triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.7

Thu Jul 10 2014 10:56:53

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.

For more information on this project and to access to the source of this file, look at https://github.com/amd/triSYCL

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

The Doxygen version of the implementation itself is in http://amd.github.io/triSYCL/Doxygen/triSYCL/html and http://amd.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf

Ronan.Keryell at AMD point com

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Chapter 2

Todo List

Member __CL_ENABLE_EXCEPTIONS

Use a macro to check instead if the OpenCL header has been included before.

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

Member cl::sycl::barrier (int barrier_type)

To be implemented

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 (storage< T > &store, range< dimensions > r)

To be implemented

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

Add const to the SYCL specification

Member cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > b, id< dimensions > base_index, range< dimensions > sub_range)

To be implemented

Update the specification to replace index by id

Member cl::sycl::buffer< T, dimensions >::buffer (cl_mem mem_object, queue from_queue, event available_event)

To be implemented

Improve the specification to allow CLHPP objects too

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. :-)

Member cl::sycl::error_handler::default_handler

add this concept to the specification?

Member cl::sycl::error_handler::report_error (exception &error)=0

Add "virtual void" to the specification

Member cl::sycl::exception::get_buffer ()

Update specification to replace 0 by nullptr and add the templated buffer

to be implemented

Member cl::sycl::exception::get_cl_code ()

to be implemented

Member cl::sycl::exception::get_image ()

Update specification to replace 0 by nullptr and add the templated buffer

to be implemented

Member cl::sycl::exception::get_queue ()

Update specification to replace 0 by nullptr

Member cl::sycl::exception::get_sycl_code ()

to be implemented

use something else instead of cl_int to be usable without OpenCL

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::image< dimensions >

implement image

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 r, Program p, ParallelForFunctor f)

deal with Program

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()

Class cl::sycl::platform

triSYCL Implementation

Member cl::sycl::platform::get ()

Add cl.hpp version to the specification

Member cl::sycl::platform::get_info ()

It looks like in the specification the cl::detail:: is lacking to fit the cl.hpp version. Or is it to be redefined in SYCL too?

Member cl::sycl::platform::has_extension (const STRING_CLASS extension_name)

Should it be a param type instead of a STRING?

extend to any type of C++-string like object

Member cl::sycl::platform::platform (cl_platform_id platform id, const error_handler &handler=error_handler::default_handler)

improve specification to accept also a cl.hpp object

Member cl::sycl::platform::platform (const error_handler &handler=error_handler::default_handler)

Add copy/move constructor to the implementation

Add const to the specification

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

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

remove from the SYCL specification and use a range-less parallel_for version with default construction of a 1-element range?

Member cl::sycl::storage< T >::element

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

Member cl::sycl::storage< T >::get_size ()=0

This is inconsistent in the specification with get_size() in buffer which returns the byte size. Is it to be renamed to get_count()?

Member cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::operator[] (ItemImpl< dimensionality > Index) const

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

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

Member cl::sycl::trisycl::GroupImpl< N >::operator[] (int index)

add it to the specification?

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

Member cl::sycl::trisycl::ItemImpl< dims >::ItemImpl (NDRangeImpl< dims > ndr)

a constructor from a nd_range too in the specification?

Member cl::sycl::trisycl::NDRangeImpl< dims >::get_offset ()

get_offset() is lacking in the specification

Member cl::sycl::trisycl::RangeImpl< Dimensions >::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

Member STRING_CLASS

this should be more local, such as SYCL_STRING_CLASS or _SYCL_STRING_CLASS

use a typedef or a using instead of a macro?

implement __NO_STD_STRING

Table 3.2 in provisional specification is wrong: STRING_CLASS not at the right place

Member VECTOR_CLASS

this should be more local, such as SYCL_VECTOR_CLASS or _SYCL_VECTOR_CLASS

use a typedef or a using instead of a macro?

implement __NO_STD_VECTOR

Table 3.1 in provisional specification is wrong: VECTOR_CLASS not at the right place

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 namespaces with brief descriptions:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 Debugging and tracing support

Classes

- struct debug< T >

  Class used to trace the construction and destruction of classes that inherit from it. More...

8.1.1 Detailed Description

8.1.2 Class Documentation

8.1.2.1 struct debug

template<typename T>struct debug< T >

Class used to trace the construction and destruction of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Definition at line 25 of file sycl-debug.hpp.

Public Member Functions

- debug ()

  Trace the construction with the compiler-dependent mangled named.
- ~debug ()

  Trace the construction with the compiler-dependent mangled named.

8.1.2.1.1 Constructor & Destructor Documentation

8.1.2.1.1.1 template<typename T > debug< T >::debug ( ) [inline]

Trace the construction with the compiler-dependent mangled named.

Definition at line 28 of file sycl-debug.hpp.

```
00028           {
00029     std::cerr << " TRISYCL_DEBUG: Constructor of " << typeid(*this).name()
00030               << " " << (void*) this << std::endl;
00031   }
```

8.1.2.1.1.2 template<typename T > debug< T >::~debug ( ) [inline]

Trace the construction with the compiler-dependent mangled named.

Definition at line 34 of file sycl-debug.hpp.

```
00034           {
00035     std::cerr << " TRISYCL_DEBUG: ~ Destructor of " << typeid(*this).name()
00036               << " " << (void*) this << std::endl;
00037   }
```

8.2 Expressing parallelism through kernels

Classes

- struct cl::sycl::trisycl::RangeImpl< Dimensions >

  Define a multi-dimensional index range. More...
- struct cl::sycl::trisycl::IdImpl< N >

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct cl::sycl::trisycl::NDRangeImpl< dims >

  The implementation of a ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- struct cl::sycl::trisycl::ItemImpl< dims >

  The implementation of a SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct cl::sycl::trisycl::GroupImpl< N >

  The implementation of a SYCL group index to specify a work_group in a parallel_for_workitem. More...
- struct cl::sycl::trisycl::ParallelForIterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends calling f. More...
- struct cl::sycl::trisycl::ParallelOpenMPForIterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct cl::sycl::trisycl::ParallelForIterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- struct cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation. More...
- struct cl::sycl::id< dims >

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct cl::sycl::nd_range< dims >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- 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. More...
- struct cl::sycl::group< dims >

  A group index used in a parallel_for_workitem to specify a work_group. More...

Functions

- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator/ (RangeImpl< Dimensions > dividend, RangeImpl< Dimensions > divisor)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator* (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator+ (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)
- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator/ (range< Dimensions > dividend, range< Dimensions > divisor)
- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator* (range< Dimensions > a, range< Dimensions > b)
- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator+ (range< Dimensions > a, range< Dimensions > b)
- 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

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

  SYCL single_task launches a computation without parallelism at launch time.

- 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<>.

- 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<>

- 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)

  Loop on the work-groups.

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

  Loop on the work-items inside a work-group.

8.2.1 Detailed Description

8.2.2 Class Documentation

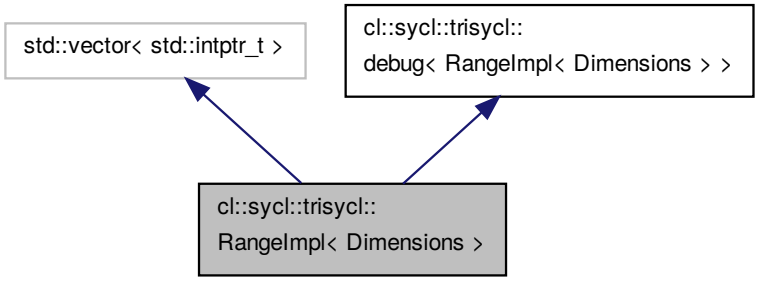
8.2.2.1 struct cl::sycl::trisycl::RangeImpl

template<std::size_t Dimensions = 1U>struct cl::sycl::trisycl::RangeImpl< Dimensions >

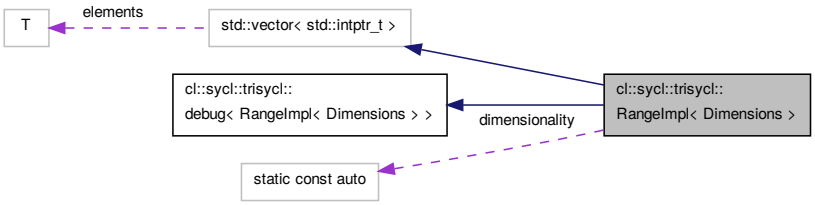
Define a multi-dimensional index range.

Definition at line 31 of file sycl-implementation.hpp.

Inheritance diagram for cl::sycl::trisycl::RangeImpl< Dimensions >:



Collaboration diagram for cl::sycl::trisycl::RangeImpl< Dimensions >:



Public Member Functions

- RangeImpl & getImpl ()
- const RangeImpl & getImpl () const
- RangeImpl ()
- RangeImpl (const RangeImpl &init)
- RangeImpl (std::initializer_list< std::intptr_t > l)
- auto get (int index)

  Return the given coordinate.
- void display ()

Static Public Attributes

- static const auto dimensionality = Dimensions

8.2.2.1.1 Constructor & Destructor Documentation

8.2.2.1.1.1 template<std::size_t Dimensions = 1U> cl::sycl::trisycl::RangeImpl< Dimensions >::RangeImpl ( ) [inline]

Definition at line 61 of file sycl-implementation.hpp.

```
00061 : vector(Dimensions) {}
```

8.2.2.1.1.2 template<std::size_t Dimensions = 1U> cl::sycl::trisycl::RangeImpl< Dimensions >::RangeImpl (const RangeImpl< Dimensions > & *init*) [inline]

Definition at line 65 of file sycl-implementation.hpp.

```
00065 : vector(init) {}
```

8.2.2.1.1.3 template<std::size_t Dimensions = 1U> cl::sycl::trisycl::RangeImpl< Dimensions >::RangeImpl (std::initializer_list< std::intptr_t > *l*) [inline]

Definition at line 69 of file sycl-implementation.hpp.

```
00069                                                :
00070     std::vector<std::intptr_t>(l) {
00071     // The number of elements must match the dimension
00072     assert(Dimensions == l.size());
00073   }
```

8.2.2.1.2 Member Function Documentation

8.2.2.1.2.1 template<std::size_t Dimensions = 1U> void cl::sycl::trisycl::RangeImpl< Dimensions >::display ( ) [inline]

Definition at line 88 of file sycl-implementation.hpp.

```
00088                 {
00089     std::clog << typeid(this).name() << ": ";
00090     for (int i = 0; i < dimensionality; i++)
00091       std::clog << " " << get(i);
00092     std::clog << std::endl;
00093   }
```

8.2.2.1.2.2 template<std::size_t Dimensions = 1U> auto cl::sycl::trisycl::RangeImpl< Dimensions >::get (int *index*) [inline]

Return the given coordinate.

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

Definition at line 83 of file sycl-implementation.hpp.

```
00083                       {
00084     return (*this)[index];
00085   }
```

8.2.2.1.2.3 template<std::size_t Dimensions = 1U> RangeImpl& cl::sycl::trisycl::RangeImpl< Dimensions >::getImpl ( ) [inline]

Definition at line 38 of file sycl-implementation.hpp.

```
00038 { return *this; };
```

8.2.2.1.2.4 template<std::size_t Dimensions = 1U> const RangeImpl& cl::sycl::trisycl::RangeImpl< Dimensions >::getImpl ( ) const [inline]

Definition at line 42 of file sycl-implementation.hpp.

```
00042 { return *this; };
```

8.2.2.1.3 Member Data Documentation

8.2.2.1.3.1 template<std::size_t Dimensions = 1U> const auto cl::sycl::trisycl::RangeImpl< Dimensions >::dimensionality = Dimensions [static]

Definition at line 35 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::RangeImpl< dims >::display().

8.2.2.2 struct cl::sycl::trisycl::IdImpl

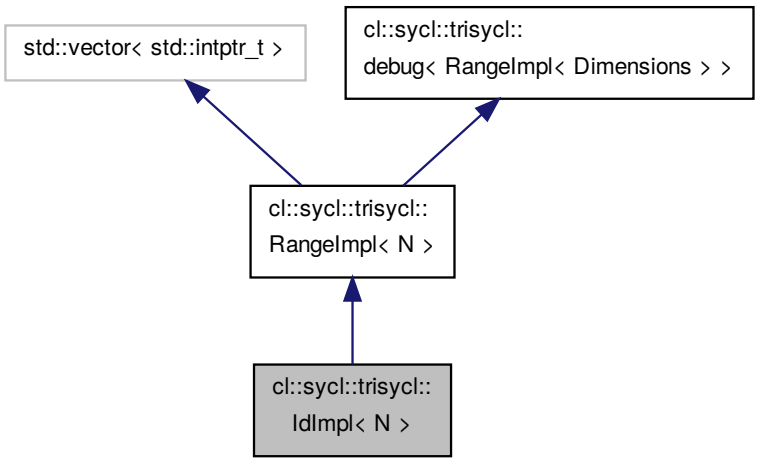
template<std::size_t N = 1U>struct cl::sycl::trisycl::IdImpl< N >

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

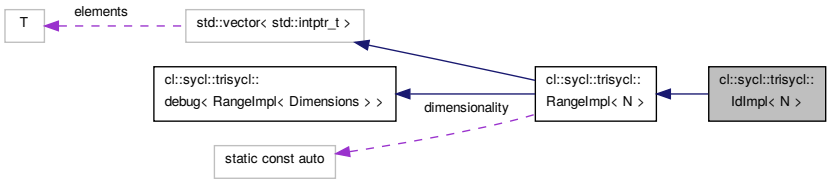
Just rely on the range implementation

Definition at line 146 of file sycl-implementation.hpp.

Inheritance diagram for cl::sycl::trisycl::IdImpl< N >:



Collaboration diagram for cl::sycl::trisycl::IdImpl< N >:



Public Member Functions

- IdImpl (const RangeImpl< N > &init)
- IdImpl ()=default

Additional Inherited Members

8.2.2.2.1 Constructor & Destructor Documentation

8.2.2.2.1.1 template<std::size_t N = 1U> cl::sycl::trisycl::IdImpl< N >::IdImpl (const RangeImpl< N > & *init*) [inline]

Definition at line 151 of file sycl-implementation.hpp.

```
00151 : RangeImpl<N>(init) {}
```

8.2.2.2.1.2 template<std::size_t N = 1U> cl::sycl::trisycl::IdImpl< N >::IdImpl ( ) [default]

8.2.2.3 struct cl::sycl::trisycl::NDRangeImpl

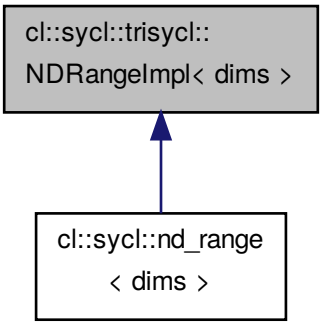
template<std::size_t dims = 1U>struct cl::sycl::trisycl::NDRangeImpl< dims >

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

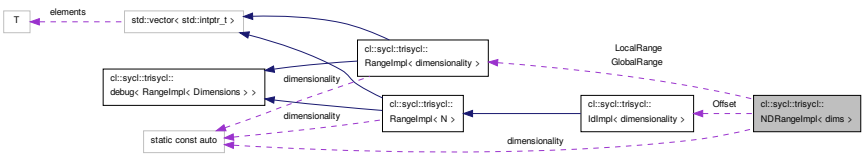
The local offset is used to translate the iteration space origin if needed.

Definition at line 166 of file sycl-implementation.hpp.

Inheritance diagram for cl::sycl::trisycl::NDRangeImpl< dims >:



Collaboration diagram for cl::sycl::trisycl::NDRangeImpl< dims >:



Public Member Functions

- NDRangeImpl (RangeImpl< dimensionality > global_size, RangeImpl< dimensionality > local_size, IdImpl< dimensionality > offset)
- NDRangeImpl & getImpl ()
- const NDRangeImpl & getImpl () const
- RangeImpl< dimensionality > get_global_range ()
- RangeImpl< dimensionality > get_local_range ()
- RangeImpl< dimensionality > get_group_range ()

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

Public Attributes

- RangeImpl< dimensionality > GlobalRange
- RangeImpl< dimensionality > LocalRange
- IdImpl< dimensionality > Offset

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.3.1 Constructor & Destructor Documentation

8.2.2.3.1.1 template<std::size_t dims = 1U> cl::sycl::trisycl::NDRangeImpl< dims >::NDRangeImpl (RangeImpl< dimensionality > *global_size*, RangeImpl< dimensionality > *local_size*, IdImpl< dimensionality > *offset*) [inline]

Definition at line 176 of file sycl-implementation.hpp.

```
00178                                                :
00179     GlobalRange(global_size),
00180     LocalRange(local_size),
00181     Offset(offset) {}
```

8.2.2.3.2 Member Function Documentation

8.2.2.3.2.1 template<std::size_t dims = 1U> RangeImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::get_global_range ( ) [inline]

Definition at line 191 of file sycl-implementation.hpp.

```
00191 { return GlobalRange; }
```

8.2.2.3.2.2 template<std::size_t dims = 1U> RangeImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::get_group_range ( ) [inline]

Get the range of work-groups needed to run this ND-range.

Definition at line 196 of file sycl-implementation.hpp.

```
00196 { return GlobalRange/LocalRange; }
```

8.2.2.3.2.3 template<std::size_t dims = 1U> RangeImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::get_local_range ( ) [inline]

Definition at line 193 of file sycl-implementation.hpp.

```
00193 { return LocalRange; }
```

8.2.2.3.2.4 template<std::size_t dims = 1U> IdImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::get_offset ( ) [inline]

Todo get_offset() is lacking in the specification

Definition at line 199 of file sycl-implementation.hpp.

```
00199 { return Offset; }
```

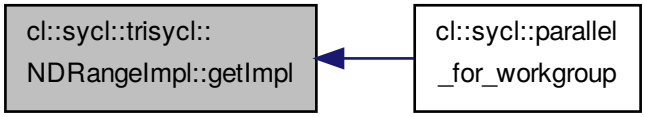
8.2.2.3.2.5 template<std::size_t dims = 1U> NDRangeImpl& cl::sycl::trisycl::NDRangeImpl< dims >::getImpl ( ) [inline]

Definition at line 184 of file sycl-implementation.hpp.

Referenced by cl::sycl::parallel_for_workgroup().

```
00184 { return *this; };
```

Here is the caller graph for this function:



8.2.2.3.2.6 template<std::size_t dims = 1U> const NDRangeImpl& cl::sycl::trisycl::NDRangeImpl< dims >::getImpl ( ) const [inline]

Definition at line 188 of file sycl-implementation.hpp.

```
00188 { return *this; };
```

8.2.2.3.3 Member Data Documentation

8.2.2.3.3.1 template<std::size_t dims = 1U> const auto cl::sycl::trisycl::NDRangeImpl< dims >::dimensionality = dims [static]

Definition at line 170 of file sycl-implementation.hpp.

8.2.2.3.3.2 template<std::size_t dims = 1U> RangeImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::GlobalRange

Definition at line 172 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::NDRangeImpl< N >::get_global_range(), cl::sycl::trisycl::GroupImpl< dims >::get_global_range(), and cl::sycl::trisycl::NDRangeImpl< N >::get_group_range().

8.2.2.3.3.3 template<std::size_t dims = 1U> RangeImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::LocalRange

Definition at line 173 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::NDRangeImpl< N >::get_group_range(), cl::sycl::trisycl::NDRangeImpl< N >::get_local_range(), and cl::sycl::trisycl::GroupImpl< dims >::get_local_range().

8.2.2.3.3.4 template<std::size_t dims = 1U> IdImpl<dimensionality> cl::sycl::trisycl::NDRangeImpl< dims >::Offset

Definition at line 174 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::NDRangeImpl< N >::get_offset().

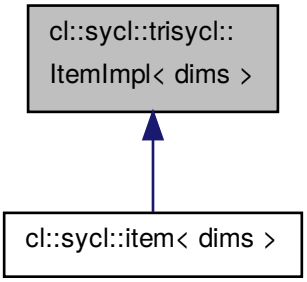
8.2.2.4 struct cl::sycl::trisycl::ItemImpl

template<std::size_t dims = 1U>struct cl::sycl::trisycl::ItemImpl< dims >

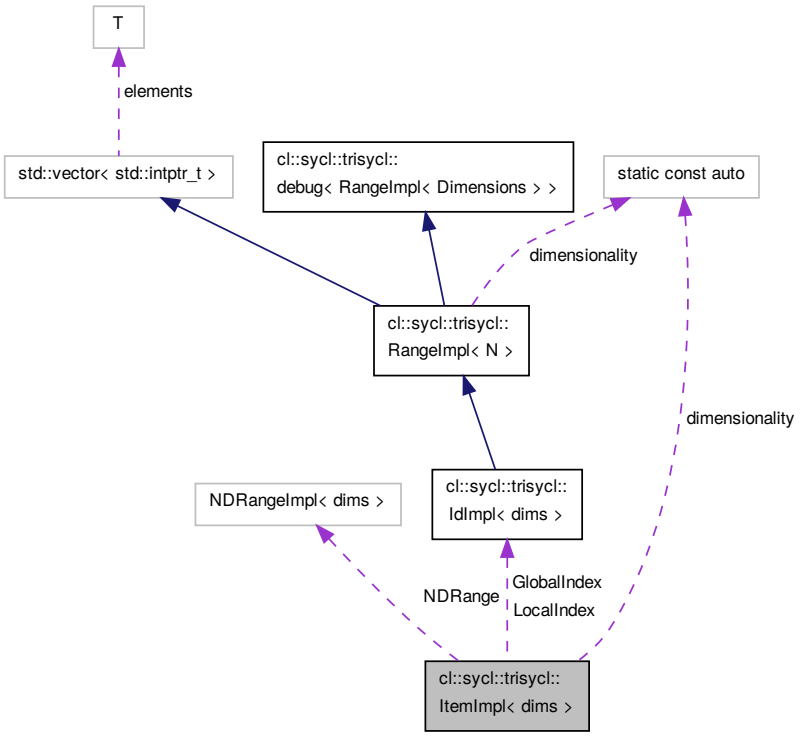
The implementation of a SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges.

Definition at line 209 of file sycl-implementation.hpp.

Inheritance diagram for cl::sycl::trisycl::ItemImpl< dims >:



Collaboration diagram for cl::sycl::trisycl::ItemImpl< dims >:



Public Member Functions

- ItemImpl (RangeImpl< dims > global_size, RangeImpl< dims > local_size)
- ItemImpl (NDRangeImpl< dims > ndr)
- auto get_global (int dimension)
- auto get_local (int dimension)
- auto get_global ()
- auto get_local ()
- void set_local (IdImpl< dims > Index)
- void set_global (IdImpl< dims > Index)
- auto get_local_range ()
- auto get_global_range ()

Public Attributes

- IdImpl< dims > GlobalIndex
- IdImpl< dims > LocalIndex
- NDRangeImpl< dims > NDRange

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.4.1 Constructor & Destructor Documentation

8.2.2.4.1.1 template<std::size_t dims = 1U> cl::sycl::trisycl::ItemImpl< dims >::ItemImpl (RangeImpl< dims > *global_size,* RangeImpl< dims > *local_size*) [inline]

Definition at line 219 of file sycl-implementation.hpp.

```
00219                                                                    :
00220     NDRange(global_size, local_size) {}
```

8.2.2.4.1.2 template<std::size_t dims = 1U> cl::sycl::trisycl::ItemImpl< dims >::ItemImpl (NDRangeImpl< dims > *ndr*) [inline]

Todo a constructor from a nd_range too in the specification?

Definition at line 223 of file sycl-implementation.hpp.

```
00223 : NDRange(ndr) {}
```

8.2.2.4.2 Member Function Documentation

8.2.2.4.2.1 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_global (int *dimension*) [inline]

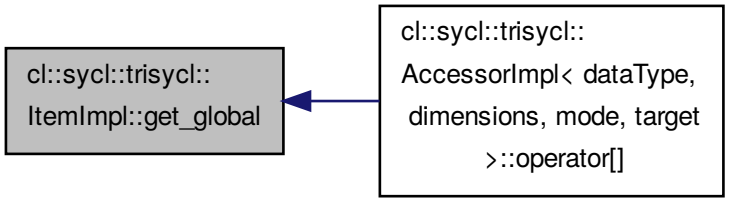
Definition at line 225 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::GlobalIndex.

Referenced by cl::sycl::trisycl::AccessorImpl< dataType, dimensions, mode, target >::operator[]().

```
00225 { return GlobalIndex[dimension]; }
```

Here is the caller graph for this function:



8.2.2.4.2.2 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_global ( ) [inline]

Definition at line 229 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::GlobalIndex.

```
00229 { return GlobalIndex; }
```

8.2.2.4.2.3 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_global_range ( ) [inline]

Definition at line 241 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::NDRange.

```
00241 { return NDRange.get_global_range(); }
```

8.2.2.4.2.4 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_local (int *dimension*) [inline]

Definition at line 227 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::LocalIndex.

```
00227 { return LocalIndex[dimension]; }
```

8.2.2.4.2.5 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_local ( ) [inline]

Definition at line 231 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::LocalIndex.

```
00231 { return LocalIndex; }
```

8.2.2.4.2.6 template<std::size_t dims = 1U> auto cl::sycl::trisycl::ItemImpl< dims >::get_local_range ( ) [inline]

Definition at line 239 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::NDRange.

```
00239 { return NDRange.get_local_range(); }
```

8.2.2.4.2.7 template<std::size_t dims = 1U> void cl::sycl::trisycl::ItemImpl< dims >::set_global (IdImpl< dims > *Index*) [inline]

Definition at line 237 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::GlobalIndex.

```
00237 { GlobalIndex = Index; }
```

8.2.2.4.2.8 template<std::size_t dims = 1U> void cl::sycl::trisycl::ItemImpl< dims >::set_local (IdImpl< dims > *Index*) [inline]

Definition at line 234 of file sycl-implementation.hpp.

References cl::sycl::trisycl::ItemImpl< dims >::LocalIndex.

```
00234 { LocalIndex = Index; }
```

8.2.2.4.3 Member Data Documentation

8.2.2.4.3.1 template<std::size_t dims = 1U> const auto cl::sycl::trisycl::ItemImpl< dims >::dimensionality = dims [static]

Definition at line 213 of file sycl-implementation.hpp.

8.2.2.4.3.2 template<std::size_t dims = 1U> IdImpl<dims> cl::sycl::trisycl::ItemImpl< dims >::GlobalIndex

Definition at line 215 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::ItemImpl< dims >::get_global(), and cl::sycl::trisycl::ItemImpl< dims >::set_global().

8.2.2.4.3.3 template<std::size_t dims = 1U> IdImpl<dims> cl::sycl::trisycl::ItemImpl< dims >::LocalIndex

Definition at line 216 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::ItemImpl< dims >::get_local(), and cl::sycl::trisycl::ItemImpl< dims >::set_local().

8.2.2.4.3.4 template<std::size_t dims = 1U> NDRangeImpl<dims> cl::sycl::trisycl::ItemImpl< dims >::NDRange

Definition at line 217 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::ItemImpl< dims >::get_global_range(), and cl::sycl::trisycl::ItemImpl< dims >::get↩_local_range().

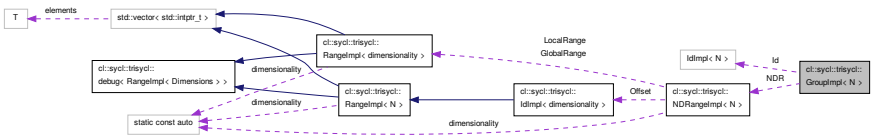
8.2.2.5 struct cl::sycl::trisycl::GroupImpl

template<std::size_t N = 1U>struct cl::sycl::trisycl::GroupImpl< N >

The implementation of a SYCL group index to specify a work_group in a parallel_for_workitem.

Definition at line 251 of file sycl-implementation.hpp.

Collaboration diagram for cl::sycl::trisycl::GroupImpl< N >:



Public Member Functions

- GroupImpl (const GroupImpl &g)
- GroupImpl (const NDRangeImpl< N > &ndr)
- GroupImpl (const NDRangeImpl< N > &ndr, const IdImpl< N > &i)
- GroupImpl & getImpl ()

  Return a reference to the implementation itself.
- const GroupImpl & getImpl () const

  Return a const reference to the implementation itself.
- IdImpl< N > get_group_id ()

  Return the id of this work-group.
- RangeImpl< N > get_local_range ()

  Return the local range associated to this work-group.
- RangeImpl< N > get_global_range ()

  Return the global range associated to this work-group.
- auto & operator[] (int index)

  Return the group coordinate in the given dimension.

Public Attributes

- const NDRangeImpl< N > & NDR

  Keep a reference on the nd_range to serve potential query on it.
- IdImpl< N > Id

  The coordinate of the group item.

8.2.2.5.1 Constructor & Destructor Documentation

8.2.2.5.1.1 template<std::size_t N = 1U> cl::sycl::trisycl::GroupImpl< N >::GroupImpl (const GroupImpl< N > & *g*) [inline]

Definition at line 257 of file sycl-implementation.hpp.

```
00257 : NDR(g.NDR), Id(g.Id) {}
```

8.2.2.5.1.2 template<std::size_t N = 1U> cl::sycl::trisycl::GroupImpl< N >::GroupImpl (const NDRangeImpl< N > & *ndr*) [inline]

Definition at line 259 of file sycl-implementation.hpp.

```
00259 : NDR(ndr) {}
```

8.2.2.5.1.3 template<std::size_t N = 1U> cl::sycl::trisycl::GroupImpl< N >::GroupImpl (const NDRangeImpl< N > & *ndr*, const IdImpl< N > & *i*) [inline]

Definition at line 261 of file sycl-implementation.hpp.

```
00261                                                     :
00262     NDR(ndr), Id(i) {}
```

8.2.2.5.2 Member Function Documentation

8.2.2.5.2.1 template<std::size_t N = 1U> RangeImpl<N> cl::sycl::trisycl::GroupImpl< N >::get_global_range ( ) [inline]

Return the global range associated to this work-group.

Definition at line 277 of file sycl-implementation.hpp.

```
00277 { return NDR.GlobalRange; }
```

8.2.2.5.2.2 template<std::size_t N = 1U> Id Impl<N> cl::sycl::trisycl::GroupImpl< N >::get_group_id ( ) [inline]

Return the id of this work-group.

Definition at line 271 of file sycl-implementation.hpp.

```
00271 { return Id; }
```

8.2.2.5.2.3 template<std::size_t N = 1U> Range Impl<N> cl::sycl::trisycl::GroupImpl< N >::get_local_range ( ) [inline]

Return the local range associated to this work-group.

Definition at line 274 of file sycl-implementation.hpp.

```
00274 { return NDR.LocalRange; }
```

8.2.2.5.2.4 template<std::size_t N = 1U> GroupImpl& cl::sycl::trisycl::GroupImpl< N >::getImpl ( ) [inline]

Return a reference to the implementation itself.

Definition at line 265 of file sycl-implementation.hpp.

```
00265 { return *this; };
```

8.2.2.5.2.5 template<std::size_t N = 1U> const GroupImpl& cl::sycl::trisycl::GroupImpl< N >::getImpl ( ) const [inline]

Return a const reference to the implementation itself.

Definition at line 268 of file sycl-implementation.hpp.

```
00268 { return *this; };
```

8.2.2.5.2.6 template<std::size_t N = 1U> auto& cl::sycl::trisycl::GroupImpl< N >::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?

Definition at line 285 of file sycl-implementation.hpp.

```
00285                                {
00286     return Id[index];
00287   }
```

8.2.2.5.3 Member Data Documentation

8.2.2.5.3.1 template<std::size_t N = 1U> Id Impl<N> cl::sycl::trisycl::GroupImpl< N >::Id

The coordinate of the group item.

Definition at line 255 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::GroupImpl< dims >::get_group_id(), and cl::sycl::trisycl::GroupImpl< dims >::operator[]().

8.2.2.5.3.2 template<std::size_t N = 1U> const NDRangeImpl<N>& cl::sycl::trisycl::GroupImpl< N >::NDR

Keep a reference on the nd_range to serve potential query on it.

Definition at line 253 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::GroupImpl< dims >::get_global_range(), and cl::sycl::trisycl::GroupImpl< dims >::get_local_range().

8.2.2.6 struct cl::sycl::trisycl::ParallelForIterate

template<int level, typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::trisycl::ParallelForIterate< level, Range, ParallelForFunctor, Id >

A recursive multi-dimensional iterator that ends calling f.

The iteration order may be changed later.

Since partial specialization of function template is not possible in C++14, use a class template instead with everything in the constructor.

Definition at line 460 of file sycl-implementation.hpp.

Public Member Functions

- ParallelForIterate (const Range &r, ParallelForFunctor &f, Id &index)

8.2.2.6.1 Constructor & Destructor Documentation

8.2.2.6.1.1 template<int level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::trisycl::ParallelForIterate< level, Range, ParallelForFunctor, Id >::ParallelForIterate (const Range & *r*, ParallelForFunctor & *f*, Id & *index*) [inline]

Definition at line 461 of file sycl-implementation.hpp.

```
00461                                                                    {
00462     for (boost::multi_array_types::index _sycl_index = 0,
00463            _sycl_end = r[Range::dimensionality - level];
00464          _sycl_index < _sycl_end;
00465          _sycl_index++) {
00466       // Set the current value of the index for this dimension
00467       index[Range::dimensionality - level] = _sycl_index;
00468       // Iterate further on lower dimensions
00469       ParallelForIterate<level - 1,
00470                          Range,
00471                          ParallelForFunctor,
00472                          Id> { r, f, index };
00473     }
00474   }
```

8.2.2.7 struct cl::sycl::trisycl::ParallelOpenMPForIterate

template<int level, typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::trisycl::ParallelOpenMPForIterate< level, Range, ParallelForFunctor, Id >

A top-level recursive multi-dimensional iterator variant using OpenMP.

Only the top-level loop uses OpenMP and go on with the normal recursive multi-dimensional.

Definition at line 484 of file sycl-implementation.hpp.

Public Member Functions

- ParallelOpenMPForIterate (const Range &r, ParallelForFunctor &f)

8.2.2.7.1 Constructor & Destructor Documentation

8.2.2.7.1.1 template<int level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::trisycl::ParallelOpenMPForIterate< level, Range, ParallelForFunctor, Id >::ParallelOpenMPForIterate (const Range & *r*, ParallelForFunctor & *f*) [inline]

Definition at line 485 of file sycl-implementation.hpp.

```
00485                                                                  {
00486     // Create the OpenMP threads before the for loop to avoid creating an
00487     // index in each iteration
00488 #pragma omp parallel
00489     {
00490       // Allocate an OpenMP thread-local index
00491       Id index;
00492       // Make a simple loop end condition for OpenMP
00493       boost::multi_array_types::index _sycl_end =
00494         r[Range::dimensionality - level];
00495       /* Distribute the iterations on the OpenMP threads. Some OpenMP
00496          "collapse" could be useful for small iteration space, but it
00497          would need some template specialization to have real contiguous
00498          loop nests */
00499 #pragma omp for
00500       for (boost::multi_array_types::index _sycl_index = 0;
00501            _sycl_index < _sycl_end;
00502            _sycl_index++) {
00503         // Set the current value of the index for this dimension
00504         index[Range::dimensionality - level] = _sycl_index;
00505         // Iterate further on lower dimensions
00506         ParallelForIterate<level - 1,
00507                            Range,
00508                            ParallelForFunctor,
00509                            Id> { r, f, index };
00510       }
00511     }
00512   }
```

8.2.2.8 struct cl::sycl::trisycl::ParallelForIterate< 0, Range, ParallelForFunctor, Id >

template<typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::trisycl::ParallelForIterate< 0, Range, ParallelForFunctor, Id >

Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id.

Definition at line 519 of file sycl-implementation.hpp.

Public Member Functions

- ParallelForIterate (const Range &r, ParallelForFunctor &f, Id &index)

8.2.2.8.1 Constructor & Destructor Documentation

8.2.2.8.1.1 template<typename Range , typename ParallelForFunctor , typename Id > cl::sycl::trisycl::ParallelForIterate< 0, Range, ParallelForFunctor, Id >::ParallelForIterate (const Range & *r*, ParallelForFunctor & *f*, Id & *index*) [inline]

Definition at line 520 of file sycl-implementation.hpp.

```
00520                                                                  {
00521     f(index);
00522   }
```

8.2.2.9 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.

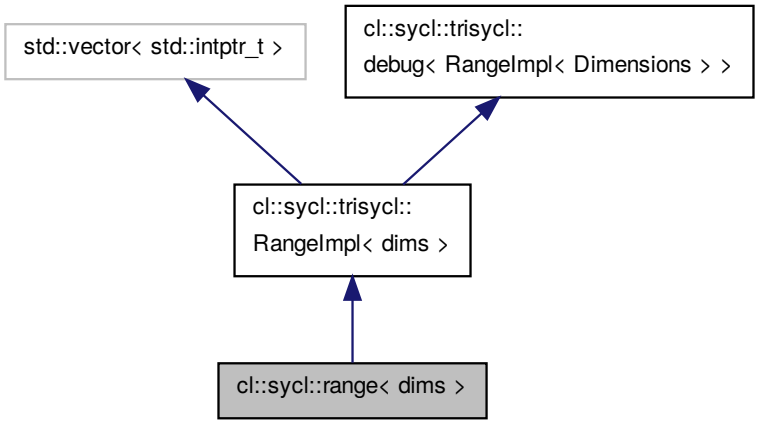
Todo use std::size_t dims instead of int dims in the specification?

Todo add to the norm this default parameter value?

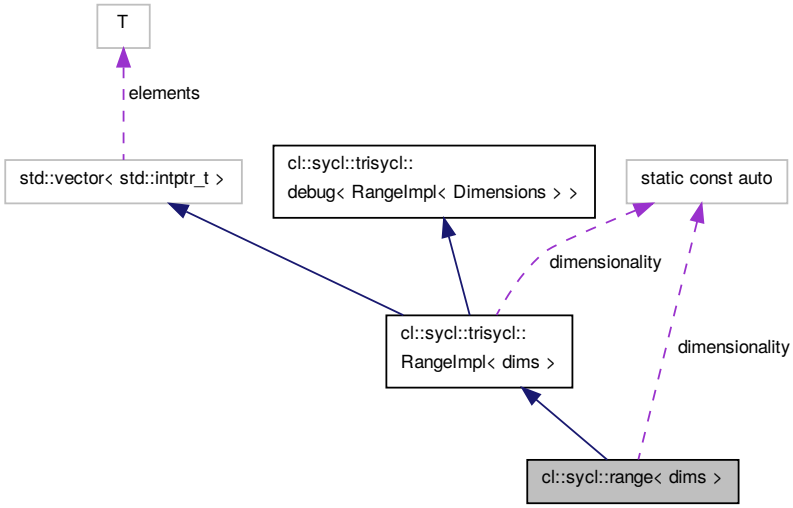
Todo add to the norm some way to specify an offset?

Definition at line 175 of file sycl.hpp.

Inheritance diagram for cl::sycl::range< dims >:



Collaboration diagram for cl::sycl::range< dims >:



Public Types

- using Impl = RangeImpl< dims >

Public Member Functions

- range (Impl &r)

  Construct a range from an implementation, used by nd_range() for example.
- range (const Impl &r)
- range (range< dims > &r)
- range (const range< dims > &r)
- range (std::initializer_list< std::intptr_t > l)

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

  Return the range size in the give dimension.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.9.1 Member Typedef Documentation

8.2.2.9.1.1 template<int dims = 1> using cl::sycl::range< dims >::Impl = RangeImpl<dims>

Definition at line 183 of file sycl.hpp.

8.2.2.9.2 Constructor & Destructor Documentation

8.2.2.9.2.1 template<int dims = 1> cl::sycl::range< dims >::range (Impl & *r*) [inline]

Construct a range from an implementation, used by nd_range() for example.

This is internal and should not appear in the specification

Definition at line 189 of file sycl.hpp.

```
00189 : Impl(r) {}
```

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

Definition at line 191 of file sycl.hpp.

```
00191 : Impl(r) {}
```

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

Definition at line 194 of file sycl.hpp.

```
00194 : Impl(r.getImpl()) {}
```

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

Definition at line 196 of file sycl.hpp.

```
00196 : Impl(r.getImpl()) {}
```

8.2.2.9.2.5 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

Definition at line 204 of file sycl.hpp.

```
00204 : Impl(l) {}
```

8.2.2.9.2.6 template<int dims = 1> cl::sycl::range< dims >::range (std::intptr_t *x*) [inline]

To have implicit conversion from 1 integer.

Definition at line 215 of file sycl.hpp.

```
00215                         : range { x } {
00216     static_assert(dims == 1, "A range with 1 size value should be 1-D");
00217   }
```

8.2.2.9.2.7 template<int dims = 1> cl::sycl::range< dims >::range (std::intptr_t *x*, std::intptr_t *y*) [inline]

A 2-D constructor from 2 integers.

Definition at line 221 of file sycl.hpp.

```
00221                                       : range { x, y } {
00222     static_assert(dims == 2, "A range with 2 size values should be 2-D");
00223   }
```

8.2.2.9.2.8 template<int dims = 1> cl::sycl::range< dims >::range (std::intptr_t *x*, std::intptr_t *y*, std::intptr_t *z*) [inline]

A 3-D constructor from 3 integers.

Definition at line 227 of file sycl.hpp.

```
00227                                                 : range { x, y, z } {
00228     static_assert(dims == 3, "A range with 3 size values should be 3-D");
00229   }
```

8.2.2.9.3 Member Function Documentation

8.2.2.9.3.1 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?

Definition at line 241 of file sycl.hpp.

```
00241                       {
00242     return (*this)[index];
00243   }
```

8.2.2.9.4 Member Data Documentation

8.2.2.9.4.1 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?

Definition at line 179 of file sycl.hpp.

8.2.2.10 struct cl::sycl::id

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

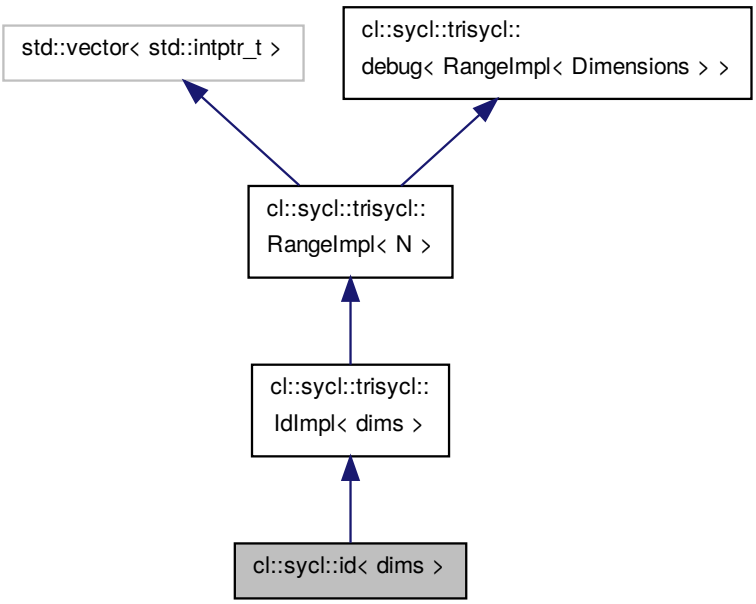
Define a multi-dimensional index, used for example to locate a work item.

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

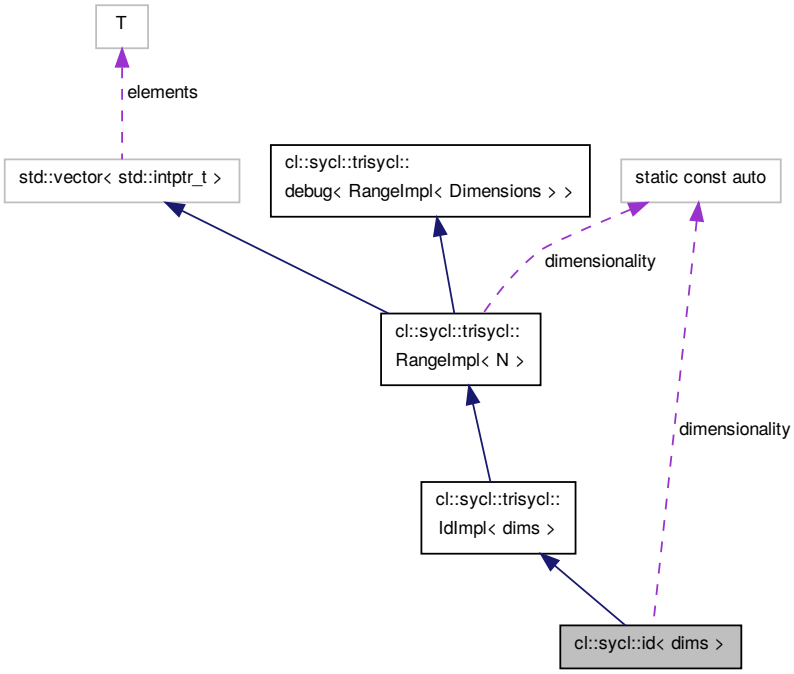
Todo It would be nice to have [] working everywhere, provide both get_...() and get_...(int dim) equivalent to get_...()[int dim] Well it is already the case for item. So not needed for id? Indeed [] is mentioned in text of page 59 but not in class description.

Definition at line 303 of file sycl.hpp.

Inheritance diagram for cl::sycl::id< dims >:



Collaboration diagram for cl::sycl::id< dims >:



Public Types

- using Impl = IdImpl< dims >

Public Member Functions

- id (const Impl &init)

  Since the runtime needs to construct an id from its implementation for example in item methods, define some hidden constructor here.
- id ()

  Create a zero id.
- id (const id &init)

  Create an id with the same value of another one.
- id (const range< dims > &r)

  Create an id from a given range.
- id (std::initializer_list< std::intptr_t > l)

  Create a n-D range from a positive integer-like list.
- id (std::intptr_t s)

  To have implicit conversion from 1 integer.
- int get (int index)

  Return the id size in the given dimension.
- auto & operator[] (int index)

  Return the id size in the given dimension.

Static Public Attributes

- static const auto dimensionality = dims

8.2.2.10.1 Member Typedef Documentation

8.2.2.10.1.1 template<int dims = 1> using cl::sycl::id< dims >::Impl = IdImpl<dims>

Definition at line 312 of file sycl.hpp.

8.2.2.10.2 Constructor & Destructor Documentation

8.2.2.10.2.1 template<int dims = 1> cl::sycl::id< dims >::id (const Impl & *init*) [inline]

Since the runtime needs to construct an id from its implementation for example in item methods, define some hidden constructor here.

Definition at line 317 of file sycl.hpp.

```
00317 : Impl(init) {}
```

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

Create a zero id.

Todo Add it to the specification?

Definition at line 325 of file sycl.hpp.

```
00325 : Impl() {}
```

8.2.2.10.2.3 template<int dims = 1> cl::sycl::id< dims >::id (const id< dims > & *init*) [inline]

Create an id with the same value of another one.

Definition at line 329 of file sycl.hpp.

```
00329 : Impl(init.getImpl()) {}
```

8.2.2.10.2.4 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) ?

Definition at line 338 of file sycl.hpp.

```
00338 : Impl(r.getImpl()) {}
```

8.2.2.10.2.5 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<>...

Definition at line 346 of file sycl.hpp.

```
00346 : Impl(l) {}
```

8.2.2.10.2.6 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

Definition at line 353 of file sycl.hpp.

```
00353                     : id({ s }) {
00354     static_assert(dims == 1, "A range with 1 size should be 1-D");
00355   }
```

8.2.2.10.3 Member Function Documentation

8.2.2.10.3.1 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?

Definition at line 362 of file sycl.hpp.

```
00362                     {
00363     return (*this)[index];
00364   }
```

8.2.2.10.3.2 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?

Definition at line 376 of file sycl.hpp.

```
00376                               {
00377     return (*this).getImpl()[index];
00378   }
```

8.2.2.10.4 Member Data Documentation

8.2.2.10.4.1 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?

Definition at line 307 of file sycl.hpp.

8.2.2.11 struct cl::sycl::nd_range

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

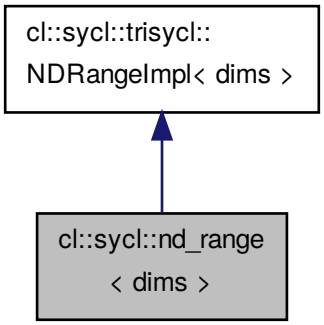
A ND-range, made by a global and local range, to specify work-group and work-item organization.

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

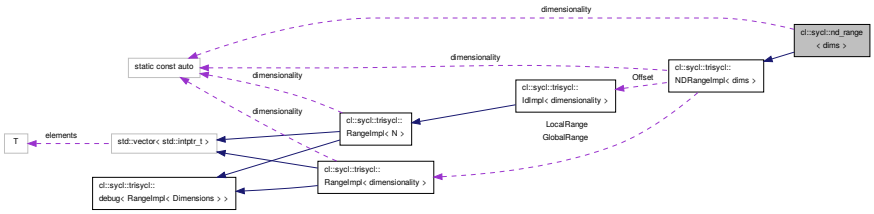
Todo add copy constructors in the specification

Definition at line 392 of file sycl.hpp.

Inheritance diagram for cl::sycl::nd_range< dims >:



Collaboration diagram for cl::sycl::nd_range< dims >:



Public Types

- using Impl = NDRangeImpl< dims >

Public Member Functions

- nd_range (const Impl &init)

  Since the runtime needs to construct a nd_range from its implementation for example in parallel_for stuff, define some hidden constructor here.
- nd_range (range< dims > global_size, range< dims > local_size, id< dims > offset=id< dims >())

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

Additional Inherited Members

8.2.2.11.1 Member Typedef Documentation

8.2.2.11.1.1 template<int dims = 1> using cl::sycl::nd_range< dims >::Impl = NDRangeImpl<dims>

Definition at line 403 of file sycl.hpp.

8.2.2.11.2 Constructor & Destructor Documentation

8.2.2.11.2.1 template<int dims = 1> cl::sycl::nd_range< dims >::nd_range (const Impl & *init*) [inline]

Since the runtime needs to construct a nd_range from its implementation for example in parallel_for stuff, define some hidden constructor here.

Definition at line 408 of file sycl.hpp.

```
00408 : Impl(init) {}
```

8.2.2.11.2.2 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

Definition at line 416 of file sycl.hpp.

```
00418                                       :
00419     Impl(global_size.getImpl(), local_size.getImpl(), offset.getImpl()) {}
```

8.2.2.11.3 Member Function Documentation

8.2.2.11.3.1 template<int dims = 1> range<dims> cl::sycl::nd_range< dims >::get_global_range ( ) [inline]

Get the global iteration space range.

Definition at line 423 of file sycl.hpp.

```
00423 { return Impl::get_global_range(); }
```

8.2.2.11.3.2 template<int dims = 1> range<dims> cl::sycl::nd_range< dims >::get_group_range ( ) [inline]

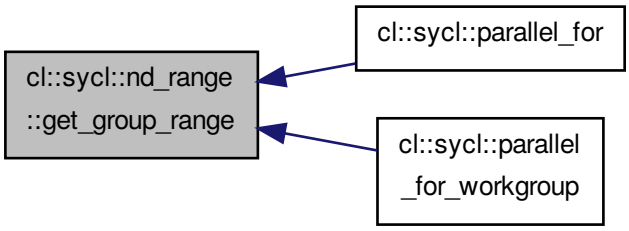
Get the range of work-groups needed to run this ND-range.

Definition at line 431 of file sycl.hpp.

Referenced by cl::sycl::parallel_for(), and cl::sycl::parallel_for_workgroup().

```
00431 { return Impl::get_group_range(); }
```

Here is the caller graph for this function:



8.2.2.11.3.3 template<int dims = 1> range<dims> cl::sycl::nd_range< dims >::get_local_range ( ) [inline]

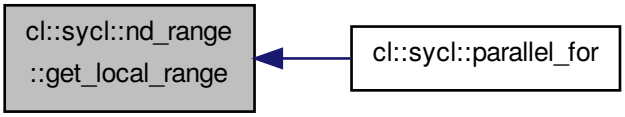
Get the local part of the iteration space range.

Definition at line 427 of file sycl.hpp.

Referenced by cl::sycl::parallel_for().

```
00427 { return Impl::get_local_range(); }
```

Here is the caller graph for this function:



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

Todo get_offset() is lacking in the specification

Definition at line 435 of file sycl.hpp.

```
00435 { return Impl::get_offset(); }
```

8.2.2.11.4 Member Data Documentation

8.2.2.11.4.1 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?

Definition at line 398 of file sycl.hpp.

8.2.2.12 struct cl::sycl::item

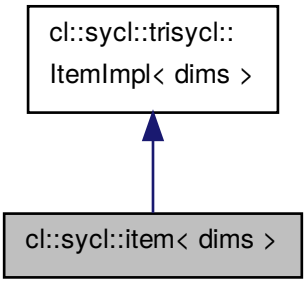
template<int dims = 1>struct cl::sycl::item< dims >

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

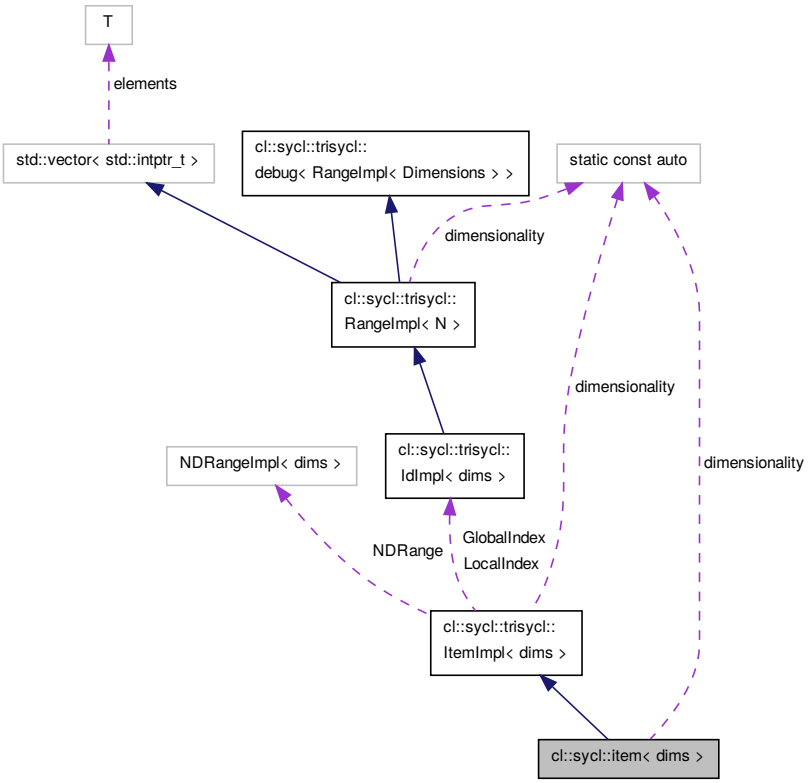
Todo Add to the specification: get_nd_range() to be coherent with providing get_local...() and get_global...() and what about the offset?

Definition at line 448 of file sycl.hpp.

Inheritance diagram for cl::sycl::item< dims >:



Collaboration diagram for cl::sycl::item< dims >:



Public Types

- using Impl = ItemImpl< dims >

Public Member Functions

- item (range< dims > global_size, range< dims > local_size)

  Create an item from a local size and 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

Additional Inherited Members

8.2.2.12.1 Member Typedef Documentation

8.2.2.12.1.1 template<int dims = 1> using cl::sycl::item< dims >::Impl = ItemImpl<dims>

Definition at line 455 of file sycl.hpp.

8.2.2.12.2 Constructor & Destructor Documentation

8.2.2.12.2.1 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?

Definition at line 463 of file sycl.hpp.

```
00463                                                  :
00464     Impl(global_size, local_size) {}
```

8.2.2.12.2.2 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?

Definition at line 470 of file sycl.hpp.

```
00470 : Impl(ndr) {}
```

8.2.2.12.3 Member Function Documentation

8.2.2.12.3.1 template<int dims = 1> int cl::sycl::item< dims >::get_global (int *dimension*) [inline]

Return the global coordinate in the given dimension.

Definition at line 474 of file sycl.hpp.

```
00474 { return Impl::get_global(dimension); }
```

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

Get the whole global id coordinate.

Definition at line 483 of file sycl.hpp.

```
00483 { return Impl::get_global(); }
```

8.2.2.12.3.3 template<int dims = 1> range<dims> cl::sycl::item< dims >::get_global_range ( ) [inline]

Get the global range where this item rely in.

Definition at line 492 of file sycl.hpp.

```
00492 { return Impl::get_global_range(); }
```

8.2.2.12.3.4 template<int dims = 1> int cl::sycl::item< dims >::get_local (int *dimension*) [inline]

Return the local coordinate (that is in the work-group) in the given dimension.

Definition at line 479 of file sycl.hpp.

```
00479 { return Impl::get_local(dimension); }
```

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

Get the whole local id coordinate (which is respective to the work-group)

Definition at line 488 of file sycl.hpp.

```
00488 { return Impl::get_local(); }
```

8.2.2.12.3.6 template<int dims = 1> range<dims> cl::sycl::item< dims >::get_local_range ( ) [inline]

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

Definition at line 495 of file sycl.hpp.

```
00495 { return Impl::get_local_range(); }
```

8.2.2.12.4 Member Data Documentation

8.2.2.12.4.1 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?

Definition at line 451 of file sycl.hpp.

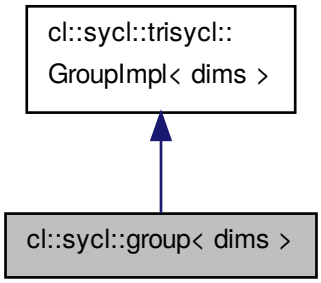
8.2.2.13 struct cl::sycl::group

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

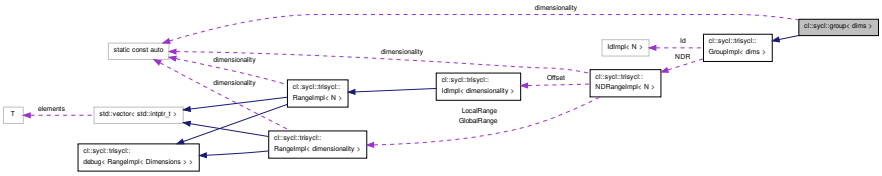
A group index used in a parallel_for_workitem to specify a work_group.

Definition at line 507 of file sycl.hpp.

Inheritance diagram for cl::sycl::group< dims >:



Collaboration diagram for cl::sycl::group< dims >:



Public Types

- using Impl = GroupImpl< dims >

Public Member Functions

- group (const NDRangeImpl< dims > &NDR, const IdImpl< dims > &ID)

  Since the runtime needs to construct a group with the right content, define some hidden constructor for this.
- group (const NDRangeImpl< dims > &NDR)

  Some internal constructor without group id initialization.
- group (const group &g)
- id< dims > get_group_id ()
- range< dims > get_local_range ()

  Get the local range for this work_group.
- range< dims > get_global_range ()

  Get the local range for this work_group.
- nd_range< dims > get_nr_range ()
- int get (int index)

  Return the group coordinate in the given dimension.
- auto & operator[] (int index)

  Return the group coordinate in the given dimension.

Static Public Attributes

- static const auto dimensionality = dims

Additional Inherited Members

8.2.2.13.1 Member Typedef Documentation

8.2.2.13.1.1 template<int dims = 1> using cl::sycl::group< dims >::Impl = GroupImpl<dims>

Definition at line 514 of file sycl.hpp.

8.2.2.13.2 Constructor & Destructor Documentation

8.2.2.13.2.1 template<int dims = 1> cl::sycl::group< dims >::group (const NDRangeImpl< dims > & *NDR*, const IdImpl< dims > & *ID*) [inline]

Since the runtime needs to construct a group with the right content, define some hidden constructor for this.

Since it is internal, directly use the implementation

Definition at line 520 of file sycl.hpp.

```
00520 : Impl(NDR, ID) {}
```

8.2.2.13.2.2 template<int dims = 1> cl::sycl::group< dims >::group (const NDRangeImpl< dims > & *NDR*) [inline]

Some internal constructor without group id initialization.

Definition at line 524 of file sycl.hpp.

```
00524 : Impl(NDR) {}
```

8.2.2.13.2.3 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

Definition at line 530 of file sycl.hpp.

```
00530 : Impl(g.getImpl()) {}
```

8.2.2.13.3 Member Function Documentation

8.2.2.13.3.1 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?

Definition at line 560 of file sycl.hpp.

```
00560                       {
00561     return (*this)[index];
00562   }
```

8.2.2.13.3.2 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<>

Definition at line 547 of file sycl.hpp.

```
00547 { return Impl::get_global_range(); }
```

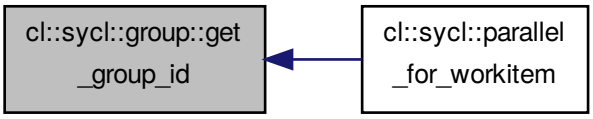
8.2.2.13.3.3 template<int dims = 1> id<dims> cl::sycl::group< dims >::get_group_id ( ) [inline]

Definition at line 533 of file sycl.hpp.

Referenced by cl::sycl::parallel_for_workitem().

```
00533 { return Impl::get_group_id(); }
```

Here is the caller graph for this function:



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

Get the local range for this work_group.

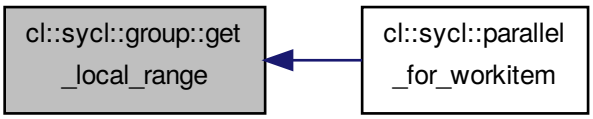
Todo Update the specification to return a range<dims> instead of an id<>

Definition at line 540 of file sycl.hpp.

Referenced by cl::sycl::parallel_for_workitem().

```
00540 { return Impl::get_local_range(); }
```

Here is the caller graph for this function:



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

Todo Why the offset is not available here?

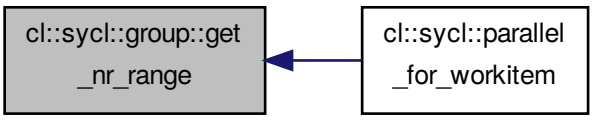
Todo Also provide this access to the current nd_range

Definition at line 552 of file sycl.hpp.

Referenced by cl::sycl::parallel_for_workitem().

```
00552 { return Impl::NDR; }
```

Here is the caller graph for this function:



8.2.2.13.3.6 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?

Definition at line 571 of file sycl.hpp.

```
00571                                {
00572     return (*this).getImpl()[index];
00573   }
```

8.2.2.13.4 Member Data Documentation

8.2.2.13.4.1 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?

Definition at line 510 of file sycl.hpp.

8.2.3 Function Documentation

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

```
#include <include/CL/sycl.hpp>
```

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

Definition at line 1164 of file sycl.hpp.

```
01164                                  {
01165   return F;
01166 }
```

8.2.3.2 template<std::size_t Dimensions> RangeImpl<Dimensions> cl::sycl::trisycl::operator* (RangeImpl< Dimensions > *a*, RangeImpl< Dimensions > *b*)

```
#include <include/CL/implementation/sycl-implementation.hpp>
```

Definition at line 116 of file sycl-implementation.hpp.

```
00117                                                                {
00118   RangeImpl<Dimensions> result;
00119
00120   for (int i = 0; i < Dimensions; i++)
00121     result[i] = a[i] * b[i];
00122
00123   return result;
00124 }
```

8.2.3.3 template<size_t Dimensions> range<Dimensions> cl::sycl::operator* (range< Dimensions > *a*, range< Dimensions > *b*)

```
#include <include/CL/sycl.hpp>
```

Definition at line 267 of file sycl.hpp.

```
00268                                                                {
00269   range<Dimensions> result;
00270
00271   for (int i = 0; i < Dimensions; i++)
00272     result[i] = a[i] * b[i];
00273
00274   return result;
00275 }
```

8.2.3.4 template<std::size_t Dimensions> RangeImpl<Dimensions> cl::sycl::trisycl::operator+ (RangeImpl< Dimensions > *a*, RangeImpl< Dimensions > *b*)

```
#include <include/CL/implementation/sycl-implementation.hpp>
```

Definition at line 129 of file sycl-implementation.hpp.

```
00130                                                                {
00131   RangeImpl<Dimensions> result;
00132
00133   for (int i = 0; i < Dimensions; i++)
00134     result[i] = a[i] + b[i];
00135
00136   return result;
00137 }
```

8.2.3.5 template<size_t Dimensions> range<Dimensions> cl::sycl::operator+ (range< Dimensions > *a*, range< Dimensions > *b*)

```
#include <include/CL/sycl.hpp>
```

Definition at line 280 of file sycl.hpp.

```
00281                                                                {
00282   range<Dimensions> result;
00283
00284   for (int i = 0; i < Dimensions; i++)
00285     result[i] = a[i] + b[i];
00286
00287   return result;
00288 }
```

8.2.3.6 template<std::size_t Dimensions> RangeImpl<Dimensions> cl::sycl::trisycl::operator/ (RangeImpl< Dimensions > *dividend*, RangeImpl< Dimensions > *divisor*)

```
#include <include/CL/implementation/sycl-implementation.hpp>
```

Definition at line 103 of file sycl-implementation.hpp.

```
00104                                                                    {
00105   RangeImpl<Dimensions> result;
00106
00107   for (int i = 0; i < Dimensions; i++)
00108     result[i] = (dividend[i] + divisor[i] - 1)/divisor[i];
00109
00110   return result;
00111 }
```

8.2.3.7 template<size_t Dimensions> range<Dimensions> cl::sycl::operator/ (range< Dimensions > *dividend,* range< Dimensions > *divisor*)

`#include <include/CL/sycl.hpp>`

Definition at line 254 of file sycl.hpp.

```
00255                                                          {
00256   range<Dimensions> result;
00257
00258   for (int i = 0; i < Dimensions; i++)
00259     result[i] = (dividend[i] + divisor[i] - 1)/divisor[i];
00260
00261   return result;
00262 }
```

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

`#include <include/CL/sycl.hpp>`

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<>

Definition at line 1191 of file sycl.hpp.

Referenced by cl::sycl::parallel_for().

```
01192                                       {
01193 #ifdef _OPENMP
01194   // Use OpenMP for the top loop level
01195   ParallelOpenMPForIterate<Dimensions,
01196                            range<Dimensions>,
01197                            ParallelForFunctor,
01198                            id<Dimensions>> { r, f };
01199 #else
01200   // In a sequential execution there is only one index processed at a time
01201   id<Dimensions> index;
01202   ParallelForIterate<Dimensions,
01203                      range<Dimensions>,
01204                      ParallelForFunctor,
01205                      id<Dimensions>> { r, f, index };
01206 #endif
01207 }
```

Here is the caller graph for this function:

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

`#include <include/CL/sycl.hpp>`

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

Todo Add an OpenMP implementation

Todo Deal with incomplete work-groups

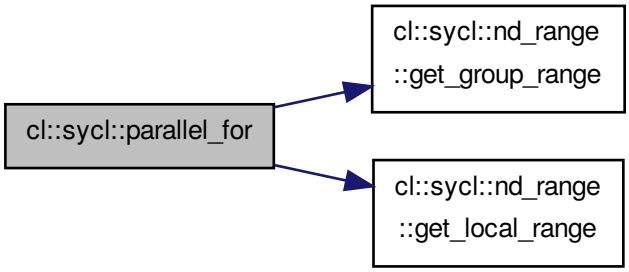
Todo Implement with parallel_for_workgroup()/parallel_for_workitem()

Definition at line 1219 of file sycl.hpp.

References cl::sycl::nd_range< dims >::get_group_range(), and cl::sycl::nd_range< dims >::get_local_range().

```
01220                                       {
01221   // In a sequential execution there is only one index processed at a time
01222   item<Dimensions> Index { r };
01223   // To iterate on the work-group
01224   id<Dimensions> Group;
01225   range<Dimensions> GroupRange = r.get_group_range();
01226   // To iterate on the local work-item
01227   id<Dimensions> Local;
01228   range<Dimensions> LocalRange = r.get_local_range();
01229
01230   // Reconstruct the item from its group and local id
01231   auto reconstructItem = [&] (id<Dimensions> L) {
01232     //Local.display();
01233     // Reconstruct the global item
01234     Index.set_local(Local);
01235     Index.set_global(Local + LocalRange*Group);
01236     // Call the user kernel at last
01237     f(Index);
01238   };
01239
01240   /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
01241      into another functor that iterate inside the work-group and then
01242      calls f */
01243   auto iterateInWorkGroup = [&] (id<Dimensions> G) {
01244     //Group.display();
01245     // Then iterate on the local work-groups
01246     ParallelForIterate<Dimensions,
01247                        range<Dimensions>,
01248                        decltype(reconstructItem),
01249                        id<Dimensions>> { LocalRange, reconstructItem, Local };
01250   };
01251
01252   // First iterate on all the work-groups
01253   ParallelForIterate<Dimensions,
01254                      range<Dimensions>,
01255                      decltype(iterateInWorkGroup),
01256                      id<Dimensions>> { GroupRange, iterateInWorkGroup, Group };
01257 }
```

Here is the call graph for this function:



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

```
#include <include/CL/sycl.hpp>
```

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

Todo deal with Program

Definition at line 1262 of file sycl.hpp.

References cl::sycl::parallel_for().

```
01262                                                                   {
01263   /// \todo deal with Program
01264   parallel_for(r, f);
01265 }
```

Here is the call graph for this function:

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

```
#include <include/CL/sycl.hpp>
```

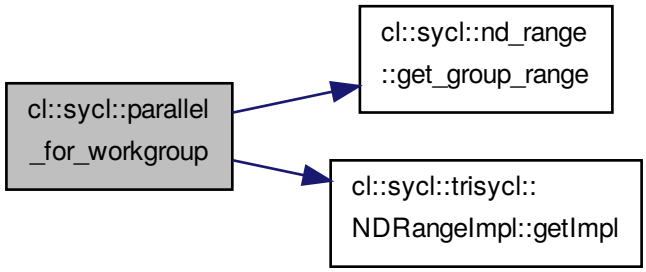
Loop on the work-groups.

Definition at line 1270 of file sycl.hpp.

References cl::sycl::nd_range< dims >::get_group_range(), and cl::sycl::trisycl::NDRangeImpl< dims >::getImpl().

```
01271                                                 {
01272   // In a sequential execution there is only one index processed at a time
01273   group<Dimensions> Group(r.getImpl());
01274
01275   // Reconstruct the item from its group and local id
01276   auto callWithGroup = [&] (group<Dimensions> G) {
01277     //G.Id.display();
01278     // Call the user kernel with the group as parameter
01279     f(G);
01280   };
01281   // First iterate on all the work-groups
01282   ParallelForIterate<Dimensions,
01283                      range<Dimensions>,
01284                      ParallelForFunctor,
01285                      group<Dimensions>> {
01286     r.get_group_range(),
01287     f,
01288     Group };
01289 }
```

Here is the call graph for this function:



8.2.3.12 template<int Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for_workitem (group< Dimensions > *g*, ParallelForFunctor *f*)

`#include <include/CL/sycl.hpp>`

Loop on the work-items inside a work-group.

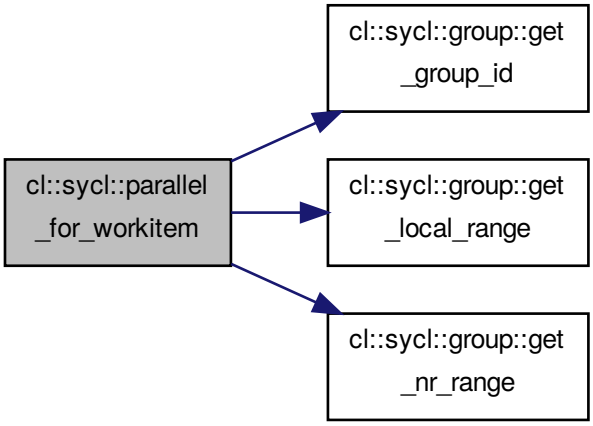
Definition at line 1294 of file sycl.hpp.

References cl::sycl::group< dims >::get_group_id(), cl::sycl::group< dims >::get_local_range(), and cl::sycl::group< dims >::get_nr_range().

```
01294                                                                 {
01295   // In a sequential execution there is only one index processed at a time
01296   item<Dimensions> Index { g.get_nr_range() };
01297   // To iterate on the local work-item
01298   id<Dimensions> Local;
01299
01300   // Reconstruct the item from its group and local id
01301   auto reconstructItem = [&] (id<Dimensions> L) {
01302     //Local.display();
01303     //L.display();
01304     // Reconstruct the global item
01305     Index.set_local(Local);
01306     // \todo Some strength reduction here
01307     Index.set_global(Local + g.get_local_range()*g.get_group_id());
01308     // Call the user kernel at last
01309     f(Index);
01310   };
01311
01312   // Then iterate on all the work-items of the work-group
```

```
01313   ParallelForIterate<Dimensions,
01314                      range<Dimensions>,
01315                      decltype(reconstructItem),
01316                      id<Dimensions>> {
01317     g.get_local_range(),
01318     reconstructItem,
01319     Local };
01320 }
```

Here is the call graph for this function:



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

`#include <include/CL/sycl.hpp>`

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

Todo remove from the SYCL specification and use a range-less parallel_for version with default construction of a 1-element range?

Definition at line 1178 of file sycl.hpp.

```
01178 { F(); }
```

8.3 Data access and storage in SYCL

Namespaces

- cl::sycl::access

  Describe the type of access by kernels.

Classes

- struct cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >

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

- struct cl::sycl::trisycl::BufferImpl< 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. More...

- 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. More...

- struct cl::sycl::storage< T >

  Abstract the way storage is managed to allow the programmer to control the storage management of buffers. More...

- 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. More...

8.3.1 Detailed Description

8.3.2 Class Documentation

8.3.2.1 struct cl::sycl::trisycl::AccessorImpl

template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer>struct cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >

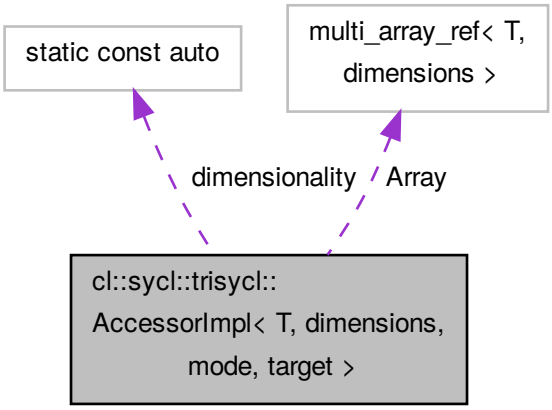
The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way.

This implementation rely on boost::multi_array to provides this nice syntax and behaviour.

Right now the aim of this class is just to access to the buffer in a read-write mode, even if capturing the multi_array_ref from a lambda make it const (since in some example we have lambda with [=] and without mutable). The access::mode is not used yet.

Definition at line 317 of file sycl-implementation.hpp.

Collaboration diagram for cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >:



Public Types

- typedef boost::multi_array_ref < T, dimensions > ArrayViewType
- typedef std::remove_const < ArrayViewType >::type WritableArrayViewType
- using element = T
- using value_type = T

Public Member Functions

- AccessorImpl (BufferImpl< T, dimensions > &targetBuffer)

  The only way to construct an AccessorImpl is from an existing buffer.
- auto & operator[] (std::size_t Index) const

  This is when we access to AccessorImpl[] that we override the const if any.
- auto & operator[] (IdImpl< dimensionality > Index) const

  This is when we access to AccessorImpl[] that we override the const if any.
- auto & operator[] (ItemImpl< dimensionality > Index) const

Public Attributes

- ArrayViewType Array

Static Public Attributes

- static const auto dimensionality = dimensions

8.3.2.1.1 Member Typedef Documentation

8.3.2.1.1.1 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> typedef boost::multi_array_ref<T, dimensions> cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::ArrayViewType

Definition at line 319 of file sycl-implementation.hpp.

8.3.2.1.1.2 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> using cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::element = T

Definition at line 329 of file sycl-implementation.hpp.

8.3.2.1.1.3 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> using cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::value_type = T

Definition at line 330 of file sycl-implementation.hpp.

8.3.2.1.1.4 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> typedef std::remove_const<ArrayViewType>::type cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::WritableArrayViewType

Definition at line 323 of file sycl-implementation.hpp.

8.3.2.1.2 Constructor & Destructor Documentation

8.3.2.1.2.1 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::AccessorImpl (BufferImpl< T, dimensions > & *targetBuffer*) [inline]

The only way to construct an AccessorImpl is from an existing buffer.

Definition at line 335 of file sycl-implementation.hpp.

```
00335                                                    :
00336     Array(targetBuffer.Access) {}
```

8.3.2.1.3 Member Function Documentation

8.3.2.1.3.1 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> auto& cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::operator[] (std::size_t *Index*) const [inline]

This is when we access to AccessorImpl[] that we override the const if any.

Definition at line 339 of file sycl-implementation.hpp.

```
00339                                          {
00340     return (const_cast<WritableArrayViewType &>(Array))[Index];
00341   }
```

8.3.2.1.3.2 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> auto& cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::operator[] (IdImpl< dimensionality > *Index*) const [inline]

This is when we access to AccessorImpl[] that we override the const if any.

Definition at line 344 of file sycl-implementation.hpp.

```
00344                                                  {
00345     return (const_cast<WritableArrayViewType &>(Array))(Index);
00346   }
```

8.3.2.1.3.3 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> auto& cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::operator[] (ItemImpl< dimensions > *Index*) const [inline]

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

Definition at line 349 of file sycl-implementation.hpp.

```
00349                                                    {
00350     return (const_cast<WritableArrayViewType &>(Array))(Index.get_global());
00351   }
```

8.3.2.1.4 Member Data Documentation

8.3.2.1.4.1 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> ArrayViewType cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::Array

Definition at line 320 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::AccessorImpl< dataType, dimensions, mode, target >::operator[]().

8.3.2.1.4.2 template<typename T, std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> const auto cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >::dimensionality = dimensions [static]

Definition at line 326 of file sycl-implementation.hpp.

8.3.2.2 struct cl::sycl::trisycl::BufferImpl

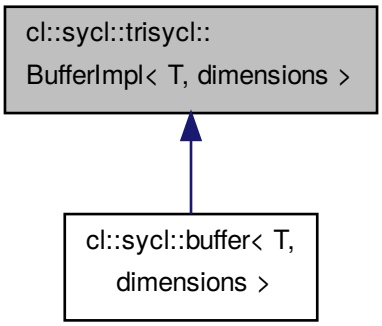
template<typename T, std::size_t dimensions>struct cl::sycl::trisycl::BufferImpl< T, dimensions >

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.

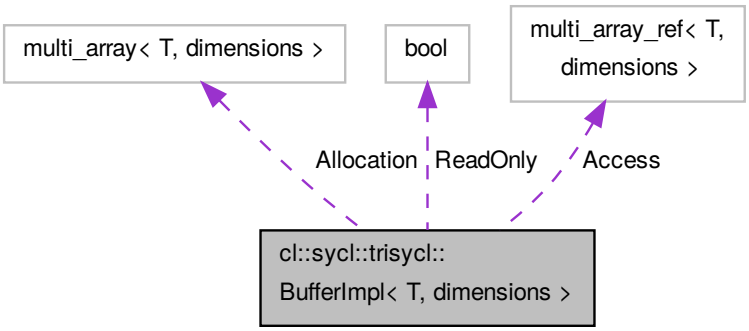
In the case we initialize it from a pointer, for now we just wrap the data with boost::multi_array_ref to provide the VLA semantics without any storage.

Definition at line 295 of file sycl-implementation.hpp.

Inheritance diagram for cl::sycl::trisycl::BufferImpl< T, dimensions >:



Collaboration diagram for cl::sycl::trisycl::BufferImpl< T, dimensions >:



Public Types

- using Implementation = boost::multi_array_ref< T, dimensions >
- using element = T
- using value_type = T

Public Member Functions

- BufferImpl (RangeImpl< dimensions > const &r)
  Create a new BufferImpl of size.
- BufferImpl (T *host_data, RangeImpl< dimensions > r)
  Create a new BufferImpl from.
- BufferImpl (const T *host_data, RangeImpl< dimensions > r)
  Create a new read only BufferImpl from.
- BufferImpl (const T *start_iterator, const T *end_iterator)
  Create a new allocated 1D BufferImpl from the given elements.
- BufferImpl (const BufferImpl< T, dimensions > &b)
  Create a new BufferImpl from an old one, with a new allocation.
- template<access::mode mode, access::target target = access::global_buffer>
  AccessorImpl< T, dimensions,
  mode, target > get_access ()
  Create a new sub-BufferImplImpl without allocation to have separate accessors later.

Public Attributes

- boost::multi_array< T, dimensions > Allocation
- boost::multi_array_ref< T,
  dimensions > Access
- bool ReadOnly

8.3.2.2.1 Member Typedef Documentation

8.3.2.2.1.1 template<typename T, std::size_t dimensions> using cl::sycl::trisycl::BufferImpl< T, dimensions >::element = T

Definition at line 367 of file sycl-implementation.hpp.

8.3.2.2.1.2 template<typename T, std::size_t dimensions> using cl::sycl::trisycl::BufferImpl< T, dimensions >::Implementation = boost::multi_array_ref<T, dimensions>

Definition at line 365 of file sycl-implementation.hpp.

8.3.2.2.1.3 template<typename T, std::size_t dimensions> using cl::sycl::trisycl::BufferImpl< T, dimensions >::value_type = T

Definition at line 368 of file sycl-implementation.hpp.

8.3.2.2.2 Constructor & Destructor Documentation

8.3.2.2.2.1 template<typename T, std::size_t dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl (RangeImpl< dimensions > const & *r*) [inline]

Create a new BufferImpl of size.

Parameters

r	

Definition at line 379 of file sycl-implementation.hpp.

```
00379                                                 : Allocation(r),
00380                                                   Access(Allocation),
00381                                                   ReadOnly(false) {}
```

8.3.2.2.2.2 template<typename T, std::size_t dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl (T * *host_data*, RangeImpl< dimensions > *r*) [inline]

Create a new BufferImpl from.

Parameters

host_data	of size
r	without further allocation

Definition at line 386 of file sycl-implementation.hpp.

```
00386                                                       : Access(host_data, r),
00387                                                         ReadOnly(false) {}
```

8.3.2.2.2.3 template<typename T, std::size_t dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl (const T * *host_data*, RangeImpl< dimensions > *r*) [inline]

Create a new read only BufferImpl from.

Parameters

host_data	of size
r	without further allocation

Definition at line 392 of file sycl-implementation.hpp.

```
00392                                                       :
00393     Access(host_data, r),
00394     ReadOnly(true) {}
```

8.3.2.2.2.4 template<typename T, std::size_t dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl (const T * *start_iterator*, const T * *end_iterator*) [inline]

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

Todo

Definition at line 401 of file sycl-implementation.hpp.

References cl::sycl::trisycl::BufferImpl< T, dimensions >::Allocation.

```
00401                                                                    :
00402     // The size of a multi_array is set at creation time
00403     Allocation(boost::extents[std::distance(start_iterator, end_iterator)]),
00404     Access(Allocation) {
00405     /* Then assign Allocation since this is the only multi_array
00406        method with this iterator interface */
00407     Allocation.assign(start_iterator, end_iterator);
00408   }
```

8.3.2.2.5 template<typename T, std::size_t dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl (const BufferImpl< T, dimensions > & *b*) [inline]

Create a new BufferImpl from an old one, with a new allocation.

Definition at line 412 of file sycl-implementation.hpp.

```
00412                                                 : Allocation(b.Access),
00413                                                   Access(Allocation),
00414                                                   ReadOnly(false) {}
```

8.3.2.2.3 Member Function Documentation

8.3.2.2.3.1 template<typename T, std::size_t dimensions> template<access::mode mode, access::target target = access::global_buffer> AccessorImpl<T, dimensions, mode, target> cl::sycl::trisycl::BufferImpl< T, dimensions >::get_access ( ) [inline]

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

Return an accessor of the required mode

Parameters

M	

Definition at line 438 of file sycl-implementation.hpp.

```
00438                                                  {
00439     return { *this };
00440   }
```

8.3.2.2.4 Member Data Documentation

8.3.2.2.4.1 template<typename T, std::size_t dimensions> boost::multi_array_ref<T, dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::Access

Definition at line 373 of file sycl-implementation.hpp.

8.3.2.2.4.2 template<typename T, std::size_t dimensions> boost::multi_array<T, dimensions> cl::sycl::trisycl::BufferImpl< T, dimensions >::Allocation

Definition at line 371 of file sycl-implementation.hpp.

Referenced by cl::sycl::trisycl::BufferImpl< T, dimensions >::BufferImpl().

8.3.2.2.4.3 template<typename T, std::size_t dimensions> bool cl::sycl::trisycl::BufferImpl< T, dimensions >::ReadOnly

Definition at line 375 of file sycl-implementation.hpp.

8.3.2.3 struct cl::sycl::accessor

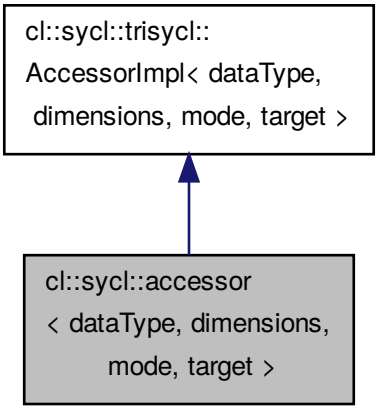
template<typename dataType, size_t dimensions, access::mode mode, access::target target = access::global_buffer>struct cl::sycl::accessor< dataType, dimensions, mode, target >

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

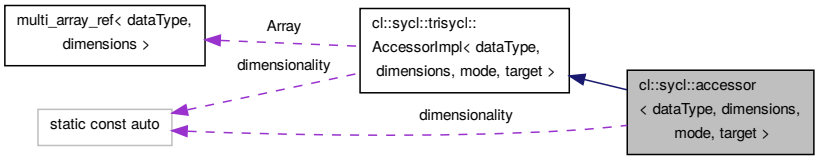
Todo Implement it for images according so section 3.3.4.5

Definition at line 872 of file sycl.hpp.

Inheritance diagram for cl::sycl::accessor< dataType, dimensions, mode, target >:



Collaboration diagram for cl::sycl::accessor< dataType, dimensions, mode, target >:



Public Types

- using element = dataType
- using value_type = dataType
- using Impl = AccessorImpl< dataType, dimensions, mode, target >

Public Member Functions

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

  Create an accessor to the given buffer.

- dataType & operator[] (id< dimensionality > Index) const

  Get the element specified by the given id.
- dataType & operator[] (size_t Index) const

  Get the element specified by the given index in the case we are mono-dimensional.
- dataType & operator[] (item< dimensionality > Index) const

  Get the element specified by the given item.

Static Public Attributes

- static const auto dimensionality = dimensions

Additional Inherited Members

8.3.2.3.1 Member Typedef Documentation

8.3.2.3.1.1 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

Definition at line 878 of file sycl.hpp.

8.3.2.3.1.2 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> using cl::sycl::accessor< dataType, dimensions, mode, target >::Impl = AccessorImpl<dataType, dimensions, mode, target>

Definition at line 884 of file sycl.hpp.

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

Definition at line 879 of file sycl.hpp.

8.3.2.3.2 Constructor & Destructor Documentation

8.3.2.3.2.1 template<typename dataType , size_t dimensions, access::mode mode, access::target target = access::global_buffer> cl::sycl::accessor< dataType, dimensions, mode, target >::accessor (buffer< dataType, dimensions > & *targetBuffer*) [inline]

Create an accessor to the given buffer.

Definition at line 889 of file sycl.hpp.

```
00889                                                    :
00890     Impl(targetBuffer) {}
```

8.3.2.3.3 Member Function Documentation

8.3.2.3.3.1 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

Definition at line 898 of file sycl.hpp.

```
00898                                                    {
00899     return Impl::operator[](Index);
00900   }
```

8.3.2.3.3.2 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>?

Definition at line 909 of file sycl.hpp.

```
00909                                          {
00910     return Impl::operator[](Index);
00911   }
```

8.3.2.3.3.3 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

Definition at line 918 of file sycl.hpp.

```
00918                                                    {
00919     return Impl::operator[](Index);
00920   }
```

8.3.2.3.4 Member Data Documentation

8.3.2.3.4.1 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

Definition at line 875 of file sycl.hpp.

8.3.2.4 struct cl::sycl::storage

template<typename T>struct cl::sycl::storage< T >

Abstract the way storage is managed to allow the programmer to control the storage management of buffers.

Parameters

T	the type of the elements of the underlying data

The user is responsible for ensuring that their storage class implementation is thread-safe.

Definition at line 935 of file sycl.hpp.

Public Types

- using element = T
- using value_type = T

Public Member Functions

- virtual size_t get_size ()=0

  Method called by SYCL system to get the number of elements of type T of the underlying data.
- virtual T * get_host_data ()=0

  Method called by the SYCL system to know where that data is held in host memory.
- virtual const T * get_initial_data ()=0

  Method called by the SYCL system at the point of construction to request the initial contents of the buffer.
- virtual T * get_final_data ()=0

  Method called at the point of construction to request where the content of the buffer should be finally stored to.
- virtual void destroy ()=0

  Method called when the associated memory object is destroyed.
- virtual void in_use ()=0

  Method called when a command_group which accesses the data is added to a queue.
- virtual void completed ()=0

  Method called when the final enqueued command has completed.

8.3.2.4.1 Member Typedef Documentation

8.3.2.4.1.1 template<typename T> using cl::sycl::storage< T >::element = T

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

Definition at line 938 of file sycl.hpp.

8.3.2.4.1.2 template<typename T> using cl::sycl::storage< T >::value_type = T

Definition at line 939 of file sycl.hpp.

8.3.2.4.2 Member Function Documentation

8.3.2.4.2.1 template<typename T> virtual void cl::sycl::storage< T >::completed ( ) [pure virtual]

Method called when the final enqueued command has completed.

8.3.2.4.2.2 template<typename T> virtual void cl::sycl::storage< T >::destroy ( ) [pure virtual]

Method called when the associated memory object is destroyed.

This method is only called once, so if a memory object is copied multiple times, only when the last copy of the memory object is destroyed is the destroy method called.

Exceptions thrown by the destroy method will be caught and ignored.

8.3.2.4.2.3 template<typename T> virtual T* cl::sycl::storage< T >::get_final_data ( ) [pure virtual]

Method called at the point of construction to request where the content of the buffer should be finally stored to.

Returns

the address of where the buffer will be written to in host memory.

If the address is nullptr, then this phase is skipped.

If get_host_data() returns the same pointer as get_initial_data() and/or get_final_data() then the SYCL system should determine whether copying is actually necessary or not.

8.3.2.4.2.4 template<typename T> virtual T* cl::sycl::storage< T >::get_host_data ( ) [pure virtual]

Method called by the SYCL system to know where that data is held in host memory.

Returns

the address or nullptr if SYCL has to manage the temporary storage of the data.

8.3.2.4.2.5 template<typename T> virtual const T* cl::sycl::storage< T >::get_initial_data ( ) [pure virtual]

Method called by the SYCL system at the point of construction to request the initial contents of the buffer.

Returns

the address of the data to use or nullptr to skip this data initialization

8.3.2.4.2.6 template<typename T> virtual size_t cl::sycl::storage< T >::get_size ( ) [pure virtual]

Method called by SYCL system to get the number of elements of type T of the underlying data.

Todo This is inconsistent in the specification with get_size() in buffer which returns the byte size. Is it to be renamed to get_count()?

8.3.2.4.2.7 template<typename T> virtual void cl::sycl::storage< T >::in_use ( ) [pure virtual]

Method called when a command_group which accesses the data is added to a queue.

After completed is called, there may be further calls of in_use() if new work is enqueued that operates on the memory object.

8.3.2.5 struct cl::sycl::buffer

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

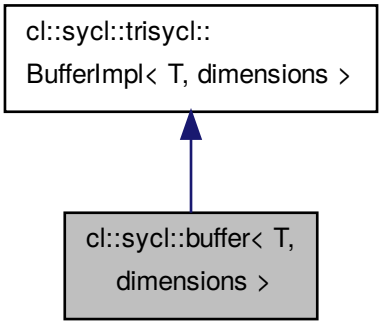
A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on.

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.

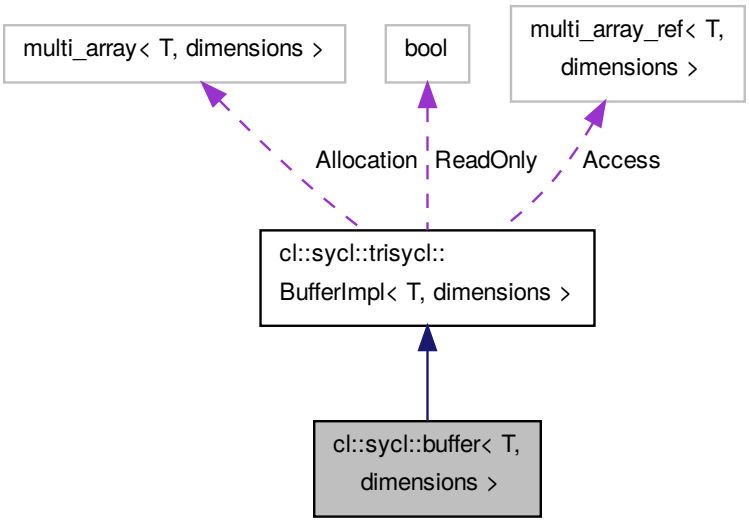
Todo there is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Definition at line 583 of file sycl.hpp.

Inheritance diagram for cl::sycl::buffer< T, dimensions >:



Collaboration diagram for cl::sycl::buffer< T, dimensions >:



Public Types

- using element = T
- using value_type = T
- using Impl = BufferImpl< T, dimensions >

Public Member Functions

- buffer (const range< dimensions > &r)

Create a new buffer with storage managed by SYCL.

- buffer (T *host_data, range< dimensions > r)

  Create a new buffer with associated host memory.
- buffer (const T *host_data, range< dimensions > r)

  Create a new read only buffer with associated host memory.
- buffer (storage< T > &store, range< dimensions > r)

  Create a new buffer from a storage abstraction provided by the user.
- buffer (const T *start_iterator, const T *end_iterator)

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

  Create a new buffer copy that shares the data with the origin buffer.
- buffer (buffer< T, dimensions > b, id< dimensions > base_index, range< dimensions > sub_range)

  Create a new sub-buffer without allocation to have separate accessors later.
- buffer (cl_mem mem_object, queue from_queue, event available_event)

  Create a buffer from an existing OpenCL memory object associated to a context after waiting for an event signaling the availability of the OpenCL data.
- template<access::mode mode, access::target target = access::global_buffer>
  accessor< T, dimensions, mode,
  target > get_access ()

  Get an accessor to the buffer with the required mode.

Additional Inherited Members

8.3.2.5.1 Member Typedef Documentation

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

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

Definition at line 1025 of file sycl.hpp.

8.3.2.5.1.2 template<typename T, int dimensions> using cl::sycl::buffer< T, dimensions >::Impl = BufferImpl<T, dimensions>

Definition at line 1030 of file sycl.hpp.

8.3.2.5.1.3 template<typename T, int dimensions> using cl::sycl::buffer< T, dimensions >::value_type = T

Definition at line 1026 of file sycl.hpp.

8.3.2.5.2 Constructor & Destructor Documentation

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

Create a new buffer with storage managed by SYCL.

Parameters

r	defines the size

Definition at line 1037 of file sycl.hpp.

```
01037 : Impl(r.getImpl()) {}
```

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

Create a new buffer with associated host memory.

Parameters

host_data	points to the storage and values used by the buffer
r	defines the size

Definition at line 1046 of file sycl.hpp.

```
01046 : Impl(host_data, r.getImpl()) {}
```

8.3.2.5.2.3 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 with associated host memory.

Parameters

host_data	points to the storage and values used by the buffer
r	defines the size

Definition at line 1055 of file sycl.hpp.

```
01055                                                 :
01056     Impl(host_data, r.getImpl()) {}
```

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

Create a new buffer from a storage abstraction provided by the user.

Parameters

store	is the storage back-end to use for the buffer
r	defines the size

The storage object has to exist during all the life of the buffer object.

Todo To be implemented

Definition at line 1070 of file sycl.hpp.

```
01070 { assert(0); }
```

8.3.2.5.2.5 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 initialized from the given elements.

Parameters

start_iterator	points to the first element to copy
end_iterator	points to just after the last element to copy

Todo Add const to the SYCL specification

Definition at line 1081 of file sycl.hpp.

```
01081                                                         :
01082     Impl(start_iterator, end_iterator) {}
```

8.3.2.5.2.6 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > & *b*) [inline]

Create a new buffer copy that shares the data with the origin buffer.

Parameters

b	is the buffer to copy from

The system use reference counting to deal with data lifetime

Definition at line 1091 of file sycl.hpp.

```
01091 : Impl(b) {}
```

8.3.2.5.2.7 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (buffer< T, dimensions > *b*, id< dimensions > *base_index*, range< dimensions > *sub_range*) [inline]

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

Parameters

b	is the buffer with the real data
base_index	specifies the origin of the sub-buffer inside the buffer b
sub_range	specifies the size of the sub-buffer

Todo To be implemented

Todo Update the specification to replace index by id

Definition at line 1108 of file sycl.hpp.

```
01110                                       { assert(0); }
```

8.3.2.5.2.8 template<typename T, int dimensions> cl::sycl::buffer< T, dimensions >::buffer (cl_mem *mem_object*, queue *from_queue*, event *available_event*) [inline]

Create a buffer from an existing OpenCL memory object associated to a context after waiting for an event signaling the availability of the OpenCL data.

Parameters

mem_object	is the OpenCL memory object to use
from_queue	is the queue associated to the memory object
available_event	specifies the event to wait for if non null

Todo To be implemented

Todo Improve the specification to allow CLHPP objects too

Definition at line 1128 of file sycl.hpp.

```
01130                                   { assert(0); }
```

8.3.2.5.3 Member Function Documentation

8.3.2.5.3.1 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]

Get an accessor to the buffer with the required mode.

Parameters

mode	is the requested access mode
target	is the type of object to be accessed

Definition at line 1144 of file sycl.hpp.

```
01144                                                                {
01145     return { *this };
01146   }
```

8.4 Error handling

Namespaces

- cl::sycl::trisycl

Classes

- struct cl::sycl::exception

  Encapsulate a SYCL error information. More...
- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...

8.4.1 Detailed Description

8.4.2 Class Documentation

8.4.2.1 struct cl::sycl::exception

Encapsulate a SYCL error information.

Definition at line 596 of file sycl.hpp.

Public Member Functions

- cl_int get_cl_code ()

  Get the OpenCL error code.
- cl_int get_sycl_code ()

  Get the SYCL-specific error code.
- queue * get_queue ()

  Get the queue that caused the error.
- template<typename T , int dimensions>
  buffer< T, dimensions > * get_buffer ()

  Get the buffer that caused the error.
- template<int dimensions>
  image< dimensions > * get_image ()

  Get the image that caused the error.

8.4.2.1.1 Member Function Documentation

8.4.2.1.1.1 template<typename T , int dimensions> buffer<T, dimensions>* cl::sycl::exception::get_buffer ( ) [inline]

Get the buffer that caused the error.

Returns

nullptr if not a buffer error

Todo Update specification to replace 0 by nullptr and add the templated buffer

Todo to be implemented

Definition at line 637 of file sycl.hpp.

```
00637                                                                   {
00638     assert(0); }
```

8.4.2.1.1.2 cl_int cl::sycl::exception::get_cl_code ( ) [inline]

Get the OpenCL error code.

Returns

0 if not an OpenCL error

Todo to be implemented

Definition at line 604 of file sycl.hpp.

```
00604 { assert(0); }
```

8.4.2.1.1.3 template<int dimensions> image<dimensions>* cl::sycl::exception::get_image ( ) [inline]

Get the image that caused the error.

Returns

nullptr if not a image error

Todo Update specification to replace 0 by nullptr and add the templated buffer

Todo to be implemented

Definition at line 650 of file sycl.hpp.

```
00650 { assert(0); }
```

8.4.2.1.1.4 queue* cl::sycl::exception::get_queue ( ) [inline]

Get the queue that caused the error.

Returns

nullptr if not a queue error

Todo Update specification to replace 0 by nullptr

Definition at line 625 of file sycl.hpp.

```
00625 { assert(0); }
```

8.4.2.1.1.5 cl_int cl::sycl::exception::get_sycl_code ( ) [inline]

Get the SYCL-specific error code.

Returns

0 if not a SYCL-specific error

Todo to be implemented

Todo use something else instead of cl_int to be usable without OpenCL

Definition at line 616 of file sycl.hpp.

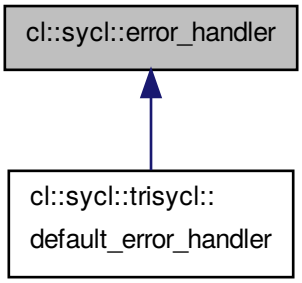
```
00616 { assert(0); }
```

8.4.2.2 struct cl::sycl::error_handler

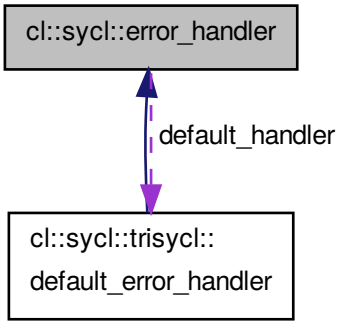
User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

Definition at line 664 of file sycl.hpp.

Inheritance diagram for cl::sycl::error_handler:



Collaboration diagram for cl::sycl::error_handler:



Public Member Functions

- virtual void report_error (exception &error)=0

  The method to define to be called in the case of an error.

Static Public Attributes

- static trisycl::default_error_handler default_handler

  Add a default_handler to be used by default.

8.4.2.2.1 Member Function Documentation

8.4.2.2.1.1 virtual void cl::sycl::error_handler::report_error (exception & *error*) [pure virtual]

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implemented in cl::sycl::trisycl::default_error_handler.

8.4.2.2.2 Member Data Documentation

8.4.2.2.2.1 trisycl::default_error_handler cl::sycl::error_handler::default_handler [static]

Add a default_handler to be used by default.

Todo add this concept to the specification?

Definition at line 675 of file sycl.hpp.

8.5 Platforms, contexts, devices and queues

Classes

- struct cl::sycl::device

  SYCL device. More...
- struct cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- struct cl::sycl::gpu_selector

  Select the best GPU, if any. More...
- struct cl::sycl::context

  SYCL context. More...
- struct cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct cl::sycl::platform

  Abstract the OpenCL platform. More...
- 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. More...

8.5.1 Detailed Description

8.5.2 Class Documentation

8.5.2.1 struct cl::sycl::device

SYCL device.

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

Definition at line 702 of file sycl.hpp.

Public Member Functions

- device ()

8.5.2.1.1 Constructor & Destructor Documentation

8.5.2.1.1.1 cl::sycl::device::device ( ) [inline]

Definition at line 703 of file sycl.hpp.

```
00703 {}
```

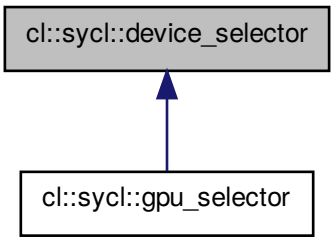
8.5.2.2 struct cl::sycl::device_selector

The SYCL heuristics to select a device.

The device with the highest score is selected

Definition at line 710 of file sycl.hpp.

Inheritance diagram for cl::sycl::device_selector:



Public Member Functions

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

8.5.2.2.1 Member Function Documentation

8.5.2.2.1.1 virtual int cl::sycl::device_selector::operator() (device *dev*) [pure virtual]

Implemented in cl::sycl::gpu_selector.

8.5.2.3 struct cl::sycl::gpu_selector

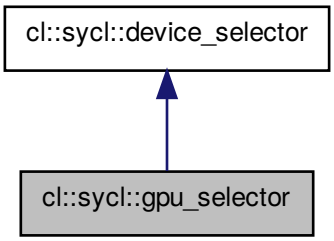
Select the best GPU, if any.

Todo to be implemented

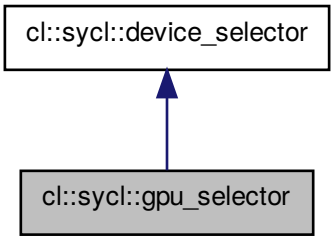
Todo to be named device_selector::gpu instead in the specification?

Definition at line 721 of file sycl.hpp.

Inheritance diagram for cl::sycl::gpu_selector:



Collaboration diagram for cl::sycl::gpu_selector:



Public Member Functions

- int operator() (device dev) override

8.5.2.3.1 Member Function Documentation

8.5.2.3.1.1 int cl::sycl::gpu_selector::operator() (device *dev*) [inline], [override], [virtual]

Implements cl::sycl::device_selector.

Definition at line 723 of file sycl.hpp.

```
00723 { return 1; }
```

8.5.2.4 struct cl::sycl::context

SYCL context.

The implementation is quite minimal for now. :-)

Definition at line 731 of file sycl.hpp.

Public Member Functions

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

8.5.2.4.1 Constructor & Destructor Documentation

8.5.2.4.1.1 cl::sycl::context::context ( ) [inline]

Definition at line 732 of file sycl.hpp.

```
00732 {}
```

8.5.2.4.1.2 cl::sycl::context::context (gpu_selector *s*) [inline]

Definition at line 735 of file sycl.hpp.

```
00735 {}
```

8.5.2.4.1.3 cl::sycl::context::context (device_selector & *s*) [inline]

Definition at line 737 of file sycl.hpp.

```
00737 {}
```

8.5.2.5 struct cl::sycl::queue

SYCL queue, similar to the OpenCL queue concept.

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

Definition at line 744 of file sycl.hpp.

Public Member Functions

- queue ()
- queue (context c)

8.5.2.5.1 Constructor & Destructor Documentation

8.5.2.5.1.1 cl::sycl::queue::queue ( ) [inline]

Definition at line 745 of file sycl.hpp.

```
00745 {}
```

8.5.2.5.1.2 cl::sycl::queue::queue (context *c*) [inline]

Definition at line 747 of file sycl.hpp.

```
00747 {}
```

8.5.2.6 struct cl::sycl::platform

Abstract the OpenCL platform.

Todo triSYCL Implementation

Definition at line 755 of file sycl.hpp.

Public Member Functions

- platform (const error_handler &handler=error_handler::default_handler)

  Construct a default platform and provide an optional error_handler to deals with errors.
- platform (cl_platform_id platform id, const error_handler &handler=error_handler::default_handler)

  Create a SYCL platform from an existing OpenCL one and provide an optional error_handler to deals with errors.
- platform (cl_platform_id platform id, int &error_code)

  Create a SYCL platform from an existing OpenCL one and provide an integer place-holder to return the OpenCL error code, if any.
- ~platform ()

  Destructor of the SYCL abstraction.

- cl_platform_id get ()

  Get the OpenCL platform_id underneath.
- template<cl_int name>
  cl::detail::param_traits
  < cl_platform_info, name >
  ::param_type get_info ()

  Get the OpenCL information about the requested parameter.
- bool is_host ()

  Test if this platform is a host platform.
- bool has_extension (const STRING_CLASS extension_name)

  Test if an extension is available on the platform.

Static Public Member Functions

- static VECTOR_CLASS< platform > get_platforms ()

  Get the list of all the platforms available to the application.
- static VECTOR_CLASS< device > get_devices (cl_device_type device_type=CL_DEVICE_TYPE_ALL)

  Get all the devices of a given type available to the application.

8.5.2.6.1 Constructor & Destructor Documentation

8.5.2.6.1.1 cl::sycl::platform::platform (const error_handler & *handler* = error_handler::default_handler) [inline]

Construct a default platform and provide an optional error_handler to deals with errors.

Todo Add copy/move constructor to the implementation

Todo Add const to the specification

Definition at line 764 of file sycl.hpp.

```
00764 {}
```

8.5.2.6.1.2 cl::sycl::platform::platform (cl_platform_id **platform** *id*, const error_handler & *handler* = error_handler::default_handler) [inline]

Create a SYCL platform from an existing OpenCL one and provide an optional error_handler to deals with errors.

Todo improve specification to accept also a cl.hpp object

Definition at line 772 of file sycl.hpp.

```
00773                                                                    {}
```

8.5.2.6.1.3 cl::sycl::platform::platform (cl_platform_id **platform** *id*, int & *error_code*) [inline]

Create a SYCL platform from an existing OpenCL one and provide an integer place-holder to return the OpenCL error code, if any.

Definition at line 777 of file sycl.hpp.

```
00778                                {}
```

8.5.2.6.1.4 cl::sycl::platform::~platform ( ) [inline]

Destructor of the SYCL abstraction.

Definition at line 782 of file sycl.hpp.

```
00782 {}
```

8.5.2.6.2 Member Function Documentation

8.5.2.6.2.1 cl_platform_id cl::sycl::platform::get ( ) [inline]

Get the OpenCL platform_id underneath.

Todo Add cl.hpp version to the specification

Definition at line 790 of file sycl.hpp.

```
00790 { assert(0); }
```

8.5.2.6.2.2 static VECTOR_CLASS<device> cl::sycl::platform::get_devices (cl_device_type *device_type* = CL_DEVICE_TYPE_ALL) [inline], [static]

Get all the devices of a given type available to the application.

By default returns all the devices.

Definition at line 804 of file sycl.hpp.

```
00804                                                          {
00805     assert(0);
00806   }
```

8.5.2.6.2.3 template<cl_int name> cl::detail::param_traits<cl_platform_info, name>::param_type cl::sycl::platform::get_info ( ) [inline]

Get the OpenCL information about the requested parameter.

Todo It looks like in the specification the cl::detail:: is lacking to fit the cl.hpp version. Or is it to be redefined in SYCL too?

Definition at line 816 of file sycl.hpp.

```
00816             {
00817     assert(0);
00818   }
```

8.5.2.6.2.4 static VECTOR_CLASS<platform> cl::sycl::platform::get_platforms ( ) [inline], [static]

Get the list of all the platforms available to the application.

Definition at line 795 of file sycl.hpp.

```
00795 { assert(0); }
```

8.5.2.6.2.5 bool cl::sycl::platform::has_extension (const STRING_CLASS *extension_name*) [inline]

Test if an extension is available on the platform.

Todo Should it be a param type instead of a STRING?

Todo extend to any type of C++-string like object

Definition at line 835 of file sycl.hpp.

```
00835                                                    {
00836     assert(0);
00837   }
```

8.5.2.6.2.6 bool cl::sycl::platform::is_host ( ) [inline]

Test if this platform is a host platform.

Definition at line 823 of file sycl.hpp.

```
00823                  {
00824     // Right now, this is a host-only implementation :-)
00825     return true;
00826   }
```

8.5.2.7 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.

Definition at line 849 of file sycl.hpp.

Public Member Functions

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

8.5.2.7.1 Constructor & Destructor Documentation

8.5.2.7.1.1 template<typename Functor > cl::sycl::command_group::command_group (queue *Q*, Functor *F*) [inline]

Definition at line 851 of file sycl.hpp.

```
00851                                    {
00852     F();
00853   }
```

Chapter 9

Namespace Documentation

9.1 cl Namespace Reference

SYCL dwells in the cl::sycl namespace.

Namespaces

- sycl

9.1.1 Detailed Description

SYCL dwells in the cl::sycl namespace.

9.2 cl::sycl Namespace Reference

Namespaces

- access

  Describe the type of access by kernels.
- trisycl

Classes

- struct accessor

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

  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. More...
- struct command_group

  SYCL command group gather all the commands needed to execute one or more kernels in a kind of atomic way. More...
- struct context

  SYCL context. More...
- struct device

  SYCL device. More...

- struct device_selector

  The SYCL heuristics to select a device. More...
- struct error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- struct exception

  Encapsulate a SYCL error information. More...
- struct gpu_selector

  Select the best GPU, if any. More...
- struct group

  A group index used in a parallel_for_workitem to specify a work_group. More...
- struct id

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct image
- struct item

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

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

  Abstract the OpenCL platform. More...
- struct queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct range

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

  Abstract the way storage is managed to allow the programmer to control the storage management of buffers. More...

Functions

- template<size_t Dimensions>
  range< Dimensions > operator/ (range< Dimensions > dividend, range< Dimensions > divisor)
- template<size_t Dimensions>
  range< Dimensions > operator* (range< Dimensions > a, range< Dimensions > b)
- template<size_t Dimensions>
  range< Dimensions > operator+ (range< Dimensions > a, range< Dimensions > b)
- template<typename KernelName , typename Functor >
  Functor kernel_lambda (Functor F)

  kernel_lambda specify a kernel to be launch with a single_task or parallel_for
- void single_task (std::function< void(void)> F)

  SYCL single_task launches a computation without parallelism at launch time.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f)

  SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  A variation of SYCL parallel_for to take into account a nd_range<>
- template<typename Range , typename Program , typename ParallelForFunctor >
  void 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 parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

Loop on the work-groups.

- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Loop on the work-items inside a work-group.
- void barrier (int barrier_type)

  The kernel synchronization barrier.

Variables

- int const CL_LOCAL_MEM_FENCE = 123

9.2.1 Function Documentation

9.2.1.1 void cl::sycl::barrier (int *barrier_type*)

The kernel synchronization barrier.

Todo To be implemented

Definition at line 1331 of file sycl.hpp.

```
01331 {}
```

9.2.2 Variable Documentation

9.2.2.1 int const cl::sycl::CL_LOCAL_MEM_FENCE = 123

Definition at line 1333 of file sycl.hpp.

9.3 cl::sycl::access Namespace Reference

Describe the type of access by kernels.

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 }

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

9.3.1 Detailed Description

Describe the type of access by kernels.

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

9.3.2 Enumeration Type Documentation

9.3.2.1 enum cl::sycl::access::mode

This describes the type of the access mode to be used via accessor.

Enumerator

read

write

atomic

read_write

discard_read_write

Definition at line 120 of file sycl.hpp.

```
00120             {
00121     read = 42, //< Why not? Insist on the fact that read_write != read + write
00122     write,
00123     atomic,
00124     read_write,
00125     discard_read_write
00126   };
```

9.3.2.2 enum cl::sycl::access::target

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

Enumerator

global_buffer

constant_buffer

local

image

host_buffer

host_image

image_array

cl_buffer

cl_image

Definition at line 131 of file sycl.hpp.

```
00131               {
00132     global_buffer = 2014, //< Just pick a random number...
00133     constant_buffer,
00134     local,
00135     image,
00136     host_buffer,
00137     host_image,
00138     image_array,
00139     cl_buffer,
00140     cl_image
00141   };
```

9.4 cl::sycl::trisycl Namespace Reference

Classes

- struct AccessorImpl

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

  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. More...
- struct debug

  Class used to trace the construction and destruction of classes that inherit from it.
- struct default_error_handler
- struct GroupImpl

  The implementation of a SYCL group index to specify a work_group in a parallel_for_workitem. More...
- struct IdImpl

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct ItemImpl

  The implementation of a SYCL item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct NDRangeImpl

  The implementation of a ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- struct ParallelForIterate

  A recursive multi-dimensional iterator that ends calling f. More...
- struct ParallelForIterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- struct ParallelOpenMPForIterate

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct RangeImpl

  Define a multi-dimensional index range. More...

Functions

- template<std::size_t Dimensions>
  RangeImpl< Dimensions > operator/ (RangeImpl< Dimensions > dividend, RangeImpl< Dimensions > divisor)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > operator* (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > operator+ (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)

Chapter 10

Class Documentation

10.1 cl::sycl::trisycl::debug< T > Struct Template Reference

Class used to trace the construction and destruction of classes that inherit from it.

`#include "sycl-implementation.hpp"`

Public Member Functions

- debug ()

  Trace the construction with the compiler-dependent mangled named.
- ~debug ()

  Trace the construction with the compiler-dependent mangled named.

10.1.1 Detailed Description

template<typename T>struct cl::sycl::trisycl::debug< T >

Class used to trace the construction and destruction of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Definition at line 26 of file sycl-implementation.hpp.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 template<typename T> cl::sycl::trisycl::debug< T >::debug ( ) [inline]

Trace the construction with the compiler-dependent mangled named.

Definition at line 29 of file sycl-implementation.hpp.

```
00031                 : std::vector<std::intptr_t>, debug<RangeImpl<Dimensions>> {
00032   static_assert(1 <= Dimensions && Dimensions <= 3,
```

10.1.2.2 template<typename T> cl::sycl::trisycl::debug< T >::~debug ( ) [inline]

Trace the construction with the compiler-dependent mangled named.

Definition at line 35 of file sycl-implementation.hpp.

```
00038                       { return *this; };
```

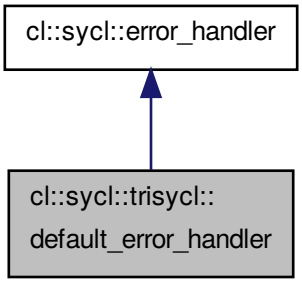
The documentation for this struct was generated from the following file:

- include/CL/implementation/sycl-implementation.hpp

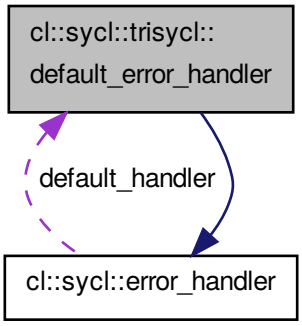
10.2 cl::sycl::trisycl::default_error_handler Struct Reference

`#include "sycl.hpp"`

Inheritance diagram for cl::sycl::trisycl::default_error_handler:



Collaboration diagram for cl::sycl::trisycl::default_error_handler:



Public Member Functions

- void report_error (exception &error) override

  The method to define to be called in the case of an error.

Additional Inherited Members

10.2.1 Detailed Description

Definition at line 681 of file sycl.hpp.

10.2.2 Member Function Documentation

10.2.2.1 void cl::sycl::trisycl::default_error_handler::report_error (exception & *error*) [inline], [override], [virtual]

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implements cl::sycl::error_handler.

Definition at line 683 of file sycl.hpp.

```
00683                                                {
00684     }
```

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

- include/CL/sycl.hpp

10.3 cl::sycl::image< dimensions > Struct Template Reference

```
#include "sycl.hpp"
```

10.3.1 Detailed Description

template<int dimensions>struct cl::sycl::image< dimensions >

Todo implement image

Definition at line 586 of file sycl.hpp.

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

- include/CL/sycl.hpp

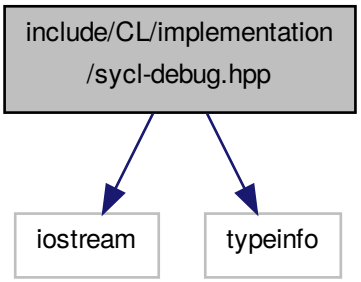
Chapter 11

File Documentation

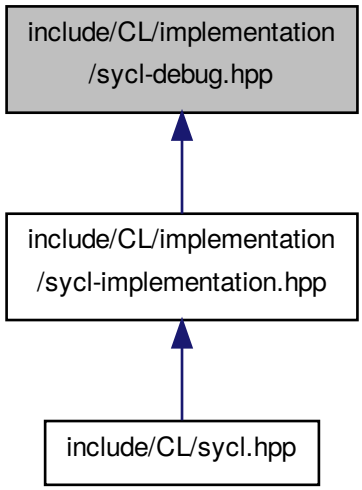
11.1 include/CL/implementation/sycl-debug.hpp File Reference

```
#include <iostream>
#include <typeinfo>
```

Include dependency graph for sycl-debug.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct debug< T >

  Class used to trace the construction and destruction of classes that inherit from it. More...

11.2 sycl-debug.hpp

```
00001 /** \file This is a small class to track constructor/destructor invocations
00002
00003     Define the TRISYCL_DEBUG CPP flag to have an output.
00004
00005     To use it in some class C, make C inherit from debug<C>.
00006
00007     Ronan.Keryell at AMD dot com
00008  */
00009
00010 #ifdef TRISYCL_DEBUG
00011 #include <iostream>
00012 #include <typeinfo>
00013 #endif
00014
00015 /** \addtogroup debug_trace Debugging and tracing support
00016     @{
00017 */
00018
00019 /** Class used to trace the construction and destruction of classes that
00020     inherit from it
00021
00022     \param T is the real type name to be used in the debug output.
00023  */
00024 template <typename T>
00025 struct debug {
00026 #ifdef TRISYCL_DEBUG
00027   /// Trace the construction with the compiler-dependent mangled named
00028   debug() {
00029     std::cerr << " TRISYCL_DEBUG: Constructor of " << typeid(*this).name()
00030               << " " << (void*) this << std::endl;
00031   }
00032
00033   /// Trace the construction with the compiler-dependent mangled named
00034   ~debug() {
```

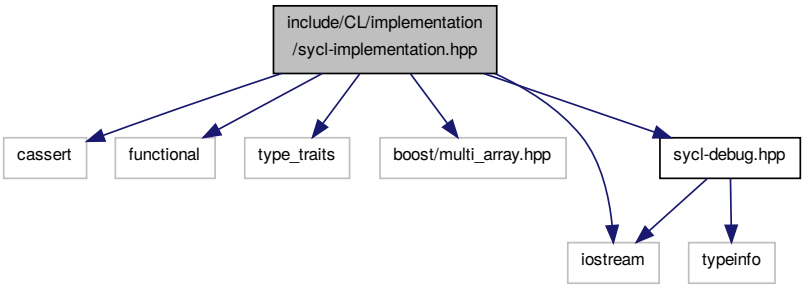
```
00035     std::cerr << " TRISYCL_DEBUG: ~ Destructor of " << typeid(*this).name()
00036               << " " << (void*) this << std::endl;
00037   }
00038 #endif
00039 };
00040
00041 /// @} End the debug_trace Doxygen group
```

11.3 include/CL/implementation/sycl-implementation.hpp File Reference

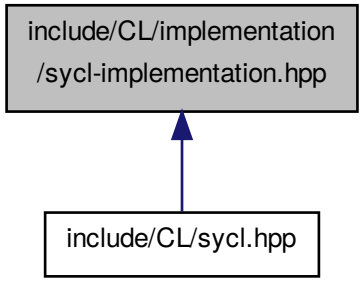
This is a simple C++ sequential OpenCL SYCL implementation to experiment with the OpenCL CL provisional specification.

```
#include <cassert>
#include <functional>
#include <type_traits>
#include "boost/multi_array.hpp"
#include <iostream>
#include "sycl-debug.hpp"
```

Include dependency graph for sycl-implementation.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct cl::sycl::trisycl::debug< T >

Class used to trace the construction and destruction of classes that inherit from it.

- struct cl::sycl::trisycl::RangeImpl< Dimensions >

  Define a multi-dimensional index range. More...

- struct cl::sycl::trisycl::IdImpl< N >

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

- struct cl::sycl::trisycl::NDRangeImpl< dims >

  The implementation of a ND-range, made by a global and local range, to specify work-group and work-item organization. More...

- struct cl::sycl::trisycl::ItemImpl< dims >

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

- struct cl::sycl::trisycl::GroupImpl< N >

  The implementation of a SYCL group index to specify a work_group in a parallel_for_workitem. More...

- struct cl::sycl::trisycl::BufferImpl< 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. More...

- struct cl::sycl::trisycl::AccessorImpl< T, dimensions, mode, target >

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

- struct cl::sycl::trisycl::BufferImpl< 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. More...

- struct cl::sycl::trisycl::ParallelForIterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends calling f. More...

- struct cl::sycl::trisycl::ParallelOpenMPForIterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...

- struct cl::sycl::trisycl::ParallelForIterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...

Namespaces

- cl

  SYCL dwells in the cl::sycl namespace.

- cl::sycl
- cl::sycl::trisycl

Functions

- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator/ (RangeImpl< Dimensions > dividend, RangeImpl< Dimensions > divisor)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator* (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)
- template<std::size_t Dimensions>
  RangeImpl< Dimensions > cl::sycl::trisycl::operator+ (RangeImpl< Dimensions > a, RangeImpl< Dimensions > b)

11.3.1 Detailed Description

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

Ronan.Keryell at AMD point com

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file sycl-implementation.hpp.

11.4 sycl-implementation.hpp

```
00001 /** \file
00002
00003     This is a simple C++ sequential OpenCL SYCL implementation to
00004     experiment with the OpenCL CL provisional specification.
00005
00006     Ronan.Keryell at AMD point com
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cassert>
00013 #include <functional>
00014 #include <type_traits>
00015 #include "boost/multi_array.hpp"
00016 #include <iostream>
00017
00018 /// SYCL dwells in the cl::sycl namespace
00019 namespace cl {
00020 namespace sycl {
00021 namespace trisycl {
00022
00023 #include "sycl-debug.hpp"
00024
00025 /** \addtogroup parallelism
00026     @{
00027 */
00028
00029 /// Define a multi-dimensional index range
00030 template <std::size_t Dimensions = 1U>
00031 struct RangeImpl : std::vector<std::intptr_t>, debug<RangeImpl<Dimensions>> {
00032   static_assert(1 <= Dimensions && Dimensions <= 3,
00033                 "Dimensions are between 1 and 3");
00034
00035   static const auto dimensionality = Dimensions;
00036
00037   // Return a reference to the implementation itself
00038   RangeImpl &getImpl() { return *this; };
00039
00040
00041   // Return a const reference to the implementation itself
00042   const RangeImpl &getImpl() const { return *this; };
00043
00044
00045   /* Inherit the constructors from the parent
00046
00047      Using a std::vector is overkill but std::array has no default
00048      constructors and I am lazy to reimplement them
00049
00050      Use std::intptr_t as a signed version of a std::size_t to allow
00051      computations with negative offsets
00052
00053      \todo in the specification: add some accessors. But it seems they are
00054      implicitly convertible to vectors of the same size in the
00055      specification
00056   */
00057   using std::vector<std::intptr_t>::vector;
00058
00059
00060   // By default, create a vector of Dimensions 0 elements
00061   RangeImpl() : vector(Dimensions) {}
00062
00063
00064   // Copy constructor to initialize from another range
00065   RangeImpl(const RangeImpl &init) : vector(init) {}
00066
00067
00068   // Create a n-D range from an integer-like list
```

```
00069   RangeImpl(std::initializer_list<std::intptr_t> l) :
00070     std::vector<std::intptr_t>(l) {
00071     // The number of elements must match the dimension
00072     assert(Dimensions == l.size());
00073   }
00074
00075
00076   /** Return the given coordinate
00077
00078       \todo explain in the specification (table 3.29, not only in the
00079       text) that [] works also for id, and why not range?
00080
00081       \todo add also [] for range in the specification
00082   */
00083   auto get(int index) {
00084     return (*this)[index];
00085   }
00086
00087   // To debug
00088   void display() {
00089     std::clog << typeid(this).name() << ": ";
00090     for (int i = 0; i < dimensionality; i++)
00091       std::clog << " " << get(i);
00092     std::clog << std::endl;
00093   }
00094
00095 };
00096
00097
00098 // Add some operations on range to help with OpenCL work-group scheduling
00099 // \todo use an element-wise template instead of copy past below for / and *
00100
00101 // An element-wise division of ranges, with upper rounding
00102 template <std::size_t Dimensions>
00103 RangeImpl<Dimensions> operator /(
      RangeImpl<Dimensions> dividend,
00104                                  RangeImpl<Dimensions> divisor) {
00105   RangeImpl<Dimensions> result;
00106
00107   for (int i = 0; i < Dimensions; i++)
00108     result[i] = (dividend[i] + divisor[i] - 1)/divisor[i];
00109
00110   return result;
00111 }
00112
00113
00114 // An element-wise multiplication of ranges
00115 template <std::size_t Dimensions>
00116 RangeImpl<Dimensions> operator *(
      RangeImpl<Dimensions> a,
00117                                  RangeImpl<Dimensions> b) {
00118   RangeImpl<Dimensions> result;
00119
00120   for (int i = 0; i < Dimensions; i++)
00121     result[i] = a[i] * b[i];
00122
00123   return result;
00124 }
00125
00126
00127 // An element-wise addition of ranges
00128 template <std::size_t Dimensions>
00129 RangeImpl<Dimensions> operator +(
      RangeImpl<Dimensions> a,
00130                                  RangeImpl<Dimensions> b) {
00131   RangeImpl<Dimensions> result;
00132
00133   for (int i = 0; i < Dimensions; i++)
00134     result[i] = a[i] + b[i];
00135
00136   return result;
00137 }
00138
00139
00140 /** Define a multi-dimensional index, used for example to locate a work
00141     item
00142
00143     Just rely on the range implementation
00144 */
00145 template <std::size_t N = 1U>
00146 struct IdImpl: RangeImpl<N> {
00147   using RangeImpl<N>::RangeImpl;
00148
00149   /* Since the copy constructor is called with RangeImpl<N>, declare this
00150      constructor to forward it */
00151   IdImpl(const RangeImpl<N> &init) : RangeImpl<N>(init) {}
00152
```

```
00153   // Add back the default constructors canceled by the previous declaration
00154   IdImpl() = default;
00155
00156 };
00157
00158
00159 /** The implementation of a ND-range, made by a global and local range, to
00160     specify work-group and work-item organization.
00161
00162     The local offset is used to translate the iteration space origin if
00163     needed.
00164 */
00165 template <std::size_t dims = 1U>
00166 struct NDRangeImpl {
00167   static_assert(1 <= dims && dims <= 3,
00168                 "Dimensions are between 1 and 3");
00169
00170   static const auto dimensionality = dims;
00171
00172   RangeImpl<dimensionality> GlobalRange;
00173   RangeImpl<dimensionality> LocalRange;
00174   IdImpl<dimensionality> Offset;
00175
00176   NDRangeImpl(RangeImpl<dimensionality> global_size,
00177               RangeImpl<dimensionality> local_size,
00178               IdImpl<dimensionality> offset) :
00179     GlobalRange(global_size),
00180     LocalRange(local_size),
00181     Offset(offset) {}
00182
00183   // Return a reference to the implementation itself
00184   NDRangeImpl &getImpl() { return *this; };
00185
00186
00187   // Return a const reference to the implementation itself
00188   const NDRangeImpl &getImpl() const { return *this; };
00189
00190
00191   RangeImpl<dimensionality> get_global_range() { return
      GlobalRange; }
00192
00193   RangeImpl<dimensionality> get_local_range() { return
      LocalRange; }
00194
00195   /// Get the range of work-groups needed to run this ND-range
00196   RangeImpl<dimensionality> get_group_range() { return
      GlobalRange/LocalRange; }
00197
00198   /// \todo get_offset() is lacking in the specification
00199   IdImpl<dimensionality> get_offset() { return
      Offset; }
00200
00201 };
00202
00203
00204 /** The implementation of a SYCL item stores information on a work-item
00205     within a work-group, with some more context such as the definition
00206     ranges.
00207  */
00208 template <std::size_t dims = 1U>
00209 struct ItemImpl {
00210   static_assert(1 <= dims && dims <= 3,
00211                 "Dimensions are between 1 and 3");
00212
00213   static const auto dimensionality = dims;
00214
00215   IdImpl<dims> GlobalIndex;
00216   IdImpl<dims> LocalIndex;
00217   NDRangeImpl<dims> NDRange;
00218
00219   ItemImpl(RangeImpl<dims> global_size, RangeImpl<dims> local_size) :
00220     NDRange(global_size, local_size) {}
00221
00222   /// \todo a constructor from a nd_range too in the specification?
00223   ItemImpl(NDRangeImpl<dims> ndr) : NDRange(ndr) {}
00224
00225   auto get_global(int dimension) { return GlobalIndex[dimension]; }
00226
00227   auto get_local(int dimension) { return LocalIndex[dimension]; }
00228
00229   auto get_global() { return GlobalIndex; }
00230
00231   auto get_local() { return LocalIndex; }
00232
00233   // For the implementation, need to set the local index
00234   void set_local(IdImpl<dims> Index) { LocalIndex = Index; }
00235
```

```
00236   // For the implementation, need to set the global index
00237   void set_global(IdImpl<dims> Index) { GlobalIndex = Index; }
00238
00239   auto get_local_range() { return NDRange.get_local_range(); }
00240
00241   auto get_global_range() { return NDRange.get_global_range(); }
00242
00243   /// \todo Add to the specification: get_nd_range() and what about the offset?
00244 };
00245
00246
00247 /** The implementation of a SYCL group index to specify a work_group in a
00248     parallel_for_workitem
00249 */
00250 template <std::size_t N = 1U>
00251 struct GroupImpl {
00252   /// Keep a reference on the nd_range to serve potential query on it
00253   const NDRangeImpl<N> &NDR;
00254   /// The coordinate of the group item
00255   IdImpl<N> Id;
00256
00257   GroupImpl(const GroupImpl &g) : NDR(g.NDR), Id(g.Id) {}
00258
00259   GroupImpl(const NDRangeImpl<N> &ndr) : NDR(ndr) {}
00260
00261   GroupImpl(const NDRangeImpl<N> &ndr, const IdImpl<N> &i) :
00262     NDR(ndr), Id(i) {}
00263
00264   /// Return a reference to the implementation itself
00265   GroupImpl &getImpl() { return *this; };
00266
00267   /// Return a const reference to the implementation itself
00268   const GroupImpl &getImpl() const { return *this; };
00269
00270   /// Return the id of this work-group
00271   IdImpl<N> get_group_id() { return Id; }
00272
00273   /// Return the local range associated to this work-group
00274   RangeImpl<N> get_local_range() { return NDR.
      LocalRange; }
00275
00276   /// Return the global range associated to this work-group
00277   RangeImpl<N> get_global_range() { return NDR.
      GlobalRange; }
00278
00279   /** Return the group coordinate in the given dimension
00280
00281       \todo add it to the specification?
00282
00283       \todo is it supposed to be an int? A cl_int? a size_t?
00284   */
00285   auto &operator[](int index) {
00286     return Id[index];
00287   }
00288
00289 };
00290
00291 /// @} End the parallelism Doxygen group
00292
00293
00294 // Forward declaration for use in accessor
00295 template <typename T, std::size_t dimensions> struct BufferImpl;
00296
00297
00298 /** \addtogroup data
00299     @{
00300 */
00301
00302 /** The accessor abstracts the way buffer data are accessed inside a
00303     kernel in a multidimensional variable length array way.
00304
00305     This implementation rely on boost::multi_array to provides this nice
00306     syntax and behaviour.
00307
00308     Right now the aim of this class is just to access to the buffer in a
00309     read-write mode, even if capturing the multi_array_ref from a lambda
00310     make it const (since in some example we have lambda with [=] and
00311     without mutable). The access::mode is not used yet.
00312 */
00313 template <typename T,
00314           std::size_t dimensions,
00315           access::mode mode,
00316           access::target target = access::global_buffer>
00317 struct AccessorImpl {
00318   // The implementation is a multi_array_ref wrapper
00319   typedef boost::multi_array_ref<T, dimensions> ArrayViewType;
00320   ArrayViewType Array;
```

```
00321
00322   // The same type but writable
00323   typedef typename std::remove_const<ArrayViewType>::type WritableArrayViewType;
00324
00325   // \todo in the specification: store the dimension for user request
00326   static const auto dimensionality = dimensions;
00327   // \todo in the specification: store the types for user request as STL
00328   // or C++AMP
00329   using element = T;
00330   using value_type = T;
00331
00332
00333   /// The only way to construct an AccessorImpl is from an existing buffer
00334   // \todo fix the specification to rename target that shadows template parm
00335   AccessorImpl(BufferImpl<T, dimensions> &targetBuffer) :
00336     Array(targetBuffer.Access) {}
00337
00338   /// This is when we access to AccessorImpl[] that we override the const if any
00339   auto &operator[](std::size_t Index) const {
00340     return (const_cast<WritableArrayViewType &>(Array))[Index];
00341   }
00342
00343   /// This is when we access to AccessorImpl[] that we override the const if any
00344   auto &operator[](IdImpl<dimensionality> Index) const {
00345     return (const_cast<WritableArrayViewType &>(Array))(Index);
00346   }
00347
00348   /// \todo Add in the specification because use by HPC-GPU slide 22
00349   auto &operator[](ItemImpl<dimensionality> Index) const {
00350     return (const_cast<WritableArrayViewType &>(Array))(Index.get_global());
00351   }
00352 };
00353
00354
00355 /** A SYCL buffer is a multidimensional variable length array (à la C99
00356     VLA or even Fortran before) that is used to store data to work on.
00357
00358     In the case we initialize it from a pointer, for now we just wrap the
00359     data with boost::multi_array_ref to provide the VLA semantics without
00360     any storage.
00361 */
00362 template <typename T,
00363           std::size_t dimensions = 1U>
00364 struct BufferImpl {
00365   using Implementation = boost::multi_array_ref<T, dimensions>;
00366   // Extension to SYCL: provide pieces of STL container interface
00367   using element = T;
00368   using value_type = T;
00369
00370   // If some allocation is requested, it is managed by this multi_array
00371   boost::multi_array<T, dimensions> Allocation;
00372   // This is the multi-dimensional interface to the data
00373   boost::multi_array_ref<T, dimensions> Access;
00374   // If the data are read-only, store the information for later optimization
00375   bool ReadOnly ;
00376
00377
00378   /// Create a new BufferImpl of size \param r
00379   BufferImpl(RangeImpl<dimensions> const &r) :
      Allocation(r),
00380                                                Access(Allocation),
00381                                                ReadOnly(false) {}
00382
00383
00384   /** Create a new BufferImpl from \param host_data of size \param r without
00385       further allocation */
00386   BufferImpl(T * host_data, RangeImpl<dimensions> r) :
      Access(host_data, r),
00387                                                        ReadOnly(false) {}
00388
00389
00390   /** Create a new read only BufferImpl from \param host_data of size \param r
00391       without further allocation */
00392   BufferImpl(const T * host_data, RangeImpl<dimensions> r) :
00393     Access(host_data, r),
00394     ReadOnly(true) {}
00395
00396
00397   /// \todo
00398   //BufferImpl(storage<T> &store, range<dimensions> r)
00399
00400   /// Create a new allocated 1D BufferImpl from the given elements
00401   BufferImpl(const T * start_iterator, const T * end_iterator) :
00402     // The size of a multi_array is set at creation time
00403     Allocation(boost::extents[std::distance(start_iterator, end_iterator)]),
00404     Access(Allocation) {
00405     /* Then assign Allocation since this is the only multi_array
```

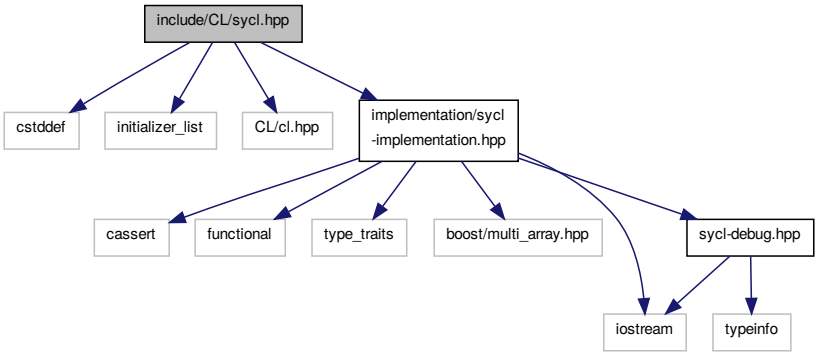
```
00406       method with this iterator interface */
00407     Allocation.assign(start_iterator, end_iterator);
00408   }
00409
00410
00411   /// Create a new BufferImpl from an old one, with a new allocation
00412   BufferImpl(const BufferImpl<T, dimensions> &b) :
      Allocation(b.Access),
00413                                                    Access(Allocation),
00414                                                    ReadOnly(false) {}
00415
00416
00417   /** Create a new sub-BufferImplImpl without allocation to have separate
00418       accessors later */
00419   /* \todo
00420   BufferImpl(BufferImpl<T, dimensions> b,
00421              index<dimensions> base_index,
00422              range<dimensions> sub_range)
00423   */
00424
00425   // Allow CLHPP objects too?
00426   // \todo
00427   /*
00428   BufferImpl(cl_mem mem_object,
00429              queue from_queue,
00430              event available_event)
00431   */
00432
00433   // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00434
00435   /// Return an accessor of the required mode \param M
00436   template <access::mode mode,
00437             access::target target=access::global_buffer>
00438   AccessorImpl<T, dimensions, mode, target>
      get_access() {
00439     return { *this };
00440   }
00441
00442 };
00443
00444 /// @} to end the data Doxygen group
00445
00446
00447 /** \addtogroup parallelism
00448     @{
00449 */
00450
00451 /** A recursive multi-dimensional iterator that ends calling f
00452
00453     The iteration order may be changed later.
00454
00455     Since partial specialization of function template is not possible in
00456     C++14, use a class template instead with everything in the
00457     constructor.
00458 */
00459 template <int level, typename Range, typename ParallelForFunctor, typename Id>
00460 struct ParallelForIterate {
00461   ParallelForIterate(const Range &r, ParallelForFunctor &f, Id &index) {
00462     for (boost::multi_array_types::index _sycl_index = 0,
00463            _sycl_end = r[Range::dimensionality - level];
00464          _sycl_index < _sycl_end;
00465          _sycl_index++) {
00466       // Set the current value of the index for this dimension
00467       index[Range::dimensionality - level] = _sycl_index;
00468       // Iterate further on lower dimensions
00469       ParallelForIterate<level - 1,
00470                          Range,
00471                          ParallelForFunctor,
00472                          Id> { r, f, index };
00473     }
00474   }
00475 };
00476
00477
00478 /** A top-level recursive multi-dimensional iterator variant using OpenMP
00479
00480     Only the top-level loop uses OpenMP and go on with the normal
00481     recursive multi-dimensional.
00482 */
00483 template <int level, typename Range, typename ParallelForFunctor, typename Id>
00484 struct ParallelOpenMPForIterate {
00485   ParallelOpenMPForIterate(const Range &r, ParallelForFunctor &f) {
00486     // Create the OpenMP threads before the for loop to avoid creating an
00487     // index in each iteration
00488 #pragma omp parallel
00489     {
00490       // Allocate an OpenMP thread-local index
```

```
00491       Id index;
00492       // Make a simple loop end condition for OpenMP
00493       boost::multi_array_types::index _sycl_end =
00494         r[Range::dimensionality - level];
00495       /* Distribute the iterations on the OpenMP threads. Some OpenMP
00496          "collapse" could be useful for small iteration space, but it
00497          would need some template specialization to have real contiguous
00498          loop nests */
00499 #pragma omp for
00500       for (boost::multi_array_types::index _sycl_index = 0;
00501            _sycl_index < _sycl_end;
00502            _sycl_index++) {
00503         // Set the current value of the index for this dimension
00504         index[Range::dimensionality - level] = _sycl_index;
00505         // Iterate further on lower dimensions
00506         ParallelForIterate<level - 1,
00507                            Range,
00508                            ParallelForFunctor,
00509                            Id> { r, f, index };
00510       }
00511     }
00512   }
00513 };
00514
00515
00516 /** Stop the recursion when level reaches 0 by simply calling the
00517     kernel functor with the constructed id */
00518 template <typename Range, typename ParallelForFunctor, typename Id>
00519 struct ParallelForIterate<0, Range, ParallelForFunctor, Id> {
00520   ParallelForIterate(const Range &r, ParallelForFunctor &f, Id &index) {
00521     f(index);
00522   }
00523 };
00524
00525 /// @} End the parallelism Doxygen group
00526
00527 }
00528 }
00529 }
00530
00531 /*
00532     # Some Emacs stuff:
00533     ### Local Variables:
00534     ### ispell-local-dictionary: "american"
00535     ### eval: (flyspell-prog-mode)
00536     ### End:
00537 */
```

11.5 include/CL/sycl.hpp File Reference

```
#include <cstddef>
#include <initializer_list>
#include <CL/cl.hpp>
#include "implementation/sycl-implementation.hpp"
```

Include dependency graph for sycl.hpp:



Classes

- struct cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to launch parallel computation. More...
- struct cl::sycl::id< dims >

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct cl::sycl::nd_range< dims >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- 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. More...
- struct cl::sycl::group< dims >

  A group index used in a parallel_for_workitem to specify a work_group. More...
- 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. More...
- struct cl::sycl::image< dimensions >
- struct cl::sycl::exception

  Encapsulate a SYCL error information. More...
- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- struct cl::sycl::trisycl::default_error_handler
- struct cl::sycl::device

  SYCL device. More...
- struct cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- struct cl::sycl::gpu_selector

  Select the best GPU, if any. More...
- struct cl::sycl::context

  SYCL context. More...
- struct cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct cl::sycl::platform

  Abstract the OpenCL platform. More...
- 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. More...
- 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. More...
- struct cl::sycl::storage< T >

  Abstract the way storage is managed to allow the programmer to control the storage management of buffers. More...
- 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. More...

Namespaces

- cl

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

  Describe the type of access by kernels.
- cl::sycl::trisycl

Macros

- #define TRISYCL_IMPL(...) __VA_ARGS__
- #define __CL_ENABLE_EXCEPTIONS

  Define TRISYCL_OPENCL to add OpenCL.
- #define VECTOR_CLASS std::vector

  The vector type to be used as SYCL vector.
- #define STRING_CLASS std::string

  The string type to be used as SYCL string.

Enumerations

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

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

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

Functions

- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator/ (range< Dimensions > dividend, range< Dimensions > divisor)
- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator* (range< Dimensions > a, range< Dimensions > b)
- template<size_t Dimensions>
  range< Dimensions > cl::sycl::operator+ (range< Dimensions > a, range< Dimensions > b)
- 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
- void cl::sycl::single_task (std::function< void(void)> F)

  SYCL single_task launches a computation without parallelism at launch time.
- 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<>.
- 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<>
- 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)

  Loop on the work-groups.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Loop on the work-items inside a work-group.
- void cl::sycl::barrier (int barrier_type)

  The kernel synchronization barrier.

Variables

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

11.5.1 Macro Definition Documentation

11.5.1.1 #define __CL_ENABLE_EXCEPTIONS

Define TRISYCL_OPENCL to add OpenCL.

triSYCL can indeed work without OpenCL if only host support is needed.

Right now it is set by Doxygen to generate the documentation.

Todo Use a macro to check instead if the OpenCL header has been included before.

But what is the right one? **OPENCL_CL_H? __OPENCL_C_VERSION**? CL_HPP_? Mostly CL_HPP_ to be able to use param_traits<> from cl.hpp...

Definition at line 63 of file sycl.hpp.

11.5.1.2 #define STRING_CLASS std::string

The string type to be used as SYCL string.

Todo this should be more local, such as SYCL_STRING_CLASS or _SYCL_STRING_CLASS

Todo use a typedef or a using instead of a macro?

Todo implement __NO_STD_STRING

Todo Table 3.2 in provisional specification is wrong: STRING_CLASS not at the right place

Definition at line 95 of file sycl.hpp.

11.5.1.3 #define TRISYCL_IMPL(...) __VA_ARGS__

Definition at line 43 of file sycl.hpp.

11.5.1.4 #define VECTOR_CLASS std::vector

The vector type to be used as SYCL vector.

Todo this should be more local, such as SYCL_VECTOR_CLASS or _SYCL_VECTOR_CLASS

Todo use a typedef or a using instead of a macro?

Todo implement __NO_STD_VECTOR

Todo Table 3.1 in provisional specification is wrong: VECTOR_CLASS not at the right place

Definition at line 80 of file sycl.hpp.

11.6 sycl.hpp

```
00001 /** \file
00002
00003     \mainpage
00004
00005     This is a simple C++ sequential OpenCL SYCL C++ header file to
00006     experiment with the OpenCL CL provisional specification.
00007
00008     For more information about OpenCL SYCL:
00009     http://www.khronos.org/opencl/sycl/
00010
00011     The aim of this file is mainly to define the interface of SYCL so that
00012     the specification documentation can be derived from it through tools
00013     like Doxygen or Sphinx. This explains why there are many functions and
00014     classes that are here only to do some forwarding in some inelegant way.
00015     This file is documentation driven and not implementation-style driven.
00016
00017     For more information on this project and to access to the source of
00018     this file, look at https://github.com/amd/triSYCL
00019
00020     The Doxygen version of the API in
00021     http://amd.github.io/triSYCL/Doxygen/SYCL/html and
00022     http://amd.github.io/triSYCL/Doxygen/SYCL/SYCL-API-refman.pdf
00023
00024     The Doxygen version of the implementation itself is in
00025     http://amd.github.io/triSYCL/Doxygen/triSYCL/html and
00026     http://amd.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf
00027
00028
00029     Ronan.Keryell at AMD point com
00030
00031     This file is distributed under the University of Illinois Open Source
00032     License. See LICENSE.TXT for details.
00033 */
00034
00035
00036 /* To remove some implementation details from the SYCL API documentation,
00037    rely on the preprocessor when this preprocessor symbol is defined */
00038 #ifdef TRISYCL_HIDE_IMPLEMENTATION
00039 // Remove the content of TRISYCL_IMPL...
00040 #define TRISYCL_IMPL(...)
00041 #else
00042 // ... or keep the content of TRISYCL_IMPL
00043 #define TRISYCL_IMPL(...) __VA_ARGS__
00044 #endif
00045
00046 #include <cstddef>
00047 #include <initializer_list>
00048
00049
00050 /** Define TRISYCL_OPENCL to add OpenCL
00051
00052     triSYCL can indeed work without OpenCL if only host support is needed.
00053
00054     Right now it is set by Doxygen to generate the documentation.
00055
00056     \todo Use a macro to check instead if the OpenCL header has been
00057     included before.
00058
00059     But what is the right one? __OPENCL_CL_H? __OPENCL_C_VERSION__? CL_HPP_?
00060     Mostly CL_HPP_ to be able to use param_traits<> from cl.hpp...
00061 */
00062 #ifdef TRISYCL_OPENCL
00063 #define __CL_ENABLE_EXCEPTIONS
00064 #include <CL/cl.hpp>
00065 #endif
00066
00067
00068 /** The vector type to be used as SYCL vector
00069
00070     \todo this should be more local, such as SYCL_VECTOR_CLASS or
00071     _SYCL_VECTOR_CLASS
00072
00073     \todo use a typedef or a using instead of a macro?
00074
00075     \todo implement __NO_STD_VECTOR
00076
00077     \todo Table 3.1 in provisional specification is wrong: VECTOR_CLASS
00078     not at the right place
00079  */
00080 #define VECTOR_CLASS std::vector
00081
00082
00083 /** The string type to be used as SYCL string
00084
```

```
00085     \todo this should be more local, such as SYCL_STRING_CLASS or
00086     _SYCL_STRING_CLASS
00087
00088     \todo use a typedef or a using instead of a macro?
00089
00090     \todo implement __NO_STD_STRING
00091
00092     \todo Table 3.2 in provisional specification is wrong: STRING_CLASS
00093     not at the right place
00094  */
00095 #define STRING_CLASS std::string
00096
00097
00098 // SYCL dwells in the cl::sycl namespace
00099 namespace cl {
00100 namespace sycl {
00101
00102
00103 /** \addtogroup data Data access and storage in SYCL
00104
00105     @{
00106 */
00107
00108 /** Describe the type of access by kernels.
00109
00110     \todo This values should be normalized to allow separate compilation
00111     with different implementations?
00112 */
00113 namespace access {
00114   /* By using "enum mode" here instead of "enum struct mode", we have for
00115      example "write" appearing both as cl::sycl::access::mode::write and
00116      cl::sycl::access::write, instead of only the last one. This seems
00117      more conform to the specification. */
00118
00119   /// This describes the type of the access mode to be used via accessor
00120   enum mode {
00121     read = 42, //< Why not? Insist on the fact that read_write != read + write
00122     write,
00123     atomic,
00124     read_write,
00125     discard_read_write
00126   };
00127
00128   /** The target enumeration describes the type of object to be accessed
00129      via the accessor
00130    */
00131   enum target {
00132     global_buffer = 2014, //< Just pick a random number...
00133     constant_buffer,
00134     local,
00135     image,
00136     host_buffer,
00137     host_image,
00138     image_array,
00139     cl_buffer,
00140     cl_image
00141   };
00142
00143 }
00144
00145 /// @} End the data Doxygen group
00146
00147 }
00148 }
00149
00150
00151
00152 #include "implementation/sycl-implementation.hpp"
00153
00154
00155 /// SYCL dwells in the cl::sycl namespace
00156 namespace cl {
00157 namespace sycl {
00158
00159 using namespace trisycl;
00160
00161 /** \addtogroup parallelism Expressing parallelism through kernels
00162     @{
00163 */
00164
00165 /** A SYCL range defines a multi-dimensional index range that can
00166     be used to launch parallel computation.
00167
00168     \todo use std::size_t dims instead of int dims in the specification?
00169
00170     \todo add to the norm this default parameter value?
00171
```

```
00172     \todo add to the norm some way to specify an offset?
00173 */
00174 template <int dims = 1>
00175 struct range TRISYCL_IMPL(: public RangeImpl<dims>) {
00176
00177   /// \todo add this Boost::multi_array or STL concept to the
00178   /// specification?
00179   static const auto dimensionality = dims;
00180
00181 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00182   // A shortcut name to the implementation
00183   using Impl = RangeImpl<dims>;
00184
00185   /** Construct a range from an implementation, used by nd_range() for
00186       example
00187
00188       This is internal and should not appear in the specification */
00189   range(Impl &r) : Impl(r) {}
00190
00191   range(const Impl &r) : Impl(r) {}
00192 #endif
00193
00194   range(range<dims> &r) : Impl(r.getImpl()) {}
00195
00196   range(const range<dims> &r) : Impl(r.getImpl()) {}
00197
00198
00199   /** Create a n-D range from a positive integer-like list
00200
00201       \todo This is not the same as the range(dim1,...) constructor from
00202       the specification
00203    */
00204   range(std::initializer_list<std::intptr_t> l) : Impl(l) {}
00205
00206
00207   /* A variadic template cannot be used because of conflicts with the
00208      constructor taking 2 iterators... So let's go verbose.
00209
00210      \todo Add a make_range() helper too to avoid specifying the
00211      dimension? Generalize this helper to anything?
00212   */
00213
00214   /// To have implicit conversion from 1 integer
00215   range(std::intptr_t x) : range { x } {
00216     static_assert(dims == 1, "A range with 1 size value should be 1-D");
00217   }
00218
00219
00220   /// A 2-D constructor from 2 integers
00221   range(std::intptr_t x, std::intptr_t y) : range { x, y } {
00222     static_assert(dims == 2, "A range with 2 size values should be 2-D");
00223   }
00224
00225
00226   /// A 3-D constructor from 3 integers
00227   range(std::intptr_t x, std::intptr_t y, std::intptr_t z) : range { x, y, z } {
00228     static_assert(dims == 3, "A range with 3 size values should be 3-D");
00229   }
00230
00231
00232   /** Return the range size in the give dimension
00233
00234       \todo explain in the specification (table 3.29, not only in the
00235       text) that [] works also for id, and why not range?
00236
00237       \todo add also [] for range in the specification
00238
00239       \todo is it supposed to be an int? A cl_int? a size_t?
00240   */
00241   int get(int index) {
00242     return (*this)[index];
00243   }
00244
00245 };
00246
00247
00248 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00249 // Add some operations on range to help with OpenCL work-group scheduling
00250 // \todo use an element-wise template instead of copy past below for / and *
00251
00252 // An element-wise division of ranges, with upper rounding
00253 template <size_t Dimensions>
00254 range<Dimensions> operator /(range<Dimensions> dividend,
00255                              range<Dimensions> divisor) {
00256   range<Dimensions> result;
00257
00258   for (int i = 0; i < Dimensions; i++)
```

```
00259     result[i] = (dividend[i] + divisor[i] - 1)/divisor[i];
00260
00261   return result;
00262 }
00263
00264
00265 // An element-wise multiplication of ranges
00266 template <size_t Dimensions>
00267 range<Dimensions> operator *(range<Dimensions> a,
00268                              range<Dimensions> b) {
00269   range<Dimensions> result;
00270
00271   for (int i = 0; i < Dimensions; i++)
00272     result[i] = a[i] * b[i];
00273
00274   return result;
00275 }
00276
00277
00278 // An element-wise addition of ranges
00279 template <size_t Dimensions>
00280 range<Dimensions> operator +(range<Dimensions> a,
00281                              range<Dimensions> b) {
00282   range<Dimensions> result;
00283
00284   for (int i = 0; i < Dimensions; i++)
00285     result[i] = a[i] + b[i];
00286
00287   return result;
00288 }
00289 #endif
00290
00291
00292 /** Define a multi-dimensional index, used for example to locate a work item
00293
00294     \todo The definition of id and item seem completely broken in the
00295     current specification. The whole 3.4.1 is to be updated.
00296
00297     \todo It would be nice to have [] working everywhere, provide both
00298     get_...() and get_...(int dim) equivalent to get_...()[int dim]
00299     Well it is already the case for item. So not needed for id?
00300     Indeed [] is mentioned in text of page 59 but not in class description.
00301 */
00302 template <int dims = 1>
00303 struct id TRISYCL_IMPL(: public IdImpl<dims>) {
00304
00305   /// \todo add this Boost::multi_array or STL concept to the
00306   /// specification?
00307   static const auto dimensionality = dims;
00308
00309
00310 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00311   // A shortcut name to the implementation
00312   using Impl = IdImpl<dims>;
00313
00314
00315   /** Since the runtime needs to construct an id from its implementation
00316       for example in item methods, define some hidden constructor here */
00317   id(const Impl &init) : Impl(init) {}
00318 #endif
00319
00320
00321   /** Create a zero id
00322
00323       \todo Add it to the specification?
00324   */
00325   id() : Impl() {}
00326
00327
00328   /// Create an id with the same value of another one
00329   id(const id &init) : Impl(init.getImpl()) {}
00330
00331   /** Create an id from a given range
00332       \todo Is this necessary?
00333
00334       \todo why in the specification
00335       id<int dims>(range<dims>global_size, range<dims> local_size)
00336       ?
00337   */
00338   id(const range<dims> &r) : Impl(r.getImpl()) {}
00339
00340
00341   /** Create a n-D range from a positive integer-like list
00342
00343       \todo Add this to the specification? Since it is said to be usable
00344       as a std::vector<>...
00345   */
```

```
00346   id(std::initializer_list<std::intptr_t> l) : Impl(l) {}
00347
00348
00349   /** To have implicit conversion from 1 integer
00350
00351       \todo Extension to the specification
00352   */
00353   id(std::intptr_t s) : id({ s }) {
00354     static_assert(dims == 1, "A range with 1 size should be 1-D");
00355   }
00356
00357
00358   /** Return the id size in the given dimension
00359
00360       \todo is it supposed to be an int? A cl_int? a size_t?
00361   */
00362   int get(int index) {
00363     return (*this)[index];
00364   }
00365
00366
00367   /** Return the id size in the given dimension
00368
00369       \todo explain in the specification (table 3.29, not only in the
00370       text) that [] works also for id, and why not range?
00371
00372       \todo add also [] for range in the specification
00373
00374       \todo is it supposed to be an int? A cl_int? a size_t?
00375   */
00376   auto &operator[](int index) {
00377     return (*this).getImpl()[index];
00378   }
00379
00380 };
00381
00382
00383 /** A ND-range, made by a global and local range, to specify work-group
00384     and work-item organization.
00385
00386     The local offset is used to translate the iteration space origin if
00387     needed.
00388
00389     \todo add copy constructors in the specification
00390 */
00391 template <int dims = 1>
00392 struct nd_range TRISYCL_IMPL(: NDRangeImpl<dims>) {
00393   static_assert(1 <= dims && dims <= 3,
00394                 "Dimensions are between 1 and 3");
00395
00396   /// \todo add this Boost::multi_array or STL concept to the
00397   /// specification?
00398   static const auto dimensionality = dims;
00399
00400
00401 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00402   // A shortcut name to the implementation
00403   using Impl = NDRangeImpl<dims>;
00404
00405   /** Since the runtime needs to construct a nd_range from its
00406       implementation for example in parallel_for stuff, define some hidden
00407       constructor here */
00408   nd_range(const Impl &init) : Impl(init) {}
00409 #endif
00410
00411
00412   /** Construct a ND-range with all the details available in OpenCL
00413
00414       By default use a zero offset, that is iterations start at 0
00415    */
00416   nd_range(range<dims> global_size,
00417            range<dims> local_size,
00418            id<dims> offset = id<dims>()) :
00419     Impl(global_size.getImpl(), local_size.getImpl(), offset.getImpl()) {}
00420
00421
00422   /// Get the global iteration space range
00423   range<dims> get_global_range() { return Impl::get_global_range(); }
00424
00425
00426   /// Get the local part of the iteration space range
00427   range<dims> get_local_range() { return Impl::get_local_range(); }
00428
00429
00430   /// Get the range of work-groups needed to run this ND-range
00431   range<dims> get_group_range() { return Impl::get_group_range(); }
00432
```

```
00433
00434   /// \todo get_offset() is lacking in the specification
00435   range<dims> get_offset() { return Impl::get_offset(); }
00436
00437 };
00438
00439
00440 /** A SYCL item stores information on a work-item within a work-group,
00441     with some more context such as the definition ranges.
00442
00443     \todo Add to the specification: get_nd_range() to be coherent with
00444     providing get_local...() and get_global...() and what about the
00445     offset?
00446 */
00447 template <int dims = 1>
00448 struct item TRISYCL_IMPL(: ItemImpl<dims>) {
00449   /// \todo add this Boost::multi_array or STL concept to the
00450   /// specification?
00451   static const auto dimensionality = dims;
00452
00453 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00454   // A shortcut name to the implementation
00455   using Impl = ItemImpl<dims>;
00456 #endif
00457
00458
00459   /** Create an item from a local size and local size
00460
00461       \todo what is the meaning of this constructor for a programmer?
00462   */
00463   item(range<dims> global_size, range<dims> local_size) :
00464     Impl(global_size, local_size) {}
00465
00466
00467   /** \todo a constructor from a nd_range too in the specification if the
00468       previous one has a meaning?
00469    */
00470   item(nd_range<dims> ndr) : Impl(ndr) {}
00471
00472
00473   /// Return the global coordinate in the given dimension
00474   int get_global(int dimension) { return Impl::get_global(dimension); }
00475
00476
00477   /// Return the local coordinate (that is in the work-group) in the given
00478   /// dimension
00479   int get_local(int dimension) { return Impl::get_local(dimension); }
00480
00481
00482   /// Get the whole global id coordinate
00483   id<dims> get_global() { return Impl::get_global(); }
00484
00485
00486   /// Get the whole local id coordinate (which is respective to the
00487   /// work-group)
00488   id<dims> get_local() { return Impl::get_local(); }
00489
00490
00491   /// Get the global range where this item rely in
00492   range<dims> get_global_range() { return Impl::get_global_range(); }
00493
00494   /// Get the local range (the dimension of the work-group) for this item
00495   range<dims> get_local_range() { return Impl::get_local_range(); }
00496
00497   /// \todo Why the offset is not available here?
00498
00499   /// \todo Also provide access to the current nd_range?
00500
00501 };
00502
00503
00504 /** A group index used in a parallel_for_workitem to specify a work_group
00505  */
00506 template <int dims = 1>
00507 struct group TRISYCL_IMPL(: GroupImpl<dims>) {
00508   /// \todo add this Boost::multi_array or STL concept to the
00509   /// specification?
00510   static const auto dimensionality = dims;
00511
00512 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00513   // A shortcut name to the implementation
00514   using Impl = GroupImpl<dims>;
00515
00516   /** Since the runtime needs to construct a group with the right content,
00517       define some hidden constructor for this.  Since it is internal,
00518       directly use the implementation
00519   */
```

```
00520   group(const NDRangeImpl<dims> &NDR, const IdImpl<dims> &ID) :
      Impl(NDR, ID) {}
00521
00522
00523   /** Some internal constructor without group id initialization  */
00524   group(const NDRangeImpl<dims> &NDR) : Impl(NDR) {}
00525 #endif
00526
00527
00528   /// \todo in the specification, only provide a copy constructor. Any
00529   /// other constructors should be unspecified
00530   group(const group &g) : Impl(g.getImpl()) {}
00531
00532
00533   id<dims> get_group_id() { return Impl::get_group_id(); }
00534
00535   /** Get the local range for this work_group
00536
00537       \todo Update the specification to return a range<dims> instead of an
00538       id<>
00539   */
00540   range<dims> get_local_range() { return Impl::get_local_range(); }
00541
00542   /** Get the local range for this work_group
00543
00544       \todo Update the specification to return a range<dims> instead of an
00545       id<>
00546   */
00547   range<dims> get_global_range() { return Impl::get_global_range(); }
00548
00549   /// \todo Why the offset is not available here?
00550
00551   /// \todo Also provide this access to the current nd_range
00552   nd_range<dims> get_nr_range() { return Impl::NDR; }
00553
00554   /** Return the group coordinate in the given dimension
00555
00556       \todo add it to the specification?
00557
00558       \todo is it supposed to be an int? A cl_int? a size_t?
00559   */
00560   int get(int index) {
00561     return (*this)[index];
00562   }
00563
00564
00565   /** Return the group coordinate in the given dimension
00566
00567       \todo add it to the specification?
00568
00569       \todo is it supposed to be an int? A cl_int? a size_t?
00570   */
00571   auto &operator[](int index) {
00572     return (*this).getImpl()[index];
00573   }
00574
00575 };
00576
00577 /// @} End the parallelism Doxygen group
00578
00579 /* Forward definitions (outside the Doxygen addtogroup to avoid multiple
00580    definitions) */
00581 struct queue;
00582
00583 template <typename T, int dimensions> struct buffer;
00584
00585   /// \todo implement image
00586 template <int dimensions> struct image;
00587
00588
00589 /** \addtogroup error_handling Error handling
00590     @{
00591 */
00592
00593 /**
00594    Encapsulate a SYCL error information
00595 */
00596 struct exception {
00597 #ifdef TRISYCL_OPENCL
00598   /** Get the OpenCL error code
00599
00600       \returns 0 if not an OpenCL error
00601
00602       \todo to be implemented
00603   */
00604   cl_int get_cl_code() { assert(0); }
00605
```

```
00606
00607   /** Get the SYCL-specific error code
00608
00609       \returns 0 if not a SYCL-specific error
00610
00611       \todo to be implemented
00612
00613       \todo use something else instead of cl_int to be usable without
00614       OpenCL
00615   */
00616   cl_int get_sycl_code() { assert(0); }
00617 #endif
00618
00619   /** Get the queue that caused the error
00620
00621       \return nullptr if not a queue error
00622
00623       \todo Update specification to replace 0 by nullptr
00624   */
00625   queue *get_queue() { assert(0); }
00626
00627
00628   /** Get the buffer that caused the error
00629
00630       \returns nullptr if not a buffer error
00631
00632       \todo Update specification to replace 0 by nullptr and add the
00633       templated buffer
00634
00635       \todo to be implemented
00636   */
00637   template <typename T, int dimensions> buffer<T, dimensions> *
      get_buffer() {
00638     assert(0); }
00639
00640
00641   /** Get the image that caused the error
00642
00643       \returns nullptr if not a image error
00644
00645       \todo Update specification to replace 0 by nullptr and add the
00646       templated buffer
00647
00648       \todo to be implemented
00649   */
00650   template <int dimensions> image<dimensions> *get_image() { assert(0); }
00651 };
00652
00653
00654 namespace trisycl {
00655   // Create a default error handler to be used when nothing is specified
00656   struct default_error_handler;
00657 }
00658
00659
00660 /** User supplied error handler to call a user-provided function when an
00661     error happens from a SYCL object that was constructed with this error
00662     handler
00663 */
00664 struct error_handler {
00665   /** The method to define to be called in the case of an error
00666
00667       \todo Add "virtual void" to the specification
00668   */
00669   virtual void report_error(exception &error) = 0;
00670
00671   /** Add a default_handler to be used by default
00672
00673       \todo add this concept to the specification?
00674   */
00675   static trisycl::default_error_handler
      default_handler;
00676 };
00677
00678
00679 namespace trisycl {
00680
00681   struct default_error_handler : error_handler {
00682
00683     void report_error(exception &error) override {
00684     }
00685   };
00686 }
00687
00688   // \todo finish initialization
00689   //error_handler::default_handler = nullptr;
00690
```

```
00691 /// @} End the error_handling Doxygen group
00692
00693
00694 /** \addtogroup execution Platforms, contexts, devices and queues
00695     @{
00696 */
00697
00698 /** SYCL device
00699
00700     \todo The implementation is quite minimal for now. :-)
00701 */
00702 struct device {
00703   device() {}
00704 };
00705
00706 /** The SYCL heuristics to select a device
00707
00708     The device with the highest score is selected
00709 */
00710 struct device_selector {
00711   // The user-provided operator computing the score
00712   virtual int operator() (device dev) = 0;
00713 };
00714
00715 /** Select the best GPU, if any
00716
00717     \todo to be implemented
00718
00719     \todo to be named device_selector::gpu instead in the specification?
00720 */
00721 struct gpu_selector : device_selector {
00722   // The user-provided operator computing the score
00723   int operator() (device dev) override { return 1; }
00724 };
00725
00726
00727 /** SYCL context
00728
00729     The implementation is quite minimal for now. :-)
00730 */
00731 struct context {
00732   context() {}
00733
00734   // \todo fix this implementation
00735   context(gpu_selector s) {}
00736
00737   context(device_selector &s) {}
00738 };
00739
00740 /** SYCL queue, similar to the OpenCL queue concept.
00741
00742     \todo The implementation is quite minimal for now. :-)
00743 */
00744 struct queue {
00745   queue() {}
00746
00747   queue(context c) {}
00748 };
00749
00750
00751 /** Abstract the OpenCL platform
00752
00753     \todo triSYCL Implementation
00754 */
00755 struct platform {
00756
00757   /** Construct a default platform and provide an optional error_handler
00758       to deals with errors
00759
00760       \todo Add copy/move constructor to the implementation
00761
00762       \todo Add const to the specification
00763   */
00764   platform(const error_handler &handler =
      error_handler::default_handler) {}
00765
00766 #ifdef TRISYCL_OPENCL
00767   /** Create a SYCL platform from an existing OpenCL one and provide an
00768       optional error_handler to deals with errors
00769
00770       \todo improve specification to accept also a cl.hpp object
00771   */
00772   platform(cl_platform_id platform id,
00773            const error_handler &handler =
      error_handler::default_handler) {}
00774
00775   /** Create a SYCL platform from an existing OpenCL one and provide an
```

```
00776       integer place-holder to return the OpenCL error code, if any */
00777   platform(cl_platform_id platform id,
00778            int &error_code) {}
00779 #endif
00780
00781   /** Destructor of the SYCL abstraction */
00782   ~platform() {}
00783
00784
00785 #ifdef TRISYCL_OPENCL
00786   /** Get the OpenCL platform_id underneath
00787
00788    \todo Add cl.hpp version to the specification
00789   */
00790   cl_platform_id get() { assert(0); }
00791 #endif
00792
00793
00794   /** Get the list of all the platforms available to the application */
00795   static VECTOR_CLASS<platform> get_platforms() { assert(0); }
00796
00797
00798 #ifdef TRISYCL_OPENCL
00799   /** Get all the devices of a given type available to the application.
00800
00801       By default returns all the devices.
00802   */
00803   static VECTOR_CLASS<device>
00804   get_devices(cl_device_type device_type = CL_DEVICE_TYPE_ALL) {
00805     assert(0);
00806   }
00807
00808
00809   /** Get the OpenCL information about the requested parameter
00810
00811       \todo It looks like in the specification the cl::detail:: is lacking
00812       to fit the cl.hpp version. Or is it to be redefined in SYCL too?
00813   */
00814   template<cl_int name> typename
00815   cl::detail::param_traits<cl_platform_info, name>::param_type
00816   get_info() {
00817     assert(0);
00818   }
00819 #endif
00820
00821
00822   /** Test if this platform is a host platform */
00823   bool is_host() {
00824     // Right now, this is a host-only implementation :-)
00825     return true;
00826   }
00827
00828
00829   /** Test if an extension is available on the platform
00830
00831       \todo Should it be a param type instead of a STRING?
00832
00833       \todo extend to any type of C++-string like object
00834   */
00835   bool has_extension(const STRING_CLASS extension_name) {
00836     assert(0);
00837   }
00838
00839 };
00840
00841
00842 /** SYCL command group gather all the commands needed to execute one or
00843     more kernels in a kind of atomic way. Since all the parameters are
00844     captured at command group creation, one can execute the content in an
00845     asynchronous way and delayed schedule.
00846
00847     For now just execute the command group directly.
00848  */
00849 struct command_group {
00850   template <typename Functor>
00851   command_group(queue Q, Functor F) {
00852     F();
00853   }
00854 };
00855
00856 /// @} to end the execution Doxygen group
00857
00858
00859 /** \addtogroup data
00860     @{
00861 */
00862
```

```
00863 /** The accessor abstracts the way buffer data are accessed inside a
00864     kernel in a multidimensional variable length array way.
00865
00866     \todo Implement it for images according so section 3.3.4.5
00867 */
00868 template <typename dataType,
00869           size_t dimensions,
00870           access::mode mode,
00871           access::target target = access::global_buffer>
00872 struct accessor
00873 TRISYCL_IMPL(: AccessorImpl<dataType, dimensions, mode, target>) {
00874   /// \todo in the specification: store the dimension for user request
00875   static const auto dimensionality = dimensions;
00876   /// \todo in the specification: store the types for user request as STL
00877   // or C++AMP
00878   using element = dataType;
00879   using value_type = dataType;
00880
00881 #ifndef TRISYCL_HIDE_IMPLEMENTATION
00882   // Use a short-cut to the implementation because type name becomes quite
00883   // long...
00884   using Impl = AccessorImpl<dataType, dimensions, mode, target>
      ;
00885 #endif
00886
00887   /// Create an accessor to the given buffer
00888   // \todo fix the specification to rename target that shadows template parm
00889   accessor(buffer<dataType, dimensions> &targetBuffer) :
00890     Impl(targetBuffer) {}
00891
00892
00893   /** Get the element specified by the given id
00894
00895       \todo Implement the "const dataType &" version in the case the
00896       accessor is not for writing, as required by the specification
00897   */
00898   dataType &operator[](id<dimensionality> Index) const {
00899     return Impl::operator[](Index);
00900   }
00901
00902
00903   /** Get the element specified by the given index in the case we are
00904       mono-dimensional
00905
00906       \todo This is not in the specification but looks like a cool common
00907       feature. Or solving it with an implicit constructor of id<1>?
00908   */
00909   dataType &operator[](size_t Index) const {
00910     return Impl::operator[](Index);
00911   }
00912
00913
00914   /** Get the element specified by the given item
00915
00916       \todo Add in the specification because used by HPC-GPU slide 22
00917   */
00918   dataType &operator[](item<dimensionality> Index) const {
00919     return Impl::operator[](Index);
00920   }
00921
00922 };
00923
00924
00925 /** Abstract the way storage is managed to allow the programmer to control
00926     the storage management of buffers
00927
00928     \param T
00929     the type of the elements of the underlying data
00930
00931     The user is responsible for ensuring that their storage class
00932     implementation is thread-safe.
00933 */
00934 template <typename T>
00935 struct storage {
00936   /// \todo Extension to SYCL specification: provide pieces of STL
00937   /// container interface?
00938   using element = T;
00939   using value_type = T;
00940
00941
00942   /** Method called by SYCL system to get the number of elements of type T
00943       of the underlying data
00944
00945       \todo This is inconsistent in the specification with get_size() in
00946       buffer which returns the byte size. Is it to be renamed to
00947       get_count()?
00948   */
```

```
00949   virtual size_t get_size() = 0;
00950
00951
00952   /** Method called by the SYCL system to know where that data is held in
00953       host memory
00954
00955       \return the address or nullptr if SYCL has to manage the temporary
00956       storage of the data.
00957   */
00958   virtual T* get_host_data() = 0;
00959
00960
00961   /** Method called by the SYCL system at the point of construction to
00962       request the initial contents of the buffer
00963
00964       \return the address of the data to use or nullptr to skip this data
00965       initialization
00966   */
00967   virtual const T* get_initial_data() = 0;
00968
00969
00970   /** Method called at the point of construction to request where the
00971       content of the buffer should be finally stored to
00972
00973       \return the address of where the buffer will be written to in host
00974       memory.
00975
00976       If the address is nullptr, then this phase is skipped.
00977
00978       If get_host_data() returns the same pointer as get_initial_data()
00979       and/or get_final_data() then the SYCL system should determine whether
00980       copying is actually necessary or not.
00981   */
00982   virtual T* get_final_data() = 0;
00983
00984
00985   /** Method called when the associated memory object is destroyed.
00986
00987       This method is only called once, so if a memory object is copied
00988       multiple times, only when the last copy of the memory object is
00989       destroyed is the destroy method called.
00990
00991       Exceptions thrown by the destroy method will be caught and ignored.
00992   */
00993   virtual void destroy() = 0;
00994
00995
00996   /** \brief Method called when a command_group which accesses the data is
00997       added to a queue
00998
00999      After completed is called, there may be further calls of
01000       in_use() if new work is enqueued that operates on the memory object.
01001   */
01002   virtual void in_use() = 0;
01003
01004
01005   /// Method called when the final enqueued command has completed
01006   virtual void completed() = 0;
01007 };
01008
01009
01010 /** A SYCL buffer is a multidimensional variable length array (à la C99
01011     VLA or even Fortran before) that is used to store data to work on.
01012
01013     In the case we initialize it from a pointer, for now we just wrap the
01014     data with boost::multi_array_ref to provide the VLA semantics without
01015     any storage.
01016
01017     \todo there is a naming inconsistency in the specification between
01018     buffer and accessor on T versus datatype
01019 */
01020 template <typename T,
01021           int dimensions = 1>
01022 struct buffer TRISYCL_IMPL(: BufferImpl<T, dimensions>) {
01023   /// \todo Extension to SYCL specification: provide pieces of STL
01024   /// container interface?
01025   using element = T;
01026   using value_type = T;
01027
01028 #ifndef TRISYCL_HIDE_IMPLEMENTATION
01029   // Use a short-cut because type name becomes quite long...
01030   using Impl = BufferImpl<T, dimensions>;
01031 #endif
01032
01033   /** Create a new buffer with storage managed by SYCL
01034
01035       \param r defines the size
```

```
01036   */
01037   buffer(const range<dimensions> &r) : Impl(r.getImpl()) {}
01038
01039
01040   /** Create a new buffer with associated host memory
01041
01042       \param host_data points to the storage and values used by the buffer
01043
01044       \param r defines the size
01045   */
01046   buffer(T * host_data, range<dimensions> r) : Impl(host_data, r.getImpl()) {}
01047
01048
01049   /** Create a new read only buffer with associated host memory
01050
01051       \param host_data points to the storage and values used by the buffer
01052
01053       \param r defines the size
01054   */
01055   buffer(const T * host_data, range<dimensions> r) :
01056     Impl(host_data, r.getImpl()) {}
01057
01058
01059   /** Create a new buffer from a storage abstraction provided by the user
01060
01061       \param store is the storage back-end to use for the buffer
01062
01063       \param r defines the size
01064
01065       The storage object has to exist during all the life of the buffer
01066       object.
01067
01068       \todo To be implemented
01069   */
01070   buffer(storage<T> &store, range<dimensions> r) { assert(0); }
01071
01072
01073   /** Create a new allocated 1D buffer initialized from the given elements
01074
01075       \param start_iterator points to the first element to copy
01076
01077       \param end_iterator points to just after the last element to copy
01078
01079       \todo Add const to the SYCL specification
01080   */
01081   buffer(const T * start_iterator, const T * end_iterator) :
01082     Impl(start_iterator, end_iterator) {}
01083
01084
01085   /** Create a new buffer copy that shares the data with the origin buffer
01086
01087       \param b is the buffer to copy from
01088
01089       The system use reference counting to deal with data lifetime
01090   */
01091   buffer(buffer<T, dimensions> &b) : Impl(b) {}
01092
01093
01094   /** Create a new sub-buffer without allocation to have separate accessors
01095       later
01096
01097       \param b is the buffer with the real data
01098
01099       \param base_index specifies the origin of the sub-buffer inside the
01100       buffer b
01101
01102       \param sub_range specifies the size of the sub-buffer
01103
01104       \todo To be implemented
01105
01106       \todo Update the specification to replace index by id
01107   */
01108   buffer(buffer<T, dimensions> b,
01109          id<dimensions> base_index,
01110          range<dimensions> sub_range) { assert(0); }
01111
01112
01113 #ifdef TRISYCL_OPENCL
01114   /** Create a buffer from an existing OpenCL memory object associated to
01115       a context after waiting for an event signaling the availability of
01116       the OpenCL data
01117
01118       \param mem_object is the OpenCL memory object to use
01119
01120       \param from_queue is the queue associated to the memory object
01121
01122       \param available_event specifies the event to wait for if non null
```

```
01123
01124       \todo To be implemented
01125
01126       \todo Improve the specification to allow CLHPP objects too
01127   */
01128   buffer(cl_mem mem_object,
01129          queue from_queue,
01130          event available_event) { assert(0); }
01131 #endif
01132
01133
01134   // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
01135
01136   /** Get an accessor to the buffer with the required mode
01137
01138       \param mode is the requested access mode
01139
01140       \param target is the type of object to be accessed
01141   */
01142   template <access::mode mode,
01143             access::target target=access::global_buffer>
01144   accessor<T, dimensions, mode, target>
      get_access() {
01145     return { *this };
01146   }
01147
01148 };
01149
01150 /// @} to end the data Doxygen group
01151
01152
01153 /** \addtogroup parallelism
01154     @{
01155 */
01156
01157 /** kernel_lambda specify a kernel to be launch with a single_task or
01158     parallel_for
01159
01160     \todo This seems to have also the kernel_functor name in the
01161     specification
01162 */
01163 template <typename KernelName, typename Functor>
01164 Functor kernel_lambda(Functor F) {
01165   return F;
01166 }
01167
01168
01169 /** SYCL single_task launches a computation without parallelism at launch
01170     time.
01171
01172     Right now the implementation does nothing else that forwarding the
01173     execution of the given functor
01174
01175     \todo remove from the SYCL specification and use a range-less
01176     parallel_for version with default construction of a 1-element range?
01177 */
01178 void single_task(std::function<void(void)> F) { F(); }
01179
01180
01181 /** SYCL parallel_for launches a data parallel computation with parallelism
01182     specified at launch time by a range<>.
01183
01184     This implementation use OpenMP 3 if compiled with the right flag.
01185
01186     \todo It is not clear if the ParallelForFunctor is called with an id<>
01187     or with an item. Let's use id<> when called with a range<> and item<>
01188     when called with a nd_range<>
01189 */
01190 template <int Dimensions = 1, typename ParallelForFunctor>
01191 void parallel_for(range<Dimensions> r,
01192                   ParallelForFunctor f) {
01193 #ifdef _OPENMP
01194   // Use OpenMP for the top loop level
01195   ParallelOpenMPForIterate<Dimensions,
01196                            range<Dimensions>,
01197                            ParallelForFunctor,
01198                            id<Dimensions>> { r, f };
01199 #else
01200   // In a sequential execution there is only one index processed at a time
01201   id<Dimensions> index;
01202   ParallelForIterate<Dimensions,
01203                      range<Dimensions>,
01204                      ParallelForFunctor,
01205                      id<Dimensions>> { r, f, index };
01206 #endif
01207 }
01208
```

```
01209
01210 /** A variation of SYCL parallel_for to take into account a nd_range<>
01211
01212     \todo Add an OpenMP implementation
01213
01214     \todo Deal with incomplete work-groups
01215
01216     \todo Implement with parallel_for_workgroup()/parallel_for_workitem()
01217 */
01218 template <int Dimensions = 1, typename ParallelForFunctor>
01219 void parallel_for(nd_range<Dimensions> r,
01220                   ParallelForFunctor f) {
01221   // In a sequential execution there is only one index processed at a time
01222   item<Dimensions> Index { r };
01223   // To iterate on the work-group
01224   id<Dimensions> Group;
01225   range<Dimensions> GroupRange = r.get_group_range();
01226   // To iterate on the local work-item
01227   id<Dimensions> Local;
01228   range<Dimensions> LocalRange = r.get_local_range();
01229
01230   // Reconstruct the item from its group and local id
01231   auto reconstructItem = [&] (id<Dimensions> L) {
01232     //Local.display();
01233     // Reconstruct the global item
01234     Index.set_local(Local);
01235     Index.set_global(Local + LocalRange*Group);
01236     // Call the user kernel at last
01237     f(Index);
01238   };
01239
01240   /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
01241      into another functor that iterate inside the work-group and then
01242      calls f */
01243   auto iterateInWorkGroup = [&] (id<Dimensions> G) {
01244     //Group.display();
01245     // Then iterate on the local work-groups
01246     ParallelForIterate<Dimensions,
01247                        range<Dimensions>,
01248                        decltype(reconstructItem),
01249                        id<Dimensions>> { LocalRange, reconstructItem, Local };
01250   };
01251
01252   // First iterate on all the work-groups
01253   ParallelForIterate<Dimensions,
01254                      range<Dimensions>,
01255                      decltype(iterateInWorkGroup),
01256                      id<Dimensions>> { GroupRange, iterateInWorkGroup, Group };
01257 }
01258
01259
01260 /// SYCL parallel_for version that allows a Program object to be specified
01261 template <typename Range, typename Program, typename ParallelForFunctor>
01262 void parallel_for(Range r, Program p, ParallelForFunctor f) {
01263   /// \todo deal with Program
01264   parallel_for(r, f);
01265 }
01266
01267
01268 /// Loop on the work-groups
01269 template <int Dimensions = 1, typename ParallelForFunctor>
01270 void parallel_for_workgroup(nd_range<Dimensions> r,
01271                             ParallelForFunctor f) {
01272   // In a sequential execution there is only one index processed at a time
01273   group<Dimensions> Group(r.getImpl());
01274
01275   // Reconstruct the item from its group and local id
01276   auto callWithGroup = [&] (group<Dimensions> G) {
01277     //G.Id.display();
01278     // Call the user kernel with the group as parameter
01279     f(G);
01280   };
01281   // First iterate on all the work-groups
01282   ParallelForIterate<Dimensions,
01283                      range<Dimensions>,
01284                      ParallelForFunctor,
01285                      group<Dimensions>> {
01286     r.get_group_range(),
01287     f,
01288     Group };
01289 }
01290
01291
01292 /// Loop on the work-items inside a work-group
01293 template <int Dimensions = 1, typename ParallelForFunctor>
01294 void parallel_for_workitem(group<Dimensions> g, ParallelForFunctor f)
       {
```

```
01295   // In a sequential execution there is only one index processed at a time
01296   item<Dimensions> Index { g.get_nr_range() };
01297   // To iterate on the local work-item
01298   id<Dimensions> Local;
01299
01300   // Reconstruct the item from its group and local id
01301   auto reconstructItem = [&] (id<Dimensions> L) {
01302     //Local.display();
01303     //L.display();
01304     // Reconstruct the global item
01305     Index.set_local(Local);
01306     // \todo Some strength reduction here
01307     Index.set_global(Local + g.get_local_range()*g.get_group_id());
01308     // Call the user kernel at last
01309     f(Index);
01310   };
01311
01312   // Then iterate on all the work-items of the work-group
01313   ParallelForIterate<Dimensions,
01314                      range<Dimensions>,
01315                      decltype(reconstructItem),
01316                      id<Dimensions>> {
01317     g.get_local_range(),
01318     reconstructItem,
01319     Local };
01320 }
01321
01322 /// @} End the parallelism Doxygen group
01323
01324
01325
01326 /** The kernel synchronization barrier
01327
01328     \todo To be implemented
01329 */
01330 void
01331 barrier(int barrier_type) {}
01332
01333 int const CL_LOCAL_MEM_FENCE = 123;
01334
01335 }
01336 }
01337
01338 /*
01339     # Some Emacs stuff:
01340     ### Local Variables:
01341     ### ispell-local-dictionary: "american"
01342     ### eval: (flyspell-prog-mode)
01343     ### End:
01344 */
```

Index