CHIPDeviceVar* getDynGlobalVar(const void* host_var_ptr);
690
697   CHIPDeviceVar* getStatGlobalVar(const void* host_var_ptr);
698
705   CHIPDeviceVar* getGlobalVar(const void* host_var_ptr);
706
715   void registerFunctionAsKernel(std::string* module_str, const void* host_f_ptr,
716                                 const char* host_f_name);
717 };
718
725 class CHIPContext {
726  protected:
727   std::vector<CHIPDevice*> chip_devices;
728   std::vector<CHIPQueue*> chip_queues;
729   std::mutex mtx;
730
731  public:
736   CHIPContext();
741   ~CHIPContext();
742
750   bool addDevice(CHIPDevice* dev);
756   void addQueue(CHIPQueue* q);
757
763   std::vector<CHIPDevice*>& getDevices();
764
770   std::vector<CHIPQueue*>& getQueues();
771
780   hipStream_t findQueue(hipStream_t stream);
781
792   void* allocate(size_t size);
793
804   void* allocate(size_t size, CHIPMemoryType mem_type);
805
817   void* allocate(size_t size, size_t alignment, CHIPMemoryType mem_type);
818
829   virtual void* allocate_(size_t size, size_t alignment,
830                           CHIPMemoryType mem_type) = 0;
831
840   hipError_t free(void* ptr);
841
850   virtual void free_(void* ptr) = 0;
851
861   virtual hipError_t memCopy(void* dst, const void* src, size_t size,
862                              hipStream_t stream) = 0;
863
868   void finishAll();
869
```

```
879   virtual hipError_t findPointerInfo(hipDeviceptr_t* pbase, size_t* psize,
880                                      hipDeviceptr_t dptr);
881
887   unsigned int getFlags();
888
894   void setFlags(unsigned int flags);
895
901   void reset();
902
909   CHIPContext* retain();
910
917   void recordEvent(CHIPQueue* q, CHIPEvent* event);
918
926   virtual CHIPTexture* createImage(hipResourceDesc* resDesc,
927                                    hipTextureDesc* texDesc);
928 };
929
933 class CHIPBackend {
934  protected:
941   std::vector<std::string*> modules_str;
942   std::mutex mtx;
943
944   CHIPContext* active_ctx;
945   CHIPDevice* active_dev;
946   CHIPQueue* active_q;
947
948  public:
955   CHIPAllocationTracker AllocationTracker;
956   // Adds -std=c++17 requirement
957   inline static thread_local hipError_t tls_last_error = hipSuccess;
958   inline static thread_local CHIPContext* tls_active_ctx;
959
960   std::stack<CHIPExecItem*> chip_execstack;
961   std::vector<CHIPContext*> chip_contexts;
962   std::vector<CHIPQueue*> chip_queues;
963   std::vector<CHIPDevice*> chip_devices;
964
965   // TODO
966   // key for caching compiled modules. To get a cached compiled module on a
967   // particular device you must make sure that you have a module which matches
968   // the host funciton pointer and also that this module was compiled for the
969   // same device model.
970   // typedef  std::pair<const void*, std::string> ptr_dev;
971   // /**
972   //  * @brief
973   //  *
974   //  */
975   // std::unordered_map<ptr_dev, CHIPModule*> host_f_ptr_to_chipmodule_map;
976
981   CHIPBackend();
986   ~CHIPBackend();
987
995   virtual void initialize(std::string platform_str, std::string device_type_str,
996                           std::string device_ids_str);
997
1002   virtual void initialize() = 0;
1003
1008   virtual void uninitialize() = 0;
1009
1015   std::vector<CHIPQueue*>& getQueues();
1021   CHIPQueue* getActiveQueue();
1028   CHIPContext* getActiveContext();
1035   CHIPDevice* getActiveDevice();
1042   void setActiveDevice(CHIPDevice* chip_dev);
1043
1044   std::vector<CHIPDevice*>& getDevices();
1050   size_t getNumDevices();
1056   std::vector<std::string*>& getModulesStr();
1062   void addContext(CHIPContext* ctx_in);
1068   void addQueue(CHIPQueue* q_in);
1074   void addDevice(CHIPDevice* dev_in);
1080   void registerModuleStr(std::string* mod_str);
1086   void unregisterModuleStr(std::string* mod_str);
1096   hipError_t configureCall(dim3 grid, dim3 block, size_t shared, hipStream_t q);
1105   hipError_t setArg(const void* arg, size_t size, size_t offset);
1106
1117   virtual bool registerFunctionAsKernel(std::string* module_str,
1118                                         const void* host_f_ptr,
1119                                         const char* host_f_name);
1120
1127   CHIPDevice* findDeviceMatchingProps(const hipDeviceProp_t* props);
1128
1135   hipError_t addModule(CHIPModule* chip_module);
1142   hipError_t removeModule(CHIPModule* chip_module);
1143 };
1144
1148 class CHIPQueue {
```

```
1149  protected:
1150   std::mutex mtx;
1151   int priority;
1152   unsigned int flags;
1154   CHIPDevice* chip_device;
1156   CHIPContext* chip_context;
1157
1158  public:
1164   CHIPQueue(CHIPDevice* chip_dev);
1171   CHIPQueue(CHIPDevice* chip_dev, unsigned int flags);
1179   CHIPQueue(CHIPDevice* chip_dev, unsigned int flags, int priority);
1184   ~CHIPQueue();
1185
1195   virtual hipError_t memCopy(void* dst, const void* src, size_t size);
1204   virtual hipError_t memCopyAsync(void* dst, const void* src, size_t size);
1205
1214   virtual void memFill(void* dst, size_t size, const void* pattern,
1215                        size_t pattern_size);
1216
1225   virtual void memFillAsync(void* dst, size_t size, const void* pattern,
1226                             size_t pattern_size);
1227
1235   virtual hipError_t launch(CHIPExecItem* exec_item);
1236
1243   CHIPDevice* getDevice();
1249   virtual void finish();
1257   bool query();  // TODO CHIP
1266   int getPriorityRange(int lower_or_upper);  // TODO CHIP
1275   bool enqueueBarrierForEvent(CHIPEvent* e);  // TODO CHIP
1282   unsigned int getFlags();  // TODO CHIP
1289   int getPriority();  // TODO CHIP
1300   bool addCallback(hipStreamCallback_t callback, void* userData);
1310   bool memPrefetch(const void* ptr, size_t count);
1311
1323   bool launchHostFunc(const void* hostFunction, dim3 numBlocks, dim3 dimBlocks,
1324                       void** args, size_t sharedMemBytes);  // TODO CHIP
1325
1336   hipError_t launchWithKernelParams(dim3 grid, dim3 block,
1337                                     unsigned int sharedMemBytes, void** args,
1338                                     CHIPKernel* kernel);
1339
1350   hipError_t launchWithExtraParams(dim3 grid, dim3 block,
1351                                    unsigned int sharedMemBytes, void** extra,
1352                                    CHIPKernel* kernel);
1353 };
1354
1355 #endif
```

5.7 /Users/pvelesko/local/HIPxx/src/CHIPDriver.cc File Reference

Definitions of extern declared functions and objects in CHIPDriver.hh Initializing the CHIP runtime with backend selection through CHIP_BE environment variable.

```
#include "CHIPDriver.hh"
#include <string>
#include "backend/backends.hh"
```

Functions

- std::string **read_env_var** (std::string ENV_VAR)
- std::string **read_backend_selection** ()
- void **read_env_vars** (std::string &CHIPPlatformStr, std::string &CHIPDeviceTypeStr, std::string &CHIPDevice↩ Str)
- void **CHIPInitializeCallOnce** (std::string BE)

  Singleton backend initialization function called via std::call_once.
- void **CHIPInitialize** (std::string BE)

  Singleton backend initialization function outer wrapper.
- void **CHIPUninitializeCallOnce** ()

  Singleton backend uninitialization function called via std::call_once.
- void **CHIPUninitialize** ()

  Singleton backend initialization function outer wrapper.

Variables

- std::once_flag **initialized**

  Singleton backend initialization flag.
- std::once_flag **uninitialized**
- CHIPBackend * **Backend**

  Global Backend pointer through which backend-specific operations are performed.

5.7.1 Detailed Description

Definitions of extern declared functions and objects in CHIPDriver.hh Initializing the CHIP runtime with backend selection through CHIP_BE environment variable.

Author

Paulius Velesko (pvelesko@gmail.com)

Version

0.1

Date

2021-08-19

Copyright

Copyright (c) 2021

5.8 /Users/pvelesko/local/HIPxx/src/CHIPDriver.hh File Reference

Header defining global CHIP classes and functions such as CHIPBackend type pointer Backend which gets initialized at the start of execution.

```
#include <iostream>
#include <mutex>
#include "CHIPBackend.hh"
```

Functions

- void **CHIPInitialize** (std::string BE="")

  Singleton backend initialization function outer wrapper.
- void **CHIPUninitialize** ()

  Singleton backend initialization function outer wrapper.
- void **CHIPInitializeCallOnce** (std::string BE="")

  Singleton backend initialization function called via std::call_once.
- void **CHIPUninitializeCallOnce** ()

  Singleton backend uninitialization function called via std::call_once.
- std::string **read_env_var** (std::string ENV_VAR)
- std::string **read_backend_selection** ()

Variables

- CHIPBackend * **Backend**

  Global Backend pointer through which backend-specific operations are performed.
- std::once_flag **initialized**

  Singleton backend initialization flag.
- std::once_flag **uninitialized**

5.8.1 Detailed Description

Header defining global CHIP classes and functions such as CHIPBackend type pointer Backend which gets initialized at the start of execution.

Author

Paulius Velesko (pvelesko@gmail.com)

Version

0.1

Date

2021-08-19

Copyright

5.9 CHIPDriver.hh

Go to the documentation of this file.

```
1
13 #ifndef CHIP_DRIVER_H
14 #define CHIP_DRIVER_H
15 #include <iostream>
16 #include <mutex>
17
18 #include "CHIPBackend.hh"
19
25 extern CHIPBackend* Backend;
26
31 extern std::once_flag initialized;
32 extern std::once_flag uninitialized;
33
38 extern void CHIPInitialize(std::string BE = "");
39
44 extern void CHIPUninitialize();
45
50 void CHIPInitializeCallOnce(std::string BE = "");
51
56 void CHIPUninitializeCallOnce();
57
58 std::string read_env_var(std::string ENV_VAR);
59 std::string read_backend_selection();
60
61 #endif
```

5.10 common.hh

```
1 #ifndef HIP_COMMON_H
2 #define HIP_COMMON_H
3
4 #include <map>
5 #include <vector>
6 #include <stdint.h>
7 #include <string>
8
9 enum class OCLType : unsigned { POD = 0, Pointer = 1, Image = 2, Sampler = 3 };
10
11 enum class OCLSpace : unsigned {
12   Private = 0,
13   Global = 1,
14   Constant = 2,
15   Local = 3,
16   Unknown = 1000
17 };
18
19 struct OCLArgTypeInfo {
20   OCLType type;
21   OCLSpace space;
22   size_t size;
23 };
24
25 struct OCLFuncInfo {
26   std::vector<OCLArgTypeInfo> ArgTypeInfo;
27   OCLArgTypeInfo retTypeInfo;
28 };
29
30 typedef std::map<int32_t, OCLFuncInfo *> OCLFuncInfoMap;
31
32 typedef std::map<std::string, OCLFuncInfo *> OpenCLFunctionInfoMap;
33
34 bool parseSPIR(int32_t *stream, size_t numWords, OpenCLFunctionInfoMap &output);
35
36 #endif
```

5.11 logging.hh

```
1 #ifndef LOGGING_H
2 #define LOGGING_H
3
4 #include "spdlog/spdlog.h"
5 #include "spdlog/sinks/stdout_color_sinks.h"
6
7 #if !defined(SPDLOG_ACTIVE_LEVEL)
8 #define SPDLOG_ACTIVE_LEVEL SPDLOG_LEVEL_TRACE
9 #endif
10
11 extern std::once_flag SpdlogWasSetup;
12 extern void setupSpdlog();
13 extern void _setupSpdlog();
14
15 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_TRACE
16 template <typename... Args>
17 void logTrace(const char *fmt, const Args &...args) {
18   setupSpdlog();
19   spdlog::trace(fmt, std::forward<const Args>(args)...);
20 }
21 #else
22 #define logDebug(...) void(0)
23 #endif
24
25 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_DEBUG
26 template <typename... Args>
27 void logDebug(const char *fmt, const Args &...args) {
28   setupSpdlog();
29   spdlog::debug(fmt, std::forward<const Args>(args)...);
30 }
31 #else
32 #define logDebug(...) void(0)
33 #endif
34
35 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_INFO
36 template <typename... Args>
37 void logInfo(const char *fmt, const Args &...args) {
38   setupSpdlog();
39   spdlog::info(fmt, std::forward<const Args>(args)...);
40 }
41 #else
42 #define logInfo(...) void(0)
```

```
43 #endif
44
45 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_WARN
46 template <typename... Args>
47 void logWarn(const char *fmt, const Args &...args) {
48   setupSpdlog();
49   spdlog::warn(fmt, std::forward<const Args>(args)...);
50 }
51 #else
52 #define logWarn(...) void(0)
53 #endif
54
55 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_ERROR
56 template <typename... Args>
57 void logError(const char *fmt, const Args &...args) {
58   setupSpdlog();
59   spdlog::error(fmt, std::forward<const Args>(args)...);
60 }
61 #else
62 #define logError(...) void(0)
63 #endif
64
65 #if SPDLOG_ACTIVE_LEVEL <= SPDLOG_LEVEL_CRITICAL
66 template <typename... Args>
67 void logCritical(const char *fmt, const Args &...args) {
68   setupSpdlog();
69   spdlog::critical(fmt, std::forward<const Args>(args)...);
70 }
71 #else
72 #define logCritical(...) void(0)
73 #endif
74
75 #endif
```

5.12 macros.hh

```
1 #ifndef TEMP_H
2 #define TEMP_H
3
4 #define RETURN(x)                  \
5   do {                             \
6     hipError_t err = (x);          \
7     Backend->tls_last_error = err; \
8     return err;                    \
9   } while (0)
10
11 #define ERROR_IF(cond, err)                                                  \
12   if (cond) do {                                                             \
13       logError("Error {} at {}:{} code {}", err, __FILE__, __LINE__, #cond); \
14       Backend->tls_last_error = err;                                         \
15       return err;                                                            \
16   } while (0)
17
18 #endif
19
20 #define ERROR_CHECK_DEVNUM(device)                                          \
21   ERROR_IF(((device < 0) || ((size_t)device >= Backend->getNumDevices())), \
22            hipErrorInvalidDevice)
23
24 #define ERROR_CHECK_DEVHANDLE(device)                    \
25   auto I = std::find(Backend->getDevices().begin(),      \
26                      Backend->getDevices().end(), device); \
27   ERROR_IF(I == Backend->getDevices().end(), hipErrorInvalidDevice)
```

Index