OpenCL SYCL API

Generated by Doxygen 1.8.7

Mon May 26 2014 18:55:39

Contents

Chapter 1

Main Page

This is a simple C++ sequential OpenCL SYCL C++ header file to experiment with the OpenCL CL provisional specification.

For more information about OpenCL SYCL: http://www.khronos.org/opencl/sycl/

The aim of this file is mainly to define the interface of SYCL so that the specification documentation can be derived from it through tools like Doxygen or Sphinx. This explains why there are many functions and classes that are here only to do some forwarding in some inelegant way. This file is documentation driven and not implementation-style driven.

The source of this file can be found on https://github.com/amd/triSYCL and the Doxygen version of the API in http://amd.github.io/triSYCL/Doxygen/SYCL/html and http://amd.github.io/triSYCL/Doxygen/SYCL/SYCL-API-refman.pdf

Ronan.Keryell at AMD point com

Chapter 2

Todo List

Namespace cl::sycl::access

This values should be normalized to allow separate compilation with different implementations?

Class cl::sycl::accessor< dataType, dimensions, mode, target >

Implement it for images according so section 3.3.4.5

Member cl::sycl::accessor< dataType, dimensions, mode, target >::dimensionality

in the specification: store the dimension for user request

Member cl::sycl::accessor< dataType, dimensions, mode, target >::element

in the specification: store the types for user request as STL

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (id< dimensionality > Index) const

Implement the "const dataType &" version in the case the accessor is not for writing, as required by the specification

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (size_t Index) const

This is not in the specification but looks like a cool common feature. Or solving it with an implicit constructor of id<1>?

Member cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (item< dimensionality > Index) const

Add in the specification because used by HPC-GPU slide 22

Class cl::sycl::buffer< T, dimensions >

there is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Member cl::sycl::buffer< T, dimensions >::buffer (const T *start_iterator, const T *end_iterator)

Member cl::sycl::buffer< T, dimensions >::element

Extension to SYCL specification: provide pieces of STL container interface?

Class cl::sycl::device

The implementation is quite minimal for now. :-)

Class cl::sycl::gpu_selector

to be implemented

to be named device_selector::gpu instead in the specification?

Member cl::sycl::group< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::group< dims >::get (int index)

add it to the specification?

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::group< dims >::get_global_range ()

Update the specification to return a range<dims> instead of an id<>

Member cl::sycl::group< dims >::get_local_range ()

Update the specification to return a range<dims> instead of an id<>

Member cl::sycl::group< dims >::get_nr_range ()

Why the offset is not available here?

Also provide this access to the current nd_range

Member cl::sycl::group< dims >::group (const group &g)

in the specification, only provide a copy constructor. Any other constructors should be unspecified

Member cl::sycl::group< dims >::operator[] (int index)

add it to the specification?

is it supposed to be an int? A cl_int? a size_t?

Class cl::sycl::id< dims >

The definition of id and item seem completely broken in the current specification. The whole 3.4.1 is to be updated.

It would be nice to have [] working everywhere, provide both get_...() and get_...(int dim) equivalent to get_...()[int dim] Well it is already the case for item. So not needed for id? Indeed [] is mentioned in text of page 59 but not in class description.

Member cl::sycl::id< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::id< dims >::get (int index)

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::id< dims >::id ()

Add it to the specification?

Member cl::sycl::id< dims >::id (const range< dims > &r)

Is this necessary?

why in the specification id<int dims>(range<dims>global_size, range<dims> local_size) ?

Member cl::sycl::id< dims >::id (std::initializer_list< std::intptr_t > l)

Add this to the specification? Since it is said to be usable as a std::vector<>...

Member cl::sycl::id< dims >::id (std::intptr_t s)

Extension to the specification

Member cl::sycl::id< dims >::operator[] (int index)

explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

add also [] for range in the specification

is it supposed to be an int? A cl_int? a size_t?

Class cl::sycl::item< dims >

Add to the specification: get_nd_range() to be coherent with providing get_local...() and get_global...() and what about the offset?

Member cl::sycl::item< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::item< dims >::item (range< dims > global_size, range< dims > local_size)

what is the meaning of this constructor for a programmer?

Member cl::sycl::item< dims >::item (nd_range< dims > ndr)

a constructor from a nd_range too in the specification if the previous one has a meaning?

Member cl::sycl::kernel_lambda (Functor F)

This seems to have also the kernel_functor name in the specification

Class cl::sycl::nd_range< dims >

add copy constructors in the specification

Member cl::sycl::nd_range< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::nd_range< dims >::get_offset ()

get_offset() is lacking in the specification

Member cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)

It is not clear if the ParallelForFunctor is called with an id<> or with an item. Let's use id<> when called with a range<> and item<> when called with a nd_range<>

Member cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

Add an OpenMP implementation

Deal with incomplete work-groups

Implement with parallel_for_workgroup()/parallel_for_workitem()

Member cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

deal with Program

Class cl::sycl::queue

The implementation is quite minimal for now. :-)

Class cl::sycl::range< dims >

use std::size_t dims instead of int dims in the specification?

add to the norm this default parameter value?

add to the norm some way to specify an offset?

Member cl::sycl::range< dims >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::range< dims >::get (int index)

explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

add also [] for range in the specification

is it supposed to be an int? A cl_int? a size_t?

Member cl::sycl::range< dims >::range (std::initializer_list< std::intptr_t > l)

This is not the same as the range(dim1,...) constructor from the specification

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

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

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Module Documentation

7.1 Data access and storage in SYCL

Namespaces

- cl::sycl::access

Classes

- struct cl::sycl::buffer< T, dimensions >
- struct cl::sycl::accessor< dataType, dimensions, mode, target >

7.1.1 Detailed Description

7.1.2 Class Documentation

7.1.2.1 struct cl::sycl::buffer

template<typename T, int dimensions>struct cl::sycl::buffer< T, dimensions >

A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on.

In the case we initialize it from a pointer, for now we just wrap the data with boost::multi_array_ref to provide the VLA semantics without any storage.

Todo there is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Public Types

- using element = T
- using **value_type** = T

Public Member Functions

- buffer (const range< dimensions > &r)

  Create a new buffer of size.
- buffer (T ∗host_data, range< dimensions > r)
- buffer (const T ∗host_data, range< dimensions > r)

- buffer (const T ∗start_iterator, const T ∗end_iterator)

  Create a new allocated 1D buffer from the given elements.
- buffer (buffer< T, dimensions > &b)

  Create a new buffer from an old one, with a new allocation.
- template<access::mode mode, access::target target = access::global_buffer>
  accessor< T, dimensions, mode,
  target > get_access ()

  Return an accessor of the required mode.

7.1.2.1.1 Member Typedef Documentation

7.1.2.1.1.1 template<typename T, int dimensions> using **cl::sycl::buffer**< T, dimensions >**::element** = T

Todo Extension to SYCL specification: provide pieces of STL container interface?

7.1.2.1.2 Constructor & Destructor Documentation

7.1.2.1.2.1 template<typename T, int dimensions> **cl::sycl::buffer**< T, dimensions >**::buffer** (const **range**< dimensions > & *r*) [inline]

Create a new buffer of size.

Parameters

r	

7.1.2.1.2.2 template<typename T, int dimensions> **cl::sycl::buffer**< T, dimensions >**::buffer** (T ∗ *host_data,* **range**< dimensions > *r*) [inline]

Create a new buffer from

Parameters

host_data	of size
r	without further allocation

7.1.2.1.2.3 template<typename T, int dimensions> **cl::sycl::buffer**< T, dimensions >**::buffer** (const T ∗ *host_data,* **range**< dimensions > *r*) [inline]

Create a new read only buffer from

Parameters

host_data	of size
r	without further allocation

7.1.2.1.2.4 template<typename T, int dimensions> **cl::sycl::buffer**< T, dimensions >**::buffer** (const T ∗ *start_iterator,* const T ∗ *end_iterator*) [inline]

Create a new allocated 1D buffer from the given elements.

Todo

7.1.2.1.3 Member Function Documentation

7.1.2.1.3.1 template<typename T, int dimensions> template<access::mode mode, access::target target = access::global_buffer> **accessor**<T, dimensions, mode, target> **cl::sycl::buffer**< T, dimensions >**::get_access** ( ) [inline]

Return an accessor of the required mode.

Create a new sub-buffer without allocation to have separate accessors later

Parameters

M	

7.1.2.2 struct cl::sycl::accessor

template< typename dataType, size_t dimensions, access::mode mode, access::target target = access::global_buffer>struct cl↩ ::sycl::accessor< dataType, dimensions, mode, target >

The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way.

Todo Implement it for images according so section 3.3.4.5

Public Types

- using element = dataType
- using **value_type** = dataType

Public Member Functions

- accessor (buffer< dataType, dimensions > &targetBuffer)

  Create an accessor to the given buffer.
- dataType & operator[ ] (id< dimensionality > Index) const
- dataType & operator[ ] (size_t Index) const
- dataType & operator[ ] (item< dimensionality > Index) const

Static Public Attributes

- static const auto dimensionality = dimensions

7.1.2.2.1 Member Typedef Documentation

7.1.2.2.1.1 template< typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> using cl::sycl::accessor< dataType, dimensions, mode, target >::element = dataType

Todo in the specification: store the types for user request as STL

7.1.2.2.2 Member Function Documentation

7.1.2.2.2.1 template< typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[ ] (id< dimensionality > *Index*) const [inline]

Get the element specified by the given id

Todo Implement the "const dataType &" version in the case the accessor is not for writing, as required by the specification

7.1.2.2.2.2 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (size_t *Index*) const [inline]

Get the element specified by the given index in the case we are mono-dimensional

Todo This is not in the specification but looks like a cool common feature. Or solving it with an implicit constructor of id<1>?

7.1.2.2.2.3 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> dataType& cl::sycl::accessor< dataType, dimensions, mode, target >::operator[] (item< dimensionality > *Index*) const [inline]

Get the element specified by the given item

Todo Add in the specification because used by HPC-GPU slide 22

7.1.2.2.3 Member Data Documentation

7.1.2.2.3.1 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> const auto cl::sycl::accessor< dataType, dimensions, mode, target >::dimensionality = dimensions [static]

Todo in the specification: store the dimension for user request

7.2 Expressing parallelism through kernels

Classes

- struct cl::sycl::range< dims >
- struct cl::sycl::id< dims >
- struct cl::sycl::nd_range< dims >
- struct cl::sycl::item< dims >
- struct cl::sycl::group< dims >

Functions

- template<typename KernelName , typename Functor >
  Functor cl::sycl::kernel_lambda (Functor F)
- void cl::sycl::single_task (std::function< void(void)> F)
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)
- template<typename Range , typename Program , typename ParallelForFunctor >
  void cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  SYCL parallel_for_workgroup.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  SYCL parallel_for_workitem.

7.2.1 Detailed Description

7.2.2 Class Documentation

7.2.2.1 struct cl::sycl::range

template<int dims = 1>struct cl::sycl::range< dims >

A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation.

Todo use std::size_t dims instead of int dims in the specification?

Todo add to the norm this default parameter value?

Todo add to the norm some way to specify an offset?

Public Member Functions

- **range** (range< dims > &r)
- **range** (const range< dims > &r)
- range (std::initializer_list< std::intptr_t > l)
- range (std::intptr_t x)

  To have implicit conversion from 1 integer.

- range (std::intptr_t x, std::intptr_t y)

  A 2-D constructor from 2 integers.
- range (std::intptr_t x, std::intptr_t y, std::intptr_t z)

  A 3-D constructor from 3 integers.
- int get (int index)

Static Public Attributes

- static const auto dimensionality = dims

7.2.2.1.1 Constructor & Destructor Documentation

7.2.2.1.1.1 template< int dims = 1> cl::sycl::range< dims >::range (std::initializer_list< std::intptr_t > *l*) [inline]

Create a n-D range from a positive integer-like list

Todo This is not the same as the range(dim1,...) constructor from the specification

7.2.2.1.2 Member Function Documentation

7.2.2.1.2.1 template< int dims = 1> int cl::sycl::range< dims >::get (int *index*) [inline]

Return the range size in the give dimension

Todo explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

Todo add also [] for range in the specification

Todo is it supposed to be an int? A cl_int? a size_t?

7.2.2.1.3 Member Data Documentation

7.2.2.1.3.1 template< int dims = 1> const auto cl::sycl::range< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

7.2.2.2 struct cl::sycl::id

template< int dims = 1>struct cl::sycl::id< dims >

Define a multi-dimensional index, used for example to locate a work item

Todo The definition of id and item seem completely broken in the current specification. The whole 3.4.1 is to be updated.

Todo It would be nice to have [] working everywhere, provide both get_...() and get_...(int dim) equivalent to get_↩ ...()[int dim] Well it is already the case for item. So not needed for id? Indeed [] is mentioned in text of page 59 but not in class description.

Public Member Functions

- id ()
- id (const id &init)

    Create an id with the same value of another one.
- id (const range< dims > &r)
- id (std::initializer_list< std::intptr_t > l)
- id (std::intptr_t s)
- int get (int index)
- auto & operator[] (int index)

Static Public Attributes

- static const auto dimensionality = dims

7.2.2.2.1 Constructor & Destructor Documentation

7.2.2.2.1.1 template<int dims = 1> cl::sycl::id< dims >::id ( ) [inline]

Create a zero id

Todo Add it to the specification?

7.2.2.2.1.2 template<int dims = 1> cl::sycl::id< dims >::id (const range< dims > & *r*) [inline]

Create an id from a given range

Todo Is this necessary?

Todo why in the specification id<int dims>(range<dims>global_size, range<dims> local_size) ?

7.2.2.2.1.3 template<int dims = 1> cl::sycl::id< dims >::id (std::initializer_list< std::intptr_t > *l*) [inline]

Create a n-D range from a positive integer-like list

Todo Add this to the specification? Since it is said to be usable as a std::vector<>...

7.2.2.2.1.4 template<int dims = 1> cl::sycl::id< dims >::id (std::intptr_t *s*) [inline]

To have implicit conversion from 1 integer

Todo Extension to the specification

7.2.2.2.2 Member Function Documentation

7.2.2.2.2.1 template<int dims = 1> int cl::sycl::id< dims >::get (int *index*) [inline]

Return the id size in the given dimension

Todo is it supposed to be an int? A cl_int? a size_t?

7.2.2.2.2.2 template< int dims = 1 > auto& **cl::sycl::id**< dims >::operator[] (int *index*) [inline]

Return the id size in the given dimension

Todo explain in the specification (table 3.29, not only in the text) that [] works also for id, and why not range?

Todo add also [] for range in the specification

Todo is it supposed to be an int? A cl_int? a size_t?

7.2.2.2.3 Member Data Documentation

7.2.2.2.3.1 template< int dims = 1 > const auto **cl::sycl::id**< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

7.2.2.3 struct cl::sycl::nd_range

template< int dims = 1 >struct cl::sycl::nd_range< dims >

A ND-range, made by a global and local range, to specify work-group and work-item organization.

The local offset is used to translate the iteration space origin if needed.

Todo add copy constructors in the specification

Public Member Functions

- nd_range (range< dims > global_size, range< dims > local_size, id< dims > offset=id< dims >())
- range< dims > get_global_range ()

  Get the global iteration space range.
- range< dims > get_local_range ()

  Get the local part of the iteration space range.
- range< dims > get_group_range ()

  Get the range of work-groups needed to run this ND-range.
- range< dims > get_offset ()

Static Public Attributes

- static const auto dimensionality = dims

7.2.2.3.1 Constructor & Destructor Documentation

7.2.2.3.1.1 template< int dims = 1 > **cl::sycl::nd_range**< dims >::nd_range (**range**< dims > *global_size*, **range**< dims > *local_size*, **id**< dims > *offset* = **id**<dims>()) [inline]

Construct a ND-range with all the details available in OpenCL

By default use a zero offset, that is iterations start at 0

7.2.2.3.2 Member Function Documentation

7.2.2.3.2.1 template< int dims = 1 > **range**<dims> **cl::sycl::nd_range**< dims >::get_offset ( ) [inline]

Todo get_offset() is lacking in the specification

7.2.2.3.3 Member Data Documentation

7.2.2.3.3.1 template<int dims = 1> const auto **cl::sycl::nd_range**< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

7.2.2.4 struct cl::sycl::item

template<int dims = 1>struct cl::sycl::item< dims >

A SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges.

Todo Add to the specification: get_nd_range() to be coherent with providing get_local...() and get_global...() and what about the offset?

Public Member Functions

- item (range< dims > global_size, range< dims > local_size)
- item (nd_range< dims > ndr)
- int get_global (int dimension)

    Return the global coordinate in the given dimension.
- int get_local (int dimension)

    Return the local coordinate (that is in the work-group) in the given dimension.
- id< dims > get_global ()

    Get the whole global id coordinate.
- id< dims > get_local ()

    Get the whole local id coordinate (which is respective to the work-group)
- range< dims > get_global_range ()

    Get the global range where this item rely in.
- range< dims > get_local_range ()

    Get the local range (the dimension of the work-group) for this item.

Static Public Attributes

- static const auto dimensionality = dims

7.2.2.4.1 Constructor & Destructor Documentation

7.2.2.4.1.1 template<int dims = 1> **cl::sycl::item**< dims >::item (range< dims > *global_size*, range< dims > *local_size*) [inline]

Create an item from a local size and local size

Todo what is the meaning of this constructor for a programmer?

7.2.2.4.1.2 template<int dims = 1> **cl::sycl::item**< dims >::item (nd_range< dims > *ndr*) [inline]

Todo a constructor from a nd_range too in the specification if the previous one has a meaning?

7.2.2.4.2 Member Data Documentation

7.2.2.4.2.1 template<int dims = 1> const auto **cl::sycl::item**< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

7.2.2.5 struct cl::sycl::group

template<int dims = 1>struct cl::sycl::group< dims >

A group index used in a parallel_for_workitem to specify a work_group

Public Member Functions

- group (const group &g)
- id< dims > **get_group_id** ()
- range< dims > get_local_range ()
- range< dims > get_global_range ()
- nd_range< dims > get_nr_range ()
- int get (int index)
- auto & operator[] (int index)

Static Public Attributes

- static const auto dimensionality = dims

7.2.2.5.1 Constructor & Destructor Documentation

7.2.2.5.1.1 template<int dims = 1> **cl::sycl::group**< dims >::**group** (const **group**< dims > & *g*) [inline]

Todo in the specification, only provide a copy constructor. Any other constructors should be unspecified

7.2.2.5.2 Member Function Documentation

7.2.2.5.2.1 template<int dims = 1> int **cl::sycl::group**< dims >::get (int *index*) [inline]

Return the group coordinate in the given dimension

Todo add it to the specification?

Todo is it supposed to be an int? A cl_int? a size_t?

7.2.2.5.2.2 template<int dims = 1> **range**<dims> **cl::sycl::group**< dims >::get_global_range ( ) [inline]

Get the local range for this work_group

Todo Update the specification to return a range<dims> instead of an id<>

7.2.2.5.2.3 template<int dims = 1> **range**<dims> **cl::sycl::group**< dims >::get_local_range ( ) [inline]

Get the local range for this work_group

Todo Update the specification to return a range<dims> instead of an id<>

7.2.2.5.2.4 template<int dims = 1> nd_range<dims> cl::sycl::group< dims >::get_nr_range ( ) [inline]

Todo Why the offset is not available here?

Todo Also provide this access to the current nd_range

7.2.2.5.2.5 template<int dims = 1> auto& cl::sycl::group< dims >::operator[] (int *index*) [inline]

Return the group coordinate in the given dimension

Todo add it to the specification?

Todo is it supposed to be an int? A cl_int? a size_t?

7.2.2.5.3 Member Data Documentation

7.2.2.5.3.1 template<int dims = 1> const auto cl::sycl::group< dims >::dimensionality = dims [static]

Todo add this Boost::multi_array or STL concept to the specification?

7.2.3 Function Documentation

7.2.3.1 template<typename KernelName , typename Functor > Functor cl::sycl::kernel_lambda (Functor *F*)

kernel_lambda specify a kernel to be launch with a single_task or parallel_for

Todo This seems to have also the kernel_functor name in the specification

7.2.3.2 template<int Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*)

SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>.

This implementation use OpenMP 3 if compiled with the right flag.

Todo It is not clear if the ParallelForFunctor is called with an id<> or with an item. Let's use id<> when called with a range<> and item<> when called with a nd_range<>

7.2.3.3 template<int Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

A variation of SYCL parallel_for to take into account a nd_range<>

Todo Add an OpenMP implementation

Todo Deal with incomplete work-groups

Todo Implement with parallel_for_workgroup()/parallel_for_workitem()

7.2.3.4 template<typename Range , typename Program , typename ParallelForFunctor > void cl::sycl::parallel_for (Range *r,* Program *p,* ParallelForFunctor *f*)

SYCL parallel_for version that allows a Program object to be specified.

Todo deal with Program

7.2.3.5 void cl::sycl::single_task (std::function< void(void)> *F*)

SYCL single_task launches a computation without parallelism at launch time.

Right now the implementation does nothing else that forwarding the execution of the given functor

7.3 Platforms, contexts, devices and queues

Classes

- struct cl::sycl::device
- struct cl::sycl::device_selector
- struct cl::sycl::gpu_selector
- struct cl::sycl::context
- struct cl::sycl::queue
- struct cl::sycl::command_group

7.3.1 Detailed Description

7.3.2 Class Documentation

7.3.2.1 struct cl::sycl::device

SYCL device

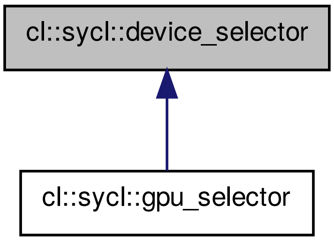
Todo The implementation is quite minimal for now. :-)

7.3.2.2 struct cl::sycl::device_selector

The SYCL heuristics to select a device

The device with the highest score is selected

Inheritance diagram for cl::sycl::device_selector:



Public Member Functions

- virtual int **operator()** (device dev)=0

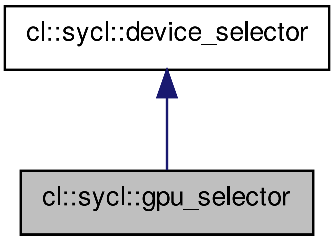
7.3.2.3 struct cl::sycl::gpu_selector

Select the best GPU, if any

Todo to be implemented

Todo to be named device_selector::gpu instead in the specification?

Inheritance diagram for cl::sycl::gpu_selector:



Public Member Functions

- int **operator()** (device dev) override

7.3.2.4 struct cl::sycl::context

SYCL context

The implementation is quite minimal for now. :-)

Public Member Functions

- **context** (gpu_selector s)
- **context** (device_selector &s)

7.3.2.5 struct cl::sycl::queue

SYCL queue, similar to the OpenCL queue concept.

Todo The implementation is quite minimal for now. :-)

Public Member Functions

- **queue** (context c)

7.3.2.6 struct cl::sycl::command_group

SYCL command group gather all the commands needed to execute one or more kernels in a kind of atomic way. Since all the parameters are captured at command group creation, one can execute the content in an asynchronous way and delayed schedule.

For now just execute the command group directly.

Public Member Functions

- template<typename Functor >
  command_group (queue Q, Functor F)

Chapter 8

Namespace Documentation

8.1 cl Namespace Reference

SYCL dwells in the cl::sycl namespace.

8.1.1 Detailed Description

SYCL dwells in the cl::sycl namespace.

8.2 cl::sycl::access Namespace Reference

Enumerations

- enum mode {
  read = 42, **write**, **atomic**, **read_write**,
  discard_read_write }

  This describes the type of the access mode to be used via accessor.
- enum target {
  global_buffer = 2014, **constant_buffer**, **local**, **image**,
  host_buffer, **host_image**, **image_array**, **cl_buffer**,
  cl_image }

8.2.1 Detailed Description

Describe the type of access by kernels.

Todo This values should be normalized to allow separate compilation with different implementations?

8.2.2 Enumeration Type Documentation

8.2.2.1 enum **cl::sycl::access::target**

The target enumeration describes the type of object to be accessed via the accessor

Chapter 9

File Documentation

9.1 include/CL/sycl.hpp File Reference

Classes

- struct cl::sycl::range< dims >
- struct cl::sycl::id< dims >
- struct cl::sycl::nd_range< dims >
- struct cl::sycl::item< dims >
- struct cl::sycl::group< dims >
- struct cl::sycl::device
- struct cl::sycl::device_selector
- struct cl::sycl::gpu_selector
- struct cl::sycl::context
- struct cl::sycl::queue
- struct cl::sycl::command_group
- struct cl::sycl::buffer< T, dimensions >
- struct cl::sycl::accessor< dataType, dimensions, mode, target >
- struct cl::sycl::buffer< T, dimensions >

Namespaces

- cl

  SYCL dwells in the cl::sycl namespace.
- cl::sycl::access

Macros

- #define **TRISYCL_IMPL**(...)

Enumerations

- enum cl::sycl::access::mode {
  read = 42, **write**, **atomic**, **read_write**,
  discard_read_write }

  This describes the type of the access mode to be used via accessor.
- enum cl::sycl::access::target {
  global_buffer = 2014, **constant_buffer**, **local**, **image**,
  host_buffer, **host_image**, **image_array**, **cl_buffer**,
  cl_image }

Functions

- template<typename KernelName , typename Functor >
  Functor cl::sycl::kernel_lambda (Functor F)
- void cl::sycl::single_task (std::function< void(void)> F)
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (range< Dimensions > r, ParallelForFunctor f)
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)
- template<typename Range , typename Program , typename ParallelForFunctor >
  void cl::sycl::parallel_for (Range r, Program p, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  SYCL parallel_for_workgroup.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  SYCL parallel_for_workitem.
- void **cl::sycl::barrier** (int barrier_type)

Variables

- int const **cl::sycl::CL_LOCAL_MEM_FENCE** = 123

Index