triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.9.1

Wed Sep 9 2015 15:36:17

Contents

1 Main Page 1

2 Todo List 3

3 Module Index 9

3.1 Modules . . . 9

4 Namespace Index 11

4.1 Namespace List . . . 11

5 Hierarchical Index 13

5.1 Class Hierarchy . . . 13

6 Class Index 17

6.1 Class List . . . 17

7 File Index 19

7.1 File List . . . 19

8 Module Documentation 21

8.1 Data access and storage in SYCL . . . 21

8.1.1 Detailed Description . . . 22

8.1.2 Class Documentation . . . 22

8.1.2.1 struct cl::sycl::detail::accessor . . . 22

8.1.2.2 struct cl::sycl::accessor . . . 26

8.1.2.3 struct cl::sycl::detail::buffer . . . 27

8.1.2.4 struct cl::sycl::buffer . . . 31

8.1.2.5 struct cl::sycl::image . . . 32

8.1.3 Typedef Documentation . . . 32

8.1.3.1 buffer_allocator . . . 32

8.1.4 Enumeration Type Documentation . . . 33

8.1.4.1 fence_space . . . 33

8.1.4.2 mode . . . 33

8.1.4.3 target . . . 33

8.2 Dealing with OpenCL address spaces . . . 35
8.2.1 Detailed Description . . . 36
8.2.2 Class Documentation . . . 36
8.2.2.1 struct cl::sycl::detail::opencl_type . . . 36
8.2.2.2 struct cl::sycl::detail::opencl_type< T, constant_address_space > . . . 36
8.2.2.3 struct cl::sycl::detail::opencl_type< T, generic_address_space > . . . 37
8.2.2.4 struct cl::sycl::detail::opencl_type< T, global_address_space > . . . 37
8.2.2.5 struct cl::sycl::detail::opencl_type< T, local_address_space > . . . 37
8.2.2.6 struct cl::sycl::detail::opencl_type< T, private_address_space > . . . 38
8.2.2.7 struct cl::sycl::detail::address_space_array . . . 38
8.2.2.8 struct cl::sycl::detail::address_space_fundamental . . . 39
8.2.2.9 struct cl::sycl::detail::address_space_object . . . 41
8.2.2.10 struct cl::sycl::detail::address_space_ptr . . . 42
8.2.2.11 struct cl::sycl::detail::address_space_base . . . 43
8.2.2.12 struct cl::sycl::detail::address_space_variable . . . 44
8.2.3 Typedef Documentation . . . 46
8.2.3.1 addr_space . . . 46
8.2.3.2 constant . . . 47
8.2.3.3 generic . . . 47
8.2.3.4 global . . . 47
8.2.3.5 local . . . 47
8.2.3.6 multi_ptr . . . 47
8.2.3.7 priv . . . 49
8.2.4 Enumeration Type Documentation . . . 49
8.2.4.1 address_space . . . 49
8.2.5 Function Documentation . . . 49
8.2.5.1 make_multi . . . 49
8.3 Platforms, contexts, devices and queues . . . 51
8.3.1 Detailed Description . . . 53
8.3.2 Class Documentation . . . 53
8.3.2.1 class cl::sycl::context . . . 53
8.3.2.2 class cl::sycl::device . . . 56
8.3.2.3 class cl::sycl::device_selector . . . 61
8.3.2.4 class cl::sycl::default_selector . . . 61
8.3.2.5 class cl::sycl::gpu_selector . . . 62
8.3.2.6 class cl::sycl::cpu_selector . . . 63
8.3.2.7 class cl::sycl::host_selector . . . 63
8.3.2.8 class cl::sycl::kernel . . . 64
8.3.2.9 class cl::sycl::handler . . . 64
8.3.2.10 class cl::sycl::platform . . . 68

8.3.2.11 class cl::sycl::queue . . . 71
8.3.3 Enumeration Type Documentation . . . 77
8.3.3.1 context . . . 77
8.3.3.2 device . . . 77
8.3.3.3 device_affinity_domain . . . 80
8.3.3.4 device_execution_capabilities . . . 81
8.3.3.5 device_partition_property . . . 81
8.3.3.6 device_partition_type . . . 81
8.3.3.7 device_type . . . 82
8.3.3.8 fp_config . . . 82
8.3.3.9 global_mem_cache_type . . . 83
8.3.3.10 local_mem_type . . . 83
8.3.3.11 platform . . . 83
8.3.3.12 queue . . . 84
8.4 Helpers to do array and tuple conversion . . . 85
8.4.1 Detailed Description . . . 85
8.4.2 Class Documentation . . . 85
8.4.2.1 struct cl::sycl::detail::expand_to_vector . . . 85
8.4.2.2 struct cl::sycl::detail::expand_to_vector< V, Tuple, true > . . . 85
8.4.3 Function Documentation . . . 86
8.4.3.1 expand . . . 86
8.4.3.2 expand . . . 86
8.4.3.3 expand . . . 86
8.4.3.4 fill_tuple . . . 87
8.4.3.5 tuple_to_array . . . 87
8.4.3.6 tuple_to_array_iterate . . . 87
8.5 Debugging and tracing support . . . 88
8.5.1 Detailed Description . . . 88
8.5.2 Class Documentation . . . 88
8.5.2.1 struct cl::sycl::detail::debug . . . 88
8.5.2.2 struct cl::sycl::detail::display_vector . . . 89
8.5.3 Function Documentation . . . 90
8.5.3.1 unimplemented . . . 90
8.6 Some helpers for the implementation . . . 91
8.6.1 Detailed Description . . . 91
8.6.2 Class Documentation . . . 91
8.6.2.1 struct cl::sycl::detail::small_array . . . 91
8.6.2.2 struct cl::sycl::detail::small_array_123 . . . 94
8.6.2.3 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > . . . 94
8.6.2.4 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > . . . 95

8.6.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > . . . 96
8.6.3 Macro Definition Documentation . . . 97
8.6.3.1 TRISYCL_BOOST_OPERATOR_VECTOR_OP . . . 97
8.6.4 Function Documentation . . . 97
8.6.4.1 linear_id . . . 97
8.7 Error handling . . . 98
8.7.1 Detailed Description . . . 98
8.7.2 Class Documentation . . . 98
8.7.2.1 struct cl::sycl::error_handler . . . 98
8.7.2.2 struct cl::sycl::trisycl::default_error_handler . . . 99
8.7.2.3 struct cl::sycl::exception . . . 99
8.7.3 Typedef Documentation . . . 101
8.7.3.1 async_handler . . . 101
8.8 Expressing parallelism through kernels . . . 102
8.8.1 Detailed Description . . . 103
8.8.2 Class Documentation . . . 103
8.8.2.1 struct cl::sycl::group . . . 103
8.8.2.2 class cl::sycl::id . . . 108
8.8.2.3 class cl::sycl::item . . . 109
8.8.2.4 struct cl::sycl::nd_item . . . 112
8.8.2.5 struct cl::sycl::nd_range . . . 119
8.8.2.6 struct cl::sycl::detail::parallel_for_iterate . . . 121
8.8.2.7 struct cl::sycl::detail::parallel_OpenMP_for_iterate . . . 122
8.8.2.8 struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id > . . . 123
8.8.2.9 class cl::sycl::range . . . 123
8.8.3 Function Documentation . . . 124
8.8.3.1 make_id . . . 124
8.8.3.2 make_id . . . 124
8.8.3.3 make_id . . . 124
8.8.3.4 make_id . . . 124
8.8.3.5 make_range . . . 124
8.8.3.6 make_range . . . 125
8.8.3.7 make_range . . . 125
8.8.3.8 make_range . . . 125
8.8.3.9 parallel_for . . . 125
8.8.3.10 parallel_for . . . 126
8.8.3.11 parallel_for_global_offset . . . 126
8.8.3.12 parallel_for_work_item . . . 127
8.8.3.13 parallel_for_workgroup . . . 127
8.8.3.14 parallel_for_workitem . . . 128

8.9 Vector types in SYCL . . . 130
8.9.1 Detailed Description . . . 130
8.9.2 Class Documentation . . . 130
8.9.2.1 class cl::sycl::vec . . . 130
8.9.3 Macro Definition Documentation . . . 132
8.9.3.1 TRISYCL_DEFINE_VEC_TYPE . . . 132
8.9.3.2 TRISYCL_DEFINE_VEC_TYPE_SIZE . . . 133

9 Namespace Documentation **135**
9.1 cl Namespace Reference . . . 135
9.1.1 Detailed Description . . . 135
9.2 cl::sycl Namespace Reference . . . 135
9.2.1 Typedef Documentation . . . 138
9.2.1.1 function_class . . . 138
9.2.1.2 mutex_class . . . 138
9.2.1.3 shared_ptr_class . . . 138
9.2.1.4 string_class . . . 138
9.2.1.5 unique_ptr_class . . . 138
9.2.1.6 vector_class . . . 138
9.2.1.7 weak_ptr_class . . . 138
9.3 cl::sycl::access Namespace Reference . . . 138
9.3.1 Detailed Description . . . 139
9.4 cl::sycl::detail Namespace Reference . . . 139
9.5 cl::sycl::info Namespace Reference . . . 141
9.5.1 Typedef Documentation . . . 143
9.5.1.1 device_exec_capabilities . . . 143
9.5.1.2 device_fp_config . . . 143
9.5.1.3 device_queue_properties . . . 143
9.5.1.4 gl_context_interop . . . 143
9.5.1.5 queue_profiling . . . 143
9.6 cl::sycl::trisycl Namespace Reference . . . 143
9.6.1 Detailed Description . . . 143

10 Class Documentation **145**
10.1 cl::sycl::detail::AccessorImpl< T, dimensions, mode, target > Struct Template Reference . . . 145
10.1.1 Detailed Description . . . 145
10.2 cl::sycl::detail::buffer_base Struct Reference . . . 145
10.2.1 Detailed Description . . . 146
10.2.2 Constructor & Destructor Documentation . . . 146
10.2.2.1 buffer_base . . . 146
10.2.3 Member Function Documentation . . . 146

10.2.3.1 get_buffer_customer . . . 146
10.2.3.2 get_last_buffer_customer . . . 147
10.2.3.3 lock . . . 147
10.2.3.4 set_last_buffer_customer . . . 147
10.2.3.5 wait . . . 147
10.2.4 Member Data Documentation . . . 148
10.2.4.1 last_buffer_customer . . . 148
10.2.4.2 protect_buffer . . . 148
10.2.4.3 read_only . . . 148
10.3 cl::sycl::detail::buffer_customer Class Reference . . . 148
10.3.1 Detailed Description . . . 149
10.3.2 Constructor & Destructor Documentation . . . 149
10.3.2.1 buffer_customer . . . 149
10.3.3 Member Function Documentation . . . 149
10.3.3.1 add . . . 149
10.3.3.2 notify_ready . . . 150
10.3.3.3 release . . . 150
10.3.3.4 set_next_generation . . . 150
10.3.3.5 wait . . . 150
10.3.3.6 wait_released . . . 151
10.3.4 Member Data Documentation . . . 151
10.3.4.1 buf . . . 151
10.3.4.2 next_generation . . . 151
10.3.4.3 ready_cv . . . 151
10.3.4.4 ready_mutex . . . 151
10.3.4.5 ready_to_use . . . 151
10.3.4.6 released_cv . . . 152
10.3.4.7 released_mutex . . . 152
10.3.4.8 user_number . . . 152
10.3.4.9 write_access . . . 152
10.4 handler_event Class Reference . . . 152
10.4.1 Detailed Description . . . 152
10.5 cl::sycl::info::param_traits< T, Param > Class Template Reference . . . 152
10.5.1 Detailed Description . . . 153
10.6 cl::sycl::detail::task Struct Reference . . . 153
10.6.1 Detailed Description . . . 153
10.6.2 Member Function Documentation . . . 153
10.6.2.1 acquire_buffers . . . 153
10.6.2.2 add . . . 154
10.6.2.3 release_buffers . . . 154

10.6.2.4 schedule . . . 154
10.6.3 Member Data Documentation . . . 155
10.6.3.1 buffers . . . 155

11 File Documentation **157**

11.1 include/CL/sycl.hpp File Reference . . . 157
11.2 sycl.hpp . . . 157
11.3 include/CL/sycl/access.hpp File Reference . . . 158
11.4 access.hpp . . . 159
11.5 include/CL/sycl/accessor/detail/accessor.hpp File Reference . . . 160
11.6 accessor.hpp . . . 160
11.7 include/CL/sycl/accessor.hpp File Reference . . . 162
11.8 accessor.hpp . . . 163
11.9 include/CL/sycl/address_space/detail/address_space.hpp File Reference . . . 164
11.9.1 Detailed Description . . . 165
11.10 address_space.hpp . . . 166
11.11 include/CL/sycl/address_space.hpp File Reference . . . 170
11.11.1 Detailed Description . . . 171
11.12 address_space.hpp . . . 171
11.13 include/CL/sycl/buffer/detail/buffer.hpp File Reference . . . 173
11.14 buffer.hpp . . . 173
11.15 include/CL/sycl/buffer.hpp File Reference . . . 175
11.16 buffer.hpp . . . 175
11.17 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference . . . 179
11.18 buffer_base.hpp . . . 180
11.19 include/CL/sycl/buffer/detail/buffer_customer.hpp File Reference . . . 181
11.20 buffer_customer.hpp . . . 181
11.21 include/CL/sycl/buffer_allocator.hpp File Reference . . . 183
11.22 buffer_allocator.hpp . . . 184
11.23 include/CL/sycl/command_group/detail/task.hpp File Reference . . . 184
11.24 task.hpp . . . 185
11.25 include/CL/sycl/context.hpp File Reference . . . 186
11.26 context.hpp . . . 187
11.27 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference . . . 189
11.27.1 Detailed Description . . . 190
11.28 array_tuple_helpers.hpp . . . 190
11.29 include/CL/sycl/detail/debug.hpp File Reference . . . 192
11.29.1 Macro Definition Documentation . . . 193
11.29.1.1 TRISYCL_DUMP . . . 193
11.29.1.2 TRISYCL_DUMP_T . . . 193

11.30debug.hpp . . . 193
11.31include/CL/sycl/detail/default_classes.hpp File Reference . . . 195
11.32default_classes.hpp . . . 195
11.33include/CL/sycl/detail/global_config.hpp File Reference . . . 197
11.33.1 Macro Definition Documentation . . . 197
11.33.1.1 __CL_ENABLE_EXCEPTIONS . . . 197
11.33.1.2 __SYCL_SINGLE_SOURCE__ . . . 197
11.33.1.3 CL_SYCL_LANGUAGE_VERSION . . . 198
11.33.1.4 CL_TRISYCL_LANGUAGE_VERSION . . . 198
11.33.1.5 TRISYCL_ASYNC . . . 198
11.34global_config.hpp . . . 198
11.35include/CL/sycl/detail/linear_id.hpp File Reference . . . 199
11.36linear_id.hpp . . . 199
11.37include/CL/sycl/detail/small_array.hpp File Reference . . . 200
11.38small_array.hpp . . . 200
11.39include/CL/sycl/detail/unimplemented.hpp File Reference . . . 204
11.40unimplemented.hpp . . . 204
11.41include/CL/sycl/device.hpp File Reference . . . 205
11.42device.hpp . . . 206
11.43include/CL/sycl/device_selector.hpp File Reference . . . 211
11.44device_selector.hpp . . . 211
11.45include/CL/sycl/error_handler.hpp File Reference . . . 213
11.46error_handler.hpp . . . 214
11.47include/CL/sycl/exception.hpp File Reference . . . 214
11.48exception.hpp . . . 215
11.49include/CL/sycl/group.hpp File Reference . . . 216
11.50group.hpp . . . 217
11.51include/CL/sycl/handler.hpp File Reference . . . 219
11.51.1 Macro Definition Documentation . . . 219
11.51.1.1 TRISYCL_parallel_for_functor_GLOBAL . . . 219
11.51.1.2 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET . . . 220
11.52handler.hpp . . . 220
11.53include/CL/sycl/handler_event.hpp File Reference . . . 223
11.54handler_event.hpp . . . 224
11.55include/CL/sycl/id.hpp File Reference . . . 224
11.56id.hpp . . . 225
11.57include/CL/sycl/image.hpp File Reference . . . 226
11.57.1 Detailed Description . . . 226
11.58image.hpp . . . 226
11.59include/CL/sycl/info/param_traits.hpp File Reference . . . 227

11.59.1 Macro Definition Documentation . . . 227
11.59.1.1 TRISYCL_INFO_PARAM_TRAITS . . . 227
11.59.1.2 TRISYCL_INFO_PARAM_TRAITS_ANY_T . . . 228
11.60 param_traits.hpp . . . 228
11.61 include/CL/sycl/item.hpp File Reference . . . 229
11.62 item.hpp . . . 229
11.63 include/CL/sycl/nd_item.hpp File Reference . . . 231
11.64 nd_item.hpp . . . 231
11.65 include/CL/sycl/nd_range.hpp File Reference . . . 234
11.66 nd_range.hpp . . . 234
11.67 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference . . . 235
11.67.1 Detailed Description . . . 236
11.68 parallelism.hpp . . . 237
11.69 include/CL/sycl/parallelism.hpp File Reference . . . 240
11.69.1 Detailed Description . . . 241
11.70 parallelism.hpp . . . 241
11.71 include/CL/sycl/platform.hpp File Reference . . . 241
11.72 platform.hpp . . . 242
11.73 include/CL/sycl/queue.hpp File Reference . . . 244
11.74 queue.hpp . . . 245
11.75 include/CL/sycl/range.hpp File Reference . . . 249
11.76 range.hpp . . . 250
11.77 include/CL/sycl/vec.hpp File Reference . . . 250
11.77.1 Detailed Description . . . 251
11.78 vec.hpp . . . 251

Index **255**

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/

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 Ronan at keryell dot FR

Chapter 2

Todo List

globalScope> 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, AccessMode, Target >

Implement it for images according so section 3.3.4.5

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality

in the specification: store the dimension for user request

Class cl::sycl::buffer< T, Dimensions, Allocator >

We have some read-write buffers and some read-only buffers, according to the constructor called. So we could have some static checking for correctness with the accessors used, but we do not have a way in the specification to have a read-only buffer type for this.

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

Think about the need of an allocator when constructing a buffer from other buffers

Class cl::sycl::context

The implementation is quite minimal for now.

Member cl::sycl::context::get_devices () const

To be implemented

Member cl::sycl::context::get_info () const

To be implemented

Member cl::sycl::context::get_platform ()

To be implemented

Class cl::sycl::cpu_selector

to be implemented

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

Class cl::sycl::default_selector

to be implemented

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

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access () const

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index) const

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

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index)

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

Member cl::sycl::detail::address_space_array< T, AS >::address_space_array (std::initializer_list< std::remove_extent_t< T >> list)

Extend to more than 1 dimension

Class cl::sycl::detail::address_space_base< T, AS >

Verify/improve to deal with const/volatile?

Member cl::sycl::detail::address_space_base< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_base< T, AS >::type

Add to the specification

Class cl::sycl::detail::address_space_fundamental< T, AS >

Verify/improve to deal with const/volatile?

Class cl::sycl::detail::address_space_object< T, AS >

Verify/improve to deal with const/volatile?

what about T having some final methods?

Member cl::sycl::detail::address_space_object< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_variable< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::buffer< T, Dimensions >::buffer (Iterator start_iterator, Iterator end_iterator)

Member cl::sycl::detail::buffer< T, Dimensions >::buffer (const buffer< T, Dimensions > &b)

Refactor the implementation to deal with buffer sharing with reference counting

Member cl::sycl::detail::buffer< T, Dimensions >::get_access ()

To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base_index, range<Dimensions> sub_range)

Allow CLHPP objects too?

Remove if not used

Member cl::sycl::detail::buffer_base::read_only

Replace this by a static read-only type for the buffer

Member cl::sycl::detail::buffer_customer::buf

Do we need to keep it?

Member cl::sycl::detail::buffer_customer::set_next_generation (std::shared_ptr< buffer_customer > bc)

Refactor this with an lock-free list?

Member cl::sycl::detail::buffer_customer::write_access

Needed?

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

Add an OpenMP implementation

Deal with incomplete work-groups

Implement with parallel_for_workgroup()/parallel_for_workitem()

Member cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimensionality

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

Class cl::sycl::device

The implementation is quite minimal for now. :-)

Member cl::sycl::device::create_sub_devices (info::device_partition_type partitionType, info::device_partition_property partitionProperty, info::device_affinity_domain affinityDomain) const

To be implemented

Member cl::sycl::device::device (cl_device_id deviceId)

To be implemented

Member cl::sycl::device::device (const device_selector &deviceSelector)

To be implemented

Member cl::sycl::device::get () const

To be implemented

Member cl::sycl::device::get_devices (info::device_type deviceType=info::device_type::all)

To be implemented

Member cl::sycl::device::get_info () const

To be implemented

Member cl::sycl::device::get_platform () const

To be implemented

Member cl::sycl::device::has_extension (const string_class &extension) const

To be implemented

Member cl::sycl::device::is_accelerator () const

To be implemented

Member cl::sycl::device::is_cpu () const

To be implemented

Member cl::sycl::device::is_gpu () const

To be implemented

Member cl::sycl::device::is_host () const

To be implemented

Member cl::sycl::device_selector::select_device () const

To be implemented

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

How to get the real buffer type? Update: has been removed in new specification

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_group_range () const

Fix this comment and the specification

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

Add to the specification

Member cl::sycl::group< dims >::get_local_range (int dimension) const

Add to the specification

Member cl::sycl::group< dims >::get_nd_range () const

Also provide this access to the current nd_range

Member cl::sycl::group< dims >::get_offset (int dimension) const

Add to the specification

Member cl::sycl::group< dims >::get_offset () const

Add to the specification

Member cl::sycl::group< dims >::group ()=default

Make most of them protected, reserved to implementation

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

This should be private since it is only used by the triSYCL implementation

Member cl::sycl::group< dims >::group (const id< dims > &i, const nd_range< dims > &ndr)

This should be private somehow, but it is used by the validation infrastructure

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

In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Member cl::sycl::handler::parallel_for (range< Dimensions > numWorkItems, kernel sycl_kernel)

To be implemented

Member cl::sycl::handler::parallel_for (nd_range< Dimensions >, kernel syclKernel)

To be implemented

Member cl::sycl::handler::set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > acc↩ _obj)

To be implemented

Member cl::sycl::handler::set_arg (int arg_index, T scalar_value)

To be implemented

Member cl::sycl::handler::single_task (kernel syclKernel)

To be implemented

Class cl::sycl::host_selector

to be implemented

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

Class cl::sycl::image< dimensions >

implement image

Member cl::sycl::info::context

Should be unsigned int to be consistent with others?

Member cl::sycl::info::device

Should be unsigned int?

Member cl::sycl::info::queue

unsigned int?

To be implemented

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

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

Member cl::sycl::item< dims >::item ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::item< dims >::set (id< dims > Index)

Move to private and add friends

Class cl::sycl::kernel

To be implemented

Member cl::sycl::make_multi (multi_ptr< T, AS > pointer)

Implement the case with a plain pointer

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

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

Member cl::sycl::nd_item< dims >::nd_item (id< dims > global_index, nd_range< dims > ndr)

This is for validation purpose. Hide this to the programmer somehow

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

This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

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 () const

get_offset() is lacking in the specification

Member cl::sycl::parallel_for_work_item (group< Dimensions > g, ParallelForFunctor f)

To be implemented

Class cl::sycl::platform

triSYCL Implementation

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

To be implemented

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

To be implemented

Member cl::sycl::platform::get_platforms ()

To be implemented

Member cl::sycl::platform::has_extension (const string_class &extension) const

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 platformID)

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 specification this default parameter value?

add to the specification some way to specify an offset?

Namespace cl::sycl::trisycl

Refactor when updating to latest specification

Class cl::sycl::vec< DataType, NumElements >

add [] operator

add iterators on elements, with begin() and end()

having vec<> sub-classing array<> instead would solve the previous issues

move the implementation elsewhere

simplify the helpers by removing some template types since there are now inside the vec<> class.

rename in the specification element_type to value_type

Class handler_event

To be implemented

To be implemented

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Data access and storage in SYCL 21
Dealing with OpenCL address spaces 35
Platforms, contexts, devices and queues 51
Helpers to do array and tuple conversion 85
Debugging and tracing support 88
Some helpers for the implementation 91
Error handling 98
Expressing parallelism through kernels 102
Vector types in SYCL 130

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cl
The vector type to be used as SYCL vector . . . 135
cl::sycl . . . 135
cl::sycl::access
Describe the type of access by kernels . . . 138
cl::sycl::detail . . . 139
cl::sycl::info . . . 141
cl::sycl::trisycl . . . 143

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

cl::sycl::detail::AccessorImpl< T, dimensions, mode, target > . . . 145
cl::sycl::detail::address_space_base< T, AS > . . . 35
cl::sycl::detail::address_space_object< T, AS > . . . 35
cl::sycl::detail::address_space_variable< T, AS > . . . 35
cl::sycl::detail::address_space_array< T, AS > . . . 35
cl::sycl::detail::address_space_fundamental< T, AS > . . . 35
cl::sycl::detail::address_space_ptr< T, AS > . . . 35
array
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 91
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 91
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 91
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 91
cl::sycl::detail::small_array_123< BasicType, FinalType, Dims > . . . 91
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 91
cl::sycl::vec< DataType, NumElements > . . . 130
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims > . . . 91
cl::sycl::detail::small_array_123< std::size_t, id< dims >, dims > . . . 91
cl::sycl::id< dims > . . . 102
cl::sycl::id< dimensionality > . . . 102
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims > . . . 91
cl::sycl::detail::small_array_123< std::size_t, range< dims >, dims > . . . 91
cl::sycl::range< dims > . . . 102
cl::sycl::range< dimensionality > . . . 102
bitwise
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 91
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 91
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims > . . . 91
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims > . . . 91

cl::sycl::buffer< T, Dimensions, Allocator > . . . 21
cl::sycl::detail::buffer_base . . . 145
cl::sycl::detail::buffer< DataType, Dimensions > . . . 21
cl::sycl::detail::buffer< T, Dimensions > . . . 21
cl::sycl::context . . . 51
cl::sycl::detail::debug< T > . . . 88
cl::sycl::detail::debug< accessor< DataType, Dimensions, Mode, Target > > . . . 88
cl::sycl::detail::accessor< DataType, Dimensions, AccessMode, Target > . . . 21
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target > . . . 21
cl::sycl::detail::debug< accessor< T, Dimensions, Mode, Target > > . . . 88
cl::sycl::detail::accessor< T, Dimensions, Mode, Target > . . . 21
cl::sycl::detail::debug< buffer< DataType, Dimensions > > . . . 88
cl::sycl::detail::buffer< DataType, Dimensions > . . . 21
cl::sycl::detail::debug< buffer< T, Dimensions > > . . . 88
cl::sycl::detail::buffer< T, Dimensions > . . . 21
cl::sycl::detail::debug< buffer_customer > . . . 88
cl::sycl::detail::buffer_customer . . . 148
cl::sycl::detail::debug< task > . . . 88
cl::sycl::detail::task . . . 153
cl::sycl::device . . . 51
cl::sycl::device_selector . . . 51
cl::sycl::cpu_selector . . . 51
cl::sycl::default_selector . . . 51
cl::sycl::gpu_selector . . . 51
cl::sycl::host_selector . . . 51
cl::sycl::detail::display_vector< T > . . . 88
cl::sycl::detail::display_vector< FinalType > . . . 88
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 91
cl::sycl::detail::display_vector< id< dims > > . . . 88
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims > . . . 91
cl::sycl::detail::display_vector< range< dims > > . . . 88
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims > . . . 91
cl::sycl::detail::display_vector< vec< DataType, NumElements > > . . . 88
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 91
enable_shared_from_this
cl::sycl::detail::task . . . 153
cl::sycl::error_handler . . . 98
cl::sycl::trisycl::default_error_handler . . . 98
euclidean_ring_operators
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 91
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 91
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 91
cl::sycl::detail::small_array< std::size_t, id< dims >, Dims > . . . 91
cl::sycl::detail::small_array< std::size_t, range< dims >, Dims > . . . 91
cl::sycl::exception . . . 98
cl::sycl::detail::expand_to_vector< V, Tuple, expansion > . . . 85
cl::sycl::detail::expand_to_vector< V, Tuple, true > . . . 85

cl::sycl::group< dims > . . . 102
cl::sycl::handler . . . 51
handler_event . . . 152
cl::sycl::image< dimensions > . . . 21
cl::sycl::item< dims > . . . 102
cl::sycl::kernel . . . 51
cl::sycl::nd_item< dims > . . . 102
cl::sycl::nd_range< dims > . . . 102
cl::sycl::detail::opencl_type< T, AS > . . . 35
cl::sycl::detail::opencl_type< T, constant_address_space > . . . 35
cl::sycl::detail::opencl_type< T, generic_address_space > . . . 35
cl::sycl::detail::opencl_type< T, global_address_space > . . . 35
cl::sycl::detail::opencl_type< T, local_address_space > . . . 35
cl::sycl::detail::opencl_type< T, private_address_space > . . . 35
cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id > . . . 102
cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id > . . . 102
cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id > . . . 102
cl::sycl::info::param_traits< T, Param > . . . 152
cl::sycl::platform . . . 51
cl::sycl::queue . . . 51
shiftable
  cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 91
  cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 91
  cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 91
  cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 91
  cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 91
  cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 91
  cl::sycl::detail::small_array< std::size_t, id< dims >, Dims > . . . 91
  cl::sycl::detail::small_array< std::size_t, range< dims >, Dims > . . . 91
type
  cl::sycl::detail::address_space_object< T, AS > . . . 35
multi_array< DataType, Dimensions > . . . **??**
multi_array_ref< DataType, Dimensions > . . . **??**

Chapter 6

Class Index

6.1 Class List

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

cl::sycl::detail::AccessorImpl< T, dimensions, mode, target > 145

cl::sycl::detail::buffer_base
Factorize some template independent buffer aspects in a base class 145

cl::sycl::detail::buffer_customer
Keep track of the tasks waiting for the availability of a buffer generation, either to read it or to write it 148

handler_event
Handler event 152

cl::sycl::info::param_traits< T, Param >
Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value) 152

cl::sycl::detail::task
The abstraction to represent SYCL tasks executing inside command_group 153

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

include/CL/sycl.hpp . . . 157
include/CL/sycl/access.hpp . . . 158
include/CL/sycl/accessor.hpp . . . 162
include/CL/sycl/address_space.hpp
    Implement OpenCL address spaces in SYCL with C++-style . . . 170
include/CL/sycl/buffer.hpp . . . 175
include/CL/sycl/buffer_allocator.hpp . . . 183
include/CL/sycl/context.hpp . . . 186
include/CL/sycl/device.hpp . . . 205
include/CL/sycl/device_selector.hpp . . . 211
include/CL/sycl/error_handler.hpp . . . 213
include/CL/sycl/exception.hpp . . . 214
include/CL/sycl/group.hpp . . . 216
include/CL/sycl/handler.hpp . . . 219
include/CL/sycl/handler_event.hpp . . . 223
include/CL/sycl/id.hpp . . . 224
include/CL/sycl/image.hpp
    OpenCL SYCL image class . . . 226
include/CL/sycl/item.hpp . . . 229
include/CL/sycl/nd_item.hpp . . . 231
include/CL/sycl/nd_range.hpp . . . 234
include/CL/sycl/parallelism.hpp
    Implement parallel constructions to launch kernels . . . 240
include/CL/sycl/platform.hpp . . . 241
include/CL/sycl/queue.hpp . . . 244
include/CL/sycl/range.hpp . . . 249
include/CL/sycl/vec.hpp
    Implement the small OpenCL vector class . . . 250
include/CL/sycl/accessor/detail/accessor.hpp . . . 160
include/CL/sycl/address_space/detail/address_space.hpp
    Implement OpenCL address spaces in SYCL with C++-style . . . 164
include/CL/sycl/buffer/detail/buffer.hpp . . . 173
include/CL/sycl/buffer/detail/buffer_base.hpp . . . 179
include/CL/sycl/buffer/detail/buffer_customer.hpp . . . 181
include/CL/sycl/command_group/detail/task.hpp . . . 184
include/CL/sycl/detail/array_tuple_helpers.hpp
    Some helpers to do array-tuple conversions . . . 189
include/CL/sycl/detail/debug.hpp . . . 192

include/CL/sycl/detail/default_classes.hpp 195
include/CL/sycl/detail/global_config.hpp 197
include/CL/sycl/detail/linear_id.hpp 199
include/CL/sycl/detail/small_array.hpp 200
include/CL/sycl/detail/unimplemented.hpp 204
include/CL/sycl/info/param_traits.hpp 227
include/CL/sycl/parallelism/detail/parallelism.hpp
Implement the detail of the parallel constructions to launch kernels 235

Chapter 8

Module Documentation

8.1 Data access and storage in SYCL

Namespaces

- cl::sycl::access

  Describe the type of access by kernels.

Classes

- struct cl::sycl::detail::accessor< 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::accessor< DataType, Dimensions, AccessMode, Target >

  The accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- struct cl::sycl::detail::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::buffer< T, Dimensions, Allocator >

  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 >

Typedefs

- template<typename T >
  using cl::sycl::buffer_allocator = std::allocator< T >

  The default buffer allocator used by the runtime, when no allocator is defined by the user.

Enumerations

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

  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 }

  The target enumeration describes the type of object to be accessed via the accessor.
- enum cl::sycl::access::fence_space : char { cl::sycl::access::fence_space::local_space, cl::sycl::access::fence_space::global_space, cl::sycl::access::fence_space::global_and_local }

  Precise the address space a barrier needs to act on.

8.1.1 Detailed Description

8.1.2 Class Documentation

8.1.2.1 struct cl::sycl::detail::accessor

template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target>struct cl::sycl::detail::accessor< 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 48 of file accessor.hpp.

Inheritance diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:

Collaboration diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:

Public Types

- using array_view_type = boost::multi_array_ref< T, Dimensions >
- using writable_array_view_type = typename std::remove_const< array_view_type >::type
- using element = T
- using value_type = T

Public Member Functions

- accessor (detail::buffer< T, Dimensions > &target_buffer)

  The only way to construct an accessor is from an existing buffer.
- array_view_type::reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- auto & operator[ ] (id< dimensionality > index)

  To use the accessor in with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor in with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

To use an accessor with an [nd_item<>].

- auto & operator[ ] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].

- detail::buffer< T, Dimensions > & get_buffer ()

  Get the buffer used to create the accessor.

- constexpr bool is_write_access () const

  Test if the accessor as a write access right.

Public Attributes

- detail::buffer< T, Dimensions > * buf
- array_view_type array

Static Public Attributes

- static const auto dimensionality = Dimensions

8.1.2.1.1 Member Typedef Documentation

8.1.2.1.1.1 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array_view_type = boost::multi_array_ref<T, Dimensions>

Definition at line 54 of file accessor.hpp.

8.1.2.1.1.2 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::element = T

Definition at line 65 of file accessor.hpp.

8.1.2.1.1.3 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::value_type = T

Definition at line 66 of file accessor.hpp.

8.1.2.1.1.4 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::writable_array_view_type = typename std::remove_const<array_view_type>::type

Definition at line 59 of file accessor.hpp.

8.1.2.1.2 Constructor & Destructor Documentation

8.1.2.1.2.1 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (detail::buffer< T, Dimensions > & *target_buffer*) [inline]

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

Definition at line 71 of file accessor.hpp.

```
00071                                                    :
00072     buf { &target_buffer }, array { target_buffer.access } {
00073 #if TRISYCL_ASYNC
00074     if (Target == access::target::host_buffer) {
00075       // A host accessor needs to be declared *outside* a command_group
00076       assert(current_task == nullptr);
00077       // Wait for the latest generation of the buffer before the host can use it
00078       buffer_base::wait(target_buffer);
00079     }
00080     else {
```

```
00081       // A host non-host accessor needs to be declared *inside* a command_group
00082       assert(current_task != nullptr);
00083       // Register the accessor to the task dependencies
00084       current_task->add(*this);
00085     }
00086 #endif
00087   }
```

8.1.2.1.3 Member Function Documentation

8.1.2.1.3.1 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> detail::buffer<T, Dimensions>& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_buffer ( ) [inline]

Get the buffer used to create the accessor.

Definition at line 146 of file accessor.hpp.

```
00146                                                {
00147     return *buf;
00148   }
```

8.1.2.1.3.2 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access ( ) const [inline]

Test if the accessor as a write access right.

Todo to move in the access::mode enum class and add to the specification ?

Definition at line 152 of file accessor.hpp.

```
00152                                           {
00153     /** \todo to move in the access::mode enum class and add to the
00154         specification ? */
00155     return Mode == access::mode::write
00156       || Mode == access::mode::read_write
00157       || Mode == access::mode::discard_write
00158       || Mode == access::mode::discard_read_write;
00159   }
```

8.1.2.1.3.3 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> array_view_type::reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (std::size_t *index*) [inline]

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 95 of file accessor.hpp.

```
00095                                                                {
00096     return array[index];
00097   }
```

8.1.2.1.3.4 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (id< dimensionality > *index*) [inline]

To use the accessor in with [id<>].

Definition at line 101 of file accessor.hpp.

```
00101                                            {
00102     return (const_cast<writable_array_view_type &>(array))(index);
00103   }
```

8.1.2.1.3.5 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (id< dimensionality > *index*) const [inline]

To use the accessor in with [id<>].

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

Definition at line 111 of file accessor.hpp.

```
00111                                                 {
00112     return (const_cast<writable_array_view_type &>(array))(index);
00113   }
```

8.1.2.1.3.6 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (item< dimensionality > *index*) [inline]

To use an accessor with [item<>].

Definition at line 117 of file accessor.hpp.

```
00117                                               {
00118     return (*this)[index.get()];
00119   }
```

8.1.2.1.3.7 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (item< dimensionality > *index*) const [inline]

To use an accessor with [item<>].

Definition at line 123 of file accessor.hpp.

```
00123                                                     {
00124     return (*this)[index.get()];
00125   }
```

8.1.2.1.3.8 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (nd_item< dimensionality > *index*) [inline]

To use an accessor with an [nd_item<>].

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

Definition at line 132 of file accessor.hpp.

```
00132                                                   {
00133     return (*this)[index.get_global()];
00134   }
```

8.1.2.1.3.9 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ] (nd_item< dimensionality > *index*) const [inline]

To use an accessor with an [nd_item<>].

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

Definition at line 140 of file accessor.hpp.

```
00140                                                         {
00141     return (*this)[index.get_global()];
00142   }
```

8.1.2.1.4 Member Data Documentation

8.1.2.1.4.1 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> array_view_type cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array

Definition at line 56 of file accessor.hpp.

8.1.2.1.4.2 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> detail::buffer<T, Dimensions>* cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::buf

Definition at line 52 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< DataType, Dimensions, AccessMode, Target >::get_buffer().

8.1.2.1.4.3 template<typename T, std::size_t Dimensions, access::mode Mode, access::target Target> const auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::dimensionality = Dimensions [static]

Definition at line 62 of file accessor.hpp.

8.1.2.2 struct cl::sycl::accessor

template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer>struct cl::sycl::accessor< DataType, Dimensions, AccessMode, 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 38 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:

Collaboration diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:

Public Types

- using value_type = DataType
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler)

  Construct a buffer accessor from a buffer using a command group handler object from the command group scope.

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Additional Inherited Members

8.1.2.2.1 Member Typedef Documentation

8.1.2.2.1.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::const_reference = const value_type&

Definition at line 43 of file accessor.hpp.

8.1.2.2.1.2 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::reference = value_type&

Definition at line 42 of file accessor.hpp.

8.1.2.2.1.3 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer> using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::value_type = DataType

Definition at line 41 of file accessor.hpp.

8.1.2.2.2 Constructor & Destructor Documentation

8.1.2.2.2.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer> template<typename Allocator > cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > & *target_buffer,* handler & *command_group_handler*) [inline]

Construct a buffer accessor from a buffer using a command group handler object from the command group scope.

Constructor only available for access modes global_buffer, host_buffer, constant_buffer (see Table 3.25).

access_target defines the form of access being obtained. See Table 3.26.

Definition at line 58 of file accessor.hpp.

References cl::sycl::buffer< T, Dimensions, Allocator >::implementation.

```
00059                                          :
00060     detail::accessor<DataType, Dimensions, AccessMode, Target> { *target_buffer.implementation } {}
```

8.1.2.2.3 Member Data Documentation

8.1.2.2.3.1 template<typename DataType, std::size_t Dimensions, access::mode AccessMode, access::target Target = access::global_buffer> constexpr auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality = Dimensions [static]

Todo in the specification: store the dimension for user request

Definition at line 40 of file accessor.hpp.

8.1.2.3 struct cl::sycl::detail::buffer

template<typename T, std::size_t Dimensions = 1>struct cl::sycl::detail::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.

Definition at line 27 of file accessor.hpp.

Inheritance diagram for cl::sycl::detail::buffer< T, Dimensions >:

Collaboration diagram for cl::sycl::detail::buffer< T, Dimensions >:

Public Types

- using element = T
- using value_type = T

Public Member Functions

- buffer (range< Dimensions > const &r)

  Create a new read-write buffer of size.
- buffer (T *host_data, range< Dimensions > r)

  Create a new read-write buffer from.
- buffer (const T *host_data, range< Dimensions > r)

  Create a new read-only buffer from.
- template<typename Iterator >
  buffer (Iterator start_iterator, Iterator end_iterator)

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

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

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

Public Attributes

- boost::multi_array< T, Dimensions > allocation

  If some allocation is requested, it is managed by this multi_array to ease initialization from data.
- boost::multi_array_ref< T, Dimensions > access

  This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.

Additional Inherited Members

8.1.2.3.1 Member Typedef Documentation

8.1.2.3.1.1 template<typename T, std::size_t Dimensions = 1> using cl::sycl::detail::buffer< T, Dimensions >::element = T

Definition at line 41 of file buffer.hpp.

8.1.2.3.1.2 template<typename T, std::size_t Dimensions = 1> using cl::sycl::detail::buffer< T, Dimensions >::value_type = T

Definition at line 42 of file buffer.hpp.

8.1.2.3.2 Constructor & Destructor Documentation

8.1.2.3.2.1 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (range< Dimensions > const & *r*) [inline]

Create a new read-write buffer of size.

Parameters

r	

Definition at line 56 of file buffer.hpp.

```
00056                                     : buffer_base { false },
00057                                       allocation { r },
00058                                       access { allocation }
00059                                       {}
```

8.1.2.3.2.2 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (T * *host_data*, range< Dimensions > *r*) [inline]

Create a new read-write buffer from.

Parameters

host_data	of size
r	without further allocation

Definition at line 64 of file buffer.hpp.

```
00064                                         : buffer_base { false },
00065                                           access { host_data, r }
00066                                           {}
```

8.1.2.3.2.3 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (const T * *host_data*, range< Dimensions > *r*) [inline]

Create a new read-only buffer from.

Parameters

host_data	of size
r	without further allocation

Definition at line 71 of file buffer.hpp.

```
00071                                                   :
00072     /// \todo Need to solve this const buffer issue in a clean way
00073     buffer_base { true },
00074     access { const_cast<T *>(host_data), r }
00075     {}
```

8.1.2.3.2.4 template<typename T, std::size_t Dimensions = 1> template<typename Iterator > cl::sycl::detail::buffer< T, Dimensions >::buffer (Iterator *start_iterator*, Iterator *end_iterator*) [inline]

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

Todo

Definition at line 83 of file buffer.hpp.

```
00083                                                       :
00084     buffer_base { false },
00085     // The size of a multi_array is set at creation time
00086     allocation { boost::extents[std::distance(start_iterator, end_iterator)] },
00087     access { allocation }
00088     {
00089       /* Then assign allocation since this is the only multi_array
00090          method with this iterator interface */
00091       allocation.assign(start_iterator, end_iterator);
00092     }
```

8.1.2.3.2.5 template<typename T, std::size_t Dimensions = 1> cl::sycl::detail::buffer< T, Dimensions >::buffer (const buffer< T, Dimensions > & *b*) [inline]

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

Todo Refactor the implementation to deal with buffer sharing with reference counting

Definition at line 100 of file buffer.hpp.

```
00100                                          : buffer_base { b.read_only },
00101                                            allocation { b.access },
00102                                            access { allocation }
00103                                            {}
```

8.1.2.3.3 Member Function Documentation

8.1.2.3.3.1 template<typename T, std::size_t Dimensions = 1> template<access::mode Mode, access::target Target = access::global_buffer> detail::accessor<T, Dimensions, Mode, Target> cl::sycl::detail::buffer< T, Dimensions >::get_access ( ) [inline]

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

Todo To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base↩
_index, range<Dimensions> sub_range)

Todo Allow CLHPP objects too?

Return an accessor of the required mode

Parameters

M	

Todo Remove if not used

Definition at line 129 of file buffer.hpp.

```
00129                                                  {
00130     return { *this };
00131   }
```

8.1.2.3.4 Member Data Documentation

8.1.2.3.4.1 template<typename T, std::size_t Dimensions = 1> boost::multi_array_ref<T, Dimensions> cl::sycl::detail::buffer< T, Dimensions >::access

This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.

Definition at line 52 of file buffer.hpp.

8.1.2.3.4.2 template<typename T, std::size_t Dimensions = 1> boost::multi_array<T, Dimensions> cl::sycl::detail::buffer< T, Dimensions >::allocation

If some allocation is requested, it is managed by this multi_array to ease initialization from data.

Definition at line 46 of file buffer.hpp.

8.1.2.4 struct cl::sycl::buffer

template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>>struct cl::sycl::buffer< T, Dimensions, Allocator >

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.

Todo We have some read-write buffers and some read-only buffers, according to the constructor called. So we could have some static checking for correctness with the accessors used, but we do not have a way in the specification to have a read-only buffer type for this.

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

Todo Think about the need of an allocator when constructing a buffer from other buffers

Definition at line 21 of file accessor.hpp.

Collaboration diagram for cl::sycl::buffer< T, Dimensions, Allocator >:

Public Types

- using value_type = T

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &
- using allocator_type = Allocator

Public Member Functions

- buffer ()=default

  Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).
- buffer (const range< Dimensions > &r, Allocator allocator={})

  Create a new read-write buffer with storage managed by SYCL.

Public Attributes

- std::shared_ptr< detail::buffer< T, Dimensions > > implementation

  Point to the underlying buffer implementation that can be shared in the SYCL model.

8.1.2.4.1 Member Typedef Documentation

8.1.2.4.1.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::allocator_type = Allocator

Definition at line 54 of file buffer.hpp.

8.1.2.4.1.2 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::const_reference = const value_type&

Definition at line 53 of file buffer.hpp.

8.1.2.4.1.3 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::reference = value_type&

Definition at line 52 of file buffer.hpp.

8.1.2.4.1.4 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> using cl::sycl::buffer< T, Dimensions, Allocator >::value_type = T

The STL-like types.

Definition at line 51 of file buffer.hpp.

8.1.2.4.2 Constructor & Destructor Documentation

8.1.2.4.2.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer ( ) [default]

Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).

Since we just copy the shared_ptr<> above, this is where/how the sharing magic is happening with reference counting in this case.

8.1.2.4.2.2 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> cl::sycl::buffer< T, Dimensions, Allocator >::buffer (const range< Dimensions > & *r*, Allocator *allocator* = {}) [inline]

Create a new read-write buffer with storage managed by SYCL.

Parameters

r	defines the size

Definition at line 74 of file buffer.hpp.

```
00074                                                          {})
00075     : implementation { new detail::buffer<T, Dimensions> { r } } {}
```

8.1.2.4.3 Member Data Documentation

8.1.2.4.3.1 template<typename T, std::size_t Dimensions = 1, typename Allocator = buffer_allocator<T>> std::shared_ptr<detail::buffer<T, Dimensions> > cl::sycl::buffer< T, Dimensions, Allocator >::implementation

Point to the underlying buffer implementation that can be shared in the SYCL model.

Definition at line 58 of file buffer.hpp.

Referenced by cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor().

8.1.2.5 struct cl::sycl::image

template<std::size_t dimensions>struct cl::sycl::image< dimensions >

Todo implement image

Definition at line 23 of file image.hpp.

8.1.3 Typedef Documentation

8.1.3.1 template<typename T > using cl::sycl::buffer_allocator = typedef std::allocator<T>

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

The default buffer allocator used by the runtime, when no allocator is defined by the user.

Reuse the C++ default allocator.

Definition at line 28 of file buffer_allocator.hpp.

8.1.4 Enumeration Type Documentation

8.1.4.1 enum cl::sycl::access::fence_space : char [strong]

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

Precise the address space a barrier needs to act on.

Enumerator

local_space

global_space

global_and_local

Definition at line 59 of file access.hpp.

```
00059                          : char {
00060     local_space,
00061     global_space,
00062     global_and_local
00063   };
```

8.1.4.2 enum cl::sycl::access::mode

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

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

Enumerator

read Read-only access. Insist on the fact that read_write != read + write.

write Write-only access, but previous content *not* discarded.

read_write Read and write access.

discard_write Write-only access and previous content discarded.

discard_read_write Read and write access and previous content discarded.

atomic Atomic access.

Definition at line 33 of file access.hpp.

```
00033             {
00034     read = 42, ///< Read-only access. Insist on the fact that read_write != read + write
00035     write, ///< Write-only access, but previous content *not* discarded
00036     read_write, ///< Read and write access
00037     discard_write, ///< Write-only access and previous content discarded
00038     discard_read_write, ///< Read and write access and previous content discarded
00039     atomic ///< Atomic access
00040   };
```

8.1.4.3 enum cl::sycl::access::target

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

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

Definition at line 46 of file access.hpp.

```
00046               {
00047     global_buffer = 2014, //< Just pick a random number...
00048     constant_buffer,
00049     local,
00050     image,
00051     host_buffer,
00052     host_image,
00053     image_array
00054   };
```

8.2 Dealing with OpenCL address spaces

Collaboration diagram for Dealing with OpenCL address spaces:

Namespaces

- cl::sycl

Classes

- struct cl::sycl::detail::opencl_type< T, AS >

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::opencl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct cl::sycl::detail::opencl_type< T, generic_address_space >

  Add an attribute for __generic address space. More...
- struct cl::sycl::detail::opencl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct cl::sycl::detail::opencl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct cl::sycl::detail::opencl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

  Implementation of a variable with an OpenCL address space. More...

Typedefs

- template<typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.
- template<typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be an OpenCL constant pointer.
- template<typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be an OpenCL 2 generic pointer.
- template<typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

Declare a variable to be an OpenCL global pointer.

- template<typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be an OpenCL local pointer.
- template<typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

  Declare a variable to be an OpenCL private pointer.
- template<typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

8.2.1 Detailed Description

8.2.2 Class Documentation

8.2.2.1 struct cl::sycl::detail::opencl_type

template<typename T, address_space AS>struct cl::sycl::detail::opencl_type< T, AS >

Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device.

In the general case, do not add any OpenCL address space qualifier

Definition at line 27 of file address_space.hpp.

Public Types

- using type = T

8.2.2.1.1 Member Typedef Documentation

8.2.2.1.1.1 template<typename T, address_space AS> using cl::sycl::detail::opencl_type< T, AS >::type = T

Definition at line 28 of file address_space.hpp.

8.2.2.2 struct cl::sycl::detail::opencl_type< T, constant_address_space >

template<typename T>struct cl::sycl::detail::opencl_type< T, constant_address_space >

Add an attribute for __constant address space.

Definition at line 33 of file address_space.hpp.

Public Types

- using type = T

8.2.2.2.1 Member Typedef Documentation

8.2.2.2.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, constant_address_space >::type = T

Definition at line 40 of file address_space.hpp.

8.2.2.3 struct cl::sycl::detail::opencl_type< T, generic_address_space >

template<typename T>struct cl::sycl::detail::opencl_type< T, generic_address_space >

Add an attribute for __generic address space.

Definition at line 45 of file address_space.hpp.

Public Types

- using type = T

8.2.2.3.1 Member Typedef Documentation

8.2.2.3.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, generic_address_space >::type = T

Definition at line 52 of file address_space.hpp.

8.2.2.4 struct cl::sycl::detail::opencl_type< T, global_address_space >

template<typename T>struct cl::sycl::detail::opencl_type< T, global_address_space >

Add an attribute for __global address space.

Definition at line 57 of file address_space.hpp.

Public Types

- using type = T

8.2.2.4.1 Member Typedef Documentation

8.2.2.4.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, global_address_space >::type = T

Definition at line 64 of file address_space.hpp.

8.2.2.5 struct cl::sycl::detail::opencl_type< T, local_address_space >

template<typename T>struct cl::sycl::detail::opencl_type< T, local_address_space >

Add an attribute for __local address space.

Definition at line 69 of file address_space.hpp.

Public Types

- using type = T

8.2.2.5.1 Member Typedef Documentation

8.2.2.5.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, local_address_space >::type = T

Definition at line 76 of file address_space.hpp.

8.2.2.6 struct cl::sycl::detail::opencl_type< T, private_address_space >

template<typename T>struct cl::sycl::detail::opencl_type< T, private_address_space >

Add an attribute for __private address space.

Definition at line 81 of file address_space.hpp.

Public Types

- using type = T

8.2.2.6.1 Member Typedef Documentation

8.2.2.6.1.1 template<typename T > using cl::sycl::detail::opencl_type< T, private_address_space >::type = T

Definition at line 88 of file address_space.hpp.

8.2.2.7 struct cl::sycl::detail::address_space_array

template<typename T, address_space AS>struct cl::sycl::detail::address_space_array< T, AS >

Implementation of an array variable with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 95 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_array< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_array< T, AS >:

Public Types

- using super = address_space_variable< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_array (const T &array)

  Allow to create an address space array from an array.
- address_space_array (std::initializer_list< std::remove_extent_t< T >> list)

  Allow to create an address space array from an initializer list.

Additional Inherited Members

8.2.2.7.1 Member Typedef Documentation

8.2.2.7.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_array< T, AS >::super = address_space_variable<T, AS>

Keep track of the base class as a short-cut.

Definition at line 297 of file address_space.hpp.

8.2.2.7.2 Constructor & Destructor Documentation

8.2.2.7.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_array< T, AS >::address_space_array (const T & *array*) [inline]

Allow to create an address space array from an array.

Definition at line 305 of file address_space.hpp.

References cl::sycl::detail::address_space_variable< T, AS >::variable.

```
00305                                       {
00306     std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00307   };
```

8.2.2.7.2.2 template<typename T , address_space AS> cl::sycl::detail::address_space_array< T, AS >::address_space_array (std::initializer_list< std::remove_extent_t< T >> *list*) [inline]

Allow to create an address space array from an initializer list.

Todo Extend to more than 1 dimension

Definition at line 314 of file address_space.hpp.

References cl::sycl::detail::address_space_variable< T, AS >::variable.

```
00314                                                                       {
00315     std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00316   };
```

8.2.2.8 struct cl::sycl::detail::address_space_fundamental

template<typename T, address_space AS>struct cl::sycl::detail::address_space_fundamental< T, AS >

Implementation of a fundamental type with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Todo Verify/improve to deal with const/volatile?

Definition at line 98 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_fundamental< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_fundamental< T, AS >:

Public Types

- using super = address_space_variable< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_fundamental ()=default

  Also request for the default constructors that have been disabled by the declaration of another constructor.

- template<typename SomeType , cl::sycl::address_space SomeAS>
  address_space_fundamental (address_space_fundamental< SomeType, SomeAS > &v)

  Allow for example assignment of a global<float> to a priv<double> for example.

Additional Inherited Members

8.2.2.8.1 Member Typedef Documentation

8.2.2.8.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_fundamental< T, AS >::super = address_space_variable<T, AS>

Keep track of the base class as a short-cut.

Definition at line 216 of file address_space.hpp.

8.2.2.8.2 Constructor & Destructor Documentation

8.2.2.8.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental ( ) [default]

Also request for the default constructors that have been disabled by the declaration of another constructor.

This ensures for example that we can write

```
generic<float *> q;
```

without initialization.

8.2.2.8.2.2 template<typename T , address_space AS> template<typename SomeType , cl::sycl::address_space SomeAS> cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental (address_space_fundamental< SomeType, SomeAS > & *v*) [inline]

Allow for example assignment of a global<float> to a priv<double> for example.

Since it needs 2 implicit conversions, it does not work with the conversion operators already define, so add 1 more explicit conversion here so that the remaining implicit conversion can be found by the compiler.

Strangely

```
template <typename SomeType, address_space SomeAS>
address_space_base(addr_space<SomeType, SomeAS>& v)
: variable(SomeType(v)) { }
```

cannot be used here because SomeType cannot be inferred. So use address_space_base<> instead

Need to think further about it...

Definition at line 254 of file address_space.hpp.

References cl::sycl::detail::address_space_variable< T, AS >::variable.

```
00255   {
00256     /* Strangely I cannot have it working in the initializer instead, for
00257        some cases */
00258     super::variable = SomeType(v);
00259   }
```

8.2.2.9 struct cl::sycl::detail::address_space_object

template<typename T, address_space AS>struct cl::sycl::detail::address_space_object< T, AS >

Implementation of an object type with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

The class implementation is just inheriting of T so that all methods and non-member operators on T work also on address_space_object<T>

Todo Verify/improve to deal with const/volatile?

Todo what about T having some final methods?

Definition at line 101 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_object< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_object< T, AS >:

Public Types

- using opencl_type = typename opencl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.

Public Member Functions

- address_space_object (T &&v)

  Allow to create an address space version of an object or to convert one.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Additional Inherited Members

8.2.2.9.1 Member Typedef Documentation

8.2.2.9.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_object< T, AS >::opencl_type = typename opencl_type<T, AS>::type

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 341 of file address_space.hpp.

8.2.2.9.2 Constructor & Destructor Documentation

8.2.2.9.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_object< T, AS >::address_space_object (T && *v*) [inline]

Allow to create an address space version of an object or to convert one.

Definition at line 352 of file address_space.hpp.

```
00352 : opencl_type(v) { }
```

8.2.2.9.3 Member Function Documentation

8.2.2.9.3.1 template<typename T , address_space AS> cl::sycl::detail::address_space_object< T, AS >::operator opencl_type & ( ) [inline]

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 360 of file address_space.hpp.

```
00360 { return *this; }
```

8.2.2.10 struct cl::sycl::detail::address_space_ptr

template<typename T, address_space AS>struct cl::sycl::detail::address_space_ptr< T, AS >

Implementation for an OpenCL address space pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

All the address space pointers inherit from it, which makes trivial the implementation of cl::sycl::multi_ptr<T, AS>

Definition at line 104 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_ptr< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_ptr< T, AS >:

Public Types

- using super = address_space_fundamental< T, AS >

  Keep track of the base class as a short-cut.

Additional Inherited Members

8.2.2.10.1 Member Typedef Documentation

8.2.2.10.1.1 template<typename T, address_space AS> using cl::sycl::detail::address_space_ptr< T, AS >::super = address_space_fundamental<T, AS>

Keep track of the base class as a short-cut.

Definition at line 280 of file address_space.hpp.

8.2.2.11 struct cl::sycl::detail::address_space_base

template<typename T, address_space AS>struct cl::sycl::detail::address_space_base< T, AS >

Implementation of the base infrastructure to wrap something in an OpenCL address space.

Parameters

T	is the type of the basic stuff to be created
AS	is the address space to place the object into

Todo Verify/improve to deal with const/volatile?

Definition at line 135 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_base< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_base< T, AS >:

Public Types

- using type = T
  Store the base type of the object.
- using opencl_type = typename opencl_type< T, AS >::type
  Store the base type of the object with OpenCL address space modifier.

Static Public Attributes

- static auto constexpr address_space = AS
  Set the address_space identifier that can be queried to know the pointer type.

8.2.2.11.1 Member Typedef Documentation

8.2.2.11.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_base< T, AS >::opencl_type = typename opencl_type<T, AS>::type

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 146 of file address_space.hpp.

8.2.2.11.1.2 template<typename T , address_space AS> using cl::sycl::detail::address_space_base< T, AS >::type = T

Store the base type of the object.

Todo Add to the specification

Definition at line 140 of file address_space.hpp.

8.2.2.11.2 Member Data Documentation

8.2.2.11.2.1 template<typename T , address_space AS> auto constexpr cl::sycl::detail::address_space_base< T, AS >::address_space = AS [static]

Set the address_space identifier that can be queried to know the pointer type.

Definition at line 150 of file address_space.hpp.

8.2.2.12 struct cl::sycl::detail::address_space_variable

template<typename T, address_space AS>struct cl::sycl::detail::address_space_variable< T, AS >

Implementation of a variable with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 162 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_variable< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_variable< T, AS >:

Public Types

- using opencl_type = typename opencl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.
- using super = address_space_base< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_variable (const T &v)

  Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.
- address_space_variable ()=default

  Put back the default constructors canceled by the previous definition.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Protected Attributes

- opencl_type variable

Additional Inherited Members

8.2.2.12.1 Member Typedef Documentation

8.2.2.12.1.1 template<typename T , address_space AS> using cl::sycl::detail::address_space_variable< T, AS >::opencl_type = typename opencl_type<T, AS>::type

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 167 of file address_space.hpp.

8.2.2.12.1.2 template<typename T , address_space AS> using cl::sycl::detail::address_space_variable< T, AS >::super = address_space_base<T, AS>

Keep track of the base class as a short-cut.

Definition at line 170 of file address_space.hpp.

8.2.2.12.2 Constructor & Destructor Documentation

8.2.2.12.2.1 template<typename T , address_space AS> cl::sycl::detail::address_space_variable< T, AS >::address_space_variable (const T & *v*) [inline]

Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.

Definition at line 186 of file address_space.hpp.

```
00186 : variable(v) { }
```

8.2.2.12.2.2 template<typename T , address_space AS> cl::sycl::detail::address_space_variable< T, AS >::address_space_variable ( ) [default]

Put back the default constructors canceled by the previous definition.

8.2.2.12.3 Member Function Documentation

8.2.2.12.3.1 template<typename T , address_space AS> cl::sycl::detail::address_space_variable< T, AS >::operator opencl_type & ( ) [inline]

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 200 of file address_space.hpp.

References cl::sycl::detail::address_space_variable< T, AS >::variable.

```
00200 { return variable; }
```

8.2.2.12.4 Member Data Documentation

8.2.2.12.4.1 template<typename T , address_space AS> opencl_type cl::sycl::detail::address_space_variable< T, AS >::variable [protected]

Definition at line 179 of file address_space.hpp.

Referenced by cl::sycl::detail::address_space_array< T, AS >::address_space_array(), cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental(), and cl::sycl::detail::address_space_variable< T, AS >::operator opencl_type &().

8.2.3 Typedef Documentation

8.2.3.1 template<typename T , address_space AS> using cl::sycl::detail::addr_space = typedef typename std::conditional<std::is_pointer<T>::value, address_space_ptr<T, AS>, typename std::conditional<std::is_class<T>::value, address_space_object<T, AS>, typename std::conditional<std::is_array<T>::value, address_space_array<T, AS>, address_space_fundamental<T, AS> >::type>::type>::type

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

Dispatch the address space implementation according to the requested type.

Parameters

T	is the type of the object to be created
AS	is the address space to place the object into or to point to in the case of a pointer type

Definition at line 122 of file address_space.hpp.

8.2.3.2 template<typename T > using cl::sycl::constant = typedef detail::addr_space<T, constant_address_space>

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

Declare a variable to be an OpenCL constant pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

Definition at line 52 of file address_space.hpp.

8.2.3.3 template<typename T > using cl::sycl::generic = typedef detail::addr_space<T, generic_address_space>

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

Declare a variable to be an OpenCL 2 generic pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

Definition at line 62 of file address_space.hpp.

8.2.3.4 template<typename T > using cl::sycl::global = typedef detail::addr_space<T, global_address_space>

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

Declare a variable to be an OpenCL global pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

Definition at line 72 of file address_space.hpp.

8.2.3.5 template<typename T > using cl::sycl::local = typedef detail::addr_space<T, local_address_space>

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

Declare a variable to be an OpenCL local pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

Definition at line 82 of file address_space.hpp.

8.2.3.6 template<typename Pointer , address_space AS> using cl::sycl::multi_ptr = typedef detail::address_space_ptr<Pointer, AS>

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

A pointer that can be statically associated to any address-space.

Parameters

Pointer	is the pointer type
AS	is the address space to point to

Note that if *Pointer* is not a pointer type, it is an error.

Definition at line 104 of file address_space.hpp.

8.2.3.7 template<typename T > using cl::sycl::priv = typedef detail::addr_space<T, private_address_space>

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

Declare a variable to be an OpenCL private pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

Definition at line 92 of file address_space.hpp.

8.2.4 Enumeration Type Documentation

8.2.4.1 enum cl::sycl::address_space

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

Enumerate the different OpenCL 2 address spaces.

Enumerator

constant_address_space

generic_address_space

global_address_space

local_address_space

private_address_space

Definition at line 22 of file address_space.hpp.

```
00022                   {
00023   constant_address_space,
00024   generic_address_space,
00025   global_address_space,
00026   local_address_space,
00027   private_address_space,
00028 };
```

8.2.5 Function Documentation

8.2.5.1 template<typename T , address_space AS> multi_ptr<T, AS> cl::sycl::make_multi (multi_ptr< T, AS > *pointer*)

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

Construct a cl::sycl::multi_ptr<> with the right type.

Parameters

pointer	is the address with its address space to point to

Todo Implement the case with a plain pointer

Definition at line 114 of file address_space.hpp.

```
00114                                                 {
00115   return pointer;
00116 }
```

8.3 Platforms, contexts, devices and queues

Namespaces

- cl::sycl::info

Classes

- class cl::sycl::context

  SYCL context. More...
- class cl::sycl::device

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

  The SYCL heuristics to select a device. More...
- class cl::sycl::default_selector

  Devices selected by heuristics of the system. More...
- class cl::sycl::gpu_selector

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices. More...
- class cl::sycl::cpu_selector

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics. More...
- class cl::sycl::host_selector

  Selects the SYCL host CPU device that does not require an OpenCL runtime. More...
- class cl::sycl::kernel

  Kernel. More...
- class cl::sycl::handler

  Command group handler class. More...
- class cl::sycl::platform

  Abstract the OpenCL platform. More...
- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...

Enumerations

- enum cl::sycl::info::context : int { cl::sycl::info::context::reference_count, cl::sycl::info::context::num_devices, cl::sycl::info::context::gl_interop }

  Context information descriptors.
- enum cl::sycl::info::device : int { cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units, cl::sycl::info::device::max_work_item_dimensions, cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short, cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double, cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int, cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half, cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_mem_alloc_size, cl::sycl::info::device::image_support, cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info::device::image2d_max_height, cl::sycl::info::device::image2d_max_width, cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device::

::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size, cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device::max_parameter_size, cl::sycl::info::device::mem_base_addr_align, cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size, cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device::max_constant_buffer_size, cl::sycl::info::device::max_constant_args, cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_correction_support, cl::sycl::info::device::host_unified_memory, cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_available, cl::sycl::info::device::is_compiler_available, cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device::queue_properties, cl::sycl::info::device::built_in_kernels, cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device::driver_version, cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl::sycl::info::device::extensions, cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info::device::parent_device, cl::sycl::info::device::partition_max_sub_devices, cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info::device::partition_type, cl::sycl::info::device::reference_count }

Device information descriptors.

- enum cl::sycl::info::device_partition_property : int { cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition_equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_property::partition_by_affinity_domain, cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int { cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache, cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int { cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache, cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int { cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero, cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }
- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }
- enum cl::sycl::info::device_type : unsigned int { cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl::sycl::info::device_type::custom, cl::sycl::info::device_type::defaults, cl::sycl::info::device_type::host, cl::sycl::info::device_type::all }
- enum cl::sycl::info::platform : unsigned int { cl::sycl::info::platform::profile, cl::sycl::info::platform::version, cl::sycl::info::platform::name, cl::sycl::info::platform::vendor, cl::sycl::info::platform::extensions }

  Platform information descriptors.
- enum cl::sycl::info::queue : int { cl::sycl::info::queue::context, cl::sycl::info::queue::device, cl::sycl::info::queue::reference_count, cl::sycl::info::queue::properties }

  Queue information descriptors.

8.3.1 Detailed Description

8.3.2 Class Documentation

8.3.2.1 class cl::sycl::context

SYCL context.

The context class encapsulates an OpenCL context, which is implicitly created and the lifetime of the context instance defines the lifetime of the underlying OpenCL context instance.

On destruction clReleaseContext is called.

The default context is the SYCL host context containing only the SYCL host device.

Todo The implementation is quite minimal for now.

Definition at line 66 of file context.hpp.

Public Member Functions

- context (async_handler asyncHandler)

  Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.
- context (cl_context clContext, async_handler asyncHandler=nullptr)
- context (const device_selector &deviceSelector, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)

  Constructs a context object using a device_selector object.
- context (const device &dev, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)

  Constructs a context object using a device object.
- context (const platform &plt, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)

  Constructs a context object using a platform object.
- context (const vector_class< device > &deviceList, info::gl_context_interop interopFlag, async_handler asyncHandler=nullptr)
- context ()=default

  Default constructor that chooses the context according the heuristics of the default selector.
- cl_context get () const
- bool is_host () const

  Specifies whether the context is in SYCL Host Execution Mode.
- platform get_platform ()

  Returns the SYCL platform that the context is initialized for.
- vector_class< device > get_devices () const

  Returns the set of devices that are part of this context.
- template<info::context Param>
  info::param_traits< info::context, Param >::type get_info () const

  Queries OpenCL information for the under-lying cl context.

8.3.2.1.1 Constructor & Destructor Documentation

8.3.2.1.1.1 cl::sycl::context::context (**async_handler** *asyncHandler*) [inline],[explicit]

Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.

Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 76 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00076                                                 {
00077     detail::unimplemented();
00078   }
```

Here is the call graph for this function:

8.3.2.1.1.2 cl::sycl::context::context (cl_context *clContext,* async_handler *asyncHandler* = nullptr) [inline]

Definition at line 90 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00090                                                                 {
00091     detail::unimplemented();
00092   }
```

Here is the call graph for this function:

8.3.2.1.1.3 cl::sycl::context::context (const device_selector & *deviceSelector,* info::gl_context_interop *interopFlag,* async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a device_selector object.

The context is constructed with a single device retrieved from the device_selector object provided.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩ handler, if provided.

Definition at line 103 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00105                                                 {
00106     detail::unimplemented();
00107   }
```

Here is the call graph for this function:

8.3.2.1.1.4 cl::sycl::context::context (const device & *dev,* info::gl_context_interop *interopFlag,* async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a device object.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩ handler, if provided.

Definition at line 115 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00117                                                 {
00118     detail::unimplemented();
00119   }
```

Here is the call graph for this function:

8.3.2.1.1.5 cl::sycl::context::context (const platform & *plt,* info::gl_context_interop *interopFlag,* async_handler *asyncHandler* = nullptr) [inline]

Constructs a context object using a platform object.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩ handler, if provided.

Definition at line 127 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00129                                                {
00130     detail::unimplemented();
00131   }
```

Here is the call graph for this function:

8.3.2.1.1.6 cl::sycl::context::context (const vector_class< device > & *deviceList*, info::gl_context_interop *interopFlag*, async_handler *asyncHandler* = nullptr) [inline]

Definition at line 142 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00144                                                {
00145     detail::unimplemented();
00146   }
```

Here is the call graph for this function:

8.3.2.1.1.7 cl::sycl::context::context ( ) [default]

Default constructor that chooses the context according the heuristics of the default selector.

Return synchronous errors via the SYCL exception class.

Get the default constructors back.

8.3.2.1.2 Member Function Documentation

8.3.2.1.2.1 cl_context cl::sycl::context::get ( ) const [inline]

Definition at line 165 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00165                          {
00166     detail::unimplemented();
00167     return {};
00168   }
```

Here is the call graph for this function:

8.3.2.1.2.2 vector_class<device> cl::sycl::context::get_devices ( ) const [inline]

Returns the set of devices that are part of this context.

Todo To be implemented

Definition at line 189 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00189                                          {
00190     detail::unimplemented();
00191     return {};
00192   }
```

Here is the call graph for this function:

8.3.2.1.2.3 template<info::context Param> info::param_traits<info::context, Param>::type cl::sycl::context::get_info ( ) const [inline]

Queries OpenCL information for the under-lying cl context.

Todo To be implemented

Definition at line 200 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00200                                                                    {
00201     detail::unimplemented();
00202     return {};
00203   }
```

Here is the call graph for this function:

8.3.2.1.2.4 platform cl::sycl::context::get_platform ( )

Returns the SYCL platform that the context is initialized for.

Todo To be implemented

8.3.2.1.2.5 bool cl::sycl::context::is_host ( ) const [inline]

Specifies whether the context is in SYCL Host Execution Mode.

Definition at line 173 of file context.hpp.

```
00173                         {
00174     return true;
00175   }
```

8.3.2.2 class cl::sycl::device

SYCL device.

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

Definition at line 184 of file device.hpp.

Public Member Functions

- device (cl_device_id deviceId)
  Construct a device class instance using cl_device_id of the OpenCL device.
- device (const device_selector &deviceSelector)
  Construct a device class instance using the device selector provided.
- device ()=default
  The default constructor will create an instance of the SYCL host device.
- cl_device_id get () const
  Return the cl_device_id of the underlying OpenCL platform.
- bool is_host () const
  Return true if the device is a SYCL host device.
- bool is_cpu () const
  Return true if the device is an OpenCL CPU device.
- bool is_gpu () const
  Return true if the device is an OpenCL GPU device.
- bool is_accelerator () const
  Return true if the device is an OpenCL accelerator device.
- platform get_platform () const
  Return the platform of device.

- template<info::device Param>
  info::param_traits< info::device, Param >::type get_info () const

  Query the device for OpenCL info::device info.
- bool has_extension (const string_class &extension) const

  Specify whether a specific extension is supported on the device.
- vector_class< device > create_sub_devices (info::device_partition_type partitionType, info::device_↩ partition_property partitionProperty, info::device_affinity_domain affinityDomain) const

  Partition the device into sub devices based upon the properties provided.

Static Public Member Functions

- static vector_class< device > get_devices (info::device_type deviceType=info::device_type::all)

  Return a list of all available devices.

8.3.2.2.1 Constructor & Destructor Documentation

8.3.2.2.1.1 cl::sycl::device::device (cl_device_id *deviceId*) [inline],[explicit]

Construct a device class instance using cl_device_id of the OpenCL device.

Return synchronous errors via the SYCL exception class.

Retain a reference to the OpenCL device and if this device was an OpenCL subdevice the device should be released by the caller when it is no longer needed.

Todo To be implemented

Definition at line 201 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00201                                       {
00202     detail::unimplemented();
00203   }
```

Here is the call graph for this function:

8.3.2.2.1.2 cl::sycl::device::device (const device_selector & *deviceSelector*) [inline],[explicit]

Construct a device class instance using the device selector provided.

Return errors via C++ exception class.

Todo To be implemented

Definition at line 214 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00214                                                       {
00215     detail::unimplemented();
00216   }
```

Here is the call graph for this function:

8.3.2.2.1.3 cl::sycl::device::device ( ) [default]

The default constructor will create an instance of the SYCL host device.

Get the default constructors back.

8.3.2.2.2 Member Function Documentation

8.3.2.2.2.1 vector_class<device> cl::sycl::device::create_sub_devices (info::device_partition_type *partitionType*, info::device_partition_property *partitionProperty*, info::device_affinity_domain *affinityDomain*) const [inline]

Partition the device into sub devices based upon the properties provided.

Return synchronous errors via SYCL exception classes.

Todo To be implemented

Definition at line 343 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00345                                                                    {
00346     detail::unimplemented();
00347     return {};
00348   }
```

Here is the call graph for this function:

8.3.2.2.2.2 cl_device_id cl::sycl::device::get ( ) const [inline]

Return the cl_device_id of the underlying OpenCL platform.

Return synchronous errors via the SYCL exception class.

Retain a reference to the returned cl_device_id object. Caller should release it when finished.

In the case where this is the SYCL host device it will return a nullptr.

Todo To be implemented

Definition at line 240 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00240                          {
00241     detail::unimplemented();
00242     return {};
00243   }
```

Here is the call graph for this function:

8.3.2.2.2.3 static vector_class<device> cl::sycl::device::get_devices (info::device_type *deviceType* = info::device_type::all) [inline], [static]

Return a list of all available devices.

Return synchronous errors via SYCL exception classes.

Todo To be implemented

Definition at line 305 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00305                                                                {
00306     detail::unimplemented();
00307     return {};
00308   }
```

Here is the call graph for this function:

8.3.2.2.4 template<info::device Param> info::param_traits<info::device, Param>::type cl::sycl::device::get_info ( ) const [inline]

Query the device for OpenCL info::device info.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

Definition at line 319 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00319                   {
00320     detail::unimplemented();
00321     return {};
00322   }
```

Here is the call graph for this function:

8.3.2.2.5 platform cl::sycl::device::get_platform ( ) const [inline]

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

Definition at line 292 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00292                                {
00293     detail::unimplemented();
00294     return {};
00295   }
```

Here is the call graph for this function:

8.3.2.2.6 bool cl::sycl::device::has_extension (const string_class & *extension*) const [inline]

Specify whether a specific extension is supported on the device.

Todo To be implemented

Definition at line 329 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00329                                                    {
00330     detail::unimplemented();
00331     return {};
00332   }
```

Here is the call graph for this function:

8.3.2.2.7 bool cl::sycl::device::is_accelerator ( ) const [inline]

Return true if the device is an OpenCL accelerator device.

Todo To be implemented

Definition at line 280 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00280                                {
00281     detail::unimplemented();
00282     return {};
00283   }
```

Here is the call graph for this function:

8.3.2.2.2.8 bool cl::sycl::device::is_cpu ( ) const [inline]

Return true if the device is an OpenCL CPU device.

Todo To be implemented

Definition at line 260 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00260                        {
00261     detail::unimplemented();
00262     return {};
00263   }
```

Here is the call graph for this function:

8.3.2.2.2.9 bool cl::sycl::device::is_gpu ( ) const [inline]

Return true if the device is an OpenCL GPU device.

Todo To be implemented

Definition at line 270 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00270                        {
00271     detail::unimplemented();
00272     return {};
00273   }
```

Here is the call graph for this function:

8.3.2.2.2.10 bool cl::sycl::device::is_host ( ) const [inline]

Return true if the device is a SYCL host device.

Todo To be implemented

Definition at line 250 of file device.hpp.

References cl::sycl::detail::unimplemented().

```
00250                         {
00251     detail::unimplemented();
00252     return true;
00253   }
```

Here is the call graph for this function:

8.3.2.3 class cl::sycl::device_selector

The SYCL heuristics to select a device.

The device with the highest score is selected

Definition at line 26 of file device_selector.hpp.

Inheritance diagram for cl::sycl::device_selector:

Public Member Functions

- device select_device () const

  Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)
- virtual int operator() (const device &dev) const =0

  This pure virtual operator allows the customization of device selection.

8.3.2.3.1 Member Function Documentation

8.3.2.3.1.1 virtual int cl::sycl::device_selector::operator() (const device & *dev*) const [pure virtual]

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implemented in cl::sycl::host_selector, cl::sycl::cpu_selector, cl::sycl::gpu_selector, and cl::sycl::default_selector.

8.3.2.3.1.2 device cl::sycl::device_selector::select_device ( ) const [inline]

Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)

Todo To be implemented

Definition at line 35 of file device_selector.hpp.

References cl::sycl::detail::unimplemented().

```
00035                                {
00036     detail::unimplemented();
00037     return {};
00038   }
```

Here is the call graph for this function:

8.3.2.4 class cl::sycl::default_selector

Devices selected by heuristics of the system.

If no OpenCL device is found then it defaults to the SYCL host device.

Todo to be implemented

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

Definition at line 61 of file device_selector.hpp.

Inheritance diagram for cl::sycl::default_selector:

Collaboration diagram for cl::sycl::default_selector:

Public Member Functions

- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

8.3.2.4.1 Member Function Documentation

8.3.2.4.1.1 int cl::sycl::default_selector::operator() (const device & *dev*) const [inline], [override], [virtual]

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implements cl::sycl::device_selector.

Definition at line 66 of file device_selector.hpp.

References cl::sycl::detail::unimplemented().

```
00066                                                 {
00067     detail::unimplemented();
00068     return 1;
00069   }
```

Here is the call graph for this function:

8.3.2.5 class cl::sycl::gpu_selector

Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.

If no OpenCL GPU device is found the selector fails.

Select the best GPU, if any.

Todo to be implemented

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

Definition at line 85 of file device_selector.hpp.

Inheritance diagram for cl::sycl::gpu_selector:

Collaboration diagram for cl::sycl::gpu_selector:

Public Member Functions

- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

8.3.2.5.1 Member Function Documentation

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

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implements cl::sycl::device_selector.

Definition at line 90 of file device_selector.hpp.

References cl::sycl::detail::unimplemented().

```
00090                                                 {
00091     detail::unimplemented();
00092     return 1;
00093   }
```

Here is the call graph for this function:

8.3.2.6 class cl::sycl::cpu_selector

Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.

If no OpenCL CPU device is found the selector fails.

Todo to be implemented

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

Definition at line 107 of file device_selector.hpp.

Inheritance diagram for cl::sycl::cpu_selector:

Collaboration diagram for cl::sycl::cpu_selector:

Public Member Functions

- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

8.3.2.6.1 Member Function Documentation

8.3.2.6.1.1 int cl::sycl::cpu_selector::operator() (const device & *dev*) const [inline], [override], [virtual]

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implements cl::sycl::device_selector.

Definition at line 112 of file device_selector.hpp.

References cl::sycl::detail::unimplemented().

```
00112                                                 {
00113     detail::unimplemented();
00114     return 1;
00115   }
```

Here is the call graph for this function:

8.3.2.7 class cl::sycl::host_selector

Selects the SYCL host CPU device that does not require an OpenCL runtime.

Todo to be implemented

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

Definition at line 127 of file device_selector.hpp.

Inheritance diagram for cl::sycl::host_selector:

Collaboration diagram for cl::sycl::host_selector:

Public Member Functions

- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

8.3.2.7.1 Member Function Documentation

8.3.2.7.1.1 int cl::sycl::host_selector::operator() (const device & *dev*) const [inline], [override], [virtual]

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implements cl::sycl::device_selector.

Definition at line 132 of file device_selector.hpp.

References cl::sycl::detail::unimplemented().

```
00132                                                 {
00133     detail::unimplemented();
00134     return 1;
00135   }
```

Here is the call graph for this function:

8.3.2.8 class cl::sycl::kernel

Kernel.

Todo To be implemented

Definition at line 29 of file handler.hpp.

8.3.2.9 class cl::sycl::handler

Command group handler class.

A command group handler object can only be constructed by the SYCL runtime.

All of the accessors defined in the command group scope take as a parameter an instance of the command group handler and all the kernel invocation functions are methods of this class.

Definition at line 41 of file handler.hpp.

Collaboration diagram for cl::sycl::handler:

Public Member Functions

- handler ()
- template<typename DataType , std::size_t Dimensions, access::mode Mode, access::target Target = access::global_buffer>
  void set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > acc_obj)

  Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

- template<typename T >
  void set_arg (int arg_index, T scalar_value)

  Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface.
- template<typename KernelName = std::nullptr_t>
  void single_task (std::function< void(void)> F)

  Kernel invocation method of a kernel defined as a lambda or functor.
- TRISYCL_parallel_for_functor_GLOBAL (1) TRISYCL_parallel_for_functor_GLOBAL(2) TRISYCL_parallel_for_functor_GLOBAL(3) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3) template< typename KernelName

  Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.
- std::size_t ParallelForFunctor void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)
- template<typename KernelName = std::nullptr_t, std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_group (nd_range< Dimensions > r, ParallelForFunctor f)

  Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.
- void single_task (kernel syclKernel)

  Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.
- template<std::size_t Dimensions = 1>
  void parallel_for (range< Dimensions > numWorkItems, kernel sycl_kernel)

  Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 3.5.3.
- template<std::size_t Dimensions = 1>
  void parallel_for (nd_range< Dimensions >, kernel syclKernel)

  Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range, described in detail in 3.5.3.

Public Attributes

- std::shared_ptr< detail::task > current_task

  Attach the task and accessors to it.
- std::size_t Dimensions

8.3.2.9.1 Constructor & Destructor Documentation

8.3.2.9.1.1 cl::sycl::handler::handler ( ) [inline]

Definition at line 50 of file handler.hpp.

```
00050             {
00051     // Create a new task for this command_group
00052     current_task = std::make_shared<detail::task>();
00053   }
```

8.3.2.9.2 Member Function Documentation

8.3.2.9.2.1 std::size_t ParallelForFunctor void cl::sycl::handler::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*) [inline]

Definition at line 209 of file handler.hpp.

References cl::sycl::detail::parallel_for().

```
00209                                                                 {
00210     current_task->schedule([=] { detail::parallel_for(r, f); });
00211   }
```

Here is the call graph for this function:

8.3.2.9.2.2 template<std::size_t Dimensions = 1> void cl::sycl::handler::parallel_for (range< Dimensions > *numWorkItems,* kernel *sycl_kernel*) [inline]

Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 3.5.3.

Todo To be implemented

Definition at line 261 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
00262                                       {
00263     detail::unimplemented();
00264   }
```

Here is the call graph for this function:

8.3.2.9.2.3 template<std::size_t Dimensions = 1> void cl::sycl::handler::parallel_for (nd_range< Dimensions >, kernel *syclKernel*) [inline]

Kernel invocation method of a kernel defined as pointer to a kernel object, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range, described in detail in 3.5.3.

Todo To be implemented

Definition at line 275 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
00275                                                       {
00276     detail::unimplemented();
00277   }
```

Here is the call graph for this function:

8.3.2.9.2.4 template<typename KernelName = std::nullptr_t, std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::handler::parallel_for_work_group (nd_range< Dimensions > *r,* ParallelForFunctor *f*) [inline]

Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

May contain multiple kernel built-in parallel_for_work_item functions representing the execution on each work-item.

Launch num_work_groups work-groups of runtime-defined size. Described in detail in 3.5.3.

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelFor↩ Functor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 238 of file handler.hpp.

References cl::sycl::detail::parallel_for_workgroup().

```
00239                                                   {
00240     current_task->schedule([=] { detail::parallel_for_workgroup(r
      , f); });
00241   }
```

Here is the call graph for this function:

8.3.2.9.2.5 template<typename DataType , std::size_t Dimensions, access::mode Mode, access::target Target = access::global_buffer> void cl::sycl::handler::set_arg (int *arg_index,* accessor< DataType, Dimensions, Mode, Target > *acc_obj*) [inline]

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

The index value specifies which parameter of the OpenCL kernel is being set and the accessor object, which OpenCL buffer or image is going to be given as kernel argument.

Todo To be implemented

Definition at line 69 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
00070                                                                    {
00071     detail::unimplemented();
00072   }
```

Here is the call graph for this function:

8.3.2.9.2.6 template<typename T > void cl::sycl::handler::set_arg (int *arg_index,* T *scalar_value*) [inline]

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface.

The index value specifies which parameter of the OpenCL kernel is being set and the accessor object, which OpenCL buffer or image is going to be given as kernel argument.

Todo To be implemented

Definition at line 85 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
00085                                                {
00086     detail::unimplemented();
00087   }
```

Here is the call graph for this function:

8.3.2.9.2.7 template<typename KernelName = std::nullptr_t> void cl::sycl::handler::single_task (std::function< void(void)> *F*) [inline]

Kernel invocation method of a kernel defined as a lambda or functor.

If it is a lambda function or the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in 3.5.3

SYCL single_task launches a computation without parallelism at launch time.

Parameters

F	specify the kernel to be launched as a single_task
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 104 of file handler.hpp.

```
00104                                                {
00105     current_task->schedule(F);
00106   }
```

8.3.2.9.2.8 void cl::sycl::handler::single_task (kernel *syclKernel*) [inline]

Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.

Todo To be implemented

Definition at line 249 of file handler.hpp.

References cl::sycl::detail::unimplemented().

```
00249                                       {
00250     detail::unimplemented();
00251   }
```

Here is the call graph for this function:

8.3.2.9.2.9 cl::sycl::handler::TRISYCL_parallel_for_functor_GLOBAL (1)

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the global size of the range<>
offset	is the offset to be add to the id<> during iteration
f	is the kernel functor to execute
ParallelFor↩ Functor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the dimensionsKernel invocation method of a kernel defined as a lambda or functor, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelFor↩ Functor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

8.3.2.9.3 Member Data Documentation

8.3.2.9.3.1 std::shared_ptr<detail::task> cl::sycl::handler::current_task

Attach the task and accessors to it.

Definition at line 47 of file handler.hpp.

8.3.2.9.3.2 std::size_t cl::sycl::handler::Dimensions

Definition at line 207 of file handler.hpp.

8.3.2.10 class cl::sycl::platform

Abstract the OpenCL platform.

Todo triSYCL Implementation

Definition at line 81 of file platform.hpp.

Public Member Functions

- platform (cl_platform_id platformID)

  Construct a default platform and provide an optional error_handler to deals with errors.
- platform ()=default

  Default constructor for platform.
- cl_platform_id get () const

  Returns the cl_platform_id of the underlying OpenCL platform.
- vector_class< device > get_devices (info::device_type device_type=info::device_type::all) const

  Returns all the available devices for this platform, of type device type, which is defaulted to info::device_type::all.
- template<info::platform Param>
  info::param_traits< info::platform, Param >::type get_info () const

  Get the OpenCL information about the requested parameter.
- bool has_extension (const string_class &extension) const

  Test if an extension is available on the platform.
- bool is_host () const

  Test if this platform is a host platform.

Static Public Member Functions

- static vector_class< platform > get_platforms ()

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

8.3.2.10.1 Constructor & Destructor Documentation

8.3.2.10.1.1 cl::sycl::platform::platform (cl_platform_id *platformID*) [inline],[explicit]

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 94 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00094                                          {
00095     detail::unimplemented();
00096   }
```

Here is the call graph for this function:

8.3.2.10.1.2 cl::sycl::platform::platform ( ) [default]

Default constructor for platform.

It constructs a platform object to encapsulate the device returned by the default device selector.

Returns errors via the SYCL exception class.

Get back the default constructors, for this implementation.

8.3.2.10.2 Member Function Documentation

8.3.2.10.2.1 cl_platform_id cl::sycl::platform::get ( ) const [inline]

Returns the cl_platform_id of the underlying OpenCL platform.

If the platform is not a valid OpenCL platform, for example if it is the SYCL host, a nullptr will be returned.

Todo To be implemented

Definition at line 120 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00120                                {
00121     detail::unimplemented();
00122     return {};
00123   }
```

Here is the call graph for this function:

8.3.2.10.2.2 vector_class<device> cl::sycl::platform::get_devices (info::device_type *device_type* = info::device_type::all) const [inline]

Returns all the available devices for this platform, of type device type, which is defaulted to info::device_type::all.

By default returns all the devices.

Definition at line 144 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00144                                                                    {
00145     detail::unimplemented();
00146     return {};
00147   }
```

Here is the call graph for this function:

8.3.2.10.2.3 template<info::platform Param> info::param_traits<info::platform, Param>::type cl::sycl::platform::get_info ( ) const [inline]

Get the OpenCL information about the requested parameter.

Todo To be implemented

Definition at line 156 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00156                     {
00157     detail::unimplemented();
00158     return {};
00159   }
```

Here is the call graph for this function:

8.3.2.10.2.4 static vector_class<platform> cl::sycl::platform::get_platforms ( ) [inline], [static]

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

Todo To be implemented

Definition at line 131 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00131                                                {
00132     detail::unimplemented();
00133     return {};
00134   }
```

Here is the call graph for this function:

8.3.2.10.2.5 bool cl::sycl::platform::has_extension (const string_class & *extension*) const [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 168 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00168                                                        {
00169     detail::unimplemented();
00170     return {};
00171   }
```

Here is the call graph for this function:

8.3.2.10.2.6 bool cl::sycl::platform::is_host ( ) const [inline]

Test if this platform is a host platform.

Definition at line 175 of file platform.hpp.

```
00175                        {
00176     // Right now, this is a host-only implementation :-)
00177     return true;
00178   }
```

8.3.2.11 class cl::sycl::queue

SYCL queue, similar to the OpenCL queue concept.

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

Definition at line 66 of file queue.hpp.

Public Member Functions

- queue (async_handler asyncHandler)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue (const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  Creates a queue for the device provided by the device selector.
- queue (const device &syclDevice, async_handler asyncHandler=nullptr)

  A queue is created for syclDevice.
- queue (const context &syclContext, const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  This constructor chooses a device based on the provided device_selector, which needs to be in the given context.
- queue (const context &syclContext, const device &syclDevice, async_handler asyncHandler=nullptr)

  Creates a command queue using clCreateCommandQueue from a context and a device.

- queue (const context &syclContext, const device &syclDevice, info::queue_profiling profilingFlag, async_↩ handler asyncHandler=nullptr)

  Creates a command queue using clCreateCommandQueue from a context and a device.
- queue (const cl_command_queue &clQueue, async_handler asyncHandler=nullptr)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue ()=default

  Get the default constructors back.
- cl_command_queue get () const

  Return the underlying OpenCL command queue after doing a retain.
- context get_context () const

  Return the SYCL queue's context.
- device get_device () const

  Return the SYCL device the queue is associated with.
- bool is_host () const

  Return whether the queue is executing on a SYCL host device.
- void wait ()

  Performs a blocking wait for the completion all enqueued tasks in the queue.
- void wait_and_throw ()

  Perform a blocking wait for the completion all enqueued tasks in the queue.
- void throw_asynchronous ()

  Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.
- template<info::queue param>
  info::param_traits< info::queue, param >::type get_info () const

  Queries the platform for cl_command_queue info.
- handler_event submit (std::function< void(handler &)> cgf)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.
- handler_event submit (std::function< void(handler &)> cgf, queue &secondaryQueue)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.

8.3.2.11.1 Constructor & Destructor Documentation

8.3.2.11.1.1 cl::sycl::queue::queue (**async_handler** ***asyncHandler***) [inline],[explicit]

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Retain a reference to the cl_command_queue object. Caller should release the passed cl_command_queue object when it is no longer needed.

Return synchronous errors regarding the creation of the queue and report asynchronous errors via the async_↩ handler callback function in conjunction with the synchronization and throw methods.

Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 85 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00085                                                {
00086     detail::unimplemented();
00087   }
```

Here is the call graph for this function:

8.3.2.11.1.2 cl::sycl::queue::queue (const device_selector & *deviceSelector,* async_handler *asyncHandler* = nullptr) [inline]

Creates a queue for the device provided by the device selector.

If no device is selected, an error is reported.

Return synchronous errors regarding the creation of the queue and report asynchronous errors via the async_↩ handler callback function if and only if there is an async_handler provided.

Definition at line 98 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00099                                          {
00100     detail::unimplemented();
00101   }
```

Here is the call graph for this function:

8.3.2.11.1.3 cl::sycl::queue::queue (const device & *syclDevice,* async_handler *asyncHandler* = nullptr) [inline]

A queue is created for syclDevice.

Return asynchronous errors via the async_handler callback function.

Definition at line 108 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00109                                          {
00110     detail::unimplemented();
00111   };
```

Here is the call graph for this function:

8.3.2.11.1.4 cl::sycl::queue::queue (const context & *syclContext,* const device_selector & *deviceSelector,* async_handler *asyncHandler* = nullptr) [inline]

This constructor chooses a device based on the provided device_selector, which needs to be in the given context.

If no device is selected, an error is reported.

Return synchronous errors regarding the creation of the queue.

If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 125 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00127                                          {
00128     detail::unimplemented();
00129   }
```

Here is the call graph for this function:

8.3.2.11.1.5 cl::sycl::queue::queue (const context & *syclContext,* const device & *syclDevice,* async_handler *asyncHandler* = nullptr) [inline]

Creates a command queue using clCreateCommandQueue from a context and a device.

Return synchronous errors regarding the creation of the queue.

If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 141 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00143                                                {
00144     detail::unimplemented();
00145   }
```

Here is the call graph for this function:

8.3.2.11.1.6 cl::sycl::queue::queue (const context & *syclContext,* const device & *syclDevice,* info::queue_profiling *profilingFlag,* async_handler *asyncHandler* = nullptr) [inline]

Creates a command queue using clCreateCommandQueue from a context and a device.

It enables profiling on the queue if the profilingFlag is set to true.

Return synchronous errors regarding the creation of the queue. If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 159 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00162                                                {
00163     detail::unimplemented();
00164   }
```

Here is the call graph for this function:

8.3.2.11.1.7 cl::sycl::queue::queue (const cl_command_queue & *clQueue,* async_handler *asyncHandler* = nullptr) [inline]

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Return synchronous errors regarding the creation of the queue. If and only if there is an async_handler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 177 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00178                                                {
00179     detail::unimplemented();
00180   }
```

Here is the call graph for this function:

8.3.2.11.1.8 cl::sycl::queue::queue ( ) [default]

Get the default constructors back.

8.3.2.11.2 Member Function Documentation

8.3.2.11.2.1 cl_command_queue cl::sycl::queue::get ( ) const [inline]

Return the underlying OpenCL command queue after doing a retain.

This memory object is expected to be released by the developer.

Retain a reference to the returned cl_command_queue object.

Caller should release it when finished.

If the queue is a SYCL host queue then a nullptr will be returned.

Definition at line 199 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00199                                {
00200     detail::unimplemented();
00201     return {};
00202   }
```

Here is the call graph for this function:

8.3.2.11.2.2 context cl::sycl::queue::get_context ( ) const [inline]

Return the SYCL queue's context.

Report errors using SYCL exception classes.

Definition at line 210 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00210                               {
00211     detail::unimplemented();
00212     return {};
00213   }
```

Here is the call graph for this function:

8.3.2.11.2.3 device cl::sycl::queue::get_device ( ) const [inline]

Return the SYCL device the queue is associated with.

Report errors using SYCL exception classes.

Definition at line 220 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00220                              {
00221     detail::unimplemented();
00222     return {};
00223   }
```

Here is the call graph for this function:

8.3.2.11.2.4 template<info::queue param> info::param_traits<info::queue, param>::type cl::sycl::queue::get_info ( ) const [inline]

Queries the platform for cl_command_queue info.

Definition at line 272 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00272                                                                {
00273     detail::unimplemented();
00274     return {};
00275   }
```

Here is the call graph for this function:

8.3.2.11.2.5 bool cl::sycl::queue::is_host ( ) const [inline]

Return whether the queue is executing on a SYCL host device.

Definition at line 228 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00228                       {
00229     detail::unimplemented();
00230     return true;
00231   }
```

Here is the call graph for this function:

8.3.2.11.2.6 handler_event cl::sycl::queue::submit (std::function< void(handler &)> *cgf*) [inline]

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

Use an explicit functor parameter taking a handler& so we can use "auto" in submit() lambda parameter.

Definition at line 284 of file queue.hpp.

Referenced by submit().

```
00284                                                 {
00285     handler command_group_handler;
00286     cgf(command_group_handler);
00287     return {};
00288   }
```

Here is the caller graph for this function:

8.3.2.11.2.7 handler_event cl::sycl::queue::submit (std::function< void(handler &)> *cgf,* queue & *secondaryQueue*) [inline]

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

On kernel error, this command group functor, then it is scheduled for execution on the secondary queue.

Return a command group functor event, which is corresponds to the queue the command group functor is being enqueued on.

Definition at line 300 of file queue.hpp.

References submit(), and cl::sycl::detail::unimplemented().

```
00300                                                                       {
00301     detail::unimplemented();
00302     // Since it is not implemented, always submit on the main queue
00303     return submit(cgf);
00304   }
```

Here is the call graph for this function:

8.3.2.11.2.8 void cl::sycl::queue::throw_asynchronous ( ) [inline]

Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.

If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 265 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00265                           {
00266     detail::unimplemented();
00267   }
```

Here is the call graph for this function:

8.3.2.11.2.9 void cl::sycl::queue::wait ( ) [inline]

Performs a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported through SYCL exceptions.

Definition at line 238 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00238               {
00239     detail::unimplemented();
00240   }
```

Here is the call graph for this function:

8.3.2.11.2.10 void cl::sycl::queue::wait_and_throw ( ) [inline]

Perform a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported via SYCL exceptions.

Asynchronous errors will be passed to the async_handler passed to the queue on construction.

If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 253 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00253                         {
00254     detail::unimplemented();
00255   }
```

Here is the call graph for this function:

8.3.3 Enumeration Type Documentation

8.3.3.1 enum cl::sycl::info::context : int [strong]

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

Context information descriptors.

Todo Should be unsigned int to be consistent with others?

Enumerator

reference_count

num_devices

gl_interop

Definition at line 37 of file context.hpp.

```
00037                   : int {
00038   reference_count,
00039   num_devices,
00040   gl_interop
00041 };
```

8.3.3.2 enum cl::sycl::info::device : int [strong]

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

Device information descriptors.

From specs/latex/headers/deviceInfo.h in the specification

Todo Should be unsigned int?

Enumerator

device_type

vendor_id

max_compute_units

max_work_item_dimensions

max_work_item_sizes

max_work_group_size

preferred_vector_width_char

preferred_vector_width_short

preferred_vector_width_int

preferred_vector_width_long_long

preferred_vector_width_float

preferred_vector_width_double

preferred_vector_width_half

native_vector_witdth_char

native_vector_witdth_short

native_vector_witdth_int

native_vector_witdth_long_long

native_vector_witdth_float

native_vector_witdth_double

native_vector_witdth_half

max_clock_frequency

address_bits

max_mem_alloc_size

image_support

max_read_image_args

max_write_image_args

image2d_max_height

image2d_max_width

image3d_max_height

image3d_max_widht

image3d_mas_depth

image_max_buffer_size

image_max_array_size

max_samplers

max_parameter_size

mem_base_addr_align

single_fp_config

double_fp_config

global_mem_cache_type

global_mem_cache_line_size

global_mem_cache_size

global_mem_size

max_constant_buffer_size

max_constant_args

local_mem_type

local_mem_size

error_correction_support

host_unified_memory

profiling_timer_resolution

endian_little

is_available

is_compiler_available

is_linker_available

execution_capabilities

queue_properties

built_in_kernels

platform

name

vendor

driver_version

profile

device_version

opencl_version

extensions

printf_buffer_size

preferred_interop_user_sync

parent_device

partition_max_sub_devices

partition_properties

partition_affinity_domain

partition_type

reference_count

Definition at line 36 of file device.hpp.

```
00036                 : int {
00037   device_type,
00038   vendor_id,
00039   max_compute_units,
00040   max_work_item_dimensions,
00041   max_work_item_sizes,
00042   max_work_group_size,
00043   preferred_vector_width_char,
00044   preferred_vector_width_short,
00045   preferred_vector_width_int,
00046   preferred_vector_width_long_long,
00047   preferred_vector_width_float,
00048   preferred_vector_width_double,
00049   preferred_vector_width_half,
00050   native_vector_witdth_char,
00051   native_vector_witdth_short,
00052   native_vector_witdth_int,
00053   native_vector_witdth_long_long,
00054   native_vector_witdth_float,
00055   native_vector_witdth_double,
00056   native_vector_witdth_half,
00057   max_clock_frequency,
00058   address_bits,
00059   max_mem_alloc_size,
00060   image_support,
00061   max_read_image_args,
00062   max_write_image_args,
00063   image2d_max_height,
```

```
00064   image2d_max_width,
00065   image3d_max_height,
00066   image3d_max_widht,
00067   image3d_mas_depth,
00068   image_max_buffer_size,
00069   image_max_array_size,
00070   max_samplers,
00071   max_parameter_size,
00072   mem_base_addr_align,
00073   single_fp_config,
00074   double_fp_config,
00075   global_mem_cache_type,
00076   global_mem_cache_line_size,
00077   global_mem_cache_size,
00078   global_mem_size,
00079   max_constant_buffer_size,
00080   max_constant_args,
00081   local_mem_type,
00082   local_mem_size,
00083   error_correction_support,
00084   host_unified_memory,
00085   profiling_timer_resolution,
00086   endian_little,
00087   is_available,
00088   is_compiler_available,
00089   is_linker_available,
00090   execution_capabilities,
00091   queue_properties,
00092   built_in_kernels,
00093   platform,
00094   name,
00095   vendor,
00096   driver_version,
00097   profile,
00098   device_version,
00099   opencl_version,
00100   extensions,
00101   printf_buffer_size,
00102   preferred_interop_user_sync,
00103   parent_device,
00104   partition_max_sub_devices,
00105   partition_properties,
00106   partition_affinity_domain,
00107   partition_type,
00108   reference_count
00109 };
```

8.3.3.3 enum cl::sycl::info::device_affinity_domain : int [strong]

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

Enumerator

unsupported

numa

L4_cache

L3_cache

L2_cache

next_partitionable

Definition at line 119 of file device.hpp.

```
00119                                   : int {
00120   unsupported,
00121   numa,
00122   L4_cache,
00123   L3_cache,
00124   L2_cache,
00125   next_partitionable
00126 };
```

8.3.3.4 enum cl::sycl::info::device_execution_capabilities : unsigned int [strong]

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

Enumerator

exec_kernel

exec_native_kernel

Definition at line 160 of file device.hpp.

```
00160                                          : unsigned int {
00161   exec_kernel,
00162   exec_native_kernel
00163 };
```

8.3.3.5 enum cl::sycl::info::device_partition_property : int [strong]

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

Enumerator

unsupported

partition_equally

partition_by_counts

partition_by_affinity_domain

partition_affinity_domain_next_partitionable

Definition at line 111 of file device.hpp.

```
00111                                       : int {
00112   unsupported,
00113   partition_equally,
00114   partition_by_counts,
00115   partition_by_affinity_domain,
00116   partition_affinity_domain_next_partitionable
00117 };
```

8.3.3.6 enum cl::sycl::info::device_partition_type : int [strong]

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

Enumerator

no_partition

numa

L4_cache

L3_cache

L2_cache

L1_cache

Definition at line 128 of file device.hpp.

```
00128                                    : int {
00129   no_partition,
00130   numa,
00131   L4_cache,
00132   L3_cache,
00133   L2_cache,
00134   L1_cache
00135 };
```

8.3.3.7 enum cl::sycl::info::device_type : unsigned int [strong]

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

Enumerator

cpu

gpu

accelerator

custom

defaults

host

all

Definition at line 28 of file platform.hpp.

```
00028                         : unsigned int {
00029   cpu,
00030   gpu,
00031   accelerator,
00032   custom,
00033   defaults,
00034   host,
00035   all
00036 };
```

8.3.3.8 enum cl::sycl::info::fp_config : int [strong]

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

Enumerator

denorm

inf_nan

round_to_nearest

round_to_zero

round_to_inf

fma

correctly_rounded_divide_sqrt

soft_float

Definition at line 143 of file device.hpp.

```
00143                       : int {
00144   denorm,
00145   inf_nan,
00146   round_to_nearest,
00147   round_to_zero,
00148   round_to_inf,
00149   fma,
00150   correctly_rounded_divide_sqrt,
00151   soft_float
00152 };
```

8.3.3.9 enum cl::sycl::info::global_mem_cache_type : int [strong]

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

Enumerator

none

read_only

write_only

Definition at line 154 of file device.hpp.

```
00154                                  : int {
00155   none,
00156   read_only,
00157   write_only
00158 };
```

8.3.3.10 enum cl::sycl::info::local_mem_type : int [strong]

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

Enumerator

none

local

global

Definition at line 137 of file device.hpp.

```
00137                              : int {
00138   none,
00139   local,
00140   global
00141 };
```

8.3.3.11 enum cl::sycl::info::platform : unsigned int [strong]

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

Platform information descriptors.

A SYCL platform can be queried for all of the following information using the get_info function. All SYCL contexts have valid devices for them, including the SYCL host device.

Enumerator

profile Returns the profile name (as a string_class) supported by the im- plementation. Can be either FULL PROFILE or EMBEDDED PROFILE.

version Returns the OpenCL software driver version string in the form major number.minor number (as a string_class)

name Returns the name of the platform (as a string_class)

vendor Returns the string provided by the platform vendor (as a string_class)

extensions Returns a space-separated list of extension names supported by the platform (as a string_class)

Definition at line 45 of file platform.hpp.

```
00045                     : unsigned int {
00046   /** Returns the profile name (as a string_class) supported by the im-
00047       plementation.
00048
00049       Can be either FULL PROFILE or EMBEDDED PROFILE.
00050   */
00051   profile,
00052   /** Returns the OpenCL software driver version string in the form major
00053       number.minor number (as a string_class)
00054   */
00055   version,
00056   /** Returns the name of the platform (as a string_class)
00057   */
00058   name,
00059   /** Returns the string provided by the platform vendor (as a string_class)
00060   */
00061   vendor,
00062   /** Returns a space-separated list of extension names supported by the
00063       platform (as a string_class)
00064   */
00065   extensions
00066 };
```

8.3.3.12 enum cl::sycl::info::queue : int [strong]

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

Queue information descriptors.

From specification C.4

Todo unsigned int?

Todo To be implemented

Enumerator

context

device

reference_count

properties

Definition at line 45 of file queue.hpp.

```
00045                   : int {
00046   context,
00047   device,
00048   reference_count,
00049   properties
00050 };
```

8.4 Helpers to do array and tuple conversion

Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std::index_sequence< Is...>)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (Tuple t)
- template<typename Value , size_t... Is>
  static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (Value e, std::index_sequence< Is...>)

  Construct a tuple from a value.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

8.4.1 Detailed Description

8.4.2 Class Documentation

8.4.2.1 struct cl::sycl::detail::expand_to_vector

template<typename V, typename Tuple, bool expansion = false>struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization.

Definition at line 65 of file array_tuple_helpers.hpp.

Static Public Member Functions

- static auto expand (Tuple t)

8.4.2.2 struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

template<typename V, typename Tuple>struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

Specialization in the case we ask for expansion.

Definition at line 77 of file array_tuple_helpers.hpp.

Static Public Member Functions

- template<typename Value , size_t... Is>
  static auto fill_tuple (Value e, std::index_sequence< Is...>)

  Construct a tuple from a value.
- static auto expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.

8.4.3 Function Documentation

8.4.3.1 template<typename V , typename Tuple , bool expansion = false> static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (Tuple *t*) [inline], [static]

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

Definition at line 70 of file array_tuple_helpers.hpp.

Referenced by cl::sycl::detail::expand().

```
00070 { return t; }
```

Here is the caller graph for this function:

8.4.3.2 template<typename V , typename Tuple > static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (Tuple *t*) [inline], [static]

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

We expand the 1-element tuple by replicating into a tuple with the size of the vector.

Definition at line 109 of file array_tuple_helpers.hpp.

```
00109                                {
00110     return fill_tuple(std::get<0>(t),
00111                       std::make_index_sequence<V::dimension>{});
00112   }
```

8.4.3.3 template<typename V , typename Tuple > auto cl::sycl::detail::expand (Tuple *t*)

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

Create the array data of V from a tuple of initializer.

If there is only 1 initializer, this is a scalar initialization of a vector and the value is expanded to all the vector elements first.

Definition at line 123 of file array_tuple_helpers.hpp.

References cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand().

```
00123                     {
00124   return tuple_to_array<V>(expand_to_vector<V,
00125                             decltype(t),
00126                             /* Only ask the expansion to all vector
00127                                element if there only a scalar
00128                                initializer */
00129                             std::tuple_size<Tuple>::value == 1>{}.expand(t));
00130 }
```

Here is the call graph for this function:

8.4.3.4 template<typename V , typename Tuple > template<typename Value , size_t... Is> static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (Value *e*, std::index_sequence< Is...>) [inline], [static]

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

Construct a tuple from a value.

Parameters

value	is used to initialize each tuple element
size	is the number of elements of the tuple to be generated

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 93 of file array_tuple_helpers.hpp.

```
00093                                                          {
00094     /* The effect is like a static for-loop with Is counting from 0 to
00095        dimension-1 and thus replicating the pattern to have
00096        make_tuple( (0, e), (1, e), ... (n - 1, e) )
00097
00098        Since the "," operator is just here to throw away the Is value
00099        (which is needed for the pack expansion...), at the end this is
00100       equivalent to:
00101       make_tuple( e, e, ..., e )
00102    */
00103    return std::make_tuple(((void)Is, e)...);
00104  }
```

8.4.3.5 template<typename V , typename Tuple > auto cl::sycl::detail::tuple_to_array (Tuple *t*)

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

Construct an array from initializer elements provided as a tuple.

Definition at line 53 of file array_tuple_helpers.hpp.

```
00053                              {
00054   /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
00055      so that tuple_to_array_iterate can statically iterate on it */
00056   return tuple_to_array_iterate<V>(t,
00057                                    std::make_index_sequence<std::tuple_size<Tuple>::value>{});
00058 }
```

8.4.3.6 template<typename V , typename Tuple , size_t... Is> std::array<typename V::element_type, V::dimension> cl::sycl::detail::tuple_to_array_iterate (Tuple *t*, std::index_sequence< Is...>)

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

Helper to construct an array from initializer elements provided as a tuple.

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 37 of file array_tuple_helpers.hpp.

```
00037                                                  {
00038   /* The effect is like a static for-loop with Is counting from 0 to
00039      dimension-1 and thus constructing a uniform initialization { }
00040      construction from each tuple element:
00041      { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }
00042
00043      The static cast is here to avoid the warning when there is a loss
00044      of precision, for example when initializing an int from a float.
00045   */
00046   return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
00047 }
```

8.5 Debugging and tracing support

Classes

- struct cl::sycl::detail::debug< T >

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Functions

- void cl::sycl::detail::unimplemented ()

  Display an "unimplemented" message.

8.5.1 Detailed Description

8.5.2 Class Documentation

8.5.2.1 struct cl::sycl::detail::debug

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

Class used to trace the construction, copy-construction, move-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 62 of file debug.hpp.

Public Member Functions

- debug ()

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

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

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

  Trace the destruction with the compiler-dependent mangled named.

8.5.2.1.1 Constructor & Destructor Documentation

8.5.2.1.1.1 template<typename T> cl::sycl::detail::debug< T >::debug ( ) [inline]

Trace the construction with the compiler-dependent mangled named.

Definition at line 65 of file debug.hpp.

```
00065           {
00066     TRISYCL_DUMP("Constructor of " << typeid(*this).name()
00067                  << " " << (void*) this);
00068   }
```

8.5.2.1.1.2 template<typename T> cl::sycl::detail::debug< T >::debug (debug< T > const &) [inline]

Trace the copy construction with the compiler-dependent mangled named.

Definition at line 73 of file debug.hpp.

```
00073                         {
00074     TRISYCL_DUMP("Copy of " << typeid(*this).name() << " " << (void*) this);
00075   }
```

8.5.2.1.1.3 template<typename T> cl::sycl::detail::debug< T >::debug (debug< T > &&) [inline]

Trace the move construction with the compiler-dependent mangled named.

Definition at line 80 of file debug.hpp.

```
00080                   {
00081     TRISYCL_DUMP("Move of " << typeid(*this).name() << " " << (void*) this);
00082   }
```

8.5.2.1.1.4 template<typename T> cl::sycl::detail::debug< T >::~debug ( ) [inline]

Trace the destruction with the compiler-dependent mangled named.

Definition at line 86 of file debug.hpp.

```
00086           {
00087     TRISYCL_DUMP("~ Destructor of " << typeid(*this).name()
00088                  << " " << (void*) this);
00089   }
```

8.5.2.2 struct cl::sycl::detail::display_vector

template<typename T>struct cl::sycl::detail::display_vector< T >

Class used to display a vector-like type of classes that inherit from it.

Parameters

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

Calling the display() method dump the values on std::cout

Definition at line 102 of file debug.hpp.

Public Member Functions

- void display () const

  To debug and test.

8.5.2.2.1 Member Function Documentation

8.5.2.2.1.1 template<typename T> void cl::sycl::detail::display_vector< T >::display ( ) const [inline]

To debug and test.

Definition at line 105 of file debug.hpp.

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

```
00105                         {
00106 #ifdef TRISYCL_DEBUG
00107     std::cout << typeid(T).name() << ":";
```

```
00108 #endif
00109     // Get a pointer to the real object
00110     for (auto e : *static_cast<const T *>(this))
00111       std::cout << " " << e;
00112     std::cout << std::endl;
00113   }
```

Here is the caller graph for this function:

8.5.3 Function Documentation

8.5.3.1 void cl::sycl::detail::unimplemented ( ) [inline]

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

Display an "unimplemented" message.

Can be changed to call assert(0) or whatever.

Definition at line 25 of file unimplemented.hpp.

Referenced by cl::sycl::nd_item< dims >::barrier(), cl::sycl::context::context(), cl::sycl::device::create_sub_devices(), cl::sycl::device::device(), cl::sycl::platform::get(), cl::sycl::context::get(), cl::sycl::queue::get(), cl::sycl::device::get(), cl::sycl::queue::get_context(), cl::sycl::queue::get_device(), cl::sycl::platform::get_devices(), cl::sycl::context::get_devices(), cl::sycl::device::get_devices(), cl::sycl::platform::get_info(), cl::sycl::context::get_info(), cl::sycl::queue::get_info(), cl::sycl::device::get_info(), cl::sycl::device::get_platform(), cl::sycl::platform::get_platforms(), cl::sycl::platform::has_extension(), cl::sycl::device::has_extension(), cl::sycl::device::is_accelerator(), cl::sycl::device::is_cpu(), cl::sycl::device::is_gpu(), cl::sycl::queue::is_host(), cl::sycl::device::is_host(), cl::sycl::default_selector::operator()(), cl::sycl::gpu_selector::operator()(), cl::sycl::cpu_selector::operator()(), cl::sycl::host_selector::operator()(), cl::sycl::handler::parallel_for(), cl::sycl::platform::platform(), cl::sycl::queue::queue(), cl::sycl::device_selector::select_device(), cl::sycl::handler::set_arg(), cl::sycl::handler::single_task(), cl::sycl::queue::submit(), cl::sycl::queue::throw_asynchronous(), cl::sycl::queue::wait(), and cl::sycl::queue::wait_and_throw().

```
00025                               {
00026   std::cerr << "Error: using a non implemented feature!!!" << std::endl
00027             << "Please contribute to the open source implementation. :-)"
00028             << std::endl;
00029 }
```

Here is the caller graph for this function:

8.6 Some helpers for the implementation

Classes

- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

  Helper macro to declare a vector operation with the given side-effect operator.

Functions

- template<typename Range , typename Id >
  size_t cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.

8.6.1 Detailed Description

8.6.2 Class Documentation

8.6.2.1 struct cl::sycl::detail::small_array

template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false>struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

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

Unfortunately, even if std::array is an aggregate class allowing native list initialization, it is no longer an aggregate if we derive from an aggregate. Thus we have to redeclare the constructors.

Parameters

BasicType	is the type element, such as int
Dims	is the dimension number, typically between 1 and 3
FinalType	is the final type, such as range<> or id<>, so that boost::operator can return the right type
EnableArgsConstructor	adds a constructors from Dims variadic elements when true. It is false by default.

std::array<> provides the collection concept, with .size(), == and != too.

Definition at line 65 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:

Collaboration diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:

Public Types

- using element_type = BasicType

Public Member Functions

- template<typename SourceType >
  small_array (const SourceType src[Dims])

  A constructor from another array.
- template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor>
  small_array (const small_array< SourceBasicType, SourceFinalType, Dims, SourceEnableArgsConstructor > &src)

  A constructor from another small_array of the same size.
- template<typename... Types, bool Depend = true, typename = typename std::enable_if<EnableArgsConstructor && Depend>::type>
  small_array (const Types &...args)

  Initialize the array from a list of elements.
- template<typename SourceBasicType >
  small_array (const std::array< SourceBasicType, Dims > &src)

  Construct a small_array from a std::array.
- small_array ()=default

  Keep the synthesized constructors.
- auto get (std::size_t index) const

  Return the element of the array.
- operator FinalType ()

  Add + like operations on the id<> and others.

Static Public Attributes

- static const auto dimensionality = Dims
- static const size_t dimension = Dims

8.6.2.1.1 Member Typedef Documentation

8.6.2.1.1.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> using cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::element_type = BasicType

Definition at line 85 of file small_array.hpp.

8.6.2.1.2 Constructor & Destructor Documentation

8.6.2.1.2.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceType > cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array (const SourceType *src[Dims]*) [inline]

A constructor from another array.

Make it explicit to avoid spurious range<> constructions from int * for example

Definition at line 94 of file small_array.hpp.

```
00094                                       {
00095     // (*this)[0] is the first element of the underlying array
00096     std::copy_n(src, Dims, &(*this)[0]);
00097   }
```

8.6.2.1.2.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor> cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array (const small_array< SourceBasicType, SourceFinalType, Dims, SourceEnableArgsConstructor > & *src*) [inline]

A constructor from another small_array of the same size.

Definition at line 104 of file small_array.hpp.

```
00107                                                {
00108     std::copy_n(&src[0], Dims, &(*this)[0]);
00109   }
```

8.6.2.1.2.3 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename... Types, bool Depend = true, typename = typename std::enable_if<EnableArgsConstructor && Depend>::type> cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array (const Types &... *args*) [inline]

Initialize the array from a list of elements.

Strangely, even when using the array constructors, the initialization of the aggregate is not available. So recreate an equivalent here.

Since there are inherited types that defines some constructors with some conflicts, make it optional here, according to EnableArgsConstructor template parameter.

Definition at line 127 of file small_array.hpp.

```
00128     : std::array<BasicType, Dims> {
00129     // Allow a loss of precision in initialization with the static_cast
00130     { static_cast<BasicType>(args)... }
00131   }
```

8.6.2.1.2.4 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> template<typename SourceBasicType > cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array (const std::array< SourceBasicType, Dims > & *src*) [inline]

Construct a small_array from a std::array.

Definition at line 141 of file small_array.hpp.

```
00142   : std::array<BasicType, Dims>(src) {}
```

8.6.2.1.2.5 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_array ( ) [default]

Keep the synthesized constructors.

8.6.2.1.3 Member Function Documentation

8.6.2.1.3.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::get (std::size_t *index*) const [inline]

Return the element of the array.

Definition at line 152 of file small_array.hpp.

```
00152                                    {
00153     return (*this)[index];
00154   }
```

8.6.2.1.3.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::operator FinalType ( ) [inline]

Add + like operations on the id<> and others.

Add - like operations on the id<> and others Add * like operations on the id<> and others Add / like operations on the id<> and others Add % like operations on the id<> and others Add << like operations on the id<> and others Add >> like operations on the id<> and others Add & like operations on the id<> and others Add ^ like operations on the id<> and others Add | like operations on the id<> and others Since the boost::operator work on the small_array, add an implicit conversion to produce the expected type

Definition at line 191 of file small_array.hpp.

```
00191                           {
00192     return *static_cast<FinalType *>(this);
00193   }
```

8.6.2.1.4 Member Data Documentation

8.6.2.1.4.1 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> const size_t cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimension = Dims [static]

Definition at line 84 of file small_array.hpp.

8.6.2.1.4.2 template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false> const auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimensionality = Dims [static]

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

Definition at line 80 of file small_array.hpp.

8.6.2.2 struct cl::sycl::detail::small_array_123

template<typename BasicType, typename FinalType, std::size_t Dims>struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

A small array of 1, 2 or 3 elements with the implicit constructors.

Definition at line 200 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:

Additional Inherited Members

8.6.2.3 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

template<typename BasicType, typename FinalType>struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1.

Definition at line 212 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:

Public Member Functions

- small_array_123 (BasicType x)

  A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.
- small_array_123 ()=default

  Keep other constructors.
- operator BasicType () const

  Conversion so that an for example an id<1> can basically be used like an integer.

Additional Inherited Members

8.6.2.3.1 Constructor & Destructor Documentation

8.6.2.3.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 (BasicType *x*) [inline]

A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.

Definition at line 216 of file small_array.hpp.

```
00216                                  {
00217     (*this)[0] = x;
00218   }
```

8.6.2.3.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 ( ) [default]

Keep other constructors.

8.6.2.3.2 Member Function Documentation

8.6.2.3.2.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::operator BasicType ( ) const [inline]

Conversion so that an for example an id<1> can basically be used like an integer.

Definition at line 228 of file small_array.hpp.

```
00228                                  {
00229     return (*this)[0];
00230   }
```

8.6.2.4 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

template<typename BasicType, typename FinalType>struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

Definition at line 235 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:

Public Member Functions

- small_array_123 (BasicType x, BasicType y)

  A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.6.2.4.1 Constructor & Destructor Documentation

8.6.2.4.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (BasicType *x*, BasicType *y*) [inline]

A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.

Definition at line 239 of file small_array.hpp.

```
00239                                                {
00240     (*this)[0] = x;
00241     (*this)[1] = y;
00242   }
```

8.6.2.4.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 ( ) [default]

Keep other constructors.

8.6.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

template<typename BasicType, typename FinalType>struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Definition at line 253 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:

Public Member Functions

- small_array_123 (BasicType x, BasicType y, BasicType z)

  A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.6.2.5.1 Constructor & Destructor Documentation

8.6.2.5.1.1 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (BasicType *x*, BasicType *y*, BasicType *z*) [inline]

A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.

Definition at line 257 of file small_array.hpp.

```
00257                                                          {
00258     (*this)[0] = x;
00259     (*this)[1] = y;
00260     (*this)[2] = z;
00261   }
```

8.6.2.5.1.2 template<typename BasicType , typename FinalType > cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 ( ) [default]

Keep other constructors.

8.6.3 Macro Definition Documentation

8.6.3.1 #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(*op*)

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

Value:

```
FinalType operator op(const FinalType &rhs) {         \
    for (std::size_t i = 0; i != Dims; ++i)             \
      (*this)[i] op rhs[i];                             \
    return *this;                                       \
  }
```

Helper macro to declare a vector operation with the given side-effect operator.

Definition at line 33 of file small_array.hpp.

8.6.4 Function Documentation

8.6.4.1 template<typename Range , typename Id > size_t cl::sycl::detail::linear_id (Range *range,* Id *id,* Id *offset* = { })

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

Compute a linearized array access used in the OpenCL 2 world.

Typically for the get_global_linear_id() and get_local_linear_id() functions.

Definition at line 28 of file linear_id.hpp.

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), cl::sycl::nd_item< dims >::get_group_linear_id(), cl::sycl::group< dims >::get_linear(), cl::sycl::item< dims >::get_linear_id(), and cl::sycl::nd_item< dims >::get_local_linear_id().

```
00028                                                 {}) {
00029   auto dims = std::distance(std::begin(range), std::end(range));
00030
00031   size_t linear_id = 0;
00032   /* A good compiler should unroll this and do partial evaluation to
00033      remove the first multiplication by 0 of this Horner evaluation and
00034      remove the 0 offset evaluation */
00035     for (int i = dims - 1; i >= 0; --i)
00036       linear_id = linear_id*range[i] + id[i] - offset[i];
00037
00038     return linear_id;
00039   }
```

Here is the caller graph for this function:

8.7 Error handling

Namespaces

- cl::sycl::trisycl

Classes

- 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::exception

  Encapsulate a SYCL error information. More...

Typedefs

- using cl::sycl::async_handler = function_class< int >

8.7.1 Detailed Description

8.7.2 Class Documentation

8.7.2.1 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 32 of file error_handler.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.

8.7.2.1.1 Member Function Documentation

8.7.2.1.1.1 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.

8.7.2.1.2 Member Data Documentation

8.7.2.1.2.1 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 43 of file error_handler.hpp.

8.7.2.2 struct cl::sycl::trisycl::default_error_handler

Definition at line 49 of file error_handler.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

8.7.2.2.1 Member Function Documentation

8.7.2.2.1.1 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 51 of file error_handler.hpp.

```
00051                                                {
00052     }
```

8.7.2.3 struct cl::sycl::exception

Encapsulate a SYCL error information.

Definition at line 29 of file exception.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, typename Allocator >
  buffer< T, dimensions, Allocator > * get_buffer ()
  Get the buffer that caused the error.
- template<std::size_t dimensions>
  image< dimensions > * get_image ()
  Get the image that caused the error.

8.7.2.3.1 Member Function Documentation

8.7.2.3.1.1 template<typename T , int dimensions, typename Allocator > buffer<T, dimensions, Allocator>* 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

Todo How to get the real buffer type? Update: has been removed in new specification

Definition at line 74 of file exception.hpp.

```
00074                                                {
00075     assert(0); }
```

8.7.2.3.1.2 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 37 of file exception.hpp.

```
00037 { assert(0); }
```

8.7.2.3.1.3 template<std::size_t 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 87 of file exception.hpp.

```
00087 { assert(0); }
```

8.7.2.3.1.4 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 58 of file exception.hpp.

```
00058 { assert(0); }
```

8.7.2.3.1.5 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 49 of file exception.hpp.

```
00049 { assert(0); }
```

8.7.3 Typedef Documentation

8.7.3.1 using cl::sycl::async_handler = typedef function_class<int>

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

Definition at line 24 of file exception.hpp.

8.8 Expressing parallelism through kernels

Collaboration diagram for Expressing parallelism through kernels:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Classes

- struct cl::sycl::group< dims >

  A group index used in a parallel_for_workitem to specify a work_group. More...
- class cl::sycl::id< dims >

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

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- struct cl::sycl::nd_item< dims >

  A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges. 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::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

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

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

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

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

Implementation of parallel_for with a range<> and an offset.

- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (group< Dimensions > g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

8.8.1 Detailed Description

8.8.2 Class Documentation

8.8.2.1 struct cl::sycl::group

template<std::size_t dims = 1>struct cl::sycl::group< dims >

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

Definition at line 29 of file group.hpp.

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

Public Member Functions

- group (const nd_range< dims > &ndr)

  Create a group from an nd_range<> with a 0 id<>
- group (const id< dims > &i, const nd_range< dims > &ndr)

  Create a group from an id and a nd_range<>
- group ()=default

  To be able to copy and assign group, use default constructors too.
- id< dims > get () const

  Return an id representing the index of the group within the nd_range for every dimension.
- size_t get (int dimension) const

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

  Return the index of the group in the given dimension within the nd_range<>
- range< dims > get_group_range () const

  Return a range<> representing the dimensions of the current group.
- size_t get_group_range (int dimension) const

Return element dimension from the con stituent group range.

- range< dims > get_global_range () const

  Get the local range for this work_group.

- size_t get_global_range (int dimension) const

  Return element dimension from the constituent global range.

- range< dims > get_local_range () const

  Get the local range for this work_group.

- size_t get_local_range (int dimension) const

  Return element dimension from the constituent local range.

- id< dims > get_offset () const

  Get the offset of the NDRange.

- size_t get_offset (int dimension) const

  Get the offset of the NDRange.

- nd_range< dims > get_nd_range () const
- size_t get_linear () const

  Get a linearized version of the group ID.

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- id< dims > group_id

  The coordinate of the group item.

- nd_range< dims > ndr

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

8.8.2.1.1 Constructor & Destructor Documentation

8.8.2.1.1.1 template<std::size_t dims = 1> cl::sycl::group< dims >::group (const nd_range< dims > & *ndr*) [inline]

Create a group from an nd_range<> with a 0 id<>

Todo This should be private since it is only used by the triSYCL implementation

Definition at line 49 of file group.hpp.

```
00049 : ndr { ndr } {}
```

8.8.2.1.1.2 template<std::size_t dims = 1> cl::sycl::group< dims >::group (const id< dims > & *i*, const nd_range< dims > & *ndr*) [inline]

Create a group from an id and a nd_range<>

Todo This should be private somehow, but it is used by the validation infrastructure

Definition at line 57 of file group.hpp.

```
00057                                                  :
00058     group_id { i }, ndr { ndr } {}
```

8.8.2.1.1.3 template<std::size_t dims = 1> cl::sycl::group< dims >::group ( ) [default]

To be able to copy and assign group, use default constructors too.

Todo Make most of them protected, reserved to implementation

8.8.2.1.2 Member Function Documentation

8.8.2.1.2.1 template<std::size_t dims = 1> id<dims> cl::sycl::group< dims >::get ( ) const [inline]

Return an id representing the index of the group within the nd_range for every dimension.

Definition at line 71 of file group.hpp.

References cl::sycl::group< dims >::group_id.

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

```
00071 { return group_id; }
```

Here is the caller graph for this function:

8.8.2.1.2.2 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get (int *dimension*) const [inline]

Return the index of the group in the given dimension.

Definition at line 75 of file group.hpp.

```
00075 { return get()[dimension]; }
```

8.8.2.1.2.3 template<std::size_t dims = 1> range<dims> cl::sycl::group< dims >::get_global_range ( ) const [inline]

Get the local range for this work_group.

Definition at line 110 of file group.hpp.

References cl::sycl::group< dims >::get_nd_range().

Referenced by cl::sycl::group< dims >::get_global_range().

```
00110 { return get_nd_range().get_global(); }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.1.2.4 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get_global_range (int *dimension*) const [inline]

Return element dimension from the constituent global range.

Definition at line 114 of file group.hpp.

References cl::sycl::group< dims >::get_global_range().

```
00114                                              {
00115     return get_global_range()[dimension];
00116   }
```

Here is the call graph for this function:

8.8.2.1.2.5 template<std::size_t dims = 1> range<dims> cl::sycl::group< dims >::get_group_range ( ) const [inline]

Return a range<> representing the dimensions of the current group.

This local range may have been provided by the programmer, or chosen by the runtime.

Todo Fix this comment and the specification

Definition at line 98 of file group.hpp.

References cl::sycl::group< dims >::get_nd_range().

Referenced by cl::sycl::group< dims >::get_group_range(), and cl::sycl::group< dims >::get_linear().

```
00098                                       {
00099     return get_nd_range().get_group();
00100   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.1.2.6 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get_group_range (int *dimension*) const [inline]

Return element dimension from the con stituent group range.

Definition at line 104 of file group.hpp.

References cl::sycl::group< dims >::get_group_range().

```
00104                                                 {
00105     return get_group_range()[dimension];
00106   }
```

Here is the call graph for this function:

8.8.2.1.2.7 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get_linear ( ) const [inline]

Get a linearized version of the group ID.

Definition at line 156 of file group.hpp.

References cl::sycl::group< dims >::get_group_range(), and cl::sycl::detail::linear_id().

```
00156                                {
00157     return detail::linear_id(get_group_range(), get());
00158   }
```

Here is the call graph for this function:

8.8.2.1.2.8 template<std::size_t dims = 1> range<dims> cl::sycl::group< dims >::get_local_range ( ) const [inline]

Get the local range for this work_group.

Todo Add to the specification

Definition at line 123 of file group.hpp.

References cl::sycl::group< dims >::get_nd_range().

Referenced by cl::sycl::group< dims >::get_local_range(), and cl::sycl::detail::parallel_for_workitem().

```
00123 { return get_nd_range().get_local(); }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.1.2.9 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get_local_range (int *dimension*) const [inline]

Return element dimension from the constituent local range.

Todo Add to the specification

Definition at line 130 of file group.hpp.

References cl::sycl::group< dims >::get_local_range().

```
00130                                                {
00131     return get_local_range()[dimension];
00132   }
```

Here is the call graph for this function:

8.8.2.1.2.10 template<std::size_t dims = 1> nd_range<dims> cl::sycl::group< dims >::get_nd_range ( ) const [inline]

Todo Also provide this access to the current nd_range

Definition at line 150 of file group.hpp.

References cl::sycl::group< dims >::ndr.

Referenced by cl::sycl::group< dims >::get_global_range(), cl::sycl::group< dims >::get_group_range(), cl::sycl::group< dims >::get_local_range(), cl::sycl::group< dims >::get_offset(), and cl::sycl::detail::parallel_for_workitem().

```
00150 { return ndr; }
```

Here is the caller graph for this function:

8.8.2.1.2.11 template<std::size_t dims = 1> id<dims> cl::sycl::group< dims >::get_offset ( ) const [inline]

Get the offset of the NDRange.

Todo Add to the specification

Definition at line 139 of file group.hpp.

References cl::sycl::group< dims >::get_nd_range().

```
00139 { return get_nd_range().get_offset(); }
```

Here is the call graph for this function:

8.8.2.1.2.12 template<std::size_t dims = 1> size_t cl::sycl::group< dims >::get_offset (int *dimension*) const [inline]

Get the offset of the NDRange.

Todo Add to the specification

Definition at line 146 of file group.hpp.

References cl::sycl::group< dims >::get_offset().

Referenced by cl::sycl::group< dims >::get_offset().

```
00146 { return get_offset()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.1.2.13 template<std::size_t dims = 1> auto& cl::sycl::group< dims >::operator[ ] (int *dimension*) [inline]

Return the index of the group in the given dimension within the nd_range<>

Todo In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Definition at line 85 of file group.hpp.

```
00085                                  {
00086     return group_id[dimension];
00087   }
```

8.8.2.1.3 Member Data Documentation

8.8.2.1.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::group< dims >::dimensionality = dims [static]

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

Definition at line 32 of file group.hpp.

8.8.2.1.3.2 template<std::size_t dims = 1> id<dims> cl::sycl::group< dims >::group_id [private]

The coordinate of the group item.

Definition at line 37 of file group.hpp.

Referenced by cl::sycl::group< dims >::get().

8.8.2.1.3.3 template<std::size_t dims = 1> nd_range<dims> cl::sycl::group< dims >::ndr [private]

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

Definition at line 40 of file group.hpp.

Referenced by cl::sycl::group< dims >::get_nd_range().

8.8.2.2 class cl::sycl::id

template<std::size_t dims = 1>class cl::sycl::id< dims >

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

Definition at line 31 of file id.hpp.

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

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

Public Member Functions

- id (const range< dims > &range_size)
  Construct an id from the dimensions of a range.

Additional Inherited Members

8.8.2.2.1 Constructor & Destructor Documentation

8.8.2.2.1.1 template<std::size_t dims = 1> cl::sycl::id< dims >::id (const range< dims > & *range_size*) [inline]

Construct an id from the dimensions of a range.

Use the fact we have a constructor of a small_array from a another kind of small_array

Definition at line 42 of file id.hpp.

```
00046     : detail::small_array_123<std::size_t, id<dims>, dims> { range_size } {}
```

8.8.2.3 class cl::sycl::item

template<std::size_t dims = 1>class cl::sycl::item< dims >

A SYCL item stores information on a work-item with some more context such as the definition range and offset.

Definition at line 21 of file id.hpp.

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

Public Member Functions

- item (range< dims > global_size, id< dims > global_index, id< dims > offset={})

  Create an item from a local size and an optional offset.
- item ()=default

  To be able to copy and assign item, use default constructors too.
- id< dims > get () const

  Return the constituent local or global id<> representing the work-item's position in the iteration space.
- size_t get (int dimension) const

  Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.
- auto & operator[ ] (int dimension)

  Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.
- range< dims > get_range () const

  Returns a range<> representing the dimensions of the range of possible values of the item.
- id< dims > get_offset () const

  Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.
- size_t get_linear_id () const

  Return the linearized ID in the item's range.
- void set (id< dims > Index)

  For the implementation, need to set the global index.
- void display () const

  Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- range< dims > global_range
- id< dims > global_index
- id< dims > offset

8.8.2.3.1 Constructor & Destructor Documentation

8.8.2.3.1.1 template<std::size_t dims = 1> cl::sycl::item< dims >::item (range< dims > *global_size,* id< dims > *global_index,* id< dims > *offset* = {}) [inline]

Create an item from a local size and an optional offset.

This constructor is used by the triSYCL implementation and the non-regression testing.

Definition at line 50 of file item.hpp.

```
00052                          {}) :
00053     global_range { global_size },
00054     global_index { global_index },
00055     offset { offset }
00056   {}
```

8.8.2.3.1.2 template<std::size_t dims = 1> cl::sycl::item< dims >::item ( ) [default]

To be able to copy and assign item, use default constructors too.

Todo Make most of them protected, reserved to implementation

8.8.2.3.2 Member Function Documentation

8.8.2.3.2.1 template<std::size_t dims = 1> void cl::sycl::item< dims >::display ( ) const [inline]

Display the value for debugging and validation purpose.

Definition at line 117 of file item.hpp.

References cl::sycl::detail::display_vector< range< dims > >::display(), and cl::sycl::detail::display_vector< id< dims > >::display().

```
00117                          {
00118     global_range.display();
00119     global_index.display();
00120     offset.display();
00121   }
```

Here is the call graph for this function:

8.8.2.3.2.2 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::get ( ) const [inline]

Return the constituent local or global id<> representing the work-item's position in the iteration space.

Definition at line 69 of file item.hpp.

References cl::sycl::item< dims >::global_index.

Referenced by cl::sycl::detail::accessor< DataType, Dimensions, AccessMode, Target >::operator[]().

```
00069 { return global_index; }
```

Here is the caller graph for this function:

8.8.2.3.2.3 template<std::size_t dims = 1> size_t cl::sycl::item< dims >::get (int *dimension*) const [inline]

Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.

Definition at line 75 of file item.hpp.

```
00075 { return get()[dimension]; }
```

8.8.2.3.2.4 template<std::size_t dims = 1> size_t cl::sycl::item< dims >::get_linear_id ( ) const [inline]

Return the linearized ID in the item's range.

Computed as the flatted ID after the offset is subtracted.

Definition at line 104 of file item.hpp.

References cl::sycl::item< dims >::get_offset(), cl::sycl::item< dims >::get_range(), and cl::sycl::detail::linear_id().

```
00104                                {
00105     return detail::linear_id(get_range(), get(),
      get_offset());
00106   }
```

Here is the call graph for this function:

8.8.2.3.2.5 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::get_offset ( ) const [inline]

Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.

For an item representing a local range of where no offset was passed this will always return an id of all 0 values.

Definition at line 97 of file item.hpp.

References cl::sycl::item< dims >::offset.

Referenced by cl::sycl::item< dims >::get_linear_id().

```
00097 { return offset; }
```

Here is the caller graph for this function:

8.8.2.3.2.6 template<std::size_t dims = 1> range<dims> cl::sycl::item< dims >::get_range ( ) const [inline]

Returns a range<> representing the dimensions of the range of possible values of the item.

Definition at line 87 of file item.hpp.

References cl::sycl::item< dims >::global_range.

Referenced by cl::sycl::item< dims >::get_linear_id().

```
00087 { return global_range; }
```

Here is the caller graph for this function:

8.8.2.3.2.7 template<std::size_t dims = 1> auto& cl::sycl::item< dims >::operator[ ] (int *dimension*) [inline]

Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.

Definition at line 81 of file item.hpp.

```
00081 { return global_index[dimension]; }
```

8.8.2.3.2.8 template<std::size_t dims = 1> void cl::sycl::item< dims >::set (id< dims > *Index*) [inline]

For the implementation, need to set the global index.

Todo Move to private and add friends

Definition at line 113 of file item.hpp.

```
00113 { global_index = Index; }
```

8.8.2.3.3 Member Data Documentation

8.8.2.3.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::item< dims >::dimensionality = dims [static]

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

Definition at line 35 of file item.hpp.

8.8.2.3.3.2 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::global_index [private]

Definition at line 40 of file item.hpp.

Referenced by cl::sycl::item< dims >::get().

8.8.2.3.3.3 template<std::size_t dims = 1> range<dims> cl::sycl::item< dims >::global_range [private]

Definition at line 39 of file item.hpp.

Referenced by cl::sycl::item< dims >::get_range().

8.8.2.3.3.4 template<std::size_t dims = 1> id<dims> cl::sycl::item< dims >::offset [private]

Definition at line 41 of file item.hpp.

Referenced by cl::sycl::item< dims >::get_offset().

8.8.2.4 struct cl::sycl::nd_item

template<std::size_t dims = 1>struct cl::sycl::nd_item< dims >

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

Definition at line 32 of file nd_item.hpp.

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

Public Member Functions

- nd_item (nd_range< dims > ndr)

  Create an empty nd_item<> from an nd_range<>
- nd_item (id< dims > global_index, nd_range< dims > ndr)

  Create a full nd_item.
- id< dims > get_global () const

  Return the constituent global id representing the work-item's position in the global iteration space.
- size_t get_global (int dimension) const

  Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.
- size_t get_global_linear_id () const

  Return the flattened id of the current work-item after subtracting the offset.
- id< dims > get_local () const

  Return the constituent local id representing the work-item's position within the current work-group.
- size_t get_local (int dimension) const

  Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.
- size_t get_local_linear_id () const

  Return the flattened id of the current work-item within the current work-group.
- id< dims > get_group () const

Return the constituent group group representing the work-group's position within the overall nd_range.

- size_t get_group (int dimension) const

  Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

- size_t get_group_linear_id () const

  Return the flattened id of the current work-group.

- id< dims > get_num_groups () const

  Return the number of groups in the nd_range.

- size_t get_num_groups (int dimension) const

  Return the number of groups for dimension in the nd_range.

- range< dims > get_global_range () const

  Return a range<> representing the dimensions of the nd_range<>

- range< dims > get_local_range () const

  Return a range<> representing the dimensions of the current work-group.

- id< dims > get_offset () const

  Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.

- nd_range< dims > get_nd_range () const

  Return the nd_range<> of the current execution.

- void barrier (access::fence_space flag=access::fence_space::global_and_local) const

  Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.

- void set_local (id< dims > Index)
- void set_global (id< dims > Index)

Public Attributes

- ND_range

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- id< dims > global_index
- id< dims > local_index
- nd_range< dims > ND_range

8.8.2.4.1 Constructor & Destructor Documentation

8.8.2.4.1.1 template<std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item (nd_range< dims > *ndr*) [inline]

Create an empty nd_item<> from an nd_range<>

Todo This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

Definition at line 53 of file nd_item.hpp.

```
00053 : ND_range { ndr } {}
```

8.8.2.4.1.2 template<std::size_t dims = 1> cl::sycl::nd_item< dims >::nd_item (id< dims > *global_index*, nd_range< dims > *ndr*) [inline]

Create a full nd_item.

Todo This is for validation purpose. Hide this to the programmer somehow

Definition at line 61 of file nd_item.hpp.

```
00062                               :
00063     global_index { global_index },
00064     // Compute the local index using the offset and the group size
00065     local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
```

8.8.2.4.2 Member Function Documentation

8.8.2.4.2.1 template<std::size_t dims = 1> void cl::sycl::nd_item< dims >::barrier (access::fence_space *flag* = access::fence_space::global_and_local) const [inline]

Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.

The current work-item will wait at the barrier until all work-items in the current work-group have reached the barrier.

In addition, the barrier performs a fence operation ensuring that all memory accesses in the specified address space issued before the barrier complete before those issued after the barrier

Definition at line 188 of file nd_item.hpp.

References cl::sycl::detail::unimplemented().

```
00189                                                       {
00190 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00191     /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00192        work-item of the work-group */
00193 #pragma omp barrier
00194 #else
00195     // \todo To be implemented efficiently otherwise
00196     detail::unimplemented();
00197 #endif
00198   }
```

Here is the call graph for this function:

8.8.2.4.2.2 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_global ( ) const [inline]

Return the constituent global id representing the work-item's position in the global iteration space.

Definition at line 80 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::global_index.

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id(), cl::sycl::nd_item< dims >::get_group(), and cl::sycl::detail::accessor< DataType, Dimensions, AccessMode, Target >::operator[]().

```
00080 { return global_index; }
```

Here is the caller graph for this function:

8.8.2.4.2.3 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_global (int *dimension*) const [inline]

Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.

Definition at line 87 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global().

Referenced by cl::sycl::nd_item< dims >::get_global().

```
00087 { return get_global()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.4 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_global_linear_id ( ) const [inline]

Return the flattened id of the current work-item after subtracting the offset.

Definition at line 93 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global(), cl::sycl::nd_item< dims >::get_global_range(), cl::sycl::nd_item< dims >::get_offset(), and cl::sycl::detail::linear_id().

```
00093                                       {
00094     return detail::linear_id(get_global_range(),
      get_global(), get_offset());
00095   }
```

Here is the call graph for this function:

8.8.2.4.2.5 template<std::size_t dims = 1> range<dims> cl::sycl::nd_item< dims >::get_global_range ( ) const [inline]

Return a range<> representing the dimensions of the nd_range<>

Definition at line 156 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id().

```
00156                                        {
00157     return get_nd_range().get_global();
00158   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.6 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_group ( ) const [inline]

Return the constituent group group representing the work-group's position within the overall nd_range.

Definition at line 122 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_global(), and cl::sycl::nd_item< dims >::get_local_range().

Referenced by cl::sycl::nd_item< dims >::get_group(), and cl::sycl::nd_item< dims >::get_group_linear_id().

```
00122                                 {
00123     /* Convert get_local_range() to an id<> to remove ambiguity into using
00124        implicit conversion either from range<> to id<> or the opposite */
00125     return get_global()/id<dims> { get_local_range() };
00126   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.7 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_group (int *dimension*) const [inline]

Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

Definition at line 133 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_group().

```
00133                                       {
00134     return get_group()[dimension];
00135   }
```

Here is the call graph for this function:

8.8.2.4.2.8 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_group_linear_id ( ) const [inline]

Return the flattened id of the current work-group.

Definition at line 139 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_group(), cl::sycl::nd_item< dims >::get_num_groups(), and cl::sycl↩ ::detail::linear_id().

```
00139                                       {
00140     return detail::linear_id(get_num_groups(),
      get_group());
00141   }
```

Here is the call graph for this function:

8.8.2.4.2.9 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_local ( ) const [inline]

Return the constituent local id representing the work-item's position within the current work-group.

Definition at line 101 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::local_index.

Referenced by cl::sycl::nd_item< dims >::get_local_linear_id().

```
00101 { return local_index; }
```

Here is the caller graph for this function:

8.8.2.4.2.10 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_local (int *dimension*) const [inline]

Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.

Definition at line 108 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_local().

Referenced by cl::sycl::nd_item< dims >::get_local().

```
00108 { return get_local()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.11 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_local_linear_id ( ) const [inline]

Return the flattened id of the current work-item within the current work-group.

Definition at line 114 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_local(), cl::sycl::nd_item< dims >::get_local_range(), and cl::sycl↩ ::detail::linear_id().

```
00114                                       {
00115     return detail::linear_id(get_local_range(),
      get_local());
00116   }
```

Here is the call graph for this function:

8.8.2.4.2.12 template<std::size_t dims = 1> range<dims> cl::sycl::nd_item< dims >::get_local_range ( ) const [inline]

Return a range<> representing the dimensions of the current work-group.

Definition at line 162 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

Referenced by cl::sycl::nd_item< dims >::get_group(), and cl::sycl::nd_item< dims >::get_local_linear_id().

```
00162                                        {
00163     return get_nd_range().get_local();
00164   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.13 template<std::size_t dims = 1> nd_range<dims> cl::sycl::nd_item< dims >::get_nd_range ( ) const [inline]

Return the nd_range<> of the current execution.

Definition at line 175 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::ND_range.

Referenced by cl::sycl::nd_item< dims >::get_global_range(), cl::sycl::nd_item< dims >::get_local_range(), cl::sycl::nd_item< dims >::get_num_groups(), and cl::sycl::nd_item< dims >::get_offset().

```
00175 { return ND_range; }
```

Here is the caller graph for this function:

8.8.2.4.2.14 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_num_groups ( ) const [inline]

Return the number of groups in the nd_range.

Definition at line 145 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

Referenced by cl::sycl::nd_item< dims >::get_group_linear_id(), and cl::sycl::nd_item< dims >::get_num_groups().

```
00145                                     {
00146     return get_nd_range().get_group();
00147   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.15 template<std::size_t dims = 1> size_t cl::sycl::nd_item< dims >::get_num_groups (int *dimension*) const [inline]

Return the number of groups for dimension in the nd_range.

Definition at line 150 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_num_groups().

```
00150                                                {
00151       return get_num_groups()[dimension];
00152     }
```

Here is the call graph for this function:

8.8.2.4.2.16 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::get_offset ( ) const [inline]

Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.

Definition at line 171 of file nd_item.hpp.

References cl::sycl::nd_item< dims >::get_nd_range().

Referenced by cl::sycl::nd_item< dims >::get_global_linear_id().

```
00171 { return get_nd_range().get_offset(); }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.2.4.2.17 template<std::size_t dims = 1> void cl::sycl::nd_item< dims >::set_global (id< dims > *Index*) [inline]

Definition at line 206 of file nd_item.hpp.

```
00206 { global_index = Index; }
```

8.8.2.4.2.18 template<std::size_t dims = 1> void cl::sycl::nd_item< dims >::set_local (id< dims > *Index*) [inline]

Definition at line 202 of file nd_item.hpp.

```
00202 { local_index = Index; }
```

8.8.2.4.3 Member Data Documentation

8.8.2.4.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::nd_item< dims >::dimensionality = dims [static]

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

Definition at line 35 of file nd_item.hpp.

8.8.2.4.3.2 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::global_index [private]

Definition at line 39 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_global().

8.8.2.4.3.3 template<std::size_t dims = 1> id<dims> cl::sycl::nd_item< dims >::local_index [private]

Definition at line 42 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_local().

8.8.2.4.3.4 template<std::size_t dims = 1> nd_range<dims> cl::sycl::nd_item< dims >::ND_range [private]

Definition at line 43 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< dims >::get_nd_range().

8.8.2.4.3.5 template<std::size_t dims = 1> cl::sycl::nd_item< dims >::ND_range

Initial value:

```
{ ndr }
  {}


  nd_item() = default
```

Definition at line 66 of file nd_item.hpp.

8.8.2.5 struct cl::sycl::nd_range

template<std::size_t 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 33 of file nd_range.hpp.

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

Public Member Functions

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

  Construct a ND-range with all the details available in OpenCL.
- range< dims > get_global () const

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

  Get the local part of the iteration space range.
- auto get_group () const

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

  Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = dims

Private Attributes

- range< dimensionality > global_range
- range< dimensionality > local_range
- id< dimensionality > offset

8.8.2.5.1 Constructor & Destructor Documentation

8.8.2.5.1.1 template<std::size_t dims = 1> cl::sycl::nd_range< dims >::nd_range (range< dims > *global_size*, range< dims > *local_size*, id< dims > *offset* = {}) [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 50 of file nd_range.hpp.

```
00052                               {}) :
00053     global_range { global_size }, local_range { local_size },
      offset { offset }
00054   { }
```

8.8.2.5.2 Member Function Documentation

8.8.2.5.2.1 template<std::size_t dims = 1> void cl::sycl::nd_range< dims >::display ( ) const [inline]

Display the value for debugging and validation purpose.

Definition at line 77 of file nd_range.hpp.

References cl::sycl::detail::display_vector< T >::display().

```
00077                         {
00078     global_range.display();
00079     local_range.display();
00080     offset.display();
00081   }
```

Here is the call graph for this function:

8.8.2.5.2.2 template<std::size_t dims = 1> range<dims> cl::sycl::nd_range< dims >::get_global ( ) const [inline]

Get the global iteration space range.

Definition at line 58 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::global_range.

```
00058 { return global_range; }
```

8.8.2.5.2.3 template<std::size_t dims = 1> auto cl::sycl::nd_range< dims >::get_group ( ) const [inline]

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

Definition at line 66 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::local_range.

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

```
00066                         {
00067     // \todo Assume that global_range is a multiple of local_range, element-wise
00068     return global_range/local_range;
00069   }
```

Here is the caller graph for this function:

8.8.2.5.2.4 template<std::size_t dims = 1> range<dims> cl::sycl::nd_range< dims >::get_local ( ) const [inline]

Get the local part of the iteration space range.

Definition at line 62 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::local_range.

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

```
00062 { return local_range; }
```

Here is the caller graph for this function:

8.8.2.5.2.5 template<std::size_t dims = 1> id<dims> cl::sycl::nd_range< dims >::get_offset ( ) const [inline]

Todo get_offset() is lacking in the specification

Definition at line 73 of file nd_range.hpp.

References cl::sycl::nd_range< dims >::offset.

```
00073 { return offset; }
```

8.8.2.5.3 Member Data Documentation

8.8.2.5.3.1 template<std::size_t dims = 1> constexpr auto cl::sycl::nd_range< dims >::dimensionality = dims [static]

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

Definition at line 36 of file nd_range.hpp.

8.8.2.5.3.2 template<std::size_t dims = 1> range<dimensionality> cl::sycl::nd_range< dims >::global_range [private]

Definition at line 40 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_global().

8.8.2.5.3.3 template<std::size_t dims = 1> range<dimensionality> cl::sycl::nd_range< dims >::local_range [private]

Definition at line 41 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_group(), and cl::sycl::nd_range< dims >::get_local().

8.8.2.5.3.4 template<std::size_t dims = 1> id<dimensionality> cl::sycl::nd_range< dims >::offset [private]

Definition at line 42 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< dims >::get_offset().

8.8.2.6 struct cl::sycl::detail::parallel_for_iterate

template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::detail::parallel_for_↩ iterate< 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 47 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.8.2.6.1 Constructor & Destructor Documentation

8.8.2.6.1.1 template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >::parallel_for_iterate (Range *r,* ParallelForFunctor & *f,* Id & *index*) [inline]

Definition at line 48 of file parallelism.hpp.

```
00048                                                                    {
00049     for (boost::multi_array_types::index _sycl_index = 0,
00050            _sycl_end = r[Range::dimensionality - level];
00051          _sycl_index < _sycl_end;
00052          _sycl_index++) {
00053       // Set the current value of the index for this dimension
00054       index[Range::dimensionality - level] = _sycl_index;
00055       // Iterate further on lower dimensions
00056       parallel_for_iterate<level - 1,
00057                            Range,
00058                            ParallelForFunctor,
00059                            Id> { r, f, index };
00060     }
00061   }
```

8.8.2.7 struct cl::sycl::detail::parallel_OpenMP_for_iterate

template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::detail::parallel_Open↩ MP_for_iterate< 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 71 of file parallelism.hpp.

Public Member Functions

- parallel_OpenMP_for_iterate (Range r, ParallelForFunctor &f)

8.8.2.7.1 Constructor & Destructor Documentation

8.8.2.7.1.1 template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >::parallel_OpenMP_for_iterate (Range *r,* ParallelForFunctor & *f*) [inline]

Definition at line 72 of file parallelism.hpp.

```
00072                                                                {
00073     // Create the OpenMP threads before the for loop to avoid creating an
00074     // index in each iteration
00075 #pragma omp parallel
00076     {
00077       // Allocate an OpenMP thread-local index
00078       Id index;
00079       // Make a simple loop end condition for OpenMP
00080       boost::multi_array_types::index _sycl_end = r[Range::dimensionality - level];
00081       /* Distribute the iterations on the OpenMP threads. Some OpenMP
00082          "collapse" could be useful for small iteration space, but it
00083          would need some template specialization to have real contiguous
00084          loop nests */
00085 #pragma omp for
00086       for (boost::multi_array_types::index _sycl_index = 0;
00087            _sycl_index < _sycl_end;
00088            _sycl_index++) {
```

```
00089         // Set the current value of the index for this dimension
00090         index[Range::dimensionality - level] = _sycl_index;
00091         // Iterate further on lower dimensions
00092         parallel_for_iterate<level - 1,
00093                              Range,
00094                              ParallelForFunctor,
00095                              Id> { r, f, index };
00096       }
00097     }
00098   }
```

8.8.2.8 struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

template<typename Range, typename ParallelForFunctor, typename Id>struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

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

Definition at line 105 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.8.2.8.1 Constructor & Destructor Documentation

8.8.2.8.1.1 template<typename Range , typename ParallelForFunctor , typename Id > cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >::parallel_for_iterate (Range *r,* ParallelForFunctor & *f,* Id & *index*) [inline]

Definition at line 106 of file parallelism.hpp.

```
00106                                                                   {
00107     f(index);
00108   }
```

8.8.2.9 class cl::sycl::range

template<std::size_t dims = 1>class cl::sycl::range< dims >

A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes.

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

Todo add to the specification this default parameter value?

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

Definition at line 31 of file range.hpp.

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

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

Additional Inherited Members

8.8.3 Function Documentation

8.8.3.1 auto cl::sycl::make_id (id< 1 > *i*) [inline]

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

Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 66 of file id.hpp.

```
00066 { return i; }
```

8.8.3.2 auto cl::sycl::make_id (id< 2 > *i*) [inline]

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

Definition at line 67 of file id.hpp.

```
00067 { return i; }
```

8.8.3.3 auto cl::sycl::make_id (id< 3 > *i*) [inline]

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

Definition at line 68 of file id.hpp.

```
00068 { return i; }
```

8.8.3.4 template<typename... BasicType> auto cl::sycl::make_id (BasicType... *Args*)

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

Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

Definition at line 74 of file id.hpp.

```
00074                                {
00075   // Call constructor directly to allow narrowing
00076   return id<sizeof...(Args)>(Args...);
00077 }
```

8.8.3.5 auto cl::sycl::make_range (range< 1 > *r*) [inline]

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

Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 48 of file range.hpp.

```
00048 { return r; }
```

8.8.3.6 auto cl::sycl::make_range (range< 2 > r) [inline]

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

Definition at line 49 of file range.hpp.

```
00049 { return r; }
```

8.8.3.7 auto cl::sycl::make_range (range< 3 > r) [inline]

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

Definition at line 50 of file range.hpp.

```
00050 { return r; }
```

8.8.3.8 template<typename... BasicType> auto cl::sycl::make_range (BasicType... *Args*)

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

Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

Definition at line 57 of file range.hpp.

```
00057                                    {
00058   // Call constructor directly to allow narrowing
00059   return range<sizeof...(Args)>(Args...);
00060 }
```

8.8.3.9 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for (range< Dimensions > *r*, ParallelForFunctor *f*)

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

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

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

Definition at line 118 of file parallelism.hpp.

Referenced by cl::sycl::handler::parallel_for(), and cl::sycl::detail::parallel_for_global_offset().

```
00119                                       {
00120 #ifdef _OPENMP
00121   // Use OpenMP for the top loop level
00122   parallel_OpenMP_for_iterate<Dimensions,
00123                               range<Dimensions>,
00124                               ParallelForFunctor,
00125                               id<Dimensions>> { r, f };
00126 #else
00127   // In a sequential execution there is only one index processed at a time
00128   id<Dimensions> index;
00129   parallel_for_iterate<Dimensions,
00130                        range<Dimensions>,
00131                        ParallelForFunctor,
00132                        id<Dimensions>> { r, f, index };
00133 #endif
00134 }
```

Here is the caller graph for this function:

8.8.3.10 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

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

Implement a variation of 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 165 of file parallelism.hpp.

References cl::sycl::nd_range< dims >::get_group(), and cl::sycl::nd_range< dims >::get_local().

```
00166                                       {
00167   // In a sequential execution there is only one index processed at a time
00168   nd_item<Dimensions> index { r };
00169   // To iterate on the work-group
00170   id<Dimensions> group;
00171   range<Dimensions> group_range = r.get_group();
00172   // To iterate on the local work-item
00173   id<Dimensions> local;
00174
00175   range<Dimensions> local_range = r.get_local();
00176
00177   // Reconstruct the nd_item from its group and local id
00178   auto reconstruct_item = [&] (id<Dimensions> l) {
00179     //local.display();
00180     // Reconstruct the global nd_item
00181     index.set_local(local);
00182     // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00183     index.set_global(local + id<Dimensions>(local_range)*group);
00184     // Call the user kernel at last
00185     f(index);
00186   };
00187
00188   /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
00189      into another functor that iterates inside the work-group and then
00190      calls f */
00191   auto iterate_in_work_group = [&] (id<Dimensions> g) {
00192     //group.display();
00193     // Then iterate on the local work-groups
00194     parallel_for_iterate<Dimensions,
00195                          range<Dimensions>,
00196                          decltype(reconstruct_item),
00197                          id<Dimensions>> { local_range, reconstruct_item, local };
00198   };
00199
00200   // First iterate on all the work-groups
00201   parallel_for_iterate<Dimensions,
00202                        range<Dimensions>,
00203                        decltype(iterate_in_work_group),
00204                        id<Dimensions>> { group_range, iterate_in_work_group, group };
00205 }
```

Here is the call graph for this function:

8.8.3.11 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > *global_size*, id< Dimensions > *offset*, ParallelForFunctor *f*)

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

Implementation of parallel_for with a range<> and an offset.

Definition at line 139 of file parallelism.hpp.

References cl::sycl::detail::parallel_for().

```
00141                                                 {
```

```
00142   // Reconstruct the item from its id<> and its offset
00143   auto reconstruct_item = [&] (id<Dimensions> l) {
00144     // Reconstruct the global item
00145     item<Dimensions> index { global_size, l + offset, offset };
00146     // Call the user kernel with the item<> instead of the id<>
00147     f(index);
00148   };
00149
00150   // First iterate on all the work-groups
00151   parallel_for(global_size, reconstruct_item);
00152 }
```

Here is the call graph for this function:

8.8.3.12 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::parallel_for_work_item (group< Dimensions > *g*, ParallelForFunctor *f*)

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

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

Todo To be implemented

Loop on the work-items inside a work-group

Definition at line 34 of file parallelism.hpp.

References cl::sycl::detail::parallel_for_workitem().

```
00034                                                                    {
00035     detail::parallel_for_workitem(g, f);
00036   }
```

Here is the call graph for this function:

8.8.3.13 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > *r*, ParallelForFunctor *f*)

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

Implement the loop on the work-groups.

Definition at line 210 of file parallelism.hpp.

References cl::sycl::nd_range< dims >::get_group().

Referenced by cl::sycl::handler::parallel_for_work_group().

```
00211                                                 {
00212   // In a sequential execution there is only one index processed at a time
00213   group<Dimensions> g { r };
00214
00215   // First iterate on all the work-groups
00216   parallel_for_iterate<Dimensions,
00217                        range<Dimensions>,
00218                        ParallelForFunctor,
00219                        group<Dimensions>> {
00220     r.get_group(),
00221     f,
00222     g };
00223 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.8.3.14 template<std::size_t Dimensions = 1, typename ParallelForFunctor > void cl::sycl::detail::parallel_for_workitem (group< Dimensions > *g*, ParallelForFunctor *f*)

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

Implement the loop on the work-items inside a work-group.

Definition at line 228 of file parallelism.hpp.

References cl::sycl::group< dims >::get(), cl::sycl::detail::small_array< std::size_t, range< dims >, Dims >::get(), cl::sycl::group< dims >::get_local_range(), and cl::sycl::group< dims >::get_nd_range().

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

```
00229                                                 {
00230 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00231   /* To implement barriers With OpenMP, one thread is created for each
00232      work-item in the group and thus an OpenMP barrier has the same effect
00233      of an OpenCL barrier executed by the work-items in a workgroup
00234
00235      The issue is that the parallel_for_workitem() execution is slow even
00236      when nd_item::barrier() is not used
00237   */
00238   range<Dimensions> l_r = g.get_nd_range().get_local();
00239   // \todo Implement with a reduction algorithm
00240   int tot = l_r.get(0);
00241   for (int i = 1; i < (int) Dimensions; ++i){
00242     tot *= l_r.get(i);
00243   }
00244   /* An alternative could be to use 1 to 3 loops with #pragma omp parallel
00245      for collapse(...) instead of reconstructing the iteration index from
00246      the thread number */
00247   omp_set_num_threads(tot);
00248 #pragma omp parallel
00249   {
00250     int th_id = omp_get_thread_num();
00251     nd_item<Dimensions> index { g.get_nd_range() };
00252     id<Dimensions> local; // to initialize correctly
00253
00254     if (Dimensions ==1) {
00255       local[0] = th_id;
00256     } else if (Dimensions == 2) {
00257       local[0] = th_id / l_r.get(1);
00258       local[1] = th_id - local[0]*l_r.get(1);
00259     } else if (Dimensions == 3) {
00260       int tmp = l_r.get(1)*l_r.get(2);
00261       local[0] = th_id / tmp;
00262       local[1] = (th_id - local[0]*tmp) / l_r.get(1);
00263       local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
00264     }
00265     index.set_local(local);
00266     index.set_global(local + id<Dimensions>(l_r)*g.get());
00267     f(index);
00268   }
00269 #else
00270   // In a sequential execution there is only one index processed at a time
00271   nd_item<Dimensions> index { g.get_nd_range() };
00272   // To iterate on the local work-item
00273   id<Dimensions> local;
00274
00275   // Reconstruct the nd_item from its group and local id
00276   auto reconstruct_item = [&] (id<Dimensions> l) {
00277     //local.display();
00278     //l.display();
00279     // Reconstruct the global nd_item
00280     index.set_local(local);
00281     // \todo Some strength reduction here
00282     index.set_global(local + id<Dimensions>(g.get_local_range())*g.get());
00283     // Call the user kernel at last
00284     f(index);
00285   };
00286
00287   // Then iterate on all the work-items of the work-group
00288   parallel_for_iterate<Dimensions,
00289                        range<Dimensions>,
00290                        decltype(reconstruct_item),
00291                        id<Dimensions>> {
00292     g.get_local_range(),
00293     reconstruct_item,
00294     local };
00295 #endif
00296 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.9 Vector types in SYCL

Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

8.9.1 Detailed Description

8.9.2 Class Documentation

8.9.2.1 class cl::sycl::vec

template<typename DataType, size_t NumElements>class cl::sycl::vec< DataType, NumElements >

Small OpenCL vector class.

Todo add [] operator

Todo add iterators on elements, with begin() and end()

Todo having vec<> sub-classing array<> instead would solve the previous issues

Todo move the implementation elsewhere

Todo simplify the helpers by removing some template types since there are now inside the vec<> class.

Todo rename in the specification element_type to value_type

Definition at line 42 of file vec.hpp.

Inheritance diagram for cl::sycl::vec< DataType, NumElements >:

Collaboration diagram for cl::sycl::vec< DataType, NumElements >:

Public Member Functions

- template<typename... Types>
  vec (const Types...args)

  Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)
- vec ()=default

  Use classical constructors too.

Private Types

- using basic_type = typename detail::small_array< DataType, vec< DataType, NumElements >, Num↩ Elements >

Static Private Member Functions

- template<typename V , typename Element , size_t s>
  static auto flatten (const vec< Element, s > i)

  Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.
- template<typename V , typename Type >
  static auto flatten (const Type i)

  If we do not have a vector, just forward it as a tuple up to the final initialization.
- template<typename V , typename... Types>
  static auto flatten_to_tuple (const Types...i)

  Take some initializer values and apply flattening on each value.

Additional Inherited Members

8.9.2.1.1 Member Typedef Documentation

8.9.2.1.1.1 template<typename DataType, size_t NumElements> using cl::sycl::vec< DataType, NumElements >::basic_type = typename detail::small_array<DataType, vec<DataType, NumElements>, NumElements> [private]

Definition at line 47 of file vec.hpp.

8.9.2.1.2 Constructor & Destructor Documentation

8.9.2.1.2.1 template<typename DataType, size_t NumElements> template<typename... Types> cl::sycl::vec< DataType, NumElements >::vec (const Types... *args*) [inline]

Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)

Definition at line 57 of file vec.hpp.

References cl::sycl::vec< DataType, NumElements >::vec().

```
00058     : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }
```

Here is the call graph for this function:

8.9.2.1.2.2 template<typename DataType, size_t NumElements> cl::sycl::vec< DataType, NumElements >::vec ( ) [default]

Use classical constructors too.

Referenced by cl::sycl::vec< DataType, NumElements >::vec().

Here is the caller graph for this function:

8.9.2.1.3 Member Function Documentation

8.9.2.1.3.1 template<typename DataType, size_t NumElements> template<typename V , typename Element , size_t s> static auto cl::sycl::vec< DataType, NumElements >::flatten (const vec< Element, s > *i*) [inline], [static], [private]

Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.

If we have a vector, just forward its array content since an array has also a tuple interface :-) (23.3.2.9 Tuple interface to class template array [array.tuple])

Definition at line 78 of file vec.hpp.

```
00078                                                {
00079     static_assert(s <= V::dimension,
00080                   "The element i will not fit in the vector");
00081     return static_cast<std::array<Element, s>>(i);
00082   }
```

8.9.2.1.3.2 template<typename DataType, size_t NumElements> template<typename V , typename Type > static auto cl::sycl::vec< DataType, NumElements >::flatten (const Type *i*) [inline], [static], [private]

If we do not have a vector, just forward it as a tuple up to the final initialization.

Returns

typically tuple<double>{ 2.4 } from 2.4 input for example

Definition at line 91 of file vec.hpp.

```
00091                                    {
00092     return std::make_tuple(i);
00093   }
```

8.9.2.1.3.3 template<typename DataType, size_t NumElements> template<typename V , typename... Types> static auto cl::sycl::vec< DataType, NumElements >::flatten_to_tuple (const Types... *i*) [inline], [static], [private]

Take some initializer values and apply flattening on each value.

Returns

a tuple of scalar initializer values

Definition at line 101 of file vec.hpp.

```
00101                                                  {
00102     // Concatenate the tuples returned by each flattening
00103     return std::tuple_cat(flatten<V>(i)...);
00104   }
```

8.9.3 Macro Definition Documentation

8.9.3.1 #define TRISYCL_DEFINE_VEC_TYPE(*type, actual_type*)

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

Value:

```
TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)
      \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
```

Declare the vector types of a type for all the sizes.

Definition at line 162 of file vec.hpp.

8.9.3.2 #define TRISYCL_DEFINE_VEC_TYPE_SIZE(*type, size, actual_type*) using type##size = vec<actual_type, size>;

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

A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>

Definition at line 158 of file vec.hpp.

Chapter 9

Namespace Documentation

9.1 cl Namespace Reference

The vector type to be used as SYCL vector.

Namespaces

- sycl

9.1.1 Detailed Description

The vector type to be used as SYCL vector.

The weak pointer type to be used as SYCL weak pointer.

The shared pointer type to be used as SYCL shared pointer.

The unique pointer type to be used as SYCL unique pointer.

The mutex type to be used as SYCL mutex.

The functional type to be used as SYCL function.

The string type to be used as SYCL string.

9.2 cl::sycl Namespace Reference

Namespaces

- access

  Describe the type of access by kernels.
- detail
- info
- 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...
- class context

  SYCL context. More...
- class cpu_selector

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics. More...
- class default_selector

  Devices selected by heuristics of the system. More...
- class device

  SYCL device. More...
- class 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...
- class gpu_selector

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices. More...
- struct group

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

  Command group handler class. More...
- class host_selector

  Selects the SYCL host CPU device that does not require an OpenCL runtime. More...
- class id

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

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- class kernel

  Kernel. More...
- struct nd_item

  A SYCL nd_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...
- class platform

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

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

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...
- class vec

  Small OpenCL vector class. More...

Typedefs

- template<typename T >
  using constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be an OpenCL constant pointer.
- template<typename T >
  using generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be an OpenCL 2 generic pointer.
- template<typename T >
  using global = detail::addr_space< T, global_address_space >

  Declare a variable to be an OpenCL global pointer.
- template<typename T >
  using local = detail::addr_space< T, local_address_space >

  Declare a variable to be an OpenCL local pointer.
- template<typename T >
  using priv = detail::addr_space< T, private_address_space >

  Declare a variable to be an OpenCL private pointer.
- template<typename Pointer , address_space AS>
  using multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.
- template<typename T >
  using buffer_allocator = std::allocator< T >

  The default buffer allocator used by the runtime, when no allocator is defined by the user.
- template<class T , class Alloc = std::allocator<T>>
  using vector_class = std::vector< T, Alloc >
- using string_class = std::string
- template<class R , class... ArgTypes>
  using function_class = std::function< R(ArgTypes...)>
- using mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using unique_ptr_class = std::unique_ptr< T[ ], D >
- template<class T >
  using shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using weak_ptr_class = std::weak_ptr< T >
- using async_handler = function_class< int >

Enumerations

- enum address_space {
  constant_address_space, generic_address_space, global_address_space, local_address_space,
  private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.
- auto make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_id (id< 2 > i)

- auto make_id (id< 3 > i)
- template<typename... BasicType>
  auto make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_item (group< Dimensions > g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_range (range< 2 > r)
- auto make_range (range< 3 > r)
- template<typename... BasicType>
  auto make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

9.2.1 Typedef Documentation

9.2.1.1 template<class R , class... ArgTypes> using cl::sycl::function_class = typedef std::function<R(ArgTypes...)>

Definition at line 51 of file default_classes.hpp.

9.2.1.2 using cl::sycl::mutex_class = typedef std::mutex

Definition at line 65 of file default_classes.hpp.

9.2.1.3 template<class T > using cl::sycl::shared_ptr_class = typedef std::shared_ptr<T>

Definition at line 95 of file default_classes.hpp.

9.2.1.4 using cl::sycl::string_class = typedef std::string

Definition at line 36 of file default_classes.hpp.

9.2.1.5 template<class T , class D = std::default_delete<T>> using cl::sycl::unique_ptr_class = typedef std::unique_ptr<T[ ], D>

Definition at line 80 of file default_classes.hpp.

9.2.1.6 template<class T , class Alloc = std::allocator<T>> using cl::sycl::vector_class = typedef std::vector<T, Alloc>

Definition at line 22 of file default_classes.hpp.

9.2.1.7 template<class T > using cl::sycl::weak_ptr_class = typedef std::weak_ptr<T>

Definition at line 110 of file default_classes.hpp.

9.3 cl::sycl::access Namespace Reference

Describe the type of access by kernels.

Enumerations

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

  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 }

  The target enumeration describes the type of object to be accessed via the accessor.
- enum fence_space : char { fence_space::local_space, fence_space::global_space, fence_space::global_↩
  and_local }

  Precise the address space a barrier needs to act on.

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?

9.4 cl::sycl::detail Namespace Reference

Classes

- struct accessor

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

  Implementation of an array variable with an OpenCL address space. More...
- struct address_space_base

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct address_space_fundamental

  Implementation of a fundamental type with an OpenCL address space. More...
- struct address_space_object

  Implementation of an object type with an OpenCL address space. More...
- struct address_space_ptr

  Implementation for an OpenCL address space pointer. More...
- struct address_space_variable

  Implementation of a variable with an OpenCL address space. 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 buffer_base

  Factorize some template independent buffer aspects in a base class.
- class buffer_customer

  Keep track of the tasks waiting for the availability of a buffer generation, either to read it or to write it.
- struct debug

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct display_vector

Class used to display a vector-like type of classes that inherit from it. More...

- struct expand_to_vector

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...
- struct opencl_type

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct opencl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct opencl_type< T, generic_address_space >

  Add an attribute for __generic address space. More...
- struct opencl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct opencl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct opencl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct parallel_for_iterate

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

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

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

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct small_array_123

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct small_array_123< BasicType, FinalType, 2 >
- struct small_array_123< BasicType, FinalType, 3 >
- struct task

  The abstraction to represent SYCL tasks executing inside command_group.

Typedefs

- template<typename T , address_space AS>
  using addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > tuple_to_array_iterate (Tuple t, std::index_sequence< Is...>)

  Helper to construct an array from initializer elements provided as a tuple.

- template<typename V , typename Tuple >
  auto tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto expand (Tuple t)

  Create the array data of V from a tuple of initializer.
- template<typename Range , typename Id >
  size_t linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.
- void unimplemented ()

  Display an "unimplemented" message.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelFor↩
  Functor f)

  Implementation of parallel_for with a range<> and an offset.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.

9.5 cl::sycl::info Namespace Reference

Classes

- class param_traits

  Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Typedefs

- using gl_context_interop = bool
- using device_fp_config = unsigned int
- using device_exec_capabilities = unsigned int
- using device_queue_properties = unsigned int
- using queue_profiling = bool

Enumerations

- enum context : int { context::reference_count, context::num_devices, context::gl_interop }

  Context information descriptors.

- enum device : int {
  device::device_type, device::vendor_id, device::max_compute_units, device::max_work_item_dimensions,
  device::max_work_item_sizes, device::max_work_group_size, device::preferred_vector_width_char,
  device::preferred_vector_width_short,
  device::preferred_vector_width_int, device::preferred_vector_width_long_long, device::preferred_vector_↩
  width_float, device::preferred_vector_width_double,
  device::preferred_vector_width_half, device::native_vector_witdth_char, device::native_vector_witdth_short,
  device::native_vector_witdth_int,
  device::native_vector_witdth_long_long, device::native_vector_witdth_float, device::native_vector_width_↩
  double, device::native_vector_witdth_half,
  device::max_clock_frequency, device::address_bits, device::max_mem_alloc_size, device::image_support,
  device::max_read_image_args, device::max_write_image_args, device::image2d_max_height, device↩
  ::image2d_max_width,
  device::image3d_max_height, device::image3d_max_widht, device::image3d_mas_depth, device::image_↩
  max_buffer_size,
  device::image_max_array_size, device::max_samplers, device::max_parameter_size, device::mem_base↩
  _addr_align,
  device::single_fp_config, device::double_fp_config, device::global_mem_cache_type, device::global_mem↩
  _cache_line_size,
  device::global_mem_cache_size, device::global_mem_size, device::max_constant_buffer_size, device↩
  ::max_constant_args,
  device::local_mem_type, device::local_mem_size, device::error_correction_support, device::host_unified_↩
  memory,
  device::profiling_timer_resolution, device::endian_little, device::is_available, device::is_compiler_available,
  device::is_linker_available, device::execution_capabilities, device::queue_properties, device::built_in_↩
  kernels,
  device::platform, device::name, device::vendor, device::driver_version,
  device::profile, device::device_version, device::opencl_version, device::extensions,
  device::printf_buffer_size, device::preferred_interop_user_sync, device::parent_device, device::partition_↩
  max_sub_devices,
  device::partition_properties, device::partition_affinity_domain, device::partition_type, device::reference_↩
  count }

  Device information descriptors.
- enum device_partition_property : int {
  device_partition_property::unsupported, device_partition_property::partition_equally, device_partition_↩
  property::partition_by_counts, device_partition_property::partition_by_affinity_domain,
  device_partition_property::partition_affinity_domain_next_partitionable }
- enum device_affinity_domain : int {
  device_affinity_domain::unsupported, device_affinity_domain::numa, device_affinity_domain::L4_cache,
  device_affinity_domain::L3_cache,
  device_affinity_domain::L2_cache, device_affinity_domain::next_partitionable }
- enum device_partition_type : int {
  device_partition_type::no_partition, device_partition_type::numa, device_partition_type::L4_cache, device↩
  _partition_type::L3_cache,
  device_partition_type::L2_cache, device_partition_type::L1_cache }
- enum local_mem_type : int { local_mem_type::none, local_mem_type::local, local_mem_type::global }
- enum fp_config : int {
  fp_config::denorm, fp_config::inf_nan, fp_config::round_to_nearest, fp_config::round_to_zero,
  fp_config::round_to_inf, fp_config::fma, fp_config::correctly_rounded_divide_sqrt, fp_config::soft_float }
- enum global_mem_cache_type : int { global_mem_cache_type::none, global_mem_cache_type::read_only,
  global_mem_cache_type::write_only }
- enum device_execution_capabilities : unsigned int { device_execution_capabilities::exec_kernel, device_↩
  execution_capabilities::exec_native_kernel }
- enum device_type : unsigned int {
  device_type::cpu, device_type::gpu, device_type::accelerator, device_type::custom,
  device_type::defaults, device_type::host, device_type::all }

- enum platform : unsigned int { platform::profile, platform::version, platform::name, platform::vendor, platform::extensions }

  Platform information descriptors.
- enum queue : int { queue::context, queue::device, queue::reference_count, queue::properties }

  Queue information descriptors.

9.5.1 Typedef Documentation

9.5.1.1 using cl::sycl::info::device_exec_capabilities = typedef unsigned int

Definition at line 167 of file device.hpp.

9.5.1.2 using cl::sycl::info::device_fp_config = typedef unsigned int

Definition at line 166 of file device.hpp.

9.5.1.3 using cl::sycl::info::device_queue_properties = typedef unsigned int

Definition at line 168 of file device.hpp.

9.5.1.4 using cl::sycl::info::gl_context_interop = typedef bool

Definition at line 31 of file context.hpp.

9.5.1.5 using cl::sycl::info::queue_profiling = typedef bool

Definition at line 35 of file queue.hpp.

9.6 cl::sycl::trisycl Namespace Reference

Classes

- struct default_error_handler

9.6.1 Detailed Description

Todo Refactor when updating to latest specification

Chapter 10

Class Documentation

10.1 cl::sycl::detail::AccessorImpl< T, dimensions, mode, target > Struct Template Reference

`#include <buffer_base.hpp>`

10.1.1 Detailed Description

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

Definition at line 26 of file buffer_base.hpp.

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

- include/CL/sycl/buffer/detail/buffer_base.hpp

10.2 cl::sycl::detail::buffer_base Struct Reference

Factorize some template independent buffer aspects in a base class.

`#include <buffer_base.hpp>`

Inheritance diagram for cl::sycl::detail::buffer_base:

Collaboration diagram for cl::sycl::detail::buffer_base:

Public Member Functions

- buffer_base (bool read_only)
- std::unique_lock< std::mutex > lock ()

  Lock the buffer_base structure by returning a unique_lock on the mutex.
- std::shared_ptr< buffer_customer > get_last_buffer_customer ()
- void set_last_buffer_customer (std::shared_ptr< buffer_customer > bc)

Static Public Member Functions

- template<typename T , std::size_t dimensions, access::mode mode, access::target target = access::global_buffer>
  static std::shared_ptr< buffer_customer > get_buffer_customer (AccessorImpl< T, dimensions, mode, target > &a)

Get the buffer customer associated to the latest version of the buffer.

- static void wait (buffer_base &b)

Public Attributes

- bool read_only

  If the data are read-only, store the information for later optimization.
- std::shared_ptr< buffer_customer > last_buffer_customer

  Store the buffer_customer for the last generation of this buffer.
- std::mutex protect_buffer

10.2.1 Detailed Description

Factorize some template independent buffer aspects in a base class.

Definition at line 30 of file buffer_base.hpp.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 cl::sycl::detail::buffer_base::buffer_base (bool *read_only*) [inline]

Definition at line 40 of file buffer_base.hpp.

```
00040 : read_only { read_only } {}
```

10.2.3 Member Function Documentation

10.2.3.1 template<typename T , std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> static std::shared_ptr<buffer_customer> cl::sycl::detail::buffer_base::get_buffer_customer (AccessorImpl< T, dimensions, mode, target > & *a*) [inline], [static]

Get the buffer customer associated to the latest version of the buffer.

Use atomic list?

Definition at line 64 of file buffer_base.hpp.

References get_last_buffer_customer(), and lock().

Referenced by cl::sycl::detail::task::add().

```
00064                                                                    {
00065     buffer_base &b = a.get_buffer();
00066     {
00067       /// Use atomic list?
00068       // Protect the update of last_buffer_customer in the Buffer
00069       auto lock = b.lock();
00070       std::shared_ptr<buffer_customer> bc = b.get_last_buffer_customer();
00071       auto old_bc = bc;
00072       /* When we write into a buffer, we generate a new version of it (think
00073          "SSA"). Of course we do it also when there is not yet any
00074          buffer_customer */
00075       if (!bc || a.is_write_access()) {
00076         bc = std::make_shared<buffer_customer>(b, a.is_write_access());
00077         b.set_last_buffer_customer(bc);
00078       }
00079
00080       if (old_bc)
00081         // \todo Use atomic list instead
00082         old_bc->set_next_generation(bc);
00083       else
00084         // If we just created the buffer_customer, it is ready to use
00085         bc->notify_ready();
00086
```

```
00087       return bc;
00088     }
00089   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.2.3.2 std::shared_ptr<buffer_customer> cl::sycl::detail::buffer_base::get_last_buffer_customer ( ) [inline]

Definition at line 49 of file buffer_base.hpp.

References last_buffer_customer.

Referenced by get_buffer_customer().

```
00049                                                          {
00050     return last_buffer_customer;
00051   }
```

Here is the caller graph for this function:

10.2.3.3 std::unique_lock<std::mutex> cl::sycl::detail::buffer_base::lock ( ) [inline]

Lock the buffer_base structure by returning a unique_lock on the mutex.

Definition at line 44 of file buffer_base.hpp.

Referenced by get_buffer_customer().

```
00044                                    {
00045     return std::unique_lock<std::mutex> { protect_buffer };
00046   }
```

Here is the caller graph for this function:

10.2.3.4 void cl::sycl::detail::buffer_base::set_last_buffer_customer (std::shared_ptr< buffer_customer > *bc*) [inline]

Definition at line 54 of file buffer_base.hpp.

```
00054                                                                  {
00055     last_buffer_customer = bc;
00056   }
```

10.2.3.5 static void cl::sycl::detail::buffer_base::wait (buffer_base & *b*) [inline], [static]

Definition at line 93 of file buffer_base.hpp.

References last_buffer_customer.

```
00093                                    {
00094     // If there is nobody using the buffer, no need to wait
00095     if (b.last_buffer_customer)
00096       /* In a correct SYCL program there should be no more task creation
00097          using a buffer given to use by a host accessor so this should be
00098          race free */
00099       b.last_buffer_customer->wait_released();
00100   }
```

10.2.4 Member Data Documentation

10.2.4.1 std::shared_ptr<buffer_customer> cl::sycl::detail::buffer_base::last_buffer_customer

Store the buffer_customer for the last generation of this buffer.

Definition at line 36 of file buffer_base.hpp.

Referenced by get_last_buffer_customer(), and wait().

10.2.4.2 std::mutex cl::sycl::detail::buffer_base::protect_buffer

Definition at line 37 of file buffer_base.hpp.

10.2.4.3 bool cl::sycl::detail::buffer_base::read_only

If the data are read-only, store the information for later optimization.

Todo Replace this by a static read-only type for the buffer

Definition at line 33 of file buffer_base.hpp.

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

- include/CL/sycl/buffer/detail/buffer_base.hpp

10.3 cl::sycl::detail::buffer_customer Class Reference

Keep track of the tasks waiting for the availability of a buffer generation, either to read it or to write it.

`#include <buffer_customer.hpp>`

Inheritance diagram for cl::sycl::detail::buffer_customer:

Collaboration diagram for cl::sycl::detail::buffer_customer:

Public Member Functions

- buffer_customer (buffer_base &buf, bool is_write_access)
- void set_next_generation (std::shared_ptr< buffer_customer > bc)

  Set the next generation of the buffer after this.
- void add (std::shared_ptr< task > task, bool is_write_access)

  Add a new task as a customer of the buffer generation.
- void wait ()

  Wait for the buffer generation to be ready to use by a kernel task.
- void release ()

  Release the buffer generation usage by a kernel task.
- void wait_released ()

  Wait for the release of the buffer generation before the host can use it.
- void notify_ready ()

  Notify the customer tasks this buffer generation is ready to use.

Private Attributes

- buffer_base & buf

  The considered buffer.
- std::shared_ptr< buffer_customer > next_generation

  At some point use lock free list for this inside buffer_base.
- bool write_access
- bool ready_to_use
- std::condition_variable ready_cv
- std::mutex ready_mutex

  To protect the access to the condition variable.
- std::atomic< unsigned int > user_number

  Count the number of accelerator-side usage of this buffer generation.
- std::condition_variable released_cv

  To signal when the buffer generation is no longer used from the accelerator side and can be used for example through a host accessor.
- std::mutex released_mutex

  To protect the access to the condition variable.

10.3.1 Detailed Description

Keep track of the tasks waiting for the availability of a buffer generation, either to read it or to write it.

When we write into a buffer, we generate a new version of it (think "SSA")

Definition at line 33 of file buffer_customer.hpp.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 cl::sycl::detail::buffer_customer::buffer_customer (buffer_base & *buf,* bool *is_write_access*) [inline]

Definition at line 58 of file buffer_customer.hpp.

```
00059     : buf { buf },  write_access { is_write_access },
00060       ready_to_use { false }, user_number { 0 } {
00061   }
```

10.3.3 Member Function Documentation

10.3.3.1 void cl::sycl::detail::buffer_customer::add (std::shared_ptr< task > *task,* bool *is_write_access*) [inline]

Add a new task as a customer of the buffer generation.

Definition at line 74 of file buffer_customer.hpp.

References TRISYCL_DUMP_T.

```
00074                                                       {
00075     write_access = is_write_access;
00076     user_number++;
00077     TRISYCL_DUMP_T("buffer_customer::add() now user_number = " <<
      user_number);
00078   }
```

10.3.3.2 void cl::sycl::detail::buffer_customer::notify_ready ( ) [inline]

Notify the customer tasks this buffer generation is ready to use.

Definition at line 125 of file buffer_customer.hpp.

References TRISYCL_DUMP_T.

```
00125                       {
00126     {
00127       std::unique_lock<std::mutex> ul { ready_mutex };
00128       // \todo This lock can be avoided if ready_to_use is atomic
00129       ready_to_use = true;
00130     }
00131     TRISYCL_DUMP_T("buffer_customer::notify_ready()");
00132     ready_cv.notify_all();
00133   }
```

10.3.3.3 void cl::sycl::detail::buffer_customer::release ( ) [inline]

Release the buffer generation usage by a kernel task.

Definition at line 91 of file buffer_customer.hpp.

References TRISYCL_DUMP_T.

```
00091                   {
00092     user_number--;
00093     TRISYCL_DUMP_T("buffer_customer::release() now user_number = " <<
      user_number);
00094     if (user_number == 0) {
00095       /* If there is no task using this generation of the buffer, first
00096          notify the host accessors waiting for it, if any */
00097       released_cv.notify_all();
00098
00099       /* And then make the next generation ready if any. Note that if the
00100          SYCL program is race condition-free, there should be no host
00101          accessor waiting for a generation which is not the last one...
00102
00103          \todo: add some SYCL semantics runtime verification
00104       */
00105       if (next_generation)
00106         next_generation->notify_ready();
00107     }
00108     // \todo Can we have UserNumber increasing again?
00109   }
```

10.3.3.4 void cl::sycl::detail::buffer_customer::set_next_generation (std::shared_ptr< buffer_customer > *bc*) [inline]

Set the next generation of the buffer after this.

Todo Refactor this with an lock-free list?

Definition at line 68 of file buffer_customer.hpp.

```
00068                                                       {
00069     next_generation = bc;
00070   }
```

10.3.3.5 void cl::sycl::detail::buffer_customer::wait ( ) [inline]

Wait for the buffer generation to be ready to use by a kernel task.

Definition at line 82 of file buffer_customer.hpp.

References ready_to_use.

```
00082               {
00083     {
00084       std::unique_lock<std::mutex> ul { ready_mutex };
00085       ready_cv.wait(ul, [&] { return ready_to_use; });
00086     }
00087   }
```

10.3.3.6 void cl::sycl::detail::buffer_customer::wait_released ( ) [inline]

Wait for the release of the buffer generation before the host can use it.

Definition at line 115 of file buffer_customer.hpp.

References TRISYCL_DUMP_T.

```
00115                         {
00116     TRISYCL_DUMP_T("buffer_customer::wait_released() user_number = " <<
      user_number);
00117     {
00118       std::unique_lock<std::mutex> ul { released_mutex };
00119       released_cv.wait(ul, [&] { return user_number == 0; });
00120     }
00121   }
```

10.3.4 Member Data Documentation

10.3.4.1 buffer_base& cl::sycl::detail::buffer_customer::buf [private]

The considered buffer.

Todo Do we need to keep it?

Definition at line 36 of file buffer_customer.hpp.

10.3.4.2 std::shared_ptr<buffer_customer> cl::sycl::detail::buffer_customer::next_generation [private]

At some point use lock free list for this inside buffer_base.

Definition at line 38 of file buffer_customer.hpp.

10.3.4.3 std::condition_variable cl::sycl::detail::buffer_customer::ready_cv [private]

Definition at line 44 of file buffer_customer.hpp.

10.3.4.4 std::mutex cl::sycl::detail::buffer_customer::ready_mutex [private]

To protect the access to the condition variable.

Definition at line 46 of file buffer_customer.hpp.

10.3.4.5 bool cl::sycl::detail::buffer_customer::ready_to_use [private]

Definition at line 42 of file buffer_customer.hpp.

Referenced by wait().

10.3.4.6 std::condition_variable cl::sycl::detail::buffer_customer::released_cv [private]

To signal when the buffer generation is no longer used from the accelerator side and can be used for example through a host accessor.

Definition at line 52 of file buffer_customer.hpp.

10.3.4.7 std::mutex cl::sycl::detail::buffer_customer::released_mutex [private]

To protect the access to the condition variable.

Definition at line 54 of file buffer_customer.hpp.

10.3.4.8 std::atomic<unsigned int> cl::sycl::detail::buffer_customer::user_number [private]

Count the number of accelerator-side usage of this buffer generation.

Definition at line 48 of file buffer_customer.hpp.

10.3.4.9 bool cl::sycl::detail::buffer_customer::write_access [private]

Todo Needed?

Definition at line 40 of file buffer_customer.hpp.

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

- include/CL/sycl/buffer/detail/buffer_customer.hpp

10.4 handler_event Class Reference

Handler event.

`#include <handler_event.hpp>`

10.4.1 Detailed Description

Handler event.

Todo To be implemented

Todo To be implemented

Definition at line 19 of file handler_event.hpp.

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

- include/CL/sycl/handler_event.hpp

10.5 cl::sycl::info::param_traits< T, Param > Class Template Reference

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

`#include <param_traits.hpp>`

10.5.1 Detailed Description

template<typename T, T Param>class cl::sycl::info::param_traits< T, Param >

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Definition at line 20 of file param_traits.hpp.

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

- include/CL/sycl/info/param_traits.hpp

10.6 cl::sycl::detail::task Struct Reference

The abstraction to represent SYCL tasks executing inside command_group.

`#include <task.hpp>`

Inheritance diagram for cl::sycl::detail::task:

Collaboration diagram for cl::sycl::detail::task:

Public Member Functions

- void schedule (std::function< void(void)> f)

  Add a new task to the task graph and schedule for execution.
- void acquire_buffers ()
- void release_buffers ()
- template<typename T , std::size_t dimensions, access::mode mode, access::target target = access::global_buffer>
  void add (AccessorImpl< T, dimensions, mode, target > &a)

  Register an accessor to this task.

Public Attributes

- std::vector< std::shared_ptr< buffer_customer > > buffers

  The buffers that are used by this task.

10.6.1 Detailed Description

The abstraction to represent SYCL tasks executing inside command_group.

"enable_shared_from_this" allows to access the shared_ptr behind the scene.

Definition at line 29 of file task.hpp.

10.6.2 Member Function Documentation

10.6.2.1 void cl::sycl::detail::task::acquire_buffers ( ) [inline]

Definition at line 65 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
00065                           {
00066     TRISYCL_DUMP_T("acquire_buffers()");
00067     for (auto &b : buffers)
00068       b->wait();
00069   }
```

Here is the caller graph for this function:

10.6.2.2 template<typename T , std::size_t dimensions, access::mode mode, access::target target = access::global_buffer> void cl::sycl::detail::task::add (AccessorImpl< T, dimensions, mode, target > & *a*) [inline]

Register an accessor to this task.

This is how the dependency graph is incrementally built.

Definition at line 87 of file task.hpp.

References cl::sycl::detail::buffer_base::get_buffer_customer().

```
00087                                                                   {
00088     auto bc = buffer_base::get_buffer_customer(a);
00089     // Add the task as a new client for the buffer customer of the accessor
00090     bc->add(shared_from_this(), a.isWriteAccess());
00091     buffers.push_back(bc);
00092   }
```

Here is the call graph for this function:

10.6.2.3 void cl::sycl::detail::task::release_buffers ( ) [inline]

Definition at line 72 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
00072                           {
00073     TRISYCL_DUMP_T("release_buffers()");
00074     for (auto &b : buffers)
00075       b->release();
00076   }
```

Here is the caller graph for this function:

10.6.2.4 void cl::sycl::detail::task::schedule (std::function< void(void)> *f*) [inline]

Add a new task to the task graph and schedule for execution.

To keep a copy of the task shared_ptr after the end of the command group, capture it by copy in the following lambda. This should be easier in C++17 with move semantics on capture

Definition at line 35 of file task.hpp.

References acquire_buffers(), release_buffers(), and TRISYCL_DUMP_T.

```
00035                                         {
00036     /** To keep a copy of the task shared_ptr after the end of the command
00037         group, capture it by copy in the following lambda. This should be
00038         easier in C++17 with move semantics on capture
00039     */
00040     auto task = shared_from_this();
00041     auto execution = [=] {
00042       // Wait for the required buffers to be ready
00043       task->acquire_buffers();
00044       TRISYCL_DUMP_T("Execute the kernel");
00045       // Execute the kernel
00046       f();
00047       // Release the required buffers for other uses
```

```
00048       task->release_buffers();
00049       TRISYCL_DUMP_T("Exit");
00050     };
00051 #if TRISYCL_ASYNC
00052     /* If in asynchronous execution mode, execute the functor in a new
00053        thread */
00054     std::thread thread(execution);
00055     TRISYCL_DUMP_T("Started");
00056     // Detach the thread since it will synchronize by its own means
00057     thread.detach();
00058 #else
00059     // Just a synchronous execution otherwise
00060     execution();
00061 #endif
00062   }
```

Here is the call graph for this function:

10.6.3 Member Data Documentation

10.6.3.1 std::vector<std::shared_ptr<buffer_customer> > cl::sycl::detail::task::buffers

The buffers that are used by this task.

Definition at line 32 of file task.hpp.

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

- include/CL/sycl/command_group/detail/task.hpp

Chapter 11

File Documentation

11.1 include/CL/sycl.hpp File Reference

```
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/address_space.hpp"
#include "CL/sycl/buffer.hpp"
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/error_handler.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/group.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/image.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
#include "CL/sycl/vec.hpp"
```

Include dependency graph for sycl.hpp:

11.2 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     For more information on this project and to access to the source of
00012     this file, look at https://github.com/amd/triSYCL
00013
00014     The Doxygen version of the API in
00015     http://amd.github.io/triSYCL/Doxygen/SYCL/html and
00016     http://amd.github.io/triSYCL/Doxygen/SYCL/SYCL-API-refman.pdf
```

```
00017
00018     The Doxygen version of the implementation itself is in
00019     http://amd.github.io/triSYCL/Doxygen/triSYCL/html and
00020     http://amd.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf
00021
00022
00023     Ronan.Keryell at AMD point com
00024     Ronan at keryell dot FR
00025
00026     Copyright 2014--2015 Advanced Micro Devices, Inc.
00027
00028     This file is distributed under the University of Illinois Open Source
00029     License. See LICENSE.TXT for details.
00030 */
00031
00032
00033 /** Some global triSYCL configuration */
00034 #include "CL/sycl/detail/global_config.hpp"
00035 #include "CL/sycl/detail/default_classes.hpp"
00036
00037
00038 /* All the SYCL components, one per file */
00039 #include "CL/sycl/access.hpp"
00040 #include "CL/sycl/accessor.hpp"
00041 #include "CL/sycl/address_space.hpp"
00042 #include "CL/sycl/buffer.hpp"
00043 #include "CL/sycl/context.hpp"
00044 #include "CL/sycl/device.hpp"
00045 #include "CL/sycl/device_selector.hpp"
00046 #include "CL/sycl/error_handler.hpp"
00047 //#include "CL/sycl/event.hpp"
00048 #include "CL/sycl/exception.hpp"
00049 #include "CL/sycl/group.hpp"
00050 #include "CL/sycl/handler.hpp"
00051 #include "CL/sycl/id.hpp"
00052 #include "CL/sycl/image.hpp"
00053 #include "CL/sycl/item.hpp"
00054 #include "CL/sycl/nd_item.hpp"
00055 #include "CL/sycl/nd_range.hpp"
00056 #include "CL/sycl/parallelism.hpp"
00057 #include "CL/sycl/platform.hpp"
00058 #include "CL/sycl/queue.hpp"
00059 #include "CL/sycl/range.hpp"
00060 #include "CL/sycl/vec.hpp"
00061
00062
00063 /*
00064     # Some Emacs stuff:
00065     ### Local Variables:
00066     ### ispell-local-dictionary: "american"
00067     ### eval: (flyspell-prog-mode)
00068     ### End:
00069 */
```

11.3 include/CL/sycl/access.hpp File Reference

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::access

  Describe the type of access by kernels.

Enumerations

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

  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 }

  The target enumeration describes the type of object to be accessed via the accessor.
- enum cl::sycl::access::fence_space : char { cl::sycl::access::fence_space::local_space, cl::sycl::access::fence_space::global_space, cl::sycl::access::fence_space::global_and_local }

  Precise the address space a barrier needs to act on.

11.4 access.hpp

```
00001 #ifndef TRISYCL_SYCL_ACCESS_HPP
00002 #define TRISYCL_SYCL_ACCESS_HPP
00003
00004 /** \file The OpenCL SYCL access naming space
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 // SYCL dwells in the cl::sycl namespace
00013 namespace cl {
00014 namespace sycl {
00015
00016 /** \addtogroup data Data access and storage in SYCL
00017
00018     @{
00019 */
00020
00021 /** Describe the type of access by kernels.
00022
00023     \todo This values should be normalized to allow separate compilation
00024     with different implementations?
00025 */
00026 namespace access {
00027   /* By using "enum mode" here instead of "enum struct mode", we have for
00028      example "write" appearing both as cl::sycl::access::mode::write and
00029      cl::sycl::access::write, instead of only the last one. This seems
00030      more conform to the specification. */
00031
00032   /// This describes the type of the access mode to be used via accessor
00033   enum mode {
00034     read = 42, ///< Read-only access. Insist on the fact that read_write != read + write
00035     write, ///< Write-only access, but previous content *not* discarded
00036     read_write, ///< Read and write access
00037     discard_write, ///< Write-only access and previous content discarded
00038     discard_read_write, ///< Read and write access and previous content discarded
00039     atomic ///< Atomic access
00040   };
00041
00042
00043   /** The target enumeration describes the type of object to be accessed
00044       via the accessor
00045   */
00046   enum target {
00047     global_buffer = 2014, //< Just pick a random number...
00048     constant_buffer,
00049     local,
00050     image,
00051     host_buffer,
00052     host_image,
00053     image_array
00054   };
00055
00056
00057   /** Precise the address space a barrier needs to act on
00058   */
00059   enum class fence_space : char {
00060     local_space,
00061     global_space,
00062     global_and_local
00063   };
00064
00065 }
00066
00067 /// @} End the data Doxygen group
00068
```

```
00069 }
00070 }
00071
00072 /*
00073     # Some Emacs stuff:
00074     ### Local Variables:
00075     ### ispell-local-dictionary: "american"
00076     ### eval: (flyspell-prog-mode)
00077     ### End:
00078 */
00079
00080 #endif // TRISYCL_SYCL_ACCESS_HPP
```

11.5 include/CL/sycl/accessor/detail/accessor.hpp File Reference

```
#include <cstddef>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
```

Include dependency graph for accessor.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::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::detail::accessor< T, Dimensions, Mode, Target >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::detail

11.6 accessor.hpp

```
00001 #ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL accessor<> detail behind the scene
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include <boost/multi_array.hpp>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/detail/debug.hpp"
00018 #include "CL/sycl/id.hpp"
00019 #include "CL/sycl/item.hpp"
00020 #include "CL/sycl/nd_item.hpp"
00021
```

```
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 // Forward declaration of detail::buffer for use in accessor
00027 template <typename T, std::size_t Dimensions> struct buffer;
00028
00029 /** \addtogroup data Data access and storage in SYCL
00030     @{
00031 */
00032
00033 /** The accessor abstracts the way buffer data are accessed inside a
00034     kernel in a multidimensional variable length array way.
00035
00036     This implementation rely on boost::multi_array to provides this nice
00037     syntax and behaviour.
00038
00039     Right now the aim of this class is just to access to the buffer in a
00040     read-write mode, even if capturing the multi_array_ref from a lambda
00041     make it const (since in some example we have lambda with [=] and
00042     without mutable). The access::mode is not used yet.
00043 */
00044 template <typename T,
00045           std::size_t Dimensions,
00046           access::mode Mode,
00047           access::target Target /* = access::global_buffer */>
00048 struct accessor : public detail::debug<accessor<T,
00049                                                 Dimensions,
00050                                                 Mode,
00051                                                 Target>> {
00052   detail::buffer<T, Dimensions> *buf;
00053   // The implementation is a multi_array_ref wrapper
00054   using array_view_type = boost::multi_array_ref<T, Dimensions>;
00055   // \todo Do we need this if we have a reference on buf?
00056   array_view_type array;
00057
00058   // The same type but writable
00059   using writable_array_view_type = typename
      std::remove_const<array_view_type>::type;
00060
00061   // \todo in the specification: store the dimension for user request
00062   static const auto dimensionality = Dimensions;
00063   // \todo in the specification: store the types for user request as STL
00064   // or C++AMP
00065   using element = T;
00066   using value_type = T;
00067
00068
00069   /// The only way to construct an accessor is from an existing buffer
00070   // \todo fix the specification to rename target that shadows template parm
00071   accessor(detail::buffer<T, Dimensions> &target_buffer) :
00072     buf { &target_buffer }, array { target_buffer.access } {
00073 #if TRISYCL_ASYNC
00074     if (Target == access::target::host_buffer) {
00075       // A host accessor needs to be declared *outside* a command_group
00076       assert(current_task == nullptr);
00077       // Wait for the latest generation of the buffer before the host can use it
00078       buffer_base::wait(target_buffer);
00079     }
00080     else {
00081       // A host non-host accessor needs to be declared *inside* a command_group
00082       assert(current_task != nullptr);
00083       // Register the accessor to the task dependencies
00084       current_task->add(*this);
00085     }
00086 #endif
00087   }
00088
00089
00090   /** Use the accessor with integers à la [][][]
00091
00092       Use array_view_type::reference instead of auto& because it does not
00093       work in some dimensions.
00094    */
00095   typename array_view_type::reference operator[](std::size_t index) {
00096     return array[index];
00097   }
00098
00099
00100   /// To use the accessor in with [id<>]
00101   auto &operator[](id<dimensionality> index) {
00102     return (const_cast<writable_array_view_type &>(array))(index);
00103   }
00104
00105
00106   /** To use the accessor in with [id<>]
00107
```

```
00108       This is when we access to accessor[] that we override the const
00109       if any
00110   */
00111   auto &operator[](id<dimensionality> index) const {
00112     return (const_cast<writable_array_view_type &>(array))(index);
00113   }
00114
00115
00116   /// To use an accessor with [item<>]
00117   auto &operator[](item<dimensionality> index) {
00118     return (*this)[index.get()];
00119   }
00120
00121
00122   /// To use an accessor with [item<>]
00123   auto &operator[](item<dimensionality> index) const {
00124     return (*this)[index.get()];
00125   }
00126
00127
00128   /** To use an accessor with an [nd_item<>]
00129
00130       \todo Add in the specification because used by HPC-GPU slide 22
00131   */
00132   auto &operator[](nd_item<dimensionality> index) {
00133     return (*this)[index.get_global()];
00134   }
00135
00136   /** To use an accessor with an [nd_item<>]
00137
00138       \todo Add in the specification because used by HPC-GPU slide 22
00139   */
00140   auto &operator[](nd_item<dimensionality> index) const {
00141     return (*this)[index.get_global()];
00142   }
00143
00144
00145   /// Get the buffer used to create the accessor
00146   detail::buffer<T, Dimensions> &get_buffer() {
00147     return *buf;
00148   }
00149
00150
00151   /// Test if the accessor as a write access right
00152   constexpr bool is_write_access() const {
00153     /** \todo to move in the access::mode enum class and add to the
00154         specification ? */
00155     return Mode == access::mode::write
00156       || Mode == access::mode::read_write
00157       || Mode == access::mode::discard_write
00158       || Mode == access::mode::discard_read_write;
00159   }
00160
00161 };
00162
00163 /// @} End the data Doxygen group
00164
00165 }
00166 }
00167 }
00168
00169 /*
00170     # Some Emacs stuff:
00171     ### Local Variables:
00172     ### ispell-local-dictionary: "american"
00173     ### eval: (flyspell-prog-mode)
00174     ### End:
00175 */
00176
00177 #endif // TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_HPP
```

11.7 include/CL/sycl/accessor.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/accessor.hpp"
```

Include dependency graph for accessor.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::buffer< T, Dimensions, Allocator >

  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, AccessMode, Target >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.8 accessor.hpp

```
00001 #ifndef TRISYCL_SYCL_ACCESSOR_HPP
00002 #define TRISYCL_SYCL_ACCESSOR_HPP
00003
00004 /** \file The OpenCL SYCL accessor<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/access.hpp"
00015 #include "CL/sycl/accessor/detail/accessor.hpp"
00016
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 template <typename T, std::size_t Dimensions, typename Allocator> struct buffer;
00022
00023 class handler;
00024
00025 /** \addtogroup data Data access and storage in SYCL
00026     @{
00027 */
00028
00029 /** The accessor abstracts the way buffer data are accessed inside a
00030     kernel in a multidimensional variable length array way.
00031
00032     \todo Implement it for images according so section 3.3.4.5
00033 */
00034 template <typename DataType,
00035           std::size_t Dimensions,
00036           access::mode AccessMode,
00037           access::target Target = access::global_buffer>
00038 struct accessor : detail::accessor<DataType, Dimensions, AccessMode, Target> {
00039   /// \todo in the specification: store the dimension for user request
00040   static constexpr auto dimensionality = Dimensions;
00041   using value_type = DataType;
00042   using reference = value_type&;
00043   using const_reference = const value_type&;
00044
00045   // Inherit of the constructors to have accessor constructor from detail
00046   using detail::accessor<DataType, Dimensions, AccessMode, Target>::accessor
   ;
00047
00048   /** Construct a buffer accessor from a buffer using a command group
00049       handler object from the command group scope
00050
00051       Constructor only available for access modes global_buffer,
00052       host_buffer, constant_buffer (see Table 3.25).
00053
00054       access_target defines the form of access being obtained. See Table
00055       3.26.
00056   */
00057   template <typename Allocator>
```

```
00058   accessor(buffer<DataType, Dimensions, Allocator> &
      target_buffer,
00059           handler &command_group_handler) :
00060     detail::accessor<DataType, Dimensions, AccessMode, Target> { *target_buffer.
      implementation } {}
00061
00062
00063   /** Construct a buffer accessor from a buffer given a specific range for
00064       access permissions and an offset that provides the starting point
00065       for the access range using a command group handler object from the
00066       command group scope
00067
00068       This accessor limits the processing of the buffer to the [offset,
00069       offset+range[ for every dimension. Any other parts of the buffer
00070       will be unaffected.
00071
00072       Constructor only available for access modes global_buffer,
00073       host_buffer or constant_buffer (see Table 3.25). access_target
00074       defines the form of access being obtained (see Table 3.26).
00075
00076       This accessor is recommended for discard-write and discard read
00077       write access modes, when the unaffected parts of the processing
00078       should be retained.
00079   */
00080   template <typename Allocator>
00081   accessor(buffer<DataType, Dimensions, Allocator> &target_buffer,
00082            handler &command_group_handler,
00083            range<Dimensions> offset,
00084            range<Dimensions> range) {
00085     detail::unimplemented();
00086   }
00087
00088
00089   /** Construct an accessor of dimensions Dimensions with elements of type
00090       DataType using the passed range to specify the size in each
00091       dimension
00092
00093       It needs as a parameter a command group handler object from the
00094       command group scope. Constructor only available if AccessMode is
00095       local, see Table 3.25.
00096   */
00097   accessor(range<Dimensions> allocation_size,
00098            handler &command_group_handler) {
00099     detail::unimplemented();
00100   }
00101
00102 };
00103
00104 /// @} End the data Doxygen group
00105
00106 }
00107 }
00108
00109 /*
00110     # Some Emacs stuff:
00111     ### Local Variables:
00112     ### ispell-local-dictionary: "american"
00113     ### eval: (flyspell-prog-mode)
00114     ### End:
00115 */
00116
00117 #endif // TRISYCL_SYCL_ACCESSOR_HPP
```

11.9 include/CL/sycl/address_space/detail/address_space.hpp File Reference

Implement OpenCL address spaces in SYCL with C++-style.

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

Classes

- struct cl::sycl::detail::opencl_type< T, AS >

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::opencl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct cl::sycl::detail::opencl_type< T, generic_address_space >

Add an attribute for __generic address space. More...

- struct cl::sycl::detail::opencl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct cl::sycl::detail::opencl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct cl::sycl::detail::opencl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

  Implementation of a variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Typedefs

- template<typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

11.9.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Ronan at Keryell point FR

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

Definition in file address_space.hpp.

11.10 address_space.hpp

```
00001 #ifndef TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
00002 #define TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
00003
00004 /** \file
00005
00006     Implement OpenCL address spaces in SYCL with C++-style.
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016 namespace detail {
00017
00018 /** \addtogroup address_spaces
00019     @{
00020 */
00021
00022 /** Generate a type with some real OpenCL 2 attribute if we are on an
00023     OpenCL device
00024
00025     In the general case, do not add any OpenCL address space qualifier */
00026 template <typename T, address_space AS>
00027 struct opencl_type {
00028   using type = T;
00029 };
00030
00031 /// Add an attribute for __constant address space
00032 template <typename T>
00033 struct opencl_type<T, constant_address_space> {
00034   using type = T
00035 #ifdef __SYCL_DEVICE_ONLY__
00036     /* Put the address space qualifier after the type so that we can
00037        construct pointer type with qualifier */
00038     __constant
00039 #endif
00040     ;
00041 };
00042
00043 /// Add an attribute for __generic address space
00044 template <typename T>
00045 struct opencl_type<T, generic_address_space> {
00046   using type = T
00047 #ifdef __SYCL_DEVICE_ONLY__
00048     /* Put the address space qualifier after the type so that we can
00049        construct pointer type with qualifier */
00050     __generic
00051 #endif
00052     ;
00053 };
00054
00055 /// Add an attribute for __global address space
00056 template <typename T>
00057 struct opencl_type<T, global_address_space> {
00058   using type = T
00059 #ifdef __SYCL_DEVICE_ONLY__
00060     /* Put the address space qualifier after the type so that we can
00061        construct pointer type with qualifier */
00062     __global
00063 #endif
00064     ;
00065 };
00066
00067 /// Add an attribute for __local address space
00068 template <typename T>
00069 struct opencl_type<T, local_address_space> {
00070   using type = T
00071 #ifdef __SYCL_DEVICE_ONLY__
00072     /* Put the address space qualifier after the type so that we can
00073        construct pointer type with qualifier */
00074     __local
00075 #endif
00076     ;
00077 };
00078
00079 /// Add an attribute for __private address space
00080 template <typename T>
00081 struct opencl_type<T, private_address_space> {
00082   using type = T
00083 #ifdef __SYCL_DEVICE_ONLY__
00084     /* Put the address space qualifier after the type so that we can
```

```
00085       construct pointer type with qualifier */
00086     __private
00087 #endif
00088     ;
00089 };
00090
00091
00092 /* Forward declare some classes to allow some recursion in conversion
00093    operators */
00094 template <typename SomeType, address_space SomeAS>
00095 struct address_space_array;
00096
00097 template <typename SomeType, address_space SomeAS>
00098 struct address_space_fundamental;
00099
00100 template <typename SomeType, address_space SomeAS>
00101 struct address_space_object;
00102
00103 template <typename SomeType, address_space SomeAS>
00104 struct address_space_ptr;
00105
00106 /** Dispatch the address space implementation according to the requested type
00107
00108     \param T is the type of the object to be created
00109
00110     \param AS is the address space to place the object into or to point to
00111     in the case of a pointer type
00112 */
00113 template <typename T, address_space AS>
00114 using addr_space =
00115   typename std::conditional<std::is_pointer<T>::value,
00116                             address_space_ptr<T, AS>,
00117   typename std::conditional<std::is_class<T>::value,
00118                             address_space_object<T, AS>,
00119   typename std::conditional<std::is_array<T>::value,
00120                             address_space_array<T, AS>,
00121                             address_space_fundamental<T, AS>
00122   >::type>::type>::type;
00123
00124
00125 /** Implementation of the base infrastructure to wrap something in an
00126     OpenCL address space
00127
00128     \param T is the type of the basic stuff to be created
00129
00130     \param AS is the address space to place the object into
00131
00132     \todo Verify/improve to deal with const/volatile?
00133 */
00134 template <typename T, address_space AS>
00135 struct address_space_base {
00136   /** Store the base type of the object
00137
00138       \todo Add to the specification
00139   */
00140   using type = T;
00141
00142   /** Store the base type of the object with OpenCL address space modifier
00143
00144       \todo Add to the specification
00145   */
00146   using opencl_type = typename opencl_type<T, AS>::type;
00147
00148   /** Set the address_space identifier that can be queried to know the
00149       pointer type */
00150   static auto constexpr address_space = AS;
00151
00152 };
00153
00154
00155 /** Implementation of a variable with an OpenCL address space
00156
00157     \param T is the type of the basic object to be created
00158
00159     \param AS is the address space to place the object into
00160 */
00161 template <typename T, address_space AS>
00162 struct address_space_variable : public address_space_base<T, AS> {
00163   /** Store the base type of the object with OpenCL address space modifier
00164
00165       \todo Add to the specification
00166   */
00167   using opencl_type = typename opencl_type<T, AS>::type;
00168
00169   /// Keep track of the base class as a short-cut
00170   using super = address_space_base<T, AS>;
00171
```

```
00172 protected:
00173
00174   /* C++11 helps a lot to be able to have the same constructors as the
00175      parent class here
00176
00177      \todo Add this to the list of required C++11 features needed for SYCL
00178   */
00179   opencl_type variable;
00180
00181 public:
00182
00183   /** Allow to create an address space version of an object or to convert
00184       one to be used by the classes inheriting by this one because it is
00185       not possible to directly initialize a base class member in C++ */
00186   address_space_variable(const T & v) : variable(v) { }
00187
00188
00189   /// Put back the default constructors canceled by the previous definition
00190   address_space_variable() = default;
00191
00192
00193   /** Conversion operator to allow a address_space_object<T> to be used
00194       as a T so that all the methods of a T and the built-in operators for
00195       T can be used on a address_space_object<T> too.
00196
00197       Use opencl_type so that if we take the address of it, the address
00198       space is kept.
00199   */
00200   operator opencl_type & () { return variable; }
00201
00202 };
00203
00204
00205 /** Implementation of a fundamental type with an OpenCL address space
00206
00207     \param T is the type of the basic object to be created
00208
00209     \param AS is the address space to place the object into
00210
00211     \todo Verify/improve to deal with const/volatile?
00212 */
00213 template <typename T, address_space AS>
00214 struct address_space_fundamental : public address_space_variable<T, AS> {
00215   /// Keep track of the base class as a short-cut
00216   using super = address_space_variable<T, AS>;
00217
00218   /// Inherit from base class constructors
00219   using super::address_space_variable;
00220
00221
00222   /** Also request for the default constructors that have been disabled by
00223       the declaration of another constructor
00224
00225       This ensures for example that we can write
00226       \code
00227         generic<float *> q;
00228       \endcode
00229       without initialization.
00230   */
00231   address_space_fundamental() = default;
00232
00233
00234   /** Allow for example assignment of a global<float> to a priv<double>
00235       for example
00236
00237      Since it needs 2 implicit conversions, it does not work with the
00238      conversion operators already define, so add 1 more explicit
00239      conversion here so that the remaining implicit conversion can be
00240      found by the compiler.
00241
00242      Strangely
00243      \code
00244      template <typename SomeType, address_space SomeAS>
00245      address_space_base(addr_space<SomeType, SomeAS>& v)
00246      : variable(SomeType(v)) { }
00247      \endcode
00248      cannot be used here because SomeType cannot be inferred. So use
00249      address_space_base<> instead
00250
00251      Need to think further about it...
00252   */
00253   template <typename SomeType, cl::sycl::address_space SomeAS>
00254   address_space_fundamental(
      address_space_fundamental<SomeType, SomeAS>& v)
00255   {
00256     /* Strangely I cannot have it working in the initializer instead, for
00257        some cases */
```

```
00258     super::variable = SomeType(v);
00259   }
00260
00261 };
00262
00263
00264 /** Implementation for an OpenCL address space pointer
00265
00266     \param T is the pointer type
00267
00268     Note that if \a T is not a pointer type, it is an error.
00269
00270     All the address space pointers inherit from it, which makes trivial
00271     the implementation of cl::sycl::multi_ptr<T, AS>
00272 */
00273 template <typename T, address_space AS>
00274 struct address_space_ptr : public address_space_fundamental<T, AS> {
00275   // Verify that \a T is really a pointer
00276   static_assert(std::is_pointer<T>::value,
00277                 "T must be a pointer type");
00278
00279   /// Keep track of the base class as a short-cut
00280   using super = address_space_fundamental<T, AS>;
00281
00282   /// Inherit from base class constructors
00283   using super::address_space_fundamental;
00284
00285 };
00286
00287
00288 /** Implementation of an array variable with an OpenCL address space
00289
00290     \param T is the type of the basic object to be created
00291
00292     \param AS is the address space to place the object into
00293 */
00294 template <typename T, address_space AS>
00295 struct address_space_array : public address_space_variable<T, AS>
      {
00296   /// Keep track of the base class as a short-cut
00297   using super = address_space_variable<T, AS>;
00298
00299   /// Inherit from base class constructors
00300   using super::address_space_variable;
00301
00302
00303   /** Allow to create an address space array from an array
00304    */
00305   address_space_array(const T &array) {
00306     std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00307   };
00308
00309
00310   /** Allow to create an address space array from an initializer list
00311
00312       \todo Extend to more than 1 dimension
00313   */
00314   address_space_array(std::initializer_list<std::remove_extent_t<T>> list) {
00315     std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00316   };
00317
00318 };
00319
00320
00321 /** Implementation of an object type with an OpenCL address space
00322
00323     \param T is the type of the basic object to be created
00324
00325     \param AS is the address space to place the object into
00326
00327     The class implementation is just inheriting of T so that all methods
00328     and non-member operators on T work also on address_space_object<T>
00329
00330     \todo Verify/improve to deal with const/volatile?
00331
00332     \todo what about T having some final methods?
00333 */
00334 template <typename T, address_space AS>
00335 struct address_space_object : public opencl_type<T, AS>::type,
00336                               public address_space_base<T, AS> {
00337   /** Store the base type of the object with OpenCL address space modifier
00338
00339       \todo Add to the specification
00340   */
00341   using opencl_type = typename opencl_type<T, AS>::type;
00342
00343   /* C++11 helps a lot to be able to have the same constructors as the
```

```
00344      parent class here but with an OpenCL address space
00345
00346      \todo Add this to the list of required C++11 features needed for SYCL
00347   */
00348   using opencl_type::opencl_type;
00349
00350   /** Allow to create an address space version of an object or to
00351       convert one */
00352   address_space_object(T && v) : opencl_type(v) { }
00353
00354   /** Conversion operator to allow a address_space_object<T> to be used
00355       as a T so that all the methods of a T and the built-in operators for
00356       T can be used on a address_space_object<T> too.
00357
00358       Use opencl_type so that if we take the address of it, the address
00359       space is kept. */
00360   operator opencl_type & () { return *this; }
00361
00362 };
00363
00364 /// @} End the address_spaces Doxygen group
00365
00366 }
00367 }
00368 }
00369
00370 /*
00371     # Some Emacs stuff:
00372     ### Local Variables:
00373     ### ispell-local-dictionary: "american"
00374     ### eval: (flyspell-prog-mode)
00375     ### End:
00376 */
00377
00378 #endif // TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
```

11.11 include/CL/sycl/address_space.hpp File Reference

Implement OpenCL address spaces in SYCL with C++-style.

`#include "CL/sycl/address_space/detail/address_space.hpp"`
Include dependency graph for address_space.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be an OpenCL constant pointer.
- template<typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be an OpenCL 2 generic pointer.
- template<typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

  Declare a variable to be an OpenCL global pointer.
- template<typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be an OpenCL local pointer.
- template<typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

Declare a variable to be an OpenCL private pointer.

- template<typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

11.11.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Ronan at Keryell point FR

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

Definition in file address_space.hpp.

11.12 address_space.hpp

```
00001 #ifndef TRISYCL_SYCL_ADDRESS_SPACE_HPP
00002 #define TRISYCL_SYCL_ADDRESS_SPACE_HPP
00003
00004 /** \file
00005
00006     Implement OpenCL address spaces in SYCL with C++-style.
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup address_spaces Dealing with OpenCL address spaces
00018     @{
00019 */
00020
00021 /** Enumerate the different OpenCL 2 address spaces */
00022 enum address_space {
00023   constant_address_space,
00024   generic_address_space,
00025   global_address_space,
00026   local_address_space,
00027   private_address_space,
00028 };
00029
00030 }
00031 }
00032 /// @} End the address_spaces Doxygen group
00033
00034
00035 #include "CL/sycl/address_space/detail/address_space.hpp"
00036
```

```
00037
00038 namespace cl {
00039 namespace sycl {
00040
00041 /** \addtogroup address_spaces
00042     @{
00043 */
00044
00045 /** Declare a variable to be an OpenCL constant pointer
00046
00047     \param T is the pointer type
00048
00049     Note that if \a T is not a pointer type, it is an error.
00050 */
00051 template <typename T>
00052 using constant = detail::addr_space<T, constant_address_space>
      ;
00053
00054
00055 /** Declare a variable to be an OpenCL 2 generic pointer
00056
00057     \param T is the pointer type
00058
00059     Note that if \a T is not a pointer type, it is an error.
00060 */
00061 template <typename T>
00062 using generic = detail::addr_space<T, generic_address_space>;
00063
00064
00065 /** Declare a variable to be an OpenCL global pointer
00066
00067     \param T is the pointer type
00068
00069     Note that if \a T is not a pointer type, it is an error.
00070 */
00071 template <typename T>
00072 using global = detail::addr_space<T, global_address_space>
      ;
00073
00074
00075 /** Declare a variable to be an OpenCL local pointer
00076
00077     \param T is the pointer type
00078
00079     Note that if \a T is not a pointer type, it is an error.
00080 */
00081 template <typename T>
00082 using local = detail::addr_space<T, local_address_space>;
00083
00084
00085 /** Declare a variable to be an OpenCL private pointer
00086
00087     \param T is the pointer type
00088
00089     Note that if \a T is not a pointer type, it is an error.
00090 */
00091 template <typename T>
00092 using priv = detail::addr_space<T, private_address_space>;
00093
00094
00095 /** A pointer that can be statically associated to any address-space
00096
00097     \param Pointer is the pointer type
00098
00099     \param AS is the address space to point to
00100
00101     Note that if \a Pointer is not a pointer type, it is an error.
00102 */
00103 template <typename Pointer, address_space AS>
00104 using multi_ptr = detail::address_space_ptr<Pointer, AS>;
00105
00106
00107 /** Construct a cl::sycl::multi_ptr<> with the right type
00108
00109     \param pointer is the address with its address space to point to
00110
00111     \todo Implement the case with a plain pointer
00112 */
00113 template <typename T, address_space AS>
00114 multi_ptr<T, AS> make_multi(multi_ptr<T, AS> pointer) {
00115   return pointer;
00116 }
00117
00118 }
00119 }
00120 /// @} End the parallelism Doxygen group
00121
```

```
00122 /*
00123     # Some Emacs stuff:
00124     ### Local Variables:
00125     ### ispell-local-dictionary: "american"
00126     ### eval: (flyspell-prog-mode)
00127     ### End:
00128 */
00129
00130 #endif // TRISYCL_SYCL_ADDRESS_SPACE_HPP
```

11.13 include/CL/sycl/buffer/detail/buffer.hpp File Reference

```
#include <cstddef>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::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

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.14 buffer.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
00002 #define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
00003
00004 /** \file The OpenCL SYCL buffer<> detail implementation
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include <boost/multi_array.hpp>
00015
00016 #include "CL/sycl/access.hpp"
00017 #include "CL/sycl/accessor/detail/accessor.hpp"
00018 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00019 #include "CL/sycl/range.hpp"
00020
00021 namespace cl {
00022 namespace sycl {
00023 namespace detail {
00024
00025 /** \addtogroup data Data access and storage in SYCL
00026     @{
00027 */
00028
00029 /** A SYCL buffer is a multidimensional variable length array (à la C99
00030     VLA or even Fortran before) that is used to store data to work on.
00031
```

```
00032     In the case we initialize it from a pointer, for now we just wrap the
00033     data with boost::multi_array_ref to provide the VLA semantics without
00034     any storage.
00035 */
00036 template <typename T,
00037           std::size_t Dimensions = 1>
00038 struct buffer : public detail::debug<buffer<T, Dimensions>>,
00039                 public detail::buffer_base {
00040   // Extension to SYCL: provide pieces of STL container interface
00041   using element = T;
00042   using value_type = T;
00043
00044   /** If some allocation is requested, it is managed by this multi_array
00045       to ease initialization from data */
00046   boost::multi_array<T, Dimensions> allocation;
00047   /** This is the multi-dimensional interface to the data that may point
00048       to either allocation in the case of storage managed by SYCL itself
00049       or to some other memory location in the case of host memory or
00050       storage<> abstraction use
00051   */
00052   boost::multi_array_ref<T, Dimensions> access;
00053
00054
00055   /// Create a new read-write buffer of size \param r
00056   buffer(range<Dimensions> const &r) : buffer_base { false },
00057                                        allocation { r },
00058                                        access { allocation }
00059                                        {}
00060
00061
00062   /** Create a new read-write buffer from \param host_data of size
00063       \param r without further allocation */
00064   buffer(T * host_data, range<Dimensions> r) :
      buffer_base { false },
00065                                                access { host_data, r }
00066                                                {}
00067
00068
00069   /** Create a new read-only buffer from \param host_data of size \param r
00070       without further allocation */
00071   buffer(const T * host_data, range<Dimensions> r) :
00072     /// \todo Need to solve this const buffer issue in a clean way
00073     buffer_base { true },
00074     access { const_cast<T *>(host_data), r }
00075     {}
00076
00077
00078   /// \todo
00079   //buffer(storage<T> &store, range<Dimensions> r)
00080
00081   /// Create a new allocated 1D buffer from the given elements
00082   template <typename Iterator>
00083   buffer(Iterator start_iterator, Iterator end_iterator) :
00084     buffer_base { false },
00085     // The size of a multi_array is set at creation time
00086     allocation { boost::extents[std::distance(start_iterator, end_iterator)] },
00087     access { allocation }
00088     {
00089       /* Then assign allocation since this is the only multi_array
00090          method with this iterator interface */
00091       allocation.assign(start_iterator, end_iterator);
00092     }
00093
00094
00095   /** Create a new buffer from an old one, with a new allocation
00096
00097       \todo Refactor the implementation to deal with buffer sharing with
00098       reference counting
00099   */
00100   buffer(const buffer<T, Dimensions> &b) :
      buffer_base { b.read_only },
00101                                            allocation { b.access },
00102                                            access { allocation }
00103                                            {}
00104
00105
00106   /** Create a new sub-buffer without allocation to have separate
00107       accessors later
00108
00109       \todo To implement and deal with reference counting
00110   buffer(buffer<T, Dimensions> b,
00111          index<Dimensions> base_index,
00112          range<Dimensions> sub_range)
00113   */
00114
00115   /// \todo Allow CLHPP objects too?
00116   ///
```

```
00117   /*
00118   buffer(cl_mem mem_object,
00119          queue from_queue,
00120          event available_event)
00121   */
00122
00123   // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00124
00125   /// Return an accessor of the required mode \param M
00126   /// \todo Remove if not used
00127   template <access::mode Mode,
00128             access::target Target = access::global_buffer>
00129   detail::accessor<T, Dimensions, Mode, Target>
      get_access() {
00130     return { *this };
00131   }
00132
00133 };
00134
00135 /// @} End the data Doxygen group
00136
00137 }
00138 }
00139 }
00140
00141 /*
00142     # Some Emacs stuff:
00143     ### Local Variables:
00144     ### ispell-local-dictionary: "american"
00145     ### eval: (flyspell-prog-mode)
00146     ### End:
00147 */
00148
00149 #endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
```

11.15 include/CL/sycl/buffer.hpp File Reference

```
#include <cassert>
#include <cstddef>
#include <iterator>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::buffer< T, Dimensions, Allocator >

  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

  The vector type to be used as SYCL vector.
- cl::sycl

11.16 buffer.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_HPP
```

```
00002 #define TRISYCL_SYCL_BUFFER_HPP
00003
00004 /** \file The OpenCL SYCL buffer<>
00005
00006     Ronan at Keryell point FR
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 <cstddef>
00014 #include <iterator>
00015 #include <memory>
00016
00017 #include "CL/sycl/access.hpp"
00018 #include "CL/sycl/accessor.hpp"
00019 #include "CL/sycl/buffer/detail/buffer.hpp"
00020 #include "CL/sycl/buffer_allocator.hpp"
00021 #include "CL/sycl/handler.hpp"
00022 #include "CL/sycl/id.hpp"
00023 #include "CL/sycl/range.hpp"
00024
00025 namespace cl {
00026 namespace sycl {
00027
00028 /** \addtogroup data Data access and storage in SYCL
00029     @{
00030 */
00031
00032 /** A SYCL buffer is a multidimensional variable length array (à la C99
00033     VLA or even Fortran before) that is used to store data to work on.
00034
00035     \todo We have some read-write buffers and some read-only buffers,
00036     according to the constructor called. So we could have some static
00037     checking for correctness with the accessors used, but we do not have a
00038     way in the specification to have a read-only buffer type for this.
00039
00040     \todo There is a naming inconsistency in the specification between
00041     buffer and accessor on T versus datatype
00042
00043     \todo Think about the need of an allocator when constructing a buffer
00044     from other buffers
00045 */
00046 template <typename T,
00047           std::size_t Dimensions = 1,
00048           typename Allocator = buffer_allocator<T>>
00049 struct buffer {
00050   /// The STL-like types
00051   using value_type = T;
00052   using reference = value_type&;
00053   using const_reference = const value_type&;
00054   using allocator_type = Allocator;
00055
00056   /** Point to the underlying buffer implementation that can be shared in
00057       the SYCL model */
00058   std::shared_ptr<detail::buffer<T, Dimensions>> implementation;
00059
00060   /** Use default constructors so that we can create a new buffer copy
00061       from another one, with either a l-value or an r-value (for
00062       std::move() for example).
00063
00064       Since we just copy the shared_ptr<> above, this is where/how the
00065       sharing magic is happening with reference counting in this case.
00066   */
00067   buffer() = default;
00068
00069
00070   /** Create a new read-write buffer with storage managed by SYCL
00071
00072       \param r defines the size
00073   */
00074   buffer(const range<Dimensions> &r, Allocator allocator = {})
00075     : implementation { new detail::buffer<T, Dimensions> { r } } {}
00076
00077
00078   /** Create a new read-write buffer with associated host memory
00079
00080       \param host_data points to the storage and values used by the buffer
00081
00082       \param r defines the size
00083   */
00084   buffer(T * host_data, range<Dimensions> r, Allocator allocator = {})
00085     : implementation { new detail::buffer<T, Dimensions> { host_data, r } } {}
00086
00087
00088   /** Create a new read-only buffer with associated host memory
```

```
00089
00090       \param host_data points to the storage and values used by the buffer
00091
00092       \param r defines the size
00093   */
00094   buffer(const T * host_data, range<Dimensions> r, Allocator allocator = {})
00095     : implementation { new detail::buffer<T, Dimensions> { host_data, r } } {}
00096
00097   /** Create a new buffer with associated memory, using the data in
00098       hostData
00099
00100       The ownership of the hostData is shared between the runtime and the
00101       user. In order to enable both the user application and the SYCL
00102       runtime to use the same pointer, a cl::sycl::mutex_class is
00103       used. The mutex m is locked by the runtime whenever the data is in
00104       use and unlocked otherwise. Data is synchronized with hostData, when
00105       the mutex is unlocked by the runtime.
00106   */
00107   buffer(shared_ptr_class<T> & hostData,
00108          const range<Dimensions> & bufferRange,
00109          cl::sycl::mutex_class * m = nullptr,
00110          Allocator allocator = {}) {
00111     detail::unimplemented();
00112   }
00113
00114
00115   /** Create a new buffer which is initialized by
00116       hostData
00117
00118       The SYCL runtime receives full ownership of the hostData unique_ptr
00119       and there in effect there is no synchronization with the application
00120       code using hostData.
00121 */
00122   buffer(unique_ptr_class<T> && hostData,
00123          const range<Dimensions> & bufferRange) {
00124     detail::unimplemented();
00125   }
00126
00127
00128   /** Create a new read-write allocated 1D buffer initialized from the
00129       given elements
00130
00131       \param start_iterator points to the first element to copy
00132
00133       \param end_iterator points to just after the last element to copy
00134
00135       \todo Add const to the SYCL specification.
00136
00137       \todo Generalize this for n-D and provide column-major and row-major
00138       initialization
00139
00140       \todo Allow read-only buffer construction too
00141
00142       \todo Allow initialization from ranges and collections à la STL
00143   */
00144   template <typename InputIterator,
00145             /* To force some iterator concept checking to avoid GCC 4.9
00146                diving into this when initializing from ({ int, int })
00147                which is a range<> and and not an iterator... */
00148             typename ValueType =
00149             typename std::iterator_traits<InputIterator>::value_type>
00150   buffer(InputIterator start_iterator,
00151          InputIterator end_iterator,
00152          Allocator allocator = {}) :
00153     implementation { new detail::buffer<T, Dimensions> { start_iterator,
00154                                                          end_iterator } }
00155   {}
00156
00157
00158   /** Create a new sub-buffer without allocation to have separate accessors
00159       later
00160
00161       \param b is the buffer with the real data
00162
00163       \param base_index specifies the origin of the sub-buffer inside the
00164       buffer b
00165
00166       \param sub_range specifies the size of the sub-buffer
00167
00168       \todo To be implemented
00169
00170       \todo Update the specification to replace index by id
00171   */
00172   buffer(buffer<T, Dimensions, Allocator> b,
00173          id<Dimensions> base_index,
00174          range<Dimensions> sub_range,
00175          Allocator allocator = {}) { assert(0); }
```

```
00176
00177
00178 #ifdef TRISYCL_OPENCL
00179   /** Create a buffer from an existing OpenCL memory object associated to
00180       a context after waiting for an event signaling the availability of
00181       the OpenCL data
00182
00183       \param mem_object is the OpenCL memory object to use
00184
00185       \param from_queue is the queue associated to the memory object
00186
00187       \param available_event specifies the event to wait for if non null
00188
00189       \todo To be implemented
00190
00191       \todo Improve the specification to allow CLHPP objects too
00192   */
00193   buffer(cl_mem mem_object,
00194          queue from_queue,
00195          event available_event = {},
00196          Allocator allocator = {}) { assert(0); }
00197 #endif
00198
00199
00200   // Use BOOST_DISABLE_ASSERTS at some time to disable range checking
00201
00202   /** Get an accessor to the buffer with the required mode
00203
00204       \param Mode is the requested access mode
00205
00206       \param Target is the type of object to be accessed
00207
00208       \todo Do we need for an accessor to increase the reference count of
00209       a buffer object? It does make more sense for a host-side accessor.
00210
00211       \todo Implement the modes and targets
00212   */
00213   template <access::mode Mode,
00214             access::target Target = access::global_buffer>
00215   accessor<T, Dimensions, Mode, Target>
00216   get_access(handler &command_group_handler) const {
00217     static_assert(Target != access::host_buffer,
00218                   "get_access(&cgh) for non host_buffer accessor "
00219                   "takes a command group handler");
00220     return *implementation;
00221   }
00222
00223
00224   /** Get a host accessor to the buffer with the required mode
00225
00226       \param Mode is the requested access mode
00227
00228       \todo Implement the modes
00229
00230       \todo More elegant solution
00231   */
00232   template <access::mode Mode,
00233             access::target Target = access::global_buffer>
00234   accessor<T, Dimensions, Mode, access::host_buffer>
00235   get_access() const {
00236     static_assert(Target == access::host_buffer,
00237                   "get_access() for host_buffer accessor does not "
00238                   "take a command group handler");
00239     return *implementation;
00240   }
00241
00242
00243   /// Get the range<> of the buffer
00244   auto get_range() const {
00245     /* Interpret the shape which is a pointer to the first element as an
00246        array of Dimensions elements so that the range<Dimensions>
00247        constructor is happy with this collection
00248
00249        \todo Move into detail::
00250
00251        \todo Add also a constructor in range<> to accept a const
00252        std::size_t *?
00253     */
00254     return range<Dimensions> { *(const std::size_t (*)[Dimensions])(implementation->allocation.shape()) };
00255   }
00256
00257
00258   /** Ask for read-only status of the buffer
00259
00260       \todo Add to specification
00261   */
00262   bool is_read_only() const { return implementation->read_only; }
```

```
00263
00264
00265   /** Return the use count of the data of this buffer
00266
00267       \todo Rename to use_count() to follow shared_ptr<> naming
00268   */
00269   auto get_count() const {
00270     // Rely on the shared_ptr<> use_count()
00271     return implementation.use_count();
00272   }
00273
00274
00275   /** Set destination of buffer data on destruction
00276
00277       The finalData points to the host memory to which, the outcome of all
00278       the buffer processing is going to be copied to.
00279
00280       This is the final pointer, which is going to be accessible after the
00281       destruction of the buffer and in the case where this is a valid
00282       pointer, the data are going to be copied to this host address.
00283
00284       finalData is different from the original host address, if the buffer
00285       was created associated with one. This is mainly to be used when a
00286       shared_ptr is given in the constructor and the output data will
00287       reside in a different location from the initialization data.
00288
00289       It is defined as a weak_ptr referring to a shared_ptr that is not
00290       associated with the cl::sycl::buffer, and so the cl::sycl::buffer
00291       will have no ownership of finalData.
00292   */
00293   void set_final_data(weak_ptr_class<T> & finalData) {
00294   detail::unimplemented();
00295 }
00296
00297 };
00298
00299 /// @} End the data Doxygen group
00300
00301 }
00302 }
00303
00304 /*
00305     # Some Emacs stuff:
00306     ### Local Variables:
00307     ### ispell-local-dictionary: "american"
00308     ### eval: (flyspell-prog-mode)
00309     ### End:
00310 */
00311
00312 #endif // TRISYCL_SYCL_BUFFER_HPP
```

11.17 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference

```
#include <memory>
#include <mutex>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/buffer_customer.hpp"
```

Include dependency graph for buffer_base.hpp: This graph shows which files directly or indirectly include this file:

Classes

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

  Factorize some template independent buffer aspects in a base class.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.18 buffer_base.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_BASE_HPP
00002 #define TRISYCL_SYCL_BUFFER_BASE_HPP
00003
00004 /** \file The buffer_base behind the buffers
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <memory>
00013 #include <mutex>
00014
00015 #include "CL/sycl/access.hpp"
00016 #include "CL/sycl/buffer/detail/buffer_customer.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020 namespace detail {
00021
00022 template <typename T,
00023           std::size_t dimensions,
00024           access::mode mode,
00025           access::target target = access::global_buffer>
00026 struct AccessorImpl;
00027
00028 /** Factorize some template independent buffer aspects in a base class
00029  */
00030  struct buffer_base {
00031    /// If the data are read-only, store the information for later optimization.
00032    /// \todo Replace this by a static read-only type for the buffer
00033    bool read_only;
00034
00035    /// Store the buffer_customer for the last generation of this buffer
00036    std::shared_ptr<buffer_customer> last_buffer_customer;
00037    std::mutex protect_buffer;
00038
00039
00040    buffer_base(bool read_only) : read_only { read_only } {}
00041
00042
00043    /// Lock the buffer_base structure by returning a unique_lock on the mutex
00044    std::unique_lock<std::mutex> lock() {
00045      return std::unique_lock<std::mutex> { protect_buffer };
00046    }
00047
00048
00049    std::shared_ptr<buffer_customer> get_last_buffer_customer() {
00050      return last_buffer_customer;
00051    }
00052
00053
00054    void set_last_buffer_customer(std::shared_ptr<buffer_customer> bc) {
00055      last_buffer_customer = bc;
00056    }
00057
00058   /// Get the buffer customer associated to the latest version of the buffer
00059   template <typename T,
00060             std::size_t dimensions,
00061             access::mode mode,
00062             access::target target = access::global_buffer>
00063   static std::shared_ptr<buffer_customer>
00064   get_buffer_customer(
      AccessorImpl<T, dimensions, mode, target> &a) {
00065     buffer_base &b = a.get_buffer();
00066     {
00067       /// Use atomic list?
00068       // Protect the update of last_buffer_customer in the Buffer
00069       auto lock = b.lock();
00070       std::shared_ptr<buffer_customer> bc = b.get_last_buffer_customer();
00071       auto old_bc = bc;
00072       /* When we write into a buffer, we generate a new version of it (think
00073          "SSA"). Of course we do it also when there is not yet any
00074          buffer_customer */
00075       if (!bc || a.is_write_access()) {
00076         bc = std::make_shared<buffer_customer>(b, a.is_write_access());
00077         b.set_last_buffer_customer(bc);
00078       }
00079
00080       if (old_bc)
00081         // \todo Use atomic list instead
00082         old_bc->set_next_generation(bc);
00083       else
```

```
00084         // If we just created the buffer_customer, it is ready to use
00085         bc->notify_ready();
00086
00087       return bc;
00088     }
00089   }
00090
00091
00092   // Wait for the latest generation of the buffer before the host can use it
00093    static void wait(buffer_base &b) {
00094      // If there is nobody using the buffer, no need to wait
00095      if (b.last_buffer_customer)
00096        /* In a correct SYCL program there should be no more task creation
00097           using a buffer given to use by a host accessor so this should be
00098           race free */
00099        b.last_buffer_customer->wait_released();
00100    }
00101
00102 };
00103
00104 }
00105 }
00106 }
00107
00108 /*
00109     # Some Emacs stuff:
00110     ### Local Variables:
00111     ### ispell-local-dictionary: "american"
00112     ### eval: (flyspell-prog-mode)
00113     ### End:
00114 */
00115
00116 #endif // TRISYCL_SYCL_BUFFER_BASE_HPP
```

11.19 include/CL/sycl/buffer/detail/buffer_customer.hpp File Reference

```
#include <atomic>
#include <condition_variable>
#include <memory>
#include <mutex>
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for buffer_customer.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::buffer_customer

  Keep track of the tasks waiting for the availability of a buffer generation, either to read it or to write it.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.20 buffer_customer.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_CUSTOMER_HPP
00002 #define TRISYCL_SYCL_BUFFER_CUSTOMER_HPP
00003
00004 /** \file The concept of buffer_customer behind the scene
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
```

```
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <atomic>
00013 #include <condition_variable>
00014 #include <memory>
00015 #include <mutex>
00016
00017 #include "CL/sycl/detail/debug.hpp"
00018
00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 struct buffer_base;
00024 /// \todo is it needed?
00025 struct task;
00026
00027 /** Keep track of the tasks waiting for the availability of a buffer
00028     generation, either to read it or to write it
00029
00030     When we write into a buffer, we generate a new version of it (think
00031     "SSA")
00032 */
00033 class buffer_customer : public detail::debug<buffer_customer> {
00034   /// The considered buffer
00035   /// \todo Do we need to keep it?
00036   buffer_base &buf;
00037   /// At some point use lock free list for this inside buffer_base
00038   std::shared_ptr<buffer_customer> next_generation;
00039   /// \todo Needed?
00040   bool write_access;
00041   // State when the buffer generation is ready to be used
00042   bool ready_to_use;
00043   // To signal when it is ready
00044   std::condition_variable ready_cv;
00045   /// To protect the access to the condition variable
00046   std::mutex ready_mutex;
00047   /// Count the number of accelerator-side usage of this buffer generation
00048   std::atomic<unsigned int> user_number;
00049   /** To signal when the buffer generation is no longer used from the
00050       accelerator side and can be used for example through a host
00051       accessor */
00052   std::condition_variable released_cv;
00053   /// To protect the access to the condition variable
00054   std::mutex released_mutex;
00055
00056 public:
00057
00058   buffer_customer(buffer_base &buf, bool is_write_access)
00059     : buf { buf },  write_access { is_write_access },
00060       ready_to_use { false }, user_number { 0 } {
00061   }
00062
00063
00064   /** Set the next generation of the buffer after this
00065
00066       \todo Refactor this with an lock-free list?
00067   */
00068   void set_next_generation(std::shared_ptr<buffer_customer> bc) {
00069     next_generation = bc;
00070   }
00071
00072
00073   /// Add a new task as a customer of the buffer generation
00074   void add(std::shared_ptr<task> task, bool is_write_access) {
00075     write_access = is_write_access;
00076     user_number++;
00077     TRISYCL_DUMP_T("buffer_customer::add() now user_number = " << user_number);
00078   }
00079
00080
00081   /// Wait for the buffer generation to be ready to use by a kernel task
00082   void wait() {
00083     {
00084       std::unique_lock<std::mutex> ul { ready_mutex };
00085       ready_cv.wait(ul, [&] { return ready_to_use; });
00086     }
00087   }
00088
00089
00090   /// Release the buffer generation usage by a  kernel task
00091   void release() {
00092     user_number--;
00093     TRISYCL_DUMP_T("buffer_customer::release() now user_number = " << user_number);
00094     if (user_number == 0) {
00095       /* If there is no task using this generation of the buffer, first
```

```
00096          notify the host accessors waiting for it, if any */
00097       released_cv.notify_all();
00098
00099       /* And then make the next generation ready if any. Note that if the
00100          SYCL program is race condition-free, there should be no host
00101          accessor waiting for a generation which is not the last one...
00102
00103          \todo: add some SYCL semantics runtime verification
00104       */
00105       if (next_generation)
00106         next_generation->notify_ready();
00107     }
00108     // \todo Can we have UserNumber increasing again?
00109   }
00110
00111
00112   /** Wait for the release of the buffer generation before the host can
00113       use it
00114   */
00115   void wait_released() {
00116     TRISYCL_DUMP_T("buffer_customer::wait_released() user_number = " << user_number);
00117     {
00118       std::unique_lock<std::mutex> ul { released_mutex };
00119       released_cv.wait(ul, [&] { return user_number == 0; });
00120     }
00121   }
00122
00123
00124   /// Notify the customer tasks this buffer generation is ready to use
00125   void notify_ready() {
00126     {
00127       std::unique_lock<std::mutex> ul { ready_mutex };
00128       // \todo This lock can be avoided if ready_to_use is atomic
00129       ready_to_use = true;
00130     }
00131     TRISYCL_DUMP_T("buffer_customer::notify_ready()");
00132     ready_cv.notify_all();
00133   }
00134
00135 };
00136
00137 }
00138 }
00139 }
00140
00141 /*
00142     # Some Emacs stuff:
00143     ### Local Variables:
00144     ### ispell-local-dictionary: "american"
00145     ### eval: (flyspell-prog-mode)
00146     ### End:
00147 */
00148
00149 #endif // TRISYCL_SYCL_BUFFER_CUSTOMER_HPP
```

11.21 include/CL/sycl/buffer_allocator.hpp File Reference

`#include <cstddef>`
`#include <memory>`
Include dependency graph for buffer_allocator.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<typename T >
  using cl::sycl::buffer_allocator = std::allocator< T >

The default buffer allocator used by the runtime, when no allocator is defined by the user.

11.22 buffer_allocator.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00002 #define TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00003
00004 /** \file The OpenCL SYCL buffer_allocator
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <memory>
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 /** \addtogroup data Data access and storage in SYCL
00019     @{
00020 */
00021
00022 /** The default buffer allocator used by the runtime, when no allocator is
00023     defined by the user
00024
00025     Reuse the C++ default allocator.
00026 */
00027 template <typename T>
00028 using buffer_allocator = std::allocator<T>;
00029
00030 /// @} End the data Doxygen group
00031
00032 }
00033 }
00034
00035 /*
00036     # Some Emacs stuff:
00037     ### Local Variables:
00038     ### ispell-local-dictionary: "american"
00039     ### eval: (flyspell-prog-mode)
00040     ### End:
00041 */
00042
00043 #endif // TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
```

11.23 include/CL/sycl/command_group/detail/task.hpp File Reference

```
#include <memory>
#include <thread>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_customer.hpp"
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for task.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::task

  The abstraction to represent SYCL tasks executing inside command_group.

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::detail

11.24 task.hpp

```
00001 #ifndef TRISYCL_SYCL_TASK_HPP
00002 #define TRISYCL_SYCL_TASK_HPP
00003
00004 /** \file The concept of task behind the scene
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <memory>
00013 #include <thread>
00014
00015 #include "CL/sycl/access.hpp"
00016 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00017 #include "CL/sycl/buffer/detail/buffer_customer.hpp"
00018 #include "CL/sycl/detail/debug.hpp"
00019
00020 namespace cl {
00021 namespace sycl {
00022 namespace detail {
00023
00024 /** The abstraction to represent SYCL tasks executing inside command_group
00025
00026     "enable_shared_from_this" allows to access the shared_ptr behind the
00027     scene.
00028  */
00029 struct task : std::enable_shared_from_this<task>,
00030               public detail::debug<task> {
00031   /// The buffers that are used by this task
00032   std::vector<std::shared_ptr<buffer_customer>> buffers;
00033
00034   /// Add a new task to the task graph and schedule for execution
00035   void schedule(std::function<void(void)> f) {
00036     /** To keep a copy of the task shared_ptr after the end of the command
00037         group, capture it by copy in the following lambda. This should be
00038         easier in C++17 with move semantics on capture
00039     */
00040     auto task = shared_from_this();
00041     auto execution = [=] {
00042       // Wait for the required buffers to be ready
00043       task->acquire_buffers();
00044       TRISYCL_DUMP_T("Execute the kernel");
00045       // Execute the kernel
00046       f();
00047       // Release the required buffers for other uses
00048       task->release_buffers();
00049       TRISYCL_DUMP_T("Exit");
00050     };
00051 #if TRISYCL_ASYNC
00052     /* If in asynchronous execution mode, execute the functor in a new
00053        thread */
00054     std::thread thread(execution);
00055     TRISYCL_DUMP_T("Started");
00056     // Detach the thread since it will synchronize by its own means
00057     thread.detach();
00058 #else
00059     // Just a synchronous execution otherwise
00060     execution();
00061 #endif
00062   }
00063
00064
00065   void acquire_buffers() {
00066     TRISYCL_DUMP_T("acquire_buffers()");
00067     for (auto &b : buffers)
00068       b->wait();
00069   }
00070
00071
00072   void release_buffers() {
00073     TRISYCL_DUMP_T("release_buffers()");
00074     for (auto &b : buffers)
00075       b->release();
00076   }
00077
00078
```

```
00079   /** Register an accessor to this task
00080
00081       This is how the dependency graph is incrementally built.
00082   */
00083   template <typename T,
00084             std::size_t dimensions,
00085             access::mode mode,
00086             access::target target = access::global_buffer>
00087   void add(AccessorImpl<T, dimensions, mode, target> &a) {
00088     auto bc = buffer_base::get_buffer_customer(a);
00089     // Add the task as a new client for the buffer customer of the accessor
00090     bc->add(shared_from_this(), a.isWriteAccess());
00091     buffers.push_back(bc);
00092   }
00093
00094 };
00095
00096 }
00097 }
00098 }
00099
00100 /*
00101     # Some Emacs stuff:
00102     ### Local Variables:
00103     ### ispell-local-dictionary: "american"
00104     ### eval: (flyspell-prog-mode)
00105     ### End:
00106 */
00107
00108 #endif // TRISYCL_SYCL_TASK_HPP
```

11.25 include/CL/sycl/context.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for context.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::context

  SYCL context. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::gl_context_interop = bool

Enumerations

- enum cl::sycl::info::context : int { cl::sycl::info::context::reference_count, cl::sycl::info::context::num_devices, cl::sycl::info::context::gl_interop }

Context information descriptors.

11.26 context.hpp

```
00001 #ifndef TRISYCL_SYCL_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL context
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/detail/default_classes.hpp"
00015 #include "CL/sycl/detail/unimplemented.hpp"
00016 #include "CL/sycl/device.hpp"
00017 #include "CL/sycl/device_selector.hpp"
00018 #include "CL/sycl/exception.hpp"
00019 #include "CL/sycl/info/param_traits.hpp"
00020 #include "CL/sycl/platform.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024
00025 /** \addtogroup execution Platforms, contexts, devices and queues
00026     @{
00027 */
00028
00029 namespace info {
00030
00031 using gl_context_interop = bool;
00032
00033 /** Context information descriptors
00034
00035     \todo Should be unsigned int to be consistent with others?
00036 */
00037 enum class context : int {
00038   reference_count,
00039   num_devices,
00040   gl_interop
00041 };
00042
00043
00044 /** Query the return type for get_info() on context stuff
00045
00046     \todo To be implemented
00047 */
00048 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::context, void)
00049
00050 }
00051
00052
00053 /** SYCL context
00054
00055     The context class encapsulates an OpenCL context, which is implicitly
00056     created and the lifetime of the context instance defines the lifetime
00057     of the underlying OpenCL context instance.
00058
00059     On destruction clReleaseContext is called.
00060
00061     The default context is the SYCL host context containing only the SYCL
00062     host device.
00063
00064     \todo The implementation is quite minimal for now.
00065 */
00066 class context {
00067
00068 public:
00069
00070   /** Constructs a context object for SYCL host using an async_handler for
00071       handling asynchronous errors
00072
00073       Note that the default case asyncHandler = nullptr is handled by the
00074       default constructor.
00075   */
00076   explicit context(async_handler asyncHandler) {
00077     detail::unimplemented();
00078   }
00079
```

```
00080
00081 #ifdef TRISYCL_OPENCL
00082   /* Context constructor, where the underlying OpenCL context is given as
00083      a parameter
00084
00085      The constructor executes a retain on the cl_context.
00086
00087      Return synchronous errors via the SYCL exception class and
00088      asynchronous errors are handled via the async_handler, if provided.
00089   */
00090   context(cl_context clContext, async_handler asyncHandler = nullptr) {
00091     detail::unimplemented();
00092   }
00093 #endif
00094
00095   /** Constructs a context object using a device_selector object
00096
00097       The context is constructed with a single device retrieved from the
00098       device_selector object provided.
00099
00100       Return synchronous errors via the SYCL exception class and
00101       asynchronous errors are handled via the async_handler, if provided.
00102   */
00103   context(const device_selector &deviceSelector,
00104           info::gl_context_interop interopFlag,
00105           async_handler asyncHandler = nullptr) {
00106     detail::unimplemented();
00107   }
00108
00109
00110   /** Constructs a context object using a device object
00111
00112       Return synchronous errors via the SYCL exception class and
00113       asynchronous errors are handled via the async_handler, if provided.
00114   */
00115   context(const device &dev,
00116           info::gl_context_interop interopFlag,
00117           async_handler asyncHandler = nullptr) {
00118     detail::unimplemented();
00119   }
00120
00121
00122   /** Constructs a context object using a platform object
00123
00124       Return synchronous errors via the SYCL exception class and
00125       asynchronous errors are handled via the async_handler, if provided.
00126   */
00127   context(const platform &plt,
00128           info::gl_context_interop interopFlag,
00129           async_handler asyncHandler = nullptr) {
00130     detail::unimplemented();
00131   }
00132
00133
00134   /* Constructs a context object using a vector_class of device objects
00135
00136      Return synchronous errors via the SYCL exception class and
00137      asynchronous errors are handled via the async_handler, if provided.
00138
00139      \todo Update the specification to replace vector by collection
00140      concept.
00141   */
00142   context(const vector_class<device> &deviceList,
00143           info::gl_context_interop interopFlag,
00144           async_handler asyncHandler = nullptr) {
00145     detail::unimplemented();
00146   }
00147
00148   /** Default constructor that chooses the context according the
00149       heuristics of the default selector
00150
00151       Return synchronous errors via the SYCL exception class.
00152
00153       Get the default constructors back.
00154   */
00155   context() = default;
00156
00157
00158 #ifdef TRISYCL_OPENCL
00159   /* Returns the underlying cl_context object, after retaining the cl_context.
00160
00161      Retains a reference to the returned cl_context object.
00162
00163      Caller should release it when finished.
00164   */
00165   cl_context get() const {
00166     detail::unimplemented();
```

```
00167     return {};
00168   }
00169 #endif
00170
00171
00172   /// Specifies whether the context is in SYCL Host Execution Mode.
00173   bool is_host() const {
00174     return true;
00175   }
00176
00177
00178   /** Returns the SYCL platform that the context is initialized for
00179
00180       \todo To be implemented
00181   */
00182   platform get_platform();
00183
00184
00185   /** Returns the set of devices that are part of this context
00186
00187       \todo To be implemented
00188   */
00189   vector_class<device> get_devices() const {
00190     detail::unimplemented();
00191     return {};
00192   }
00193
00194
00195   /** Queries OpenCL information for the under-lying cl context
00196
00197       \todo To be implemented
00198   */
00199   template <info::context Param>
00200   typename info::param_traits<info::context, Param>::type
      get_info() const {
00201     detail::unimplemented();
00202     return {};
00203   }
00204
00205 };
00206
00207 /// @} to end the execution Doxygen group
00208
00209 }
00210 }
00211
00212 /*
00213     # Some Emacs stuff:
00214     ### Local Variables:
00215     ### ispell-local-dictionary: "american"
00216     ### eval: (flyspell-prog-mode)
00217     ### End:
00218 */
00219
00220 #endif // TRISYCL_SYCL_CONTEXT_HPP
```

11.27 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference

Some helpers to do array-tuple conversions.

```
#include <tuple>
#include <utility>
```

Include dependency graph for array_tuple_helpers.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...

- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std::index_sequence< Is...>)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

11.27.1 Detailed Description

Some helpers to do array-tuple conversions.

Used for example to implement cl::sycl::vec<> class.

Ronan at Keryell point FR

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

Definition in file array_tuple_helpers.hpp.

11.28 array_tuple_helpers.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
00002 #define TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
00003
00004 /** \file
00005
00006     Some helpers to do array-tuple conversions
00007
00008     Used for example to implement cl::sycl::vec<> class.
00009
00010     Ronan at Keryell point FR
00011
00012     This file is distributed under the University of Illinois Open Source
00013     License. See LICENSE.TXT for details.
00014 */
00015
00016 #include <tuple>
00017 #include <utility>
00018
00019 namespace cl {
00020 namespace sycl {
00021 namespace detail {
00022
00023 /** \addtogroup array_tuple_helpers Helpers to do array and tuple conversion
00024
00025     @{
00026 */
00027
00028 /** Helper to construct an array from initializer elements provided as a
00029     tuple
00030
00031     The trick is to get the std::index_sequence<> that represent 0,
00032     1,..., dimension-1 as a variadic template pack Is that we can
00033     iterate on, in this function.
```

```
00034 */
00035 template <typename V, typename Tuple, size_t... Is>
00036 std::array<typename V::element_type, V::dimension>
00037 tuple_to_array_iterate(Tuple t, std::index_sequence<Is...>) {
00038   /* The effect is like a static for-loop with Is counting from 0 to
00039      dimension-1 and thus constructing a uniform initialization { }
00040      construction from each tuple element:
00041      { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }
00042
00043      The static cast is here to avoid the warning when there is a loss
00044      of precision, for example when initializing an int from a float.
00045   */
00046   return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
00047 }
00048
00049
00050 /** Construct an array from initializer elements provided as a tuple
00051  */
00052 template <typename V, typename Tuple>
00053 auto tuple_to_array(Tuple t) {
00054   /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
00055      so that tuple_to_array_iterate can statically iterate on it */
00056   return tuple_to_array_iterate<V>(t,
00057                                    std::make_index_sequence<std::tuple_size<Tuple>::value>{});
00058 }
00059
00060
00061 /** Allows optional expansion of a 1-element tuple to a V::dimension
00062     tuple to replicate scalar values in vector initialization
00063 */
00064 template <typename V, typename Tuple, bool expansion = false>
00065 struct expand_to_vector {
00066   static_assert(V::dimension == std::tuple_size<Tuple>::value,
00067                 "The number of elements in initialization should match the dimension of the vector");
00068
00069   // By default, act as a pass-through and do not do any expansion
00070   static auto expand(Tuple t) { return t; }
00071
00072 };
00073
00074
00075 /** Specialization in the case we ask for expansion */
00076 template <typename V, typename Tuple>
00077 struct expand_to_vector<V, Tuple, true> {
00078   static_assert(std::tuple_size<Tuple>::value == 1,
00079                 "Since it is a vector initialization from a scalar there should be only one initializer
      value");
00080
00081
00082   /** Construct a tuple from a value
00083
00084       \param value is used to initialize each tuple element
00085
00086       \param size is the number of elements of the tuple to be generated
00087
00088       The trick is to get the std::index_sequence<> that represent 0,
00089       1,..., dimension-1 as a variadic template pack Is that we can
00090       iterate on, in this function.
00091   */
00092   template <typename Value, size_t... Is>
00093   static auto fill_tuple(Value e, std::index_sequence<Is...>) {
00094     /* The effect is like a static for-loop with Is counting from 0 to
00095        dimension-1 and thus replicating the pattern to have
00096        make_tuple( (0, e), (1, e), ... (n - 1, e) )
00097
00098        Since the "," operator is just here to throw away the Is value
00099        (which is needed for the pack expansion...), at the end this is
00100        equivalent to:
00101        make_tuple( e, e, ..., e )
00102     */
00103     return std::make_tuple(((void)Is, e)...);
00104   }
00105
00106
00107   /** We expand the 1-element tuple by replicating into a tuple with the
00108       size of the vector */
00109   static auto expand(Tuple t) {
00110     return fill_tuple(std::get<0>(t),
00111                       std::make_index_sequence<V::dimension>{});
00112   }
00113
00114 };
00115
00116
00117 /** Create the array data of V from a tuple of initializer
00118
00119     If there is only 1 initializer, this is a scalar initialization of a
```

```
00120     vector and the value is expanded to all the vector elements first.
00121 */
00122 template <typename V, typename Tuple>
00123 auto expand(Tuple t) {
00124   return tuple_to_array<V>(expand_to_vector<V,
00125                            decltype(t),
00126                            /* Only ask the expansion to all vector
00127                               element if there only a scalar
00128                               initializer */
00129                            std::tuple_size<Tuple>::value == 1>{}.expand(t));
00130 }
00131
00132 }
00133 }
00134 }
00135
00136 /*
00137     # Some Emacs stuff:
00138     ### Local Variables:
00139     ### ispell-local-dictionary: "american"
00140     ### eval: (flyspell-prog-mode)
00141     ### End:
00142 */
00143
00144 #endif // TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
```

11.29 include/CL/sycl/detail/debug.hpp File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include <thread>
#include <typeinfo>
#include <boost/log/trivial.hpp>
```

Include dependency graph for debug.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::debug< T >

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

- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_DUMP(expression)

  Dump a debug message in a formatted way.

- #define TRISYCL_DUMP_T(expression)

  Same as TRISYCL_DUMP() but with thread id first.

11.29.1 Macro Definition Documentation

11.29.1.1 #define TRISYCL_DUMP(*expression*)

Value:

```
do {                \
    std::ostringstream s;                           \
    s << expression;                                \
    BOOST_LOG_TRIVIAL(debug) << s.str();            \
  } while(0)
```

Dump a debug message in a formatted way.

Use an intermediate ostringstream because there are issues with BOOST_LOG_TRIVIAL to display C strings

Definition at line 34 of file debug.hpp.

Referenced by cl::sycl::detail::debug< buffer_customer >::debug(), and cl::sycl::detail::debug< buffer_customer >::~debug().

11.29.1.2 #define TRISYCL_DUMP_T(*expression*)

Value:

```
TRISYCL_DUMP("Thread " << std::ios_base::hex                          \
             << std::this_thread::get_id() << ": " << expression)
```

Same as TRISYCL_DUMP() but with thread id first.

Definition at line 40 of file debug.hpp.

Referenced by cl::sycl::detail::task::acquire_buffers(), cl::sycl::detail::buffer_customer::add(), cl::sycl::detail::buffer_customer::notify_ready(), cl::sycl::detail::buffer_customer::release(), cl::sycl::detail::task::release_buffers(), cl::sycl::detail::task::schedule(), and cl::sycl::detail::buffer_customer::wait_released().

11.30 debug.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_DEBUG_HPP
00002 #define TRISYCL_SYCL_DETAIL_DEBUG_HPP
00003
00004 /** \file This is a small class to track constructor/destructor invocations
00005
00006     Define the TRISYCL_DEBUG CPP flag to have an output.
00007
00008     To use it in some class C, make C inherit from debug<C>.
00009
00010     Ronan at Keryell point FR
00011
00012     This file is distributed under the University of Illinois Open Source
00013     License. See LICENSE.TXT for details.
00014 */
00015
00016 #include <iostream>
00017
00018 #ifdef TRISYCL_DEBUG
00019 #include <sstream>
00020 #include <string>
00021 #include <thread>
00022 #include <typeinfo>
00023
00024 #include <boost/log/trivial.hpp>
00025
00026 // To be able to construct string literals like "blah"s
00027 using namespace std::string_literals;
00028
00029 /** Dump a debug message in a formatted way.
00030
00031     Use an intermediate ostringstream because there are issues with
00032     BOOST_LOG_TRIVIAL to display C strings
00033 */
```

```
00034 #define TRISYCL_DUMP(expression) do {                \
00035     std::ostringstream s;                            \
00036     s << expression;                                 \
00037     BOOST_LOG_TRIVIAL(debug) << s.str();             \
00038   } while(0)
00039 /// Same as TRISYCL_DUMP() but with thread id first
00040 #define TRISYCL_DUMP_T(expression)                                  \
00041   TRISYCL_DUMP("Thread " << std::ios_base::hex                      \
00042                << std::this_thread::get_id() << ": " << expression)
00043 #else
00044 #define TRISYCL_DUMP(expression) do { } while(0)
00045 #define TRISYCL_DUMP_T(expression) do { } while(0)
00046 #endif
00047
00048 namespace cl {
00049 namespace sycl {
00050 namespace detail {
00051
00052 /** \addtogroup debug_trace Debugging and tracing support
00053     @{
00054 */
00055
00056 /** Class used to trace the construction, copy-construction,
00057     move-construction and destruction of classes that inherit from it
00058
00059     \param T is the real type name to be used in the debug output.
00060 */
00061 template <typename T>
00062 struct debug {
00063 #ifdef TRISYCL_DEBUG
00064   /// Trace the construction with the compiler-dependent mangled named
00065   debug() {
00066     TRISYCL_DUMP("Constructor of " << typeid(*this).name()
00067                  << " " << (void*) this);
00068   }
00069
00070
00071   /// Trace the copy construction with the compiler-dependent mangled
00072   /// named
00073   debug(debug const &) {
00074     TRISYCL_DUMP("Copy of " << typeid(*this).name() << " " << (void*) this);
00075   }
00076
00077
00078   /// Trace the move construction with the compiler-dependent mangled
00079   /// named
00080   debug(debug &&) {
00081     TRISYCL_DUMP("Move of " << typeid(*this).name() << " " << (void*) this);
00082   }
00083
00084
00085   /// Trace the destruction with the compiler-dependent mangled named
00086   ~debug() {
00087     TRISYCL_DUMP("~ Destructor of " << typeid(*this).name()
00088                  << " " << (void*) this);
00089   }
00090 #endif
00091 };
00092
00093
00094 /** Class used to display a vector-like type of classes that inherit from
00095     it
00096
00097     \param T is the real type name to be used in the debug output.
00098
00099     Calling the display() method dump the values on std::cout
00100 */
00101 template <typename T>
00102 struct display_vector {
00103
00104   /// To debug and test
00105   void display() const {
00106 #ifdef TRISYCL_DEBUG
00107     std::cout << typeid(T).name() << ":";
00108 #endif
00109     // Get a pointer to the real object
00110     for (auto e : *static_cast<const T *>(this))
00111       std::cout << " " << e;
00112     std::cout << std::endl;
00113   }
00114
00115 };
00116
00117 /// @} End the debug_trace Doxygen group
00118
00119 }
00120 }
```

```
00121 }
00122
00123 /*
00124     # Some Emacs stuff:
00125     ### Local Variables:
00126     ### ispell-local-dictionary: "american"
00127     ### eval: (flyspell-prog-mode)
00128     ### End:
00129 */
00130
00131 #endif // TRISYCL_SYCL_DETAIL_DEBUG_HPP
```

11.31 include/CL/sycl/detail/default_classes.hpp File Reference

```
#include <memory>
#include <vector>
#include <string>
#include <functional>
#include <mutex>
```

Include dependency graph for default_classes.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<class T , class Alloc = std::allocator<T>>
  using cl::sycl::vector_class = std::vector< T, Alloc >
- using cl::sycl::string_class = std::string
- template<class R , class... ArgTypes>
  using cl::sycl::function_class = std::function< R(ArgTypes...)>
- using cl::sycl::mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using cl::sycl::unique_ptr_class = std::unique_ptr< T[], D >
- template<class T >
  using cl::sycl::shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using cl::sycl::weak_ptr_class = std::weak_ptr< T >

11.32 default_classes.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
00002 #define TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
00003
00004 /** \file The OpenCL SYCL default classes to use from the STL according to
00005     section 3.2 of SYCL 1.2 specification
00006
00007     Ronan at Keryell point FR
00008
00009     This file is distributed under the University of Illinois Open Source
00010     License. See LICENSE.TXT for details.
00011 */
00012
00013 #ifndef CL_SYCL_NO_STD_VECTOR
00014 /** The vector type to be used as SYCL vector
00015  */
00016 #include <memory>
```

```
00017 #include <vector>
00018 namespace cl {
00019 namespace sycl {
00020
00021 template <class T, class Alloc = std::allocator<T>>
00022 using vector_class = std::vector<T, Alloc>;
00023
00024 }
00025 }
00026 #endif
00027
00028
00029 #ifndef CL_SYCL_NO_STD_STRING
00030 /** The string type to be used as SYCL string
00031  */
00032 #include <string>
00033 namespace cl {
00034 namespace sycl {
00035
00036 using string_class = std::string;
00037
00038 }
00039 }
00040 #endif
00041
00042
00043 #ifndef CL_SYCL_NO_STD_FUNCTION
00044 /** The functional type to be used as SYCL function
00045  */
00046 #include <functional>
00047 namespace cl {
00048 namespace sycl {
00049
00050 template <class R, class... ArgTypes>
00051 using function_class = std::function<R(ArgTypes...)>;
00052
00053 }
00054 }
00055 #endif
00056
00057
00058 #ifndef CL_SYCL_NO_STD_MUTEX
00059 /** The mutex type to be used as SYCL mutex
00060  */
00061 #include <mutex>
00062 namespace cl {
00063 namespace sycl {
00064
00065 using mutex_class = std::mutex;
00066
00067 }
00068 }
00069 #endif
00070
00071
00072 #ifndef CL_SYCL_NO_STD_UNIQUE_PTR
00073 /** The unique pointer type to be used as SYCL unique pointer
00074  */
00075 #include <memory>
00076 namespace cl {
00077 namespace sycl {
00078
00079 template <class T, class D = std::default_delete<T>>
00080 using unique_ptr_class = std::unique_ptr<T[], D>;
00081
00082 }
00083 }
00084 #endif
00085
00086
00087 #ifndef CL_SYCL_NO_STD_SHARED_PTR
00088 /** The shared pointer type to be used as SYCL shared pointer
00089  */
00090 #include <memory>
00091 namespace cl {
00092 namespace sycl {
00093
00094 template <class T>
00095 using shared_ptr_class = std::shared_ptr<T>;
00096
00097 }
00098 }
00099 #endif
00100
00101
00102 #ifndef CL_SYCL_NO_STD_WEAK_PTR
00103 /** The weak pointer type to be used as SYCL weak pointer
```

```
00104  */
00105 #include <memory>
00106 namespace cl {
00107 namespace sycl {
00108
00109 template <class T>
00110 using weak_ptr_class = std::weak_ptr<T>;
00111
00112 }
00113 }
00114 #endif
00115
00116
00117 /*
00118     # Some Emacs stuff:
00119     ### Local Variables:
00120     ### ispell-local-dictionary: "american"
00121     ### eval: (flyspell-prog-mode)
00122     ### End:
00123 */
00124
00125 #endif // TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
```

11.33 include/CL/sycl/detail/global_config.hpp File Reference

`#include <CL/cl.hpp>`
Include dependency graph for global_config.hpp: This graph shows which files directly or indirectly include this file:

Macros

- #define CL_SYCL_LANGUAGE_VERSION 120
  This implement SYCL 1.2.
- #define CL_TRISYCL_LANGUAGE_VERSION 120
  This implement triSYCL 1.2.
- #define __SYCL_SINGLE_SOURCE__
  This source is compiled by a single source compiler.
- #define __CL_ENABLE_EXCEPTIONS
  Define TRISYCL_OPENCL to add OpenCL.
- #define TRISYCL_ASYNC 0
  Allow the asynchronous implementation of tasks.

11.33.1 Macro Definition Documentation

11.33.1.1 #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 35 of file global_config.hpp.

11.33.1.2 #define __SYCL_SINGLE_SOURCE__

This source is compiled by a single source compiler.

Definition at line 19 of file global_config.hpp.

11.33.1.3 #define CL_SYCL_LANGUAGE_VERSION 120

This implement SYCL 1.2.

Definition at line 13 of file global_config.hpp.

11.33.1.4 #define CL_TRISYCL_LANGUAGE_VERSION 120

This implement triSYCL 1.2.

Definition at line 16 of file global_config.hpp.

11.33.1.5 #define TRISYCL_ASYNC 0

Allow the asynchronous implementation of tasks.

Use asynchronous tasks by default.

Is set to 0, the functors are executed synchronously.

Definition at line 45 of file global_config.hpp.

11.34 global_config.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
00002 #define TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
00003
00004 /** \file The OpenCL SYCL details on the global triSYCL configuration
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 /// This implement SYCL 1.2
00013 #define CL_SYCL_LANGUAGE_VERSION 120
00014
00015 /// This implement triSYCL 1.2
00016 #define CL_TRISYCL_LANGUAGE_VERSION 120
00017
00018 /// This source is compiled by a single source compiler
00019 #define __SYCL_SINGLE_SOURCE__
00020
00021
00022 /** Define TRISYCL_OPENCL to add OpenCL
00023
00024     triSYCL can indeed work without OpenCL if only host support is needed.
00025
00026     Right now it is set by Doxygen to generate the documentation.
00027
00028     \todo Use a macro to check instead if the OpenCL header has been
00029     included before.
00030
00031     But what is the right one? __OPENCL_CL_H? __OPENCL_C_VERSION__? CL_HPP_?
00032     Mostly CL_HPP_ to be able to use param_traits<> from cl.hpp...
00033 */
00034 #ifdef TRISYCL_OPENCL
00035 #define __CL_ENABLE_EXCEPTIONS
00036 #include <CL/cl.hpp>
00037 #endif
00038
00039 /** Allow the asynchronous implementation of tasks */
00040 #ifndef TRISYCL_ASYNC
00041 /** Use asynchronous tasks by default.
00042
00043     Is set to 0, the functors are executed synchronously.
00044  */
00045 #define TRISYCL_ASYNC 0
00046 #endif
00047
00048 /*
00049     # Some Emacs stuff:
00050     ### Local Variables:
00051     ### ispell-local-dictionary: "american"
```

```
00052     ### eval: (flyspell-prog-mode)
00053     ### End:
00054 */
00055
00056 #endif // TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
```

11.35 include/CL/sycl/detail/linear_id.hpp File Reference

`#include <cstddef>`
Include dependency graph for linear_id.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename Range , typename Id >
  size_t cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.

11.36 linear_id.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
00002 #define TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
00003
00004 /** \file Compute linearized array access
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 namespace cl {
00015 namespace sycl {
00016 namespace detail {
00017
00018 /** \addtogroup helpers Some helpers for the implementation
00019     @{
00020 */
00021
00022 /** Compute a linearized array access used in the OpenCL 2 world
00023
00024     Typically for the get_global_linear_id() and get_local_linear_id()
00025     functions.
00026 */
00027 template <typename Range, typename Id>
00028 size_t linear_id(Range range, Id id, Id offset = {}) {
00029   auto dims = std::distance(std::begin(range), std::end(range));
00030
00031   size_t linear_id = 0;
00032   /* A good compiler should unroll this and do partial evaluation to
00033      remove the first multiplication by 0 of this Horner evaluation and
00034      remove the 0 offset evaluation */
00035     for (int i = dims - 1; i >= 0; --i)
00036       linear_id = linear_id*range[i] + id[i] - offset[i];
00037
00038     return linear_id;
00039   }
00040
00041
00042 /// @} End the helpers Doxygen group
```

```
00043
00044 }
00045 }
00046 }
00047
00048 /*
00049     # Some Emacs stuff:
00050     ### Local Variables:
00051     ### ispell-local-dictionary: "american"
00052     ### eval: (flyspell-prog-mode)
00053     ### End:
00054 */
00055
00056 #endif // TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
```

11.37 include/CL/sycl/detail/small_array.hpp File Reference

```
#include <algorithm>
#include <array>
#include <cstddef>
#include <type_traits>
#include <boost/operators.hpp>
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for small_array.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if dims = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

  Helper macro to declare a vector operation with the given side-effect operator.

11.38 small_array.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
00002 #define TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
00003
00004 /** \file This is a small array class to build range<>, id<>, etc.
00005
00006     Ronan at Keryell point FR
00007
```

```
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <algorithm>
00013 #include <array>
00014 #include <cstddef>
00015 #include <type_traits>
00016
00017 #include <boost/operators.hpp>
00018
00019 #include "CL/sycl/detail/debug.hpp"
00020
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup helpers Some helpers for the implementation
00027     @{
00028 */
00029
00030
00031 /** Helper macro to declare a vector operation with the given side-effect
00032     operator */
00033 #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)            \
00034   FinalType operator op(const FinalType &rhs) {         \
00035     for (std::size_t i = 0; i != Dims; ++i)             \
00036       (*this)[i] op rhs[i];                             \
00037     return *this;                                       \
00038   }
00039
00040
00041 /** Define a multi-dimensional index, used for example to locate a work
00042     item or a buffer element
00043
00044     Unfortunately, even if std::array is an aggregate class allowing
00045     native list initialization, it is no longer an aggregate if we derive
00046     from an aggregate. Thus we have to redeclare the constructors.
00047
00048     \param BasicType is the type element, such as int
00049
00050     \param Dims is the dimension number, typically between 1 and 3
00051
00052     \param FinalType is the final type, such as range<> or id<>, so that
00053     boost::operator can return the right type
00054
00055     \param EnableArgsConstructor adds a constructors from Dims variadic
00056     elements when true. It is false by default.
00057
00058     std::array<> provides the collection concept, with .size(), == and !=
00059     too.
00060 */
00061 template <typename BasicType,
00062           typename FinalType,
00063           std::size_t Dims,
00064           bool EnableArgsConstructor = false>
00065 struct small_array : std::array<BasicType, Dims>,
00066   // To have all the usual arithmetic operations on this type
00067   boost::euclidean_ring_operators<FinalType>,
00068   // Bitwise operations
00069   boost::bitwise<FinalType>,
00070   // Shift operations
00071   boost::shiftable<FinalType>,
00072   // Already provided by array<> lexicographically:
00073   // boost::equality_comparable<FinalType>,
00074   // boost::less_than_comparable<FinalType>,
00075   // Add a display() method
00076   detail::display_vector<FinalType> {
00077
00078   /// \todo add this Boost::multi_array or STL concept to the
00079   /// specification?
00080   static const auto dimensionality = Dims;
00081
00082   /* Note that constexpr size() from the underlying std::array provides
00083      the same functionality */
00084   static const size_t dimension = Dims;
00085   using element_type = BasicType;
00086
00087
00088   /** A constructor from another array
00089
00090       Make it explicit to avoid spurious range<> constructions from int *
00091       for example
00092   */
00093   template <typename SourceType>
00094   small_array(const SourceType src[Dims]) {
```

```
00095     // (*this)[0] is the first element of the underlying array
00096     std::copy_n(src, Dims, &(*this)[0]);
00097   }
00098
00099
00100   /// A constructor from another small_array of the same size
00101   template <typename SourceBasicType,
00102             typename SourceFinalType,
00103             bool SourceEnableArgsConstructor>
00104   small_array(const small_array<SourceBasicType,
00105               SourceFinalType,
00106               Dims,
00107               SourceEnableArgsConstructor> &src) {
00108     std::copy_n(&src[0], Dims, &(*this)[0]);
00109   }
00110
00111
00112   /** Initialize the array from a list of elements
00113
00114       Strangely, even when using the array constructors, the
00115       initialization of the aggregate is not available. So recreate an
00116       equivalent here.
00117
00118       Since there are inherited types that defines some constructors with
00119       some conflicts, make it optional here, according to
00120       EnableArgsConstructor template parameter.
00121    */
00122   template <typename... Types,
00123             // Just to make enable_if depend of the template and work
00124             bool Depend = true,
00125             typename = typename std::enable_if<EnableArgsConstructor
00126                                                && Depend>::type>
00127   small_array(const Types &... args)
00128     : std::array<BasicType, Dims> {
00129     // Allow a loss of precision in initialization with the static_cast
00130     { static_cast<BasicType>(args)... }
00131   }
00132   {
00133     static_assert(sizeof...(args) == Dims,
00134                   "The number of initializing elements should match "
00135                   "the dimension");
00136   }
00137
00138
00139   /// Construct a small_array from a std::array
00140   template <typename SourceBasicType>
00141   small_array(const std::array<SourceBasicType, Dims> &src)
00142   : std::array<BasicType, Dims>(src) {}
00143
00144
00145   /// Keep other constructors from the underlying std::array
00146   using std::array<BasicType, Dims>::array;
00147
00148   /// Keep the synthesized constructors
00149   small_array() = default;
00150
00151   /// Return the element of the array
00152   auto get(std::size_t index) const {
00153     return (*this)[index];
00154   }
00155
00156   /* Implement minimal methods boost::euclidean_ring_operators needs to
00157      generate everything */
00158   /// Add + like operations on the id<> and others
00159   TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)
00160
00161   /// Add - like operations on the id<> and others
00162   TRISYCL_BOOST_OPERATOR_VECTOR_OP(-=)
00163
00164   /// Add * like operations on the id<> and others
00165   TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)
00166
00167   /// Add / like operations on the id<> and others
00168   TRISYCL_BOOST_OPERATOR_VECTOR_OP(/=)
00169
00170   /// Add % like operations on the id<> and others
00171   TRISYCL_BOOST_OPERATOR_VECTOR_OP(%=)
00172
00173   /// Add << like operations on the id<> and others
00174   TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)
00175
00176   /// Add >> like operations on the id<> and others
00177   TRISYCL_BOOST_OPERATOR_VECTOR_OP(>>=)
00178
00179   /// Add & like operations on the id<> and others
00180   TRISYCL_BOOST_OPERATOR_VECTOR_OP(&=)
00181
```

```
00182   /// Add ^ like operations on the id<> and others
00183   TRISYCL_BOOST_OPERATOR_VECTOR_OP(^=)
00184
00185   /// Add | like operations on the id<> and others
00186   TRISYCL_BOOST_OPERATOR_VECTOR_OP(|=)
00187
00188
00189   /** Since the boost::operator work on the small_array, add an implicit
00190       conversion to produce the expected type */
00191   operator FinalType () {
00192     return *static_cast<FinalType *>(this);
00193   }
00194
00195 };
00196
00197
00198 /** A small array of 1, 2 or 3 elements with the implicit constructors */
00199 template <typename BasicType, typename FinalType, std::size_t Dims>
00200 struct small_array_123 : small_array<BasicType, FinalType, Dims> {
00201   static_assert(1 <= Dims && Dims <= 3,
00202                 "Dimensions are between 1 and 3");
00203 };
00204
00205
00206 /** Use some specializations so that some function overloads can be
00207     determined according to some implicit constructors and to have an
00208     implicit conversion from/to BasicType (such as an int typically) if
00209     dims = 1
00210 */
00211 template <typename BasicType, typename FinalType>
00212 struct small_array_123<BasicType, FinalType, 1>
00213   : public small_array<BasicType, FinalType, 1> {
00214   /// A 1-D constructor to have implicit conversion from from 1 integer
00215   /// and automatic inference of the dimensionality
00216   small_array_123(BasicType x) {
00217     (*this)[0] = x;
00218   }
00219
00220
00221   /// Keep other constructors
00222   small_array_123() = default;
00223
00224   using small_array<BasicType, FinalType, 1>::small_array;
00225
00226   /** Conversion so that an for example an id<1> can basically be used
00227       like an integer */
00228   operator BasicType() const {
00229     return (*this)[0];
00230   }
00231 };
00232
00233
00234 template <typename BasicType, typename FinalType>
00235 struct small_array_123<BasicType, FinalType, 2>
00236   : public small_array<BasicType, FinalType, 2> {
00237   /// A 2-D constructor to have implicit conversion from from 2 integers
00238   /// and automatic inference of the dimensionality
00239   small_array_123(BasicType x, BasicType y) {
00240     (*this)[0] = x;
00241     (*this)[1] = y;
00242   }
00243
00244
00245   /// Keep other constructors
00246   small_array_123() = default;
00247
00248   using small_array<BasicType, FinalType, 2>::small_array;
00249 };
00250
00251
00252 template <typename BasicType, typename FinalType>
00253 struct small_array_123<BasicType, FinalType, 3>
00254   : public small_array<BasicType, FinalType, 3> {
00255   /// A 3-D constructor to have implicit conversion from from 3 integers
00256   /// and automatic inference of the dimensionality
00257   small_array_123(BasicType x, BasicType y, BasicType z) {
00258     (*this)[0] = x;
00259     (*this)[1] = y;
00260     (*this)[2] = z;
00261   }
00262
00263
00264   /// Keep other constructors
00265   small_array_123() = default;
00266
00267   using small_array<BasicType, FinalType, 3>::small_array;
00268 };
```

```
00269
00270 /// @} End the helpers Doxygen group
00271
00272 }
00273 }
00274 }
00275
00276 /*
00277     # Some Emacs stuff:
00278     ### Local Variables:
00279     ### ispell-local-dictionary: "american"
00280     ### eval: (flyspell-prog-mode)
00281     ### End:
00282 */
00283
00284 #endif // TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
```

11.39 include/CL/sycl/detail/unimplemented.hpp File Reference

`#include <iostream>`
Include dependency graph for unimplemented.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- void cl::sycl::detail::unimplemented ()

  Display an "unimplemented" message.

11.40 unimplemented.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
00002 #define TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
00003
00004 /** \file Deal with unimplemented features
00005     Ronan at Keryell point FR
00006
00007     This file is distributed under the University of Illinois Open Source
00008     License. See LICENSE.TXT for details.
00009 */
00010
00011 #include <iostream>
00012
00013 namespace cl {
00014 namespace sycl {
00015 namespace detail {
00016
00017 /** \addtogroup debug_trace Debugging and tracing support
00018     @{
00019 */
00020
00021 /** Display an "unimplemented" message
00022
00023     Can be changed to call assert(0) or whatever.
00024 */
00025 inline void unimplemented() {
00026   std::cerr << "Error: using a non implemented feature!!!" << std::endl
00027             << "Please contribute to the open source implementation. :-)"
00028             << std::endl;
00029 }
00030
00031 /// @} End the debug_trace Doxygen group
```

```
00032
00033 }
00034 }
00035 }
00036
00037 /*
00038     # Some Emacs stuff:
00039     ### Local Variables:
00040     ### ispell-local-dictionary: "american"
00041     ### eval: (flyspell-prog-mode)
00042     ### End:
00043 */
00044
00045 #endif // TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
```

11.41 include/CL/sycl/device.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::device

  SYCL device. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::device_fp_config = unsigned int
- using cl::sycl::info::device_exec_capabilities = unsigned int
- using cl::sycl::info::device_queue_properties = unsigned int

Enumerations

- enum cl::sycl::info::device : int {
  cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units, cl::sycl::info::device::max_work_item_dimensions, cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short, cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double, cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int, cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half, cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_

mem_alloc_size, cl::sycl::info::device::image_support, cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info::device::image2d_max_height, cl::sycl::info::device::image2d_max_width, cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size, cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device::max_parameter_size, cl::sycl::info::device::mem_base_addr_align, cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size, cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device::max_constant_buffer_size, cl::sycl::info::device::max_constant_args, cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_correction_support, cl::sycl::info::device::host_unified_memory, cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_available, cl::sycl::info::device::is_compiler_available, cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device::queue_properties, cl::sycl::info::device::built_in_kernels, cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device::driver_version, cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl::sycl::info::device::extensions, cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info::device::parent_device, cl::sycl::info::device::partition_max_sub_devices, cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info::device::partition_type, cl::sycl::info::device::reference_count }

Device information descriptors.

- enum cl::sycl::info::device_partition_property : int { cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition_equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_property::partition_by_affinity_domain, cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int { cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache, cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int { cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache, cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int { cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero, cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }
- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }

11.42 device.hpp

```
00001 #ifndef TRISYCL_SYCL_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL device
00005
```

```
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/detail/default_classes.hpp"
00013 #include "CL/sycl/detail/unimplemented.hpp"
00014 #include "CL/sycl/exception.hpp"
00015 #include "CL/sycl/info/param_traits.hpp"
00016 #include "CL/sycl/platform.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 class device_selector;
00022 class platform;
00023
00024 /** \addtogroup execution Platforms, contexts, devices and queues
00025     @{
00026 */
00027
00028
00029 namespace info {
00030 /** Device information descriptors
00031
00032     From specs/latex/headers/deviceInfo.h in the specification
00033
00034     \todo Should be unsigned int?
00035 */
00036 enum class device : int {
00037   device_type,
00038   vendor_id,
00039   max_compute_units,
00040   max_work_item_dimensions,
00041   max_work_item_sizes,
00042   max_work_group_size,
00043   preferred_vector_width_char,
00044   preferred_vector_width_short,
00045   preferred_vector_width_int,
00046   preferred_vector_width_long_long,
00047   preferred_vector_width_float,
00048   preferred_vector_width_double,
00049   preferred_vector_width_half,
00050   native_vector_witdth_char,
00051   native_vector_witdth_short,
00052   native_vector_witdth_int,
00053   native_vector_witdth_long_long,
00054   native_vector_witdth_float,
00055   native_vector_witdth_double,
00056   native_vector_witdth_half,
00057   max_clock_frequency,
00058   address_bits,
00059   max_mem_alloc_size,
00060   image_support,
00061   max_read_image_args,
00062   max_write_image_args,
00063   image2d_max_height,
00064   image2d_max_width,
00065   image3d_max_height,
00066   image3d_max_widht,
00067   image3d_mas_depth,
00068   image_max_buffer_size,
00069   image_max_array_size,
00070   max_samplers,
00071   max_parameter_size,
00072   mem_base_addr_align,
00073   single_fp_config,
00074   double_fp_config,
00075   global_mem_cache_type,
00076   global_mem_cache_line_size,
00077   global_mem_cache_size,
00078   global_mem_size,
00079   max_constant_buffer_size,
00080   max_constant_args,
00081   local_mem_type,
00082   local_mem_size,
00083   error_correction_support,
00084   host_unified_memory,
00085   profiling_timer_resolution,
00086   endian_little,
00087   is_available,
00088   is_compiler_available,
00089   is_linker_available,
00090   execution_capabilities,
00091   queue_properties,
00092   built_in_kernels,
```

```
00093   platform,
00094   name,
00095   vendor,
00096   driver_version,
00097   profile,
00098   device_version,
00099   opencl_version,
00100   extensions,
00101   printf_buffer_size,
00102   preferred_interop_user_sync,
00103   parent_device,
00104   partition_max_sub_devices,
00105   partition_properties,
00106   partition_affinity_domain,
00107   partition_type,
00108   reference_count
00109 };
00110
00111 enum class device_partition_property : int {
00112   unsupported,
00113   partition_equally,
00114   partition_by_counts,
00115   partition_by_affinity_domain,
00116   partition_affinity_domain_next_partitionable
00117 };
00118
00119 enum class device_affinity_domain : int {
00120   unsupported,
00121   numa,
00122   L4_cache,
00123   L3_cache,
00124   L2_cache,
00125   next_partitionable
00126 };
00127
00128 enum class device_partition_type : int {
00129   no_partition,
00130   numa,
00131   L4_cache,
00132   L3_cache,
00133   L2_cache,
00134   L1_cache
00135 };
00136
00137 enum class local_mem_type : int {
00138   none,
00139   local,
00140   global
00141 };
00142
00143 enum class fp_config : int {
00144   denorm,
00145   inf_nan,
00146   round_to_nearest,
00147   round_to_zero,
00148   round_to_inf,
00149   fma,
00150   correctly_rounded_divide_sqrt,
00151   soft_float
00152 };
00153
00154 enum class global_mem_cache_type : int {
00155   none,
00156   read_only,
00157   write_only
00158 };
00159
00160 enum class device_execution_capabilities : unsigned int {
00161   exec_kernel,
00162   exec_native_kernel
00163 };
00164
00165
00166 using device_fp_config = unsigned int;
00167 using device_exec_capabilities = unsigned int;
00168 using device_queue_properties = unsigned int;
00169
00170
00171 /** Query the return type for get_info() on context stuff
00172
00173     \todo To be implemented, return always void.
00174 */
00175 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::device, void)
00176
00177 }
00178
00179
```

```
00180 /** SYCL device
00181
00182     \todo The implementation is quite minimal for now. :-)
00183 */
00184 class device {
00185
00186 public:
00187
00188 #ifdef TRISYCL_OPENCL
00189   /** Construct a device class instance using cl_device_id of the OpenCL
00190       device
00191
00192       Return synchronous errors via the SYCL exception
00193       class.
00194
00195       Retain a reference to the OpenCL device and if this device was an
00196       OpenCL subdevice the device should be released by the caller when it
00197       is no longer needed.
00198
00199       \todo To be implemented
00200   */
00201   explicit device(cl_device_id deviceId) {
00202     detail::unimplemented();
00203   }
00204 #endif
00205
00206
00207   /** Construct a device class instance using the device selector
00208       provided
00209
00210       Return errors via C++ exception class.
00211
00212       \todo To be implemented
00213   */
00214   explicit device(const device_selector &deviceSelector) {
00215     detail::unimplemented();
00216   }
00217
00218
00219   /** The default constructor will create an instance of the SYCL host
00220       device
00221
00222       Get the default constructors back.
00223   */
00224   device() = default;
00225
00226
00227 #ifdef TRISYCL_OPENCL
00228   /** Return the cl_device_id of the underlying OpenCL platform
00229
00230       Return synchronous errors via the SYCL exception class.
00231
00232       Retain a reference to the returned cl_device_id object. Caller
00233       should release it when finished.
00234
00235       In the case where this is the SYCL host device it will return a
00236       nullptr.
00237
00238       \todo To be implemented
00239   */
00240   cl_device_id get() const {
00241     detail::unimplemented();
00242     return {};
00243   }
00244 #endif
00245
00246   /** Return true if the device is a SYCL host device
00247
00248       \todo To be implemented
00249   */
00250   bool is_host() const {
00251     detail::unimplemented();
00252     return true;
00253   }
00254
00255
00256   /** Return true if the device is an OpenCL CPU device
00257
00258       \todo To be implemented
00259   */
00260   bool is_cpu() const {
00261     detail::unimplemented();
00262     return {};
00263   }
00264
00265
00266   /** Return true if the device is an OpenCL GPU device
```

```
00267
00268       \todo To be implemented
00269   */
00270   bool is_gpu() const {
00271     detail::unimplemented();
00272     return {};
00273   }
00274
00275
00276   /** Return true if the device is an OpenCL accelerator device
00277
00278       \todo To be implemented
00279   */
00280   bool is_accelerator() const {
00281     detail::unimplemented();
00282     return {};
00283   }
00284
00285
00286   /** Return the platform of device
00287
00288       Return synchronous errors via the SYCL exception class.
00289
00290       \todo To be implemented
00291   */
00292   platform get_platform() const {
00293     detail::unimplemented();
00294     return {};
00295   }
00296
00297
00298   /** Return a list of all available devices
00299
00300       Return synchronous errors via SYCL exception classes.
00301
00302       \todo To be implemented
00303   */
00304   static vector_class<device>
00305   get_devices(info::device_type deviceType =
      info::device_type::all) {
00306     detail::unimplemented();
00307     return {};
00308   }
00309
00310
00311   /** Query the device for OpenCL info::device info
00312
00313       Return synchronous errors via the SYCL exception class.
00314
00315       \todo To be implemented
00316   */
00317   template <info::device Param>
00318   typename info::param_traits<info::device, Param>::type
00319   get_info() const {
00320     detail::unimplemented();
00321     return {};
00322   }
00323
00324
00325   /** Specify whether a specific extension is supported on the device.
00326
00327       \todo To be implemented
00328   */
00329   bool has_extension(const string_class &extension) const {
00330     detail::unimplemented();
00331     return {};
00332   }
00333
00334
00335   /** Partition the device into sub devices based upon the properties
00336       provided
00337
00338       Return synchronous errors via SYCL exception classes.
00339
00340       \todo To be implemented
00341   */
00342   vector_class<device>
00343   create_sub_devices(info::device_partition_type partitionType
      ,
00344                      info::device_partition_property partitionProperty,
00345                      info::device_affinity_domain affinityDomain) const {
00346     detail::unimplemented();
00347     return {};
00348   }
00349
00350 };
00351
```

```
00352 /// @} to end the execution Doxygen group
00353
00354 }
00355 }
00356
00357 /*
00358     # Some Emacs stuff:
00359     ### Local Variables:
00360     ### ispell-local-dictionary: "american"
00361     ### eval: (flyspell-prog-mode)
00362     ### End:
00363 */
00364
00365 #endif // TRISYCL_SYCL_DEVICE_HPP
```

11.43 include/CL/sycl/device_selector.hpp File Reference

```
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
```

Include dependency graph for device_selector.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- class cl::sycl::default_selector

  Devices selected by heuristics of the system. More...
- class cl::sycl::gpu_selector

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices. More...
- class cl::sycl::cpu_selector

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics. More...
- class cl::sycl::host_selector

  Selects the SYCL host CPU device that does not require an OpenCL runtime. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.44 device_selector.hpp

```
00001 #ifndef TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00002 #define TRISYCL_SYCL_DEVICE_SELECTOR_HPP
00003
00004 /** \file The OpenCL SYCL device_selector
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/detail/unimplemented.hpp"
00013 #include "CL/sycl/device.hpp"
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 /** \addtogroup execution Platforms, contexts, devices and queues
```

```
00019     @{
00020 */
00021
00022 /** The SYCL heuristics to select a device
00023
00024     The device with the highest score is selected
00025 */
00026 class device_selector {
00027
00028 public:
00029
00030   /** Returns a selected device using the functor operator defined in
00031       sub-classes operator()(const device &dev)
00032
00033       \todo To be implemented
00034   */
00035   device select_device() const {
00036     detail::unimplemented();
00037     return {};
00038   }
00039
00040
00041   /**  This pure virtual operator allows the customization of device
00042        selection.
00043
00044        It defines the behavior of the device_selector functor called by
00045        the SYCL runtime on device selection. It returns a "score" for each
00046        device in the system and the highest rated device will be used
00047        by the SYCL runtime.
00048   */
00049   virtual int operator() (const device &dev) const = 0;
00050 };
00051
00052
00053 /** Devices selected by heuristics of the system
00054
00055     If no OpenCL device is found then it defaults to the SYCL host device.
00056
00057     \todo to be implemented
00058
00059     \todo to be named device_selector::default instead in the specification?
00060 */
00061 class default_selector : public device_selector {
00062
00063 public:
00064
00065   // The user-provided operator computing the score
00066   int operator() (const device &dev) const override {
00067     detail::unimplemented();
00068     return 1;
00069   }
00070
00071 };
00072
00073
00074 /** Select devices according to device type info::device::device_type::gpu
00075     from all the available OpenCL devices.
00076
00077     If no OpenCL GPU device is found the selector fails.
00078
00079     Select the best GPU, if any.
00080
00081     \todo to be implemented
00082
00083     \todo to be named device_selector::gpu instead in the specification?
00084 */
00085 class gpu_selector : public device_selector {
00086
00087 public:
00088
00089   // The user-provided operator computing the score
00090   int operator() (const device &dev) const override {
00091     detail::unimplemented();
00092     return 1;
00093   }
00094
00095 };
00096
00097
00098 /** Select devices according to device type info::device::device_type::cpu
00099     from all the available devices and heuristics
00100
00101     If no OpenCL CPU device is found the selector fails.
00102
00103     \todo to be implemented
00104
00105     \todo to be named device_selector::cpu instead in the specification?
```

```
00106 */
00107 class cpu_selector : public device_selector {
00108
00109 public:
00110
00111   // The user-provided operator computing the score
00112   int operator() (const device &dev) const override {
00113     detail::unimplemented();
00114     return 1;
00115   }
00116
00117 };
00118
00119
00120 /** Selects the SYCL host CPU device that does not require an OpenCL
00121     runtime
00122
00123     \todo to be implemented
00124
00125     \todo to be named device_selector::host instead in the specification?
00126 */
00127 class host_selector : public device_selector {
00128
00129 public:
00130
00131   // The user-provided operator computing the score
00132   int operator() (const device &dev) const override {
00133     detail::unimplemented();
00134     return 1;
00135   }
00136
00137 };
00138
00139 /// @} to end the execution Doxygen group
00140
00141 }
00142 }
00143
00144 /*
00145     # Some Emacs stuff:
00146     ### Local Variables:
00147     ### ispell-local-dictionary: "american"
00148     ### eval: (flyspell-prog-mode)
00149     ### End:
00150 */
00151
00152 #endif // TRISYCL_SYCL_DEVICE_SELECTOR_HPP
```

11.45 include/CL/sycl/error_handler.hpp File Reference

`#include "CL/sycl/exception.hpp"`

Include dependency graph for error_handler.hpp: This graph shows which files directly or indirectly include this file:

Classes

- 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

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::trisycl

11.46 error_handler.hpp

```
00001 #ifndef TRISYCL_SYCL_ERROR_HANDLER_HPP
00002 #define TRISYCL_SYCL_ERROR_HANDLER_HPP
00003
00004 /** \file The OpenCL SYCL error_handler
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/exception.hpp"
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup error_handling Error handling
00018     @{
00019 */
00020
00021 /// \todo Refactor when updating to latest specification
00022 namespace trisycl {
00023   // Create a default error handler to be used when nothing is specified
00024   struct default_error_handler;
00025 }
00026
00027
00028 /** User supplied error handler to call a user-provided function when an
00029     error happens from a SYCL object that was constructed with this error
00030     handler
00031 */
00032 struct error_handler {
00033   /** The method to define to be called in the case of an error
00034
00035       \todo Add "virtual void" to the specification
00036   */
00037   virtual void report_error(exception &error) = 0;
00038
00039   /** Add a default_handler to be used by default
00040
00041       \todo add this concept to the specification?
00042   */
00043   static trisycl::default_error_handler
     default_handler;
00044 };
00045
00046
00047 namespace trisycl {
00048
00049   struct default_error_handler : error_handler {
00050
00051     void report_error(exception &error) override {
00052     }
00053   };
00054 }
00055
00056   // \todo finish initialization
00057   //error_handler::default_handler = nullptr;
00058
00059
00060 /// @} End the error_handling Doxygen group
00061
00062 }
00063 }
00064
00065 /*
00066     # Some Emacs stuff:
00067     ### Local Variables:
00068     ### ispell-local-dictionary: "american"
00069     ### eval: (flyspell-prog-mode)
00070     ### End:
00071 */
00072
00073 #endif // TRISYCL_SYCL_ERROR_HANDLER_HPP
```

11.47 include/CL/sycl/exception.hpp File Reference

```
#include "CL/sycl/buffer.hpp"
#include "CL/sycl/image.hpp"
```

Include dependency graph for exception.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::exception

  Encapsulate a SYCL error information. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- using cl::sycl::async_handler = function_class< int >

11.48 exception.hpp

```
00001 #ifndef TRISYCL_SYCL_EXCEPTION_HPP
00002 #define TRISYCL_SYCL_EXCEPTION_HPP
00003
00004 /** \file The OpenCL SYCL exception
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/buffer.hpp"
00013 #include "CL/sycl/image.hpp"
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 class queue;
00019
00020 /** \addtogroup error_handling Error handling
00021     @{
00022 */
00023
00024 using async_handler = function_class<int/*cl::sycl::exception_list*/>;
00025
00026 /**
00027    Encapsulate a SYCL error information
00028 */
00029 struct exception {
00030 #ifdef TRISYCL_OPENCL
00031   /** Get the OpenCL error code
00032
00033       \returns 0 if not an OpenCL error
00034
00035       \todo to be implemented
00036   */
00037   cl_int get_cl_code() { assert(0); }
00038
00039
00040   /** Get the SYCL-specific error code
00041
00042       \returns 0 if not a SYCL-specific error
00043
00044       \todo to be implemented
00045
00046       \todo use something else instead of cl_int to be usable without
00047       OpenCL
00048   */
00049   cl_int get_sycl_code() { assert(0); }
00050 #endif
00051
00052   /** Get the queue that caused the error
```

```
00053
00054       \return nullptr if not a queue error
00055
00056       \todo Update specification to replace 0 by nullptr
00057   */
00058   queue *get_queue() { assert(0); }
00059
00060
00061   /** Get the buffer that caused the error
00062
00063       \returns nullptr if not a buffer error
00064
00065       \todo Update specification to replace 0 by nullptr and add the
00066       templated buffer
00067
00068       \todo to be implemented
00069
00070       \todo How to get the real buffer type? Update: has been removed in
00071       new specification
00072   */
00073   template <typename T, int dimensions, typename Allocator>
00074   buffer<T, dimensions, Allocator> *get_buffer() {
00075     assert(0); }
00076
00077
00078   /** Get the image that caused the error
00079
00080       \returns nullptr if not a image error
00081
00082       \todo Update specification to replace 0 by nullptr and add the
00083       templated buffer
00084
00085       \todo to be implemented
00086   */
00087   template <std::size_t dimensions> image<dimensions> *
      get_image() { assert(0); }
00088 };
00089
00090 /// @} End the error_handling Doxygen group
00091
00092 }
00093 }
00094
00095 /*
00096     # Some Emacs stuff:
00097     ### Local Variables:
00098     ### ispell-local-dictionary: "american"
00099     ### eval: (flyspell-prog-mode)
00100     ### End:
00101 */
00102
00103 #endif // TRISYCL_SYCL_EXCEPTION_HPP
```

11.49 include/CL/sycl/group.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for group.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::group< dims >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.50 group.hpp

```
00001 #ifndef TRISYCL_SYCL_GROUP_HPP
00002 #define TRISYCL_SYCL_GROUP_HPP
00003
00004 /** \file The OpenCL SYCL nd_item<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/detail/linear_id.hpp"
00015 #include "CL/sycl/id.hpp"
00016 #include "CL/sycl/nd_range.hpp"
00017 #include "CL/sycl/range.hpp"
00018
00019 namespace cl {
00020 namespace sycl {
00021
00022 /** \addtogroup parallelism Expressing parallelism through kernels
00023     @{
00024 */
00025
00026 /** A group index used in a parallel_for_workitem to specify a work_group
00027  */
00028 template <std::size_t dims = 1>
00029 struct group {
00030   /// \todo add this Boost::multi_array or STL concept to the
00031   /// specification?
00032   static constexpr auto dimensionality = dims;
00033
00034 private:
00035
00036   /// The coordinate of the group item
00037   id<dims> group_id;
00038
00039   /// Keep a reference on the nd_range to serve potential query on it
00040   nd_range<dims> ndr;
00041
00042 public:
00043
00044   /** Create a group from an nd_range<> with a 0 id<>
00045
00046       \todo This should be private since it is only used by the triSYCL
00047       implementation
00048   */
00049   group(const nd_range<dims> &ndr) : ndr { ndr } {}
00050
00051
00052   /** Create a group from an id and a nd_range<>
00053
00054       \todo This should be private somehow, but it is used by the
00055       validation infrastructure
00056   */
00057   group(const id<dims> &i, const nd_range<dims> &ndr ) :
00058     group_id { i }, ndr { ndr } {}
00059
00060
00061   /** To be able to copy and assign group, use default constructors too
00062
00063       \todo Make most of them protected, reserved to implementation
00064   */
00065   group() = default;
00066
00067
00068   /** Return an id representing the index of the group within the nd_range
00069       for every dimension
00070   */
00071   id<dims> get() const { return group_id; }
00072
00073
00074   /// Return the index of the group in the given dimension
00075   size_t get(int dimension) const { return get()[dimension]; }
00076
00077
00078   /** Return the index of the group in the given dimension within the
00079       nd_range<>
00080
00081       \todo In this implementation it is not const because the group<> is
00082       written in the parallel_for iterators. To fix according to the
00083       specification
00084   */
```

```
00085   auto &operator[](int dimension) {
00086     return group_id[dimension];
00087   }
00088
00089
00090   /** Return a range<> representing the dimensions of the current
00091       group
00092
00093       This local range may have been provided by the programmer, or chosen
00094       by the runtime.
00095
00096       \todo Fix this comment and the specification
00097   */
00098   range<dims> get_group_range() const {
00099     return get_nd_range().get_group();
00100   }
00101
00102
00103   /// Return element dimension from the con stituent group range
00104   size_t get_group_range(int dimension) const {
00105     return get_group_range()[dimension];
00106   }
00107
00108
00109   /// Get the local range for this work_group
00110   range<dims> get_global_range() const { return
      get_nd_range().get_global(); }
00111
00112
00113   /// Return element dimension from the constituent global range
00114   size_t get_global_range(int dimension) const {
00115     return get_global_range()[dimension];
00116   }
00117
00118
00119   /** Get the local range for this work_group
00120
00121       \todo Add to the specification
00122   */
00123   range<dims> get_local_range() const { return
      get_nd_range().get_local(); }
00124
00125
00126   /** Return element dimension from the constituent local range
00127
00128       \todo Add to the specification
00129   */
00130   size_t get_local_range(int dimension) const {
00131     return get_local_range()[dimension];
00132   }
00133
00134
00135   /**  Get the offset of the NDRange
00136
00137        \todo Add to the specification
00138   */
00139   id<dims> get_offset() const { return get_nd_range().get_offset(); }
00140
00141
00142   /**  Get the offset of the NDRange
00143
00144        \todo Add to the specification
00145   */
00146   size_t get_offset(int dimension) const { return get_offset()[dimension]; }
00147
00148
00149   /// \todo Also provide this access to the current nd_range
00150   nd_range<dims> get_nd_range() const { return ndr; }
00151
00152
00153   /** Get a linearized version of the group ID
00154
00155    */
00156   size_t get_linear() const {
00157     return detail::linear_id(get_group_range(), get());
00158   }
00159
00160 };
00161
00162 /// @} End the parallelism Doxygen group
00163
00164 }
00165 }
00166
00167 /*
00168     # Some Emacs stuff:
00169     ### Local Variables:
```

```
00170     ### ispell-local-dictionary: "american"
00171     ### eval: (flyspell-prog-mode)
00172     ### End:
00173 */
00174
00175 #endif // TRISYCL_SYCL_GROUP_HPP
```

11.51 include/CL/sycl/handler.hpp File Reference

```
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
```

Include dependency graph for handler.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::kernel

  Kernel. More...
- class cl::sycl::handler

  Command group handler class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_parallel_for_functor_GLOBAL(N)

  SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>
- #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)

11.51.1 Macro Definition Documentation

11.51.1.1 #define TRISYCL_parallel_for_functor_GLOBAL(*N*)

Value:

```
template <typename KernelName = std::nullptr_t,                      \
          typename ParallelForFunctor>                                 \
  void parallel_for(range<N> global_size,                              \
                    ParallelForFunctor f) {                            \
    current_task->schedule([=] { detail::parallel_for(global_size, f); }); \
  }
```

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

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the full size of the range<>
N	dimensionality of the iteration space
f	is the kernel functor to execute
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the dimensions

Definition at line 133 of file handler.hpp.

11.51.1.2 #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(*N*)

Value:

```
template <typename KernelName = std::nullptr_t,       \
          typename ParallelForFunctor>                \
  void parallel_for(range<N> global_size,             \
                    id<N> offset,                     \
                    ParallelForFunctor f) {           \
    current_task->schedule([=] {                      \
        detail::parallel_for_global_offset(global_size, \
                                           offset,    \
                                           f); });    \
  }
```

11.52 handler.hpp

```
00001 #ifndef TRISYCL_SYCL_HANDLER_HPP
00002 #define TRISYCL_SYCL_HANDLER_HPP
00003
00004 /** \file The OpenCL SYCL command group handler
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/accessor.hpp"
00013 #include "CL/sycl/command_group/detail/task.hpp"
00014 #include "CL/sycl/parallelism/detail/parallelism.hpp"
00015 #include "CL/sycl/detail/unimplemented.hpp"
00016 #include "CL/sycl/exception.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 /** \addtogroup execution Platforms, contexts, devices and queues
00022     @{
00023 */
00024
00025 /** Kernel
00026
00027     \todo To be implemented
00028 */
00029 class kernel {
00030 };
00031
00032
00033 /** Command group handler class
00034
00035     A command group handler object can only be constructed by the SYCL runtime.
00036
00037     All of the accessors defined in the command group scope take as a
00038     parameter an instance of the command group handler and all the kernel
00039     invocation functions are methods of this class.
00040 */
00041 class handler {
00042
00043 public:
00044
00045   /** Attach the task and accessors to it.
00046    */
00047   std::shared_ptr<detail::task> current_task;
00048
```

```
00049
00050   handler() {
00051     // Create a new task for this command_group
00052     current_task = std::make_shared<detail::task>();
00053   }
00054
00055
00056   /** Set kernel args for an OpenCL kernel which is used through the
00057       SYCL/OpenCL interop interface
00058
00059       The index value specifies which parameter of the OpenCL kernel is
00060       being set and the accessor object, which OpenCL buffer or image is
00061       going to be given as kernel argument.
00062
00063       \todo To be implemented
00064   */
00065   template <typename DataType,
00066             std::size_t Dimensions,
00067             access::mode Mode,
00068             access::target Target = access::global_buffer>
00069   void set_arg(int arg_index,
00070                accessor<DataType, Dimensions, Mode, Target>
      acc_obj) {
00071     detail::unimplemented();
00072   }
00073
00074
00075   /** Set kernel args for an OpenCL kernel which is used through the
00076       SYCL/OpenCL interoperability interface
00077
00078       The index value specifies which parameter of the OpenCL kernel is
00079       being set and the accessor object, which OpenCL buffer or image is
00080       going to be given as kernel argument.
00081
00082       \todo To be implemented
00083   */
00084   template <typename T>
00085   void set_arg(int arg_index, T scalar_value) {
00086     detail::unimplemented();
00087   }
00088
00089
00090   /** Kernel invocation method of a kernel defined as a lambda or
00091       functor. If it is a lambda function or the functor type is globally
00092       visible there is no need for the developer to provide a kernel name type
00093       (typename KernelName) for it, as described in 3.5.3
00094
00095       SYCL single_task launches a computation without parallelism at
00096       launch time.
00097
00098       \param F specify the kernel to be launched as a single_task
00099
00100       \param KernelName is a class type that defines the name to be used for
00101       the underlying kernel
00102   */
00103   template <typename KernelName = std::nullptr_t>
00104   void single_task(std::function<void(void)> F) {
00105     current_task->schedule(F);
00106   }
00107
00108
00109   /** SYCL parallel_for launches a data parallel computation with
00110       parallelism specified at launch time by a range<>
00111
00112       Kernel invocation method of a kernel defined as a lambda or functor,
00113       for the specified range and given an id or item for indexing in the
00114       indexing space defined by range.
00115
00116       If it is a lambda function or the if the functor type is globally
00117       visible there is no need for the developer to provide a kernel name
00118       type (typename KernelName) for it, as described in detail in 3.5.3
00119
00120       \param global_size is the full size of the range<>
00121
00122       \param N dimensionality of the iteration space
00123
00124       \param f is the kernel functor to execute
00125
00126       \param KernelName is a class type that defines the name to be used
00127       for the underlying kernel
00128
00129       Unfortunately, to have implicit conversion to work on the range, the
00130       function can not be templated, so instantiate it for all the
00131       dimensions
00132   */
00133 #define TRISYCL_parallel_for_functor_GLOBAL(N)                          \
00134   template <typename KernelName = std::nullptr_t,                       \
```

```
00135             typename ParallelForFunctor>                                \
00136   void parallel_for(range<N> global_size,                               \
00137                     ParallelForFunctor f) {                             \
00138     current_task->schedule([=] { detail::parallel_for(global_size, f); }); \
00139   }
00140
00141   TRISYCL_parallel_for_functor_GLOBAL(1)
00142   TRISYCL_parallel_for_functor_GLOBAL(2)
00143   TRISYCL_parallel_for_functor_GLOBAL(3)
00144
00145
00146   /** Kernel invocation method of a kernel defined as a lambda or functor,
00147       for the specified range and offset and given an id or item for
00148       indexing in the indexing space defined by range
00149
00150       If it is a lambda function or the if the functor type is globally
00151       visible there is no need for the developer to provide a kernel name
00152       type (typename KernelName) for it, as described in detail in 3.5.3
00153
00154       \param global_size is the global size of the range<>
00155
00156       \param offset is the offset to be add to the id<> during iteration
00157
00158       \param f is the kernel functor to execute
00159
00160       \param ParallelForFunctor is the kernel functor type
00161
00162       \param KernelName is a class type that defines the name to be used for
00163       the underlying kernel
00164
00165       Unfortunately, to have implicit conversion to work on the range, the
00166       function can not be templated, so instantiate it for all the
00167       dimensions
00168   */
00169 #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)     \
00170   template <typename KernelName = std::nullptr_t,       \
00171             typename ParallelForFunctor>                \
00172   void parallel_for(range<N> global_size,               \
00173                     id<N> offset,                       \
00174                     ParallelForFunctor f) {             \
00175     current_task->schedule([=] {                        \
00176         detail::parallel_for_global_offset(global_size, \
00177                                            offset,      \
00178                                            f); });      \
00179   }
00180
00181   TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1)
00182   TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2)
00183   TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3)
00184
00185
00186   /** Kernel invocation method of a kernel defined as a lambda or functor,
00187       for the specified nd_range and given an nd_item for indexing in the
00188       indexing space defined by the nd_range
00189
00190       If it is a lambda function or the if the functor type is globally
00191       visible there is no need for the developer to provide a kernel name
00192       type (typename KernelName) for it, as described in detail in 3.5.3
00193
00194       \param r defines the iteration space with the work-group layout and
00195       offset
00196
00197       \param Dimensions dimensionality of the iteration space
00198
00199       \param f is the kernel functor to execute
00200
00201       \param ParallelForFunctor is the kernel functor type
00202
00203       \param KernelName is a class type that defines the name to be used for
00204       the underlying kernel
00205   */
00206   template <typename KernelName,
00207             std::size_t Dimensions,
00208             typename ParallelForFunctor>
00209   void parallel_for(nd_range<Dimensions> r, ParallelForFunctor f) {
00210     current_task->schedule([=] { detail::parallel_for(r, f); });
00211   }
00212
00213
00214   /** Hierarchical kernel invocation method of a kernel defined as a
00215       lambda encoding the body of each work-group to launch
00216
00217       May contain multiple kernel built-in parallel_for_work_item
00218       functions representing the execution on each work-item.
00219
00220       Launch num_work_groups work-groups of runtime-defined
00221       size. Described in detail in 3.5.3.
```

```
00222
00223       \param r defines the iteration space with the work-group layout and
00224       offset
00225
00226       \param Dimensions dimensionality of the iteration space
00227
00228       \param f is the kernel functor to execute
00229
00230       \param ParallelForFunctor is the kernel functor type
00231
00232       \param KernelName is a class type that defines the name to be used for
00233       the underlying kernel
00234   */
00235   template <typename KernelName = std::nullptr_t,
00236             std::size_t Dimensions = 1,
00237             typename ParallelForFunctor>
00238   void parallel_for_work_group(nd_range<Dimensions> r,
00239                                ParallelForFunctor f) {
00240     current_task->schedule([=] { detail::parallel_for_workgroup(r, f); });
00241   }
00242
00243
00244   /** Kernel invocation method of a kernel defined as pointer to a kernel
00245       object, described in detail in 3.5.3
00246
00247       \todo To be implemented
00248   */
00249   void single_task(kernel syclKernel) {
00250     detail::unimplemented();
00251   }
00252
00253
00254   /** Kernel invocation method of a kernel defined as pointer to a kernel
00255       object, for the specified range and given an id or item for indexing
00256       in the indexing space defined by range, described in detail in 3.5.3
00257
00258       \todo To be implemented
00259   */
00260   template <std::size_t Dimensions = 1>
00261   void parallel_for(range<Dimensions> numWorkItems,
00262                     kernel sycl_kernel) {
00263     detail::unimplemented();
00264   }
00265
00266
00267   /** Kernel invocation method of a kernel defined as pointer to a kernel
00268       object, for the specified nd_range and given an nd_item for indexing
00269       in the indexing space defined by the nd_range, described in detail
00270       in 3.5.3
00271
00272       \todo To be implemented
00273   */
00274   template <std::size_t Dimensions = 1>
00275   void parallel_for(nd_range<Dimensions>, kernel syclKernel) {
00276     detail::unimplemented();
00277   }
00278
00279 };
00280
00281 /// @} End the error_handling Doxygen group
00282
00283 }
00284 }
00285
00286 /*
00287     # Some Emacs stuff:
00288     ### Local Variables:
00289     ### ispell-local-dictionary: "american"
00290     ### eval: (flyspell-prog-mode)
00291     ### End:
00292 */
00293
00294 #endif // TRISYCL_SYCL_HANDLER_HPP
```

11.53 include/CL/sycl/handler_event.hpp File Reference

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

Classes

- class handler_event

  Handler event.

11.54 handler_event.hpp

```
00001 #ifndef TRISYCL_SYCL_HANDLER_EVENT_HPP
00002 #define TRISYCL_SYCL_HANDLER_EVENT_HPP
00003
00004 /** \file The handler event
00005
00006     Implement parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 /** \todo To be implemented */
00015 /** Handler event
00016
00017     \todo To be implemented
00018 */
00019 class handler_event {
00020 /*
00021  public:
00022   event get_kernel() const;
00023   event get_complete() const;
00024   event get_end() const;
00025 */
00026 };
00027
00028
00029 /*
00030     # Some Emacs stuff:
00031     ### Local Variables:
00032     ### ispell-local-dictionary: "american"
00033     ### eval: (flyspell-prog-mode)
00034     ### End:
00035 */
00036
00037 #endif // TRISYCL_SYCL_HANDLER_EVENT_HPP
```

11.55 include/CL/sycl/id.hpp File Reference

```
#include <algorithm>
#include <cstddef>
#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for id.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::item< dims >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- class cl::sycl::id< dims >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType...Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

11.56 id.hpp

```
00001 #ifndef TRISYCL_SYCL_ID_HPP
00002 #define TRISYCL_SYCL_ID_HPP
00003
00004 /** \file The OpenCL SYCL id<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <algorithm>
00013 #include <cstddef>
00014
00015 #include "CL/sycl/detail/small_array.hpp"
00016 #include "CL/sycl/range.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 template <std::size_t dims> class item;
00022
00023 /** \addtogroup parallelism Expressing parallelism through kernels
00024     @{
00025 */
00026
00027 /** Define a multi-dimensional index, used for example to locate a work
00028     item
00029 */
00030 template <std::size_t dims = 1>
00031 class id : public detail::small_array_123<std::size_t, id<dims>, dims> {
00032
00033 public:
00034
00035   // Inherit from all the constructors
00036   using detail::small_array_123<std::size_t,
00037                                 id<dims>,
00038                                 dims>::small_array_123;
00039
00040
00041   /// Construct an id from the dimensions of a range
00042   id(const range<dims> &range_size)
00043     /** Use the fact we have a constructor of a small_array from a another
00044         kind of small_array
00045      */
00046     : detail::small_array_123<std::size_t, id<dims>, dims> { range_size } {}
00047
00048
00049   /// Construct an id from an item global_id
00050   id(const item<dims> &rhs)
00051     : detail::small_array_123<std::size_t, id<dims>, dims>
00052       { rhs.get() }
00053   {}
00054
00055   /// Keep other constructors
00056   id() = default;
00057
00058 };
00059
00060
00061 /** Implement a make_id to construct an id<> of the right dimension with
00062     implicit conversion from an initializer list for example.
00063
00064     Cannot use a template on the number of dimensions because the implicit
00065     conversion would not be tried. */
```

```
00066 inline auto make_id(id<1> i) { return i; }
00067 inline auto make_id(id<2> i) { return i; }
00068 inline auto make_id(id<3> i) { return i; }
00069
00070
00071 /** Construct an id<> from a function call with arguments, like
00072     make_id(1, 2, 3) */
00073 template<typename... BasicType>
00074 auto make_id(BasicType... Args) {
00075   // Call constructor directly to allow narrowing
00076   return id<sizeof...(Args)>(Args...);
00077 }
00078
00079 /// @} End the parallelism Doxygen group
00080
00081 }
00082 }
00083
00084 /*
00085     # Some Emacs stuff:
00086     ### Local Variables:
00087     ### ispell-local-dictionary: "american"
00088     ### eval: (flyspell-prog-mode)
00089     ### End:
00090 */
00091
00092 #endif // TRISYCL_SYCL_ID_HPP
```

11.57 include/CL/sycl/image.hpp File Reference

OpenCL SYCL image class.

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

Classes

- struct cl::sycl::image< dimensions >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.57.1 Detailed Description

OpenCL SYCL image class.

Ronan at Keryell point FR

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

Definition in file image.hpp.

11.58 image.hpp

```
00001 #ifndef TRISYCL_SYCL_IMAGE_HPP
00002 #define TRISYCL_SYCL_IMAGE_HPP
00003
00004 /** \file
00005
00006     OpenCL SYCL image class
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
```

```
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup data
00018
00019     @{
00020 */
00021
00022 /// \todo implement image
00023 template <std::size_t dimensions> struct image;
00024
00025
00026 /// @} End the data Doxygen group
00027
00028
00029 }
00030 }
00031
00032 /*
00033     # Some Emacs stuff:
00034     ### Local Variables:
00035     ### ispell-local-dictionary: "american"
00036     ### eval: (flyspell-prog-mode)
00037     ### End:
00038 */
00039
00040 #endif // TRISYCL_SYCL_IMAGE_HPP
```

11.59 include/CL/sycl/info/param_traits.hpp File Reference

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

Classes

- class cl::sycl::info::param_traits< T, Param >

  Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Macros

- #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)

  To declare a param_traits returning RETURN_TYPE for function of any T.
- #define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)

  To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

11.59.1 Macro Definition Documentation

11.59.1.1 #define TRISYCL_INFO_PARAM_TRAITS(*VALUE*, *RETURN_TYPE*)

Value:

```
template <>                                                 \
  class param_traits<decltype(VALUE), VALUE> {               \
    using type = RETURN_TYPE;                                \
  };
```

To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

Definition at line 35 of file param_traits.hpp.

11.59.1.2 #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(*T, RETURN_TYPE*)

Value:

```
template <T Param>                                          \
  class param_traits<T, Param> {                            \
    using type = RETURN_TYPE;                               \
  };
```

To declare a param_traits returning RETURN_TYPE for function of any T.

Definition at line 25 of file param_traits.hpp.

11.60 param_traits.hpp

```
00001 #ifndef TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
00002 #define TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
00003
00004 /** \file The OpenCL SYCL param_traits
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 namespace cl {
00013 namespace sycl {
00014 namespace info {
00015
00016 /** Implement a meta-function from (T, value) to T' to express the return type
00017     value of an OpenCL function of kind (T, value)
00018 */
00019 template <typename T, T Param>
00020 class param_traits {
00021 };
00022
00023
00024 /// To declare a param_traits returning RETURN_TYPE for function of any T
00025 #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)      \
00026   template <T Param>                                        \
00027   class param_traits<T, Param> {                            \
00028     using type = RETURN_TYPE;                               \
00029   };
00030
00031
00032 /** To declare a param_traits returning RETURN_TYPE for function taking a
00033     VALUE of type T
00034 */
00035 #define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)        \
00036   template <>                                               \
00037   class param_traits<decltype(VALUE), VALUE> {              \
00038     using type = RETURN_TYPE;                               \
00039   };
00040
00041 }
00042 }
00043 }
00044
00045 /*
00046     # Some Emacs stuff:
00047     ### Local Variables:
00048     ### ispell-local-dictionary: "american"
00049     ### eval: (flyspell-prog-mode)
00050     ### End:
00051 */
00052
00053 #endif // TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
```

11.61 include/CL/sycl/item.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for item.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::item< dims >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. *More...*

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.62 item.hpp

```
00001 #ifndef TRISYCL_SYCL_ITEM_HPP
00002 #define TRISYCL_SYCL_ITEM_HPP
00003
00004 /** \file The OpenCL SYCL item<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/detail/linear_id.hpp"
00015 #include "CL/sycl/id.hpp"
00016 #include "CL/sycl/range.hpp"
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 /** \addtogroup parallelism Expressing parallelism through kernels
00022     @{
00023 */
00024
00025 /** A SYCL item stores information on a work-item with some more context
00026     such as the definition range and offset.
00027 */
00028 template <std::size_t dims = 1>
00029 class item {
00030
00031 public:
00032
00033   /// \todo add this Boost::multi_array or STL concept to the
00034   /// specification?
00035   static constexpr auto dimensionality = dims;
00036
00037 private:
00038
00039   range<dims> global_range;
00040   id<dims> global_index;
00041   id<dims> offset;
00042
00043 public:
00044
00045   /** Create an item from a local size and an optional offset
00046
00047       This constructor is used by the triSYCL implementation and the
00048       non-regression testing.
```

```
00049   */
00050   item(range<dims> global_size,
00051        id<dims> global_index,
00052        id<dims> offset = {}) :
00053     global_range { global_size },
00054     global_index { global_index },
00055     offset { offset }
00056   {}
00057
00058
00059   /** To be able to copy and assign item, use default constructors too
00060
00061       \todo Make most of them protected, reserved to implementation
00062   */
00063   item() = default;
00064
00065
00066   /** Return the constituent local or global id<> representing the
00067       work-item's position in the iteration space
00068   */
00069   id<dims> get() const { return global_index; }
00070
00071
00072   /** Return the requested dimension of the constituent id<> representing
00073       the work-item's position in the iteration space
00074   */
00075   size_t get(int dimension) const { return get()[dimension]; }
00076
00077
00078   /** Return the constituent id<> l-value representing the work-item's
00079       position in the iteration space in the given dimension
00080   */
00081   auto &operator[](int dimension) { return global_index[dimension]; }
00082
00083
00084   /** Returns a range<> representing the dimensions of the range of
00085       possible values of the item
00086   */
00087   range<dims> get_range() const { return global_range; }
00088
00089
00090   /** Returns an id<> representing the n-dimensional offset provided to
00091       the parallel_for and that is added by the runtime to the global-ID
00092       of each work-item, if this item represents a global range
00093
00094       For an item representing a local range of where no offset was passed
00095       this will always return an id of all 0 values.
00096   */
00097   id<dims> get_offset() const { return offset; }
00098
00099
00100   /** Return the linearized ID in the item's range
00101
00102       Computed as the flatted ID after the offset is subtracted.
00103   */
00104   size_t get_linear_id() const {
00105     return detail::linear_id(get_range(), get(),
      get_offset());
00106   }
00107
00108
00109   /** For the implementation, need to set the global index
00110
00111       \todo Move to private and add friends
00112   */
00113   void set(id<dims> Index) { global_index = Index; }
00114
00115
00116   /// Display the value for debugging and validation purpose
00117   void display() const {
00118     global_range.display();
00119     global_index.display();
00120     offset.display();
00121   }
00122
00123 };
00124
00125 /// @} End the parallelism Doxygen group
00126
00127 }
00128 }
00129
00130 /*
00131     # Some Emacs stuff:
00132     ### Local Variables:
00133     ### ispell-local-dictionary: "american"
00134     ### eval: (flyspell-prog-mode)
```

```
00135     ### End:
00136 */
00137
00138 #endif // TRISYCL_SYCL_ITEM_HPP
```

11.63 include/CL/sycl/nd_item.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_item.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::nd_item< dims >

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.64 nd_item.hpp

```
00001 #ifndef TRISYCL_SYCL_ND_ITEM_HPP
00002 #define TRISYCL_SYCL_ND_ITEM_HPP
00003
00004 /** \file The OpenCL SYCL nd_item<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/access.hpp"
00015 #include "CL/sycl/detail/linear_id.hpp"
00016 #include "CL/sycl/detail/unimplemented.hpp"
00017 #include "CL/sycl/id.hpp"
00018 #include "CL/sycl/nd_range.hpp"
00019 #include "CL/sycl/range.hpp"
00020
00021 namespace cl {
00022 namespace sycl {
00023
00024 /** \addtogroup parallelism Expressing parallelism through kernels
00025     @{
00026 */
00027
00028 /** A SYCL nd_item stores information on a work-item within a work-group,
00029     with some more context such as the definition ranges.
00030 */
00031 template <std::size_t dims = 1>
00032 struct nd_item {
00033   /// \todo add this Boost::multi_array or STL concept to the
00034   /// specification?
00035   static constexpr auto dimensionality = dims;
00036
```

```
00037 private:
00038
00039   id<dims> global_index;
00040   /* This is a cached value since it can be computed from global_index and
00041      ND_range */
00042   id<dims> local_index;
00043   nd_range<dims> ND_range;
00044
00045 public:
00046
00047   /** Create an empty nd_item<> from an nd_range<>
00048
00049       \todo This is for the triSYCL implementation which is expected to
00050       call set_global() and set_local() later. This should be hidden to
00051       the user.
00052   */
00053   nd_item(nd_range<dims> ndr) : ND_range { ndr } {}
00054
00055
00056   /** Create a full nd_item
00057
00058       \todo This is for validation purpose. Hide this to the programmer
00059       somehow
00060   */
00061   nd_item(id<dims> global_index,
00062           nd_range<dims> ndr) :
00063     global_index { global_index },
00064     // Compute the local index using the offset and the group size
00065     local_index { (global_index - ndr.get_offset())%id<dims> { ndr.get_local() } },
00066     ND_range { ndr }
00067   {}
00068
00069
00070   /** To be able to copy and assign nd_item, use default constructors too
00071
00072       \todo Make most of them protected, reserved to implementation
00073   */
00074   nd_item() = default;
00075
00076
00077   /** Return the constituent global id representing the work-item's
00078       position in the global iteration space
00079   */
00080   id<dims> get_global() const { return global_index; }
00081
00082
00083   /** Return the constituent element of the global id representing the
00084       work-item's position in the global iteration space in the given
00085       dimension
00086   */
00087   size_t get_global(int dimension) const { return get_global()[dimension]; }
00088
00089
00090   /** Return the flattened id of the current work-item after subtracting
00091       the offset
00092   */
00093   size_t get_global_linear_id() const {
00094     return detail::linear_id(get_global_range(),
      get_global(), get_offset());
00095   }
00096
00097
00098   /** Return the constituent local id representing the work-item's
00099       position within the current work-group
00100   */
00101   id<dims> get_local() const { return local_index; }
00102
00103
00104   /** Return the constituent element of the local id representing the
00105       work-item's position within the current work-group in the given
00106       dimension
00107   */
00108   size_t get_local(int dimension) const { return get_local()[dimension]; }
00109
00110
00111   /** Return the flattened id of the current work-item within the current
00112       work-group
00113    */
00114   size_t get_local_linear_id() const {
00115     return detail::linear_id(get_local_range(),
      get_local());
00116   }
00117
00118
00119   /** Return the constituent group group representing the work-group's
00120       position within the overall nd_range
00121   */
```

```
00122   id<dims> get_group() const {
00123     /* Convert get_local_range() to an id<> to remove ambiguity into using
00124        implicit conversion either from range<> to id<> or the opposite */
00125     return get_global()/id<dims> { get_local_range() };
00126   }
00127
00128
00129   /** Return the constituent element of the group id representing the
00130       work-group;s position within the overall nd_range in the given
00131       dimension.
00132   */
00133   size_t get_group(int dimension) const {
00134     return get_group()[dimension];
00135   }
00136
00137
00138   /// Return the flattened id of the current work-group
00139   size_t get_group_linear_id() const {
00140     return detail::linear_id(get_num_groups(),
      get_group());
00141   }
00142
00143
00144   /// Return the number of groups in the nd_range
00145   id<dims> get_num_groups() const {
00146     return get_nd_range().get_group();
00147   }
00148
00149   /// Return the number of groups for dimension in the nd_range
00150   size_t get_num_groups(int dimension) const {
00151      return get_num_groups()[dimension];
00152   }
00153
00154
00155   /// Return a range<> representing the dimensions of the nd_range<>
00156   range<dims> get_global_range() const {
00157     return get_nd_range().get_global();
00158   }
00159
00160
00161   /// Return a range<> representing the dimensions of the current work-group
00162   range<dims> get_local_range() const {
00163     return get_nd_range().get_local();
00164   }
00165
00166
00167   /** Return an id<> representing the n-dimensional offset provided to the
00168       constructor of the nd_range<> and that is added by the runtime to the
00169       global-ID of each work-item
00170   */
00171   id<dims> get_offset() const { return get_nd_range().get_offset(); }
00172
00173
00174   /// Return the nd_range<> of the current execution
00175   nd_range<dims> get_nd_range() const { return
      ND_range; }
00176
00177
00178   /** Execute a barrier with memory ordering on the local address space,
00179       global address space or both based on the value of flag
00180
00181       The current work-item will wait at the barrier until all work-items
00182       in the current work-group have reached the barrier.
00183
00184       In addition, the barrier performs a fence operation ensuring that all
00185       memory accesses in the specified address space issued before the
00186       barrier complete before those issued after the barrier
00187   */
00188   void barrier(access::fence_space flag =
00189                access::fence_space::global_and_local) const {
00190 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00191     /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00192        work-item of the work-group */
00193 #pragma omp barrier
00194 #else
00195     // \todo To be implemented efficiently otherwise
00196     detail::unimplemented();
00197 #endif
00198   }
00199
00200
00201   // For the triSYCL implementation, need to set the local index
00202   void set_local(id<dims> Index) { local_index = Index; }
00203
00204
00205   // For the triSYCL implementation, need to set the global index
00206   void set_global(id<dims> Index) { global_index = Index; }
```

```
00207
00208 };
00209
00210 /// @} End the parallelism Doxygen group
00211
00212 }
00213 }
00214
00215 /*
00216     # Some Emacs stuff:
00217     ### Local Variables:
00218     ### ispell-local-dictionary: "american"
00219     ### eval: (flyspell-prog-mode)
00220     ### End:
00221 */
00222
00223 #endif // TRISYCL_SYCL_ND_ITEM_HPP
```

11.65 include/CL/sycl/nd_range.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_range.hpp: This graph shows which files directly or indirectly include this file:

Classes

- 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...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.66 nd_range.hpp

```
00001 #ifndef TRISYCL_SYCL_ND_RANGE_HPP
00002 #define TRISYCL_SYCL_ND_RANGE_HPP
00003
00004 /** \file The OpenCL SYCL nd_range<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013
00014 #include "CL/sycl/id.hpp"
00015 #include "CL/sycl/range.hpp"
00016
00017 namespace cl {
00018 namespace sycl {
00019
00020 /** \addtogroup parallelism Expressing parallelism through kernels
00021     @{
00022 */
00023
00024 /** A ND-range, made by a global and local range, to specify work-group
00025     and work-item organization.
00026
00027     The local offset is used to translate the iteration space origin if
00028     needed.
00029
00030     \todo add copy constructors in the specification
```

```
00031 */
00032 template <std::size_t dims = 1>
00033 struct nd_range {
00034   /// \todo add this Boost::multi_array or STL concept to the
00035   /// specification?
00036   static constexpr auto dimensionality = dims;
00037
00038 private:
00039
00040   range<dimensionality> global_range;
00041   range<dimensionality> local_range;
00042   id<dimensionality> offset;
00043
00044 public:
00045
00046   /** Construct a ND-range with all the details available in OpenCL
00047
00048       By default use a zero offset, that is iterations start at 0
00049    */
00050   nd_range(range<dims> global_size,
00051            range<dims> local_size,
00052            id<dims> offset = {}) :
00053     global_range { global_size }, local_range { local_size }, offset { offset }
00054   { }
00055
00056
00057   /// Get the global iteration space range
00058   range<dims> get_global() const { return global_range; }
00059
00060
00061   /// Get the local part of the iteration space range
00062   range<dims> get_local() const { return local_range; }
00063
00064
00065   /// Get the range of work-groups needed to run this ND-range
00066   auto get_group() const {
00067     // \todo Assume that global_range is a multiple of local_range, element-wise
00068     return global_range/local_range;
00069   }
00070
00071
00072   /// \todo get_offset() is lacking in the specification
00073   id<dims> get_offset() const { return offset; }
00074
00075
00076   /// Display the value for debugging and validation purpose
00077   void display() const {
00078     global_range.display();
00079     local_range.display();
00080     offset.display();
00081   }
00082
00083 };
00084
00085 /// @} End the parallelism Doxygen group
00086
00087 }
00088 }
00089
00090 /*
00091     # Some Emacs stuff:
00092     ### Local Variables:
00093     ### ispell-local-dictionary: "american"
00094     ### eval: (flyspell-prog-mode)
00095     ### End:
00096 */
00097
00098 #endif // TRISYCL_SYCL_ND_RANGE_HPP
```

11.67 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference

Implement the detail of the parallel constructions to launch kernels.

```
#include <cstddef>
#include <boost/multi_array.hpp>
#include "CL/sycl/group.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for parallelism.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

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

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

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

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

  Implementation of parallel_for with a range<> and an offset.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (group< Dimensions > g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.

11.67.1 Detailed Description

Implement the detail of the parallel constructions to launch kernels.

Ronan at keryell dot FR

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

Definition in file parallelism.hpp.

11.68 parallelism.hpp

```
00001 #ifndef TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
00002 #define TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
00003
00004 /** \file
00005
00006     Implement the detail of the parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include <cstddef>
00015 #include <boost/multi_array.hpp>
00016
00017 #include "CL/sycl/group.hpp"
00018 #include "CL/sycl/id.hpp"
00019 #include "CL/sycl/item.hpp"
00020 #include "CL/sycl/nd_item.hpp"
00021 #include "CL/sycl/nd_range.hpp"
00022 #include "CL/sycl/range.hpp"
00023
00024 #ifdef _OPENMP
00025 #include <omp.h>
00026 #endif
00027
00028
00029 /** \addtogroup parallelism
00030     @{
00031 */
00032
00033 namespace cl {
00034 namespace sycl {
00035 namespace detail {
00036
00037
00038 /** A recursive multi-dimensional iterator that ends calling f
00039
00040     The iteration order may be changed later.
00041
00042     Since partial specialization of function template is not possible in
00043     C++14, use a class template instead with everything in the
00044     constructor.
00045 */
00046 template <std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
00047 struct parallel_for_iterate {
00048   parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
00049     for (boost::multi_array_types::index _sycl_index = 0,
00050            _sycl_end = r[Range::dimensionality - level];
00051          _sycl_index < _sycl_end;
00052          _sycl_index++) {
00053       // Set the current value of the index for this dimension
00054       index[Range::dimensionality - level] = _sycl_index;
00055       // Iterate further on lower dimensions
00056       parallel_for_iterate<level - 1,
00057                            Range,
00058                            ParallelForFunctor,
00059                            Id> { r, f, index };
00060     }
00061   }
00062 };
00063
00064
00065 /** A top-level recursive multi-dimensional iterator variant using OpenMP
00066
00067     Only the top-level loop uses OpenMP and go on with the normal
00068     recursive multi-dimensional.
00069 */
00070 template <std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
00071 struct parallel_OpenMP_for_iterate {
00072   parallel_OpenMP_for_iterate(Range r, ParallelForFunctor &f) {
00073     // Create the OpenMP threads before the for loop to avoid creating an
00074     // index in each iteration
00075 #pragma omp parallel
00076     {
00077       // Allocate an OpenMP thread-local index
00078       Id index;
00079       // Make a simple loop end condition for OpenMP
00080       boost::multi_array_types::index _sycl_end = r[Range::dimensionality - level];
00081       /* Distribute the iterations on the OpenMP threads. Some OpenMP
00082          "collapse" could be useful for small iteration space, but it
00083          would need some template specialization to have real contiguous
00084          loop nests */
```

```
00085 #pragma omp for
00086       for (boost::multi_array_types::index _sycl_index = 0;
00087            _sycl_index < _sycl_end;
00088            _sycl_index++) {
00089         // Set the current value of the index for this dimension
00090         index[Range::dimensionality - level] = _sycl_index;
00091         // Iterate further on lower dimensions
00092         parallel_for_iterate<level - 1,
00093                              Range,
00094                              ParallelForFunctor,
00095                              Id> { r, f, index };
00096       }
00097     }
00098   }
00099 };
00100
00101
00102 /** Stop the recursion when level reaches 0 by simply calling the
00103     kernel functor with the constructed id */
00104 template <typename Range, typename ParallelForFunctor, typename Id>
00105 struct parallel_for_iterate<0, Range, ParallelForFunctor, Id> {
00106   parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
00107     f(index);
00108   }
00109 };
00110
00111
00112 /** Implementation of a data parallel computation with parallelism
00113     specified at launch time by a range<>.
00114
00115     This implementation use OpenMP 3 if compiled with the right flag.
00116 */
00117 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00118 void parallel_for(range<Dimensions> r,
00119                   ParallelForFunctor f) {
00120 #ifdef _OPENMP
00121   // Use OpenMP for the top loop level
00122   parallel_OpenMP_for_iterate<Dimensions,
00123                               range<Dimensions>,
00124                               ParallelForFunctor,
00125                               id<Dimensions>> { r, f };
00126 #else
00127   // In a sequential execution there is only one index processed at a time
00128   id<Dimensions> index;
00129   parallel_for_iterate<Dimensions,
00130                        range<Dimensions>,
00131                        ParallelForFunctor,
00132                        id<Dimensions>> { r, f, index };
00133 #endif
00134 }
00135
00136
00137 /** Implementation of parallel_for with a range<> and an offset */
00138 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00139 void parallel_for_global_offset(range<Dimensions> global_size,
00140                                 id<Dimensions> offset,
00141                                 ParallelForFunctor f) {
00142   // Reconstruct the item from its id<> and its offset
00143   auto reconstruct_item = [&] (id<Dimensions> l) {
00144     // Reconstruct the global item
00145     item<Dimensions> index { global_size, l + offset, offset };
00146     // Call the user kernel with the item<> instead of the id<>
00147     f(index);
00148   };
00149
00150   // First iterate on all the work-groups
00151   parallel_for(global_size, reconstruct_item);
00152 }
00153
00154
00155 /** Implement a variation of parallel_for to take into account a
00156     nd_range<>
00157
00158     \todo Add an OpenMP implementation
00159
00160     \todo Deal with incomplete work-groups
00161
00162     \todo Implement with parallel_for_workgroup()/parallel_for_workitem()
00163 */
00164 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00165 void parallel_for(nd_range<Dimensions> r,
00166                   ParallelForFunctor f) {
00167   // In a sequential execution there is only one index processed at a time
00168   nd_item<Dimensions> index { r };
00169   // To iterate on the work-group
00170   id<Dimensions> group;
00171   range<Dimensions> group_range = r.get_group();
```

```
00172   // To iterate on the local work-item
00173   id<Dimensions> local;
00174
00175   range<Dimensions> local_range = r.get_local();
00176
00177   // Reconstruct the nd_item from its group and local id
00178   auto reconstruct_item = [&] (id<Dimensions> l) {
00179     //local.display();
00180     // Reconstruct the global nd_item
00181     index.set_local(local);
00182     // Upgrade local_range to an id<> so that we can * with the group (an id<>)
00183     index.set_global(local + id<Dimensions>(local_range)*group);
00184     // Call the user kernel at last
00185     f(index);
00186   };
00187
00188   /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
00189      into another functor that iterates inside the work-group and then
00190      calls f */
00191   auto iterate_in_work_group = [&] (id<Dimensions> g) {
00192     //group.display();
00193     // Then iterate on the local work-groups
00194     parallel_for_iterate<Dimensions,
00195                          range<Dimensions>,
00196                          decltype(reconstruct_item),
00197                          id<Dimensions>> { local_range, reconstruct_item, local };
00198   };
00199
00200   // First iterate on all the work-groups
00201   parallel_for_iterate<Dimensions,
00202                        range<Dimensions>,
00203                        decltype(iterate_in_work_group),
00204                        id<Dimensions>> { group_range, iterate_in_work_group, group };
00205 }
00206
00207
00208 /// Implement the loop on the work-groups
00209 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00210 void parallel_for_workgroup(nd_range<Dimensions> r,
00211                             ParallelForFunctor f) {
00212   // In a sequential execution there is only one index processed at a time
00213   group<Dimensions> g { r };
00214
00215   // First iterate on all the work-groups
00216   parallel_for_iterate<Dimensions,
00217                        range<Dimensions>,
00218                        ParallelForFunctor,
00219                        group<Dimensions>> {
00220     r.get_group(),
00221     f,
00222     g };
00223 }
00224
00225
00226 /// Implement the loop on the work-items inside a work-group
00227 template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00228 void parallel_for_workitem(group<Dimensions> g,
00229                            ParallelForFunctor f) {
00230 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00231   /* To implement barriers With OpenMP, one thread is created for each
00232      work-item in the group and thus an OpenMP barrier has the same effect
00233      of an OpenCL barrier executed by the work-items in a workgroup
00234
00235      The issue is that the parallel_for_workitem() execution is slow even
00236      when nd_item::barrier() is not used
00237   */
00238   range<Dimensions> l_r = g.get_nd_range().get_local();
00239   // \todo Implement with a reduction algorithm
00240   int tot = l_r.get(0);
00241   for (int i = 1; i < (int) Dimensions; ++i){
00242     tot *= l_r.get(i);
00243   }
00244   /* An alternative could be to use 1 to 3 loops with #pragma omp parallel
00245      for collapse(...) instead of reconstructing the iteration index from
00246      the thread number */
00247   omp_set_num_threads(tot);
00248 #pragma omp parallel
00249   {
00250     int th_id = omp_get_thread_num();
00251     nd_item<Dimensions> index { g.get_nd_range() };
00252     id<Dimensions> local; // to initialize correctly
00253
00254     if (Dimensions ==1) {
00255       local[0] = th_id;
00256     } else if (Dimensions == 2) {
00257       local[0] = th_id / l_r.get(1);
00258       local[1] = th_id - local[0]*l_r.get(1);
```

```
00259     } else if (Dimensions == 3) {
00260       int tmp = l_r.get(1)*l_r.get(2);
00261       local[0] = th_id / tmp;
00262       local[1] = (th_id - local[0]*tmp) / l_r.get(1);
00263       local[2] = th_id - local[0]*tmp - local[1]*l_r.get(1);
00264     }
00265     index.set_local(local);
00266     index.set_global(local + id<Dimensions>(l_r)*g.get());
00267     f(index);
00268   }
00269 #else
00270   // In a sequential execution there is only one index processed at a time
00271   nd_item<Dimensions> index { g.get_nd_range() };
00272   // To iterate on the local work-item
00273   id<Dimensions> local;
00274
00275   // Reconstruct the nd_item from its group and local id
00276   auto reconstruct_item = [&] (id<Dimensions> l) {
00277     //local.display();
00278     //l.display();
00279     // Reconstruct the global nd_item
00280     index.set_local(local);
00281     // \todo Some strength reduction here
00282     index.set_global(local + id<Dimensions>(g.get_local_range())*g.
      get());
00283     // Call the user kernel at last
00284     f(index);
00285   };
00286
00287   // Then iterate on all the work-items of the work-group
00288   parallel_for_iterate<Dimensions,
00289                        range<Dimensions>,
00290                        decltype(reconstruct_item),
00291                        id<Dimensions>> {
00292     g.get_local_range(),
00293     reconstruct_item,
00294     local };
00295 #endif
00296 }
00297 /// @} End the parallelism Doxygen group
00298
00299 }
00300 }
00301 }
00302
00303 /*
00304     # Some Emacs stuff:
00305     ### Local Variables:
00306     ### ispell-local-dictionary: "american"
00307     ### eval: (flyspell-prog-mode)
00308     ### End:
00309 */
00310
00311 #endif // TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
```

11.69 include/CL/sycl/parallelism.hpp File Reference

Implement parallel constructions to launch kernels.

`#include "CL/sycl/parallelism/detail/parallelism.hpp"`
Include dependency graph for parallelism.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- template<std::size_t Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (group< Dimensions > g, ParallelForFunctor f)

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

11.69.1 Detailed Description

Implement parallel constructions to launch kernels.

Ronan at keryell dot FR

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

Definition in file parallelism.hpp.

11.70 parallelism.hpp

```
00001 #ifndef TRISYCL_SYCL_PARALLELISM_HPP
00002 #define TRISYCL_SYCL_PARALLELISM_HPP
00003
00004 /** \file
00005
00006     Implement parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include "CL/sycl/parallelism/detail/parallelism.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup parallelism
00020     @{
00021 */
00022
00023 /// SYCL parallel_for version that allows a Program object to be specified
00024 /// \todo To be implemented
00025 /* template <typename Range, typename Program, typename ParallelForFunctor>
00026 void parallel_for(Range r, Program p, ParallelForFunctor f) {
00027   /// \todo deal with Program
00028   parallel_for(r, f);
00029 }
00030 */
00031
00032   /// Loop on the work-items inside a work-group
00033   template <std::size_t Dimensions = 1, typename ParallelForFunctor>
00034   void parallel_for_work_item(group<Dimensions> g,
      ParallelForFunctor f) {
00035     detail::parallel_for_workitem(g, f);
00036   }
00037
00038
00039
00040 }
00041 }
00042
00043 /// @} End the parallelism Doxygen group
00044
00045 /*
00046     # Some Emacs stuff:
00047     ### Local Variables:
00048     ### ispell-local-dictionary: "american"
00049     ### eval: (flyspell-prog-mode)
00050     ### End:
00051 */
00052
00053 #endif // TRISYCL_SYCL_PARALLELISM_HPP
```

11.71 include/CL/sycl/platform.hpp File Reference

```
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
```

Include dependency graph for platform.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::platform

  Abstract the OpenCL platform. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Enumerations

- enum cl::sycl::info::device_type : unsigned int {
  cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl::sycl::info::device_type::custom,
  cl::sycl::info::device_type::defaults, cl::sycl::info::device_type::host, cl::sycl::info::device_type::all }
- enum cl::sycl::info::platform : unsigned int {
  cl::sycl::info::platform::profile, cl::sycl::info::platform::version, cl::sycl::info::platform::name, cl::sycl::info::platform::vendor,
  cl::sycl::info::platform::extensions }

  Platform information descriptors.

11.72 platform.hpp

```
00001 #ifndef TRISYCL_SYCL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_HPP
00003
00004 /** \file The OpenCL SYCL platform
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/context.hpp"
00013 #include "CL/sycl/detail/default_classes.hpp"
00014 #include "CL/sycl/detail/unimplemented.hpp"
00015 #include "CL/sycl/device.hpp"
00016
00017 namespace cl {
00018 namespace sycl {
00019
00020 class device_selector;
00021 class device;
00022
00023 /** \addtogroup execution Platforms, contexts, devices and queues
00024     @{
00025 */
00026 namespace info {
00027
00028 enum class device_type : unsigned int {
00029   cpu,
00030   gpu,
00031   accelerator,
00032   custom,
00033   defaults,
00034   host,
00035   all
00036 };
00037
00038
00039 /** Platform information descriptors
00040
00041     A SYCL platform can be queried for all of the following information
00042     using the get_info function. All SYCL contexts have valid devices for
```

```
00043     them, including the SYCL host device.
00044 */
00045 enum class platform : unsigned int {
00046   /** Returns the profile name (as a string_class) supported by the im-
00047       plementation.
00048
00049       Can be either FULL PROFILE or EMBEDDED PROFILE.
00050   */
00051   profile,
00052   /** Returns the OpenCL software driver version string in the form major
00053       number.minor number (as a string_class)
00054   */
00055   version,
00056   /** Returns the name of the platform (as a string_class)
00057   */
00058   name,
00059   /** Returns the string provided by the platform vendor (as a string_class)
00060   */
00061   vendor,
00062   /** Returns a space-separated list of extension names supported by the
00063       platform (as a string_class)
00064   */
00065   extensions
00066 };
00067
00068
00069 /** Query the return type for get_info() on platform stuff
00070
00071     Only return a string_class
00072 */
00073 TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::platform,
    string_class)
00074
00075 }
00076
00077 /** Abstract the OpenCL platform
00078
00079     \todo triSYCL Implementation
00080 */
00081 class platform {
00082
00083
00084 public:
00085
00086 #ifdef TRISYCL_OPENCL
00087   /** Construct a default platform and provide an optional error_handler
00088       to deals with errors
00089
00090       \todo Add copy/move constructor to the implementation
00091
00092       \todo Add const to the specification
00093   */
00094   explicit platform(cl_platform_id platformID) {
00095     detail::unimplemented();
00096   }
00097 #endif
00098
00099   /** Default constructor for platform
00100
00101       It constructs a platform object to encapsulate the device returned
00102       by the default device selector.
00103
00104       Returns errors via the SYCL exception class.
00105
00106       Get back the default constructors, for this implementation.
00107   */
00108   platform() = default;
00109
00110
00111 #ifdef TRISYCL_OPENCL
00112   /** Returns the cl_platform_id of the underlying
00113       OpenCL platform
00114
00115       If the platform is not a valid OpenCL platform, for example if it is
00116       the SYCL host, a nullptr will be returned.
00117
00118       \todo To be implemented
00119   */
00120   cl_platform_id get() const {
00121     detail::unimplemented();
00122     return {};
00123   }
00124 #endif
00125
00126
00127   /** Get the list of all the platforms available to the application
00128
```

```
00129       \todo To be implemented
00130   */
00131   static vector_class<platform> get_platforms() {
00132     detail::unimplemented();
00133     return {};
00134   }
00135
00136
00137   /** Returns all the available devices for this platform, of type device
00138       type, which is defaulted to info::device_type::all
00139
00140
00141       By default returns all the devices.
00142   */
00143   vector_class<device>
00144   get_devices(info::device_type device_type =
      info::device_type::all) const {
00145     detail::unimplemented();
00146     return {};
00147   }
00148
00149
00150   /** Get the OpenCL information about the requested parameter
00151
00152       \todo To be implemented
00153   */
00154   template <info::platform Param>
00155   typename info::param_traits<info::platform, Param>::type
00156   get_info() const {
00157     detail::unimplemented();
00158     return {};
00159   }
00160
00161
00162   /** Test if an extension is available on the platform
00163
00164       \todo Should it be a param type instead of a STRING?
00165
00166       \todo extend to any type of C++-string like object
00167   */
00168   bool has_extension(const string_class &extension) const {
00169     detail::unimplemented();
00170     return {};
00171   }
00172
00173
00174   /// Test if this platform is a host platform
00175   bool is_host() const {
00176     // Right now, this is a host-only implementation :-)
00177     return true;
00178   }
00179
00180 };
00181
00182 /// @} to end the execution Doxygen group
00183
00184 }
00185 }
00186
00187 /*
00188     # Some Emacs stuff:
00189     ### Local Variables:
00190     ### ispell-local-dictionary: "american"
00191     ### eval: (flyspell-prog-mode)
00192     ### End:
00193 */
00194
00195 #endif // TRISYCL_SYCL_PLATFORM_HPP
```

11.73 include/CL/sycl/queue.hpp File Reference

```
#include "CL/sycl/context.hpp"
```

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/handler_event.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/parallelism.hpp"
```

Include dependency graph for queue.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::queue_profiling = bool

Enumerations

- enum cl::sycl::info::queue : int { cl::sycl::info::queue::context, cl::sycl::info::queue::device, cl::sycl::info::queue::reference_count, cl::sycl::info::queue::properties }

  Queue information descriptors.

11.74 queue.hpp

```
00001 #ifndef TRISYCL_SYCL_QUEUE_HPP
00002 #define TRISYCL_SYCL_QUEUE_HPP
00003
00004 /** \file The OpenCL SYCL queue
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/context.hpp"
00013 #include "CL/sycl/detail/default_classes.hpp"
00014 #include "CL/sycl/detail/unimplemented.hpp"
00015 #include "CL/sycl/device.hpp"
00016 #include "CL/sycl/device_selector.hpp"
00017 #include "CL/sycl/handler.hpp"
00018 #include "CL/sycl/handler_event.hpp"
00019 #include "CL/sycl/exception.hpp"
00020 #include "CL/sycl/info/param_traits.hpp"
00021 #include "CL/sycl/parallelism.hpp"
00022
00023 namespace cl {
00024 namespace sycl {
00025
00026 class context;
```

```
00027 class device_selector;
00028
00029 /** \addtogroup execution Platforms, contexts, devices and queues
00030     @{
00031 */
00032
00033 namespace info {
00034
00035 using queue_profiling = bool;
00036
00037 /** Queue information descriptors
00038
00039     From specification C.4
00040
00041     \todo unsigned int?
00042
00043     \todo To be implemented
00044 */
00045 enum class queue : int {
00046   context,
00047   device,
00048   reference_count,
00049   properties
00050 };
00051
00052 /** Dummy example for get_info() on queue::context that would return a
00053     context
00054
00055     \todo Describe all the types
00056 */
00057 TRISYCL_INFO_PARAM_TRAITS(queue::context,
     context)
00058
00059 }
00060
00061
00062 /** SYCL queue, similar to the OpenCL queue concept.
00063
00064     \todo The implementation is quite minimal for now. :-)
00065 */
00066 class queue {
00067
00068 public:
00069   /** This constructor creates a SYCL queue from an OpenCL queue
00070
00071       At construction it does a retain on the queue memory object.
00072
00073       Retain a reference to the cl_command_queue object. Caller should
00074       release the passed cl_command_queue object when it is no longer
00075       needed.
00076
00077       Return synchronous errors regarding the creation of the queue and
00078       report asynchronous errors via the async_handler callback function
00079       in conjunction with the synchronization and throw methods.
00080
00081       Note that the default case asyncHandler = nullptr is handled by the
00082       default constructor.
00083
00084   */
00085   explicit queue(async_handler asyncHandler) {
00086     detail::unimplemented();
00087   }
00088
00089
00090   /** Creates a queue for the device provided by the device selector
00091
00092       If no device is selected, an error is reported.
00093
00094       Return synchronous errors regarding the creation of the queue and
00095       report asynchronous errors via the async_handler callback
00096       function if and only if there is an async_handler provided.
00097   */
00098   queue(const device_selector &deviceSelector,
00099         async_handler asyncHandler = nullptr) {
00100     detail::unimplemented();
00101   }
00102
00103
00104   /** A queue is created for syclDevice
00105
00106       Return asynchronous errors via the async_handler callback function.
00107   */
00108   queue(const device &syclDevice,
00109         async_handler asyncHandler = nullptr) {
00110     detail::unimplemented();
00111   };
00112
```

```
00113
00114   /** This constructor chooses a device based on the provided
00115       device_selector, which needs to be in the given context.
00116
00117       If no device is selected, an error is reported.
00118
00119       Return synchronous errors regarding the creation of the queue.
00120
00121       If and only if there is an asyncHandler provided, it reports
00122       asynchronous errors via the async_handler callback function in
00123       conjunction with the synchronization and throw methods.
00124   */
00125   queue(const context &syclContext,
00126         const device_selector &deviceSelector,
00127         async_handler asyncHandler = nullptr) {
00128     detail::unimplemented();
00129   }
00130
00131
00132   /** Creates a command queue using clCreateCommandQueue from a context
00133       and a device
00134
00135       Return synchronous errors regarding the creation of the queue.
00136
00137       If and only if there is an asyncHandler provided, it reports
00138       asynchronous errors via the async_handler callback function in
00139       conjunction with the synchronization and throw methods.
00140   */
00141   queue(const context &syclContext,
00142         const device &syclDevice,
00143         async_handler asyncHandler = nullptr) {
00144     detail::unimplemented();
00145   }
00146
00147
00148   /** Creates a command queue using clCreateCommandQueue from a context
00149       and a device
00150
00151       It enables profiling on the queue if the profilingFlag is set to
00152       true.
00153
00154       Return synchronous errors regarding the creation of the queue. If
00155       and only if there is an asyncHandler provided, it reports
00156       asynchronous errors via the async_handler callback function in
00157       conjunction with the synchronization and throw methods.
00158   */
00159   queue(const context &syclContext,
00160         const device &syclDevice,
00161         info::queue_profiling profilingFlag,
00162         async_handler asyncHandler = nullptr) {
00163     detail::unimplemented();
00164   }
00165
00166
00167 #ifdef TRISYCL_OPENCL
00168   /** This constructor creates a SYCL queue from an OpenCL queue
00169
00170       At construction it does a retain on the queue memory object.
00171
00172       Return synchronous errors regarding the creation of the queue. If
00173       and only if there is an async_handler provided, it reports
00174       asynchronous errors via the async_handler callback function in
00175       conjunction with the synchronization and throw methods.
00176   */
00177   queue(const cl_command_queue &clQueue,
00178         async_handler asyncHandler = nullptr) {
00179     detail::unimplemented();
00180   }
00181 #endif
00182
00183
00184   /// Get the default constructors back.
00185   queue() = default;
00186
00187
00188 #ifdef TRISYCL_OPENCL
00189   /** Return the underlying OpenCL command queue after doing a retain
00190
00191       This memory object is expected to be released by the developer.
00192
00193       Retain a reference to the returned cl_command_queue object.
00194
00195       Caller should release it when finished.
00196
00197       If the queue is a SYCL host queue then a nullptr will be returned.
00198   */
00199   cl_command_queue get() const {
```

```
00200     detail::unimplemented();
00201     return {};
00202   }
00203 #endif
00204
00205
00206   /** Return the SYCL queue's context
00207
00208       Report errors using SYCL exception classes.
00209   */
00210   context get_context() const {
00211     detail::unimplemented();
00212     return {};
00213   }
00214
00215
00216   /** Return the SYCL device the queue is associated with
00217
00218       Report errors using SYCL exception classes.
00219   */
00220   device get_device() const {
00221     detail::unimplemented();
00222     return {};
00223   }
00224
00225
00226   /** Return whether the queue is executing on a SYCL host device
00227   */
00228   bool is_host() const {
00229     detail::unimplemented();
00230     return true;
00231   }
00232
00233   /** Performs a blocking wait for the completion all enqueued tasks in
00234       the queue
00235
00236       Synchronous errors will be reported through SYCL exceptions.
00237   */
00238   void wait() {
00239     detail::unimplemented();
00240   }
00241
00242
00243   /** Perform a blocking wait for the completion all enqueued tasks in the queue
00244
00245       Synchronous errors will be reported via SYCL exceptions.
00246
00247       Asynchronous errors will be passed to the async_handler passed to the
00248       queue on construction.
00249
00250       If no async_handler was provided then asynchronous exceptions will
00251       be lost.
00252   */
00253   void wait_and_throw() {
00254     detail::unimplemented();
00255   }
00256
00257
00258   /** Checks to see if any asynchronous errors have been produced by the
00259       queue and if so reports them by passing them to the async_handler
00260       passed to the queue on construction
00261
00262       If no async_handler was provided then asynchronous exceptions will
00263       be lost.
00264   */
00265   void throw_asynchronous() {
00266     detail::unimplemented();
00267   }
00268
00269
00270   /// Queries the platform for cl_command_queue info
00271   template <info::queue param>
00272   typename info::param_traits<info::queue, param>::type
    get_info() const  {
00273     detail::unimplemented();
00274     return {};
00275   }
00276
00277
00278   /** Submit a command group functor to the queue, in order to be
00279       scheduled for execution on the device
00280
00281       Use an explicit functor parameter taking a handler& so we can use
00282       "auto" in submit() lambda parameter.
00283   */
00284   handler_event submit(std::function<void(handler &)> cgf) {
00285     handler command_group_handler;
```

```
00286     cgf(command_group_handler);
00287     return {};
00288   }
00289
00290
00291   /** Submit a command group functor to the queue, in order to be
00292       scheduled for execution on the device
00293
00294       On kernel error, this command group functor, then it is scheduled
00295       for execution on the secondary queue.
00296
00297       Return a command group functor event, which is corresponds to the
00298       queue the command group functor is being enqueued on.
00299   */
00300   handler_event submit(std::function<void(handler &)> cgf,
      queue &secondaryQueue) {
00301     detail::unimplemented();
00302     // Since it is not implemented, always submit on the main queue
00303     return submit(cgf);
00304   }
00305
00306 };
00307
00308 /// @} to end the execution Doxygen group
00309
00310 }
00311 }
00312
00313 /*
00314     # Some Emacs stuff:
00315     ### Local Variables:
00316     ### ispell-local-dictionary: "american"
00317     ### eval: (flyspell-prog-mode)
00318     ### End:
00319 */
00320
00321 #endif // TRISYCL_SYCL_QUEUE_HPP
```

11.75 include/CL/sycl/range.hpp File Reference

`#include "CL/sycl/detail/small_array.hpp"`
Include dependency graph for range.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::range< dims >

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType...Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

11.76 range.hpp

```
00001 #ifndef TRISYCL_SYCL_RANGE_HPP
00002 #define TRISYCL_SYCL_RANGE_HPP
00003
00004 /** \file The OpenCL SYCL range<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include "CL/sycl/detail/small_array.hpp"
00013
00014 namespace cl {
00015 namespace sycl {
00016
00017 /** \addtogroup parallelism Expressing parallelism through kernels
00018     @{
00019 */
00020
00021 /** A SYCL range defines a multi-dimensional index range that can be used
00022     to define launch parallel computation extent or buffer sizes.
00023
00024     \todo use std::size_t dims instead of int dims in the specification?
00025
00026     \todo add to the specification this default parameter value?
00027
00028     \todo add to the specification some way to specify an offset?
00029 */
00030 template <std::size_t dims = 1>
00031 class range : public detail::small_array_123<std::size_t, range<dims>, dims> {
00032
00033 public:
00034
00035   // Inherit of all the constructors
00036   using detail::small_array_123<std::size_t,
00037                                 range<dims>,
00038                                 dims>::small_array_123;
00039 };
00040
00041
00042 /** Implement a make_range to construct a range<> of the right dimension
00043     with implicit conversion from an initializer list for example.
00044
00045     Cannot use a template on the number of dimensions because the implicit
00046     conversion would not be tried.
00047 */
00048 inline auto make_range(range<1> r) { return r; }
00049 inline auto make_range(range<2> r) { return r; }
00050 inline auto make_range(range<3> r) { return r; }
00051
00052
00053 /** Construct a range<> from a function call with arguments, like
00054     make_range(1, 2, 3)
00055 */
00056 template<typename... BasicType>
00057 auto make_range(BasicType... Args) {
00058   // Call constructor directly to allow narrowing
00059   return range<sizeof...(Args)>(Args...);
00060 }
00061
00062 /// @} End the parallelism Doxygen group
00063
00064 }
00065 }
00066
00067 /*
00068     # Some Emacs stuff:
00069     ### Local Variables:
00070     ### ispell-local-dictionary: "american"
00071     ### eval: (flyspell-prog-mode)
00072     ### End:
00073 */
00074
00075 #endif // TRISYCL_SYCL_RANGE_HPP
```

11.77 include/CL/sycl/vec.hpp File Reference

Implement the small OpenCL vector class.

`#include "CL/sycl/detail/array_tuple_helpers.hpp"`
Include dependency graph for vec.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

11.77.1 Detailed Description

Implement the small OpenCL vector class.

Ronan at Keryell point FR

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

Definition in file vec.hpp.

11.78 vec.hpp

```
00001 #ifndef TRISYCL_SYCL_VEC_HPP
00002 #define TRISYCL_SYCL_VEC_HPP
00003
00004 /** \file
00005
00006     Implement the small OpenCL vector class
00007
00008     Ronan at Keryell point FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 #include "CL/sycl/detail/array_tuple_helpers.hpp"
00015
00016 namespace cl {
00017 namespace sycl {
00018
00019 /** \addtogroup vector Vector types in SYCL
00020
00021     @{
00022 */
00023
00024
00025 /** Small OpenCL vector class
00026
00027     \todo add [] operator
00028
```

```
00029     \todo add iterators on elements, with begin() and end()
00030
00031     \todo having vec<> sub-classing array<> instead would solve the
00032     previous issues
00033
00034     \todo move the implementation elsewhere
00035
00036     \todo simplify the helpers by removing some template types since there
00037     are now inside the vec<> class.
00038
00039     \todo rename in the specification element_type to value_type
00040 */
00041 template <typename DataType, size_t NumElements>
00042 class vec : public detail::small_array<DataType,
00043                                        vec<DataType, NumElements>,
00044                                        NumElements> {
00045   using basic_type = typename detail::small_array<DataType,
00046                                                   vec<DataType, NumElements>,
00047                                                   NumElements>;
00048
00049 public:
00050
00051   /** Construct a vec from anything from a scalar (to initialize all the
00052       elements with this value) up to an aggregate of scalar and vector
00053       types (in this case the total number of elements must match the size
00054       of the vector)
00055   */
00056   template <typename... Types>
00057   vec(const Types... args)
00058     : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }
00059
00060
00061 /// Use classical constructors too
00062   vec() = default;
00063
00064
00065   // Inherit of all the constructors
00066   using typename basic_type::small_array;
00067
00068 private:
00069
00070   /** Flattening helper that does not change scalar values but flatten a
00071       vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }
00072
00073       If we have a vector, just forward its array content since an array
00074       has also a tuple interface :-) (23.3.2.9 Tuple interface to class
00075       template array [array.tuple])
00076   */
00077   template <typename V, typename Element, size_t s>
00078   static auto flatten(const vec<Element, s> i) {
00079     static_assert(s <= V::dimension,
00080                   "The element i will not fit in the vector");
00081     return static_cast<std::array<Element, s>>(i);
00082   }
00083
00084
00085   /** If we do not have a vector, just forward it as a tuple up to the
00086       final initialization.
00087
00088       \return typically tuple<double>{ 2.4 } from 2.4 input for example
00089   */
00090   template <typename V, typename Type>
00091   static auto flatten(const Type i) {
00092     return std::make_tuple(i);
00093   }
00094
00095
00096  /** Take some initializer values and apply flattening on each value
00097
00098       \return a tuple of scalar initializer values
00099    */
00100   template <typename V, typename... Types>
00101   static auto flatten_to_tuple(const Types... i) {
00102     // Concatenate the tuples returned by each flattening
00103     return std::tuple_cat(flatten<V>(i)...);
00104   }
00105
00106
00107   /// \todo To implement
00108 #if 0
00109   vec<dataT,
00110       numElements>
00111   operator+(const vec<dataT, numElements> &rhs) const;
00112   vec<dataT, numElements>
00113   operator-(const vec<dataT, numElements> &rhs) const;
00114   vec<dataT, numElements>
00115   operator*(const vec<dataT, numElements> &rhs) const;
```

```
00116   vec<dataT, numElements>
00117   operator/(const vec<dataT, numElements> &rhs) const;
00118   vec<dataT, numElements>
00119   operator+=(const vec<dataT, numElements> &rhs);
00120   vec<dataT, numElements>
00121   operator-=(const vec<dataT, numElements> &rhs);
00122   vec<dataT, numElements>
00123   operator*=(const vec<dataT, numElements> &rhs);
00124   vec<dataT, numElements>
00125   operator/=(const vec<dataT, numElements> &rhs);
00126   vec<dataT, numElements>
00127   operator+(const dataT &rhs) const;
00128   vec<dataT, numElements>
00129   operator-(const dataT &rhs) const;
00130   vec<dataT, numElements>
00131   operator*(const dataT &rhs) const;
00132   vec<dataT, numElements>
00133   operator/(const dataT &rhs) const;
00134   vec<dataT, numElements>
00135   operator+=(const dataT &rhs);
00136   vec<dataT, numElements>
00137   operator-=(const dataT &rhs);
00138   vec<dataT, numElements>
00139   operator*=(const dataT &rhs);
00140   vec<dataT, numElements>
00141   operator/=(const dataT &rhs);
00142   vec<dataT, numElements> &operator=(const
      vec<dataT, numElements> &rhs);
00143   vec<dataT, numElements> &operator=(const dataT &rhs);
00144   bool operator==(const vec<dataT, numElements> &rhs) const;
00145   bool operator!=(const vec<dataT, numElements> &rhs) const;
00146   // Swizzle methods (see notes)
00147   swizzled_vec<T, out_dims> swizzle<int s1, ...>();
00148 #ifdef SYCL_SIMPLE_SWIZZLES
00149   swizzled_vec<T, 4> xyzw();
00150   ...
00151 #endif // #ifdef SYCL_SIMPLE_SWIZZLES
00152 #endif
00153 };
00154
00155   /** A macro to define type alias, such as for type=uchar, size=4 and
00156       real_type=unsigned char, uchar4 is equivalent to vec<float, 4>
00157   */
00158 #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) \
00159   using type##size = vec<actual_type, size>;
00160
00161   /// Declare the vector types of a type for all the sizes
00162 #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)           \
00163   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)         \
00164   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)         \
00165   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)         \
00166   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)         \
00167   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)         \
00168   TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
00169
00170   /// Declare all the possible vector type aliases
00171   TRISYCL_DEFINE_VEC_TYPE(char, char)
00172   TRISYCL_DEFINE_VEC_TYPE(uchar, unsigned char)
00173   TRISYCL_DEFINE_VEC_TYPE(short, short int)
00174   TRISYCL_DEFINE_VEC_TYPE(ushort, unsigned short int)
00175   TRISYCL_DEFINE_VEC_TYPE(int, int)
00176   TRISYCL_DEFINE_VEC_TYPE(uint, unsigned int)
00177   TRISYCL_DEFINE_VEC_TYPE(long, long int)
00178   TRISYCL_DEFINE_VEC_TYPE(ulong, unsigned long int)
00179   TRISYCL_DEFINE_VEC_TYPE(float, float)
00180   TRISYCL_DEFINE_VEC_TYPE(double, double)
00181
00182 /// @} End the vector Doxygen group
00183
00184
00185 }
00186 }
00187
00188 /*
00189     # Some Emacs stuff:
00190     ### Local Variables:
00191     ### ispell-local-dictionary: "american"
00192     ### eval: (flyspell-prog-mode)
00193     ### End:
00194 */
00195
00196 #endif // TRISYCL_SYCL_VEC_HPP
```

Index

__CL_ENABLE_EXCEPTIONS
    global_config.hpp, 197
__SYCL_SINGLE_SOURCE__
    global_config.hpp, 197
~debug
    cl::sycl::detail::debug, 89

accelerator
    Platforms, contexts, devices and queues, 82
access
    cl::sycl::detail::buffer, 30
accessor
    cl::sycl::accessor, 27
    cl::sycl::detail::accessor, 23
acquire_buffers
    cl::sycl::detail::task, 153
add
    cl::sycl::detail::buffer_customer, 149
    cl::sycl::detail::task, 154
addr_space
    Dealing with OpenCL address spaces, 46
address_bits
    Platforms, contexts, devices and queues, 78
address_space
    cl::sycl::detail::address_space_base, 44
    Dealing with OpenCL address spaces, 49
address_space_array
    cl::sycl::detail::address_space_array, 39
address_space_fundamental
    cl::sycl::detail::address_space_fundamental, 40
address_space_object
    cl::sycl::detail::address_space_object, 42
address_space_variable
    cl::sycl::detail::address_space_variable, 46
all
    Platforms, contexts, devices and queues, 82
allocation
    cl::sycl::detail::buffer, 30
allocator_type
    cl::sycl::buffer, 31
array
    cl::sycl::detail::accessor, 26
array_view_type
    cl::sycl::detail::accessor, 23
async_handler
    Error handling, 101
atomic
    Data access and storage in SYCL, 33

barrier
    cl::sycl::nd_item, 114
basic_type
    cl::sycl::vec, 131
buf
    cl::sycl::detail::accessor, 26
    cl::sycl::detail::buffer_customer, 151
buffer
    cl::sycl::buffer, 32
    cl::sycl::detail::buffer, 28, 29
buffer_allocator
    Data access and storage in SYCL, 32
buffer_base
    cl::sycl::detail::buffer_base, 146
buffer_customer
    cl::sycl::detail::buffer_customer, 149
buffers
    cl::sycl::detail::task, 155
built_in_kernels
    Platforms, contexts, devices and queues, 79

CL_SYCL_LANGUAGE_VERSION
    global_config.hpp, 197
CL_TRISYCL_LANGUAGE_VERSION
    global_config.hpp, 198
cl, 135
cl::sycl, 135
    function_class, 138
    mutex_class, 138
    shared_ptr_class, 138
    string_class, 138
    unique_ptr_class, 138
    vector_class, 138
    weak_ptr_class, 138
cl::sycl::access, 138
cl::sycl::accessor, 26
    accessor, 27
    const_reference, 26
    dimensionality, 27
    reference, 27
    value_type, 27
cl::sycl::buffer, 30
    allocator_type, 31
    buffer, 32
    const_reference, 31
    implementation, 32
    reference, 31
    value_type, 31
cl::sycl::context, 53
    context, 53–55
    get, 55

get_devices, 55
get_info, 55
get_platform, 56
is_host, 56
cl::sycl::cpu_selector, 63
operator(), 63
cl::sycl::default_selector, 61
operator(), 62
cl::sycl::detail, 139
cl::sycl::detail::AccessorImpl< T, dimensions, mode, target >, 145
cl::sycl::detail::accessor, 22
accessor, 23
array, 26
array_view_type, 23
buf, 26
dimensionality, 26
element, 23
get_buffer, 24
is_write_access, 24
operator[], 24, 25
value_type, 23
writable_array_view_type, 23
cl::sycl::detail::address_space_array, 38
address_space_array, 39
super, 39
cl::sycl::detail::address_space_base, 42
address_space, 44
opencl_type, 44
type, 44
cl::sycl::detail::address_space_fundamental, 39
address_space_fundamental, 40
super, 40
cl::sycl::detail::address_space_object, 41
address_space_object, 42
opencl_type, 41
operator opencl_type &, 42
cl::sycl::detail::address_space_ptr, 42
super, 42
cl::sycl::detail::address_space_variable, 44
address_space_variable, 46
opencl_type, 45
operator opencl_type &, 46
super, 45
variable, 46
cl::sycl::detail::buffer, 27
access, 30
allocation, 30
buffer, 28, 29
element, 28
get_access, 30
value_type, 28
cl::sycl::detail::buffer_base, 145
buffer_base, 146
get_buffer_customer, 146
get_last_buffer_customer, 147
last_buffer_customer, 148
lock, 147
protect_buffer, 148
read_only, 148
set_last_buffer_customer, 147
wait, 147
cl::sycl::detail::buffer_customer, 148
add, 149
buf, 151
buffer_customer, 149
next_generation, 151
notify_ready, 149
ready_cv, 151
ready_mutex, 151
ready_to_use, 151
release, 150
released_cv, 151
released_mutex, 152
set_next_generation, 150
user_number, 152
wait, 150
wait_released, 151
write_access, 152
cl::sycl::detail::debug, 88
~debug, 89
debug, 88, 89
cl::sycl::detail::display_vector, 89
display, 89
cl::sycl::detail::expand_to_vector, 85
cl::sycl::detail::expand_to_vector< V, Tuple, true >, 85
cl::sycl::detail::opencl_type, 36
type, 36
cl::sycl::detail::opencl_type< T, constant_address_space >, 36
type, 37
cl::sycl::detail::opencl_type< T, generic_address_space >, 37
type, 37
cl::sycl::detail::opencl_type< T, global_address_space >, 37
type, 37
cl::sycl::detail::opencl_type< T, local_address_space >, 37
type, 38
cl::sycl::detail::opencl_type< T, private_address_space >, 38
type, 38
cl::sycl::detail::parallel_OpenMP_for_iterate, 122
parallel_OpenMP_for_iterate, 122
cl::sycl::detail::parallel_for_iterate, 121
parallel_for_iterate, 122
cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >, 123
parallel_for_iterate, 123
cl::sycl::detail::small_array, 91
dimension, 94
dimensionality, 94
element_type, 92
get, 93
operator FinalType, 93

small_array, 92, 93
cl::sycl::detail::small_array_123, 94
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 94
operator BasicType, 95
small_array_123, 95
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 95
small_array_123, 96
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 96
small_array_123, 96
cl::sycl::detail::task, 153
acquire_buffers, 153
add, 154
buffers, 155
release_buffers, 154
schedule, 154
cl::sycl::device, 56
create_sub_devices, 58
device, 57
get, 58
get_devices, 58
get_info, 58
get_platform, 59
has_extension, 59
is_accelerator, 59
is_cpu, 60
is_gpu, 60
is_host, 60
cl::sycl::device_selector, 60
operator(), 61
select_device, 61
cl::sycl::error_handler, 98
default_handler, 99
report_error, 98
cl::sycl::exception, 99
get_buffer, 100
get_cl_code, 100
get_image, 100
get_queue, 100
get_sycl_code, 101
cl::sycl::gpu_selector, 62
operator(), 62
cl::sycl::group, 103
dimensionality, 108
get, 105
get_global_range, 105
get_group_range, 105, 106
get_linear, 106
get_local_range, 106
get_nd_range, 107
get_offset, 107
group, 104
group_id, 108
ndr, 108
operator[], 107
cl::sycl::handler, 64
current_task, 68
Dimensions, 68
handler, 65
parallel_for, 65, 66
parallel_for_work_group, 66
set_arg, 66, 67
single_task, 67
TRISYCL_parallel_for_functor_GLOBAL, 68
cl::sycl::host_selector, 63
operator(), 64
cl::sycl::id, 108
id, 108
cl::sycl::image, 32
cl::sycl::info, 141
device_exec_capabilities, 143
device_fp_config, 143
device_queue_properties, 143
gl_context_interop, 143
queue_profiling, 143
cl::sycl::info::param_traits< T, Param >, 152
cl::sycl::item, 109
dimensionality, 112
display, 110
get, 110
get_linear_id, 110
get_offset, 111
get_range, 111
global_index, 112
global_range, 112
item, 110
offset, 112
operator[], 111
set, 111
cl::sycl::kernel, 64
cl::sycl::nd_item, 112
barrier, 114
dimensionality, 118
get_global, 114
get_global_linear_id, 115
get_global_range, 115
get_group, 115
get_group_linear_id, 116
get_local, 116
get_local_linear_id, 116
get_local_range, 117
get_nd_range, 117
get_num_groups, 117
get_offset, 118
global_index, 118
local_index, 118
ND_range, 118
nd_item, 113
set_global, 118
set_local, 118
cl::sycl::nd_range, 119
dimensionality, 121
display, 120
get_global, 120

get_group, 120
get_local, 120
get_offset, 121
global_range, 121
local_range, 121
nd_range, 120
offset, 121
cl::sycl::platform, 68
get, 70
get_devices, 70
get_info, 70
get_platforms, 70
has_extension, 71
is_host, 71
platform, 69
cl::sycl::queue, 71
get, 74
get_context, 75
get_device, 75
get_info, 75
is_host, 75
queue, 72–74
submit, 76
throw_asynchronous, 76
wait, 76
wait_and_throw, 77
cl::sycl::range, 123
cl::sycl::trisycl, 143
cl::sycl::trisycl::default_error_handler, 99
report_error, 99
cl::sycl::vec, 130
basic_type, 131
flatten, 131, 132
flatten_to_tuple, 132
vec, 131
const_reference
cl::sycl::accessor, 26
cl::sycl::buffer, 31
constant
Dealing with OpenCL address spaces, 47
constant_address_space
Dealing with OpenCL address spaces, 49
constant_buffer
Data access and storage in SYCL, 33
context
cl::sycl::context, 53–55
Platforms, contexts, devices and queues, 77, 84
correctly_rounded_divide_sqrt
Platforms, contexts, devices and queues, 82
cpu
Platforms, contexts, devices and queues, 82
create_sub_devices
cl::sycl::device, 58
current_task
cl::sycl::handler, 68
custom
Platforms, contexts, devices and queues, 82

Data access and storage in SYCL, 21
atomic, 33
buffer_allocator, 32
constant_buffer, 33
discard_read_write, 33
discard_write, 33
fence_space, 33
global_and_local, 33
global_buffer, 33
global_space, 33
host_buffer, 34
host_image, 34
image, 33
image_array, 34
local, 33
local_space, 33
mode, 33
read, 33
read_write, 33
target, 33
write, 33
Dealing with OpenCL address spaces, 35
addr_space, 46
address_space, 49
constant, 47
constant_address_space, 49
generic, 47
generic_address_space, 49
global, 47
global_address_space, 49
local, 47
local_address_space, 49
make_multi, 49
multi_ptr, 47
priv, 49
private_address_space, 49
debug
cl::sycl::detail::debug, 88, 89
debug.hpp
TRISYCL_DUMP, 193
TRISYCL_DUMP_T, 193
Debugging and tracing support, 88
unimplemented, 90
default_handler
cl::sycl::error_handler, 99
defaults
Platforms, contexts, devices and queues, 82
denorm
Platforms, contexts, devices and queues, 82
device
cl::sycl::device, 57
Platforms, contexts, devices and queues, 77, 84
device_affinity_domain
Platforms, contexts, devices and queues, 80
device_exec_capabilities
cl::sycl::info, 143
device_execution_capabilities
Platforms, contexts, devices and queues, 80
device_fp_config

cl::sycl::info, 143
device_partition_property
Platforms, contexts, devices and queues, 81
device_partition_type
Platforms, contexts, devices and queues, 81
device_queue_properties
cl::sycl::info, 143
device_type
Platforms, contexts, devices and queues, 78, 81
device_version
Platforms, contexts, devices and queues, 79
dimension
cl::sycl::detail::small_array, 94
dimensionality
cl::sycl::accessor, 27
cl::sycl::detail::accessor, 26
cl::sycl::detail::small_array, 94
cl::sycl::group, 108
cl::sycl::item, 112
cl::sycl::nd_item, 118
cl::sycl::nd_range, 121
Dimensions
cl::sycl::handler, 68
discard_read_write
Data access and storage in SYCL, 33
discard_write
Data access and storage in SYCL, 33
display
cl::sycl::detail::display_vector, 89
cl::sycl::item, 110
cl::sycl::nd_range, 120
double_fp_config
Platforms, contexts, devices and queues, 78
driver_version
Platforms, contexts, devices and queues, 79

element
cl::sycl::detail::accessor, 23
cl::sycl::detail::buffer, 28
element_type
cl::sycl::detail::small_array, 92
endian_little
Platforms, contexts, devices and queues, 79
Error handling, 98
async_handler, 101
error_correction_support
Platforms, contexts, devices and queues, 79
exec_kernel
Platforms, contexts, devices and queues, 81
exec_native_kernel
Platforms, contexts, devices and queues, 81
execution_capabilities
Platforms, contexts, devices and queues, 79
expand
Helpers to do array and tuple conversion, 86
Expressing parallelism through kernels, 102
make_id, 124
make_range, 124, 125
parallel_for, 125
parallel_for_global_offset, 126
parallel_for_work_item, 127
parallel_for_workgroup, 127
parallel_for_workitem, 127
extensions
Platforms, contexts, devices and queues, 79, 83

fence_space
Data access and storage in SYCL, 33
fill_tuple
Helpers to do array and tuple conversion, 86
flatten
cl::sycl::vec, 131, 132
flatten_to_tuple
cl::sycl::vec, 132
fma
Platforms, contexts, devices and queues, 82
fp_config
Platforms, contexts, devices and queues, 82
function_class
cl::sycl, 138

generic
Dealing with OpenCL address spaces, 47
generic_address_space
Dealing with OpenCL address spaces, 49
get
cl::sycl::context, 55
cl::sycl::detail::small_array, 93
cl::sycl::device, 58
cl::sycl::group, 105
cl::sycl::item, 110
cl::sycl::platform, 70
cl::sycl::queue, 74
get_access
cl::sycl::detail::buffer, 30
get_buffer
cl::sycl::detail::accessor, 24
cl::sycl::exception, 100
get_buffer_customer
cl::sycl::detail::buffer_base, 146
get_cl_code
cl::sycl::exception, 100
get_context
cl::sycl::queue, 75
get_device
cl::sycl::queue, 75
get_devices
cl::sycl::context, 55
cl::sycl::device, 58
cl::sycl::platform, 70
get_global
cl::sycl::nd_item, 114
cl::sycl::nd_range, 120
get_global_linear_id
cl::sycl::nd_item, 115
get_global_range
cl::sycl::group, 105
cl::sycl::nd_item, 115

get_group
cl::sycl::nd_item, 115
cl::sycl::nd_range, 120
get_group_linear_id
cl::sycl::nd_item, 116
get_group_range
cl::sycl::group, 105, 106
get_image
cl::sycl::exception, 100
get_info
cl::sycl::context, 55
cl::sycl::device, 58
cl::sycl::platform, 70
cl::sycl::queue, 75
get_last_buffer_customer
cl::sycl::detail::buffer_base, 147
get_linear
cl::sycl::group, 106
get_linear_id
cl::sycl::item, 110
get_local
cl::sycl::nd_item, 116
cl::sycl::nd_range, 120
get_local_linear_id
cl::sycl::nd_item, 116
get_local_range
cl::sycl::group, 106
cl::sycl::nd_item, 117
get_nd_range
cl::sycl::group, 107
cl::sycl::nd_item, 117
get_num_groups
cl::sycl::nd_item, 117
get_offset
cl::sycl::group, 107
cl::sycl::item, 111
cl::sycl::nd_item, 118
cl::sycl::nd_range, 121
get_platform
cl::sycl::context, 56
cl::sycl::device, 59
get_platforms
cl::sycl::platform, 70
get_queue
cl::sycl::exception, 100
get_range
cl::sycl::item, 111
get_sycl_code
cl::sycl::exception, 101
gl_context_interop
cl::sycl::info, 143
gl_interop
Platforms, contexts, devices and queues, 77
global
Dealing with OpenCL address spaces, 47
Platforms, contexts, devices and queues, 83
global_address_space
Dealing with OpenCL address spaces, 49
global_and_local
Data access and storage in SYCL, 33
global_buffer
Data access and storage in SYCL, 33
global_config.hpp
__CL_ENABLE_EXCEPTIONS, 197
__SYCL_SINGLE_SOURCE__, 197
CL_SYCL_LANGUAGE_VERSION, 197
CL_TRISYCL_LANGUAGE_VERSION, 198
TRISYCL_ASYNC, 198
global_index
cl::sycl::item, 112
cl::sycl::nd_item, 118
global_mem_cache_line_size
Platforms, contexts, devices and queues, 78
global_mem_cache_size
Platforms, contexts, devices and queues, 78
global_mem_cache_type
Platforms, contexts, devices and queues, 78, 82
global_mem_size
Platforms, contexts, devices and queues, 78
global_range
cl::sycl::item, 112
cl::sycl::nd_range, 121
global_space
Data access and storage in SYCL, 33
gpu
Platforms, contexts, devices and queues, 82
group
cl::sycl::group, 104
group_id
cl::sycl::group, 108

handler
cl::sycl::handler, 65
handler.hpp
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET, 220
TRISYCL_parallel_for_functor_GLOBAL, 219
handler_event, 152
has_extension
cl::sycl::device, 59
cl::sycl::platform, 71
Helpers to do array and tuple conversion, 85
expand, 86
fill_tuple, 86
tuple_to_array, 87
tuple_to_array_iterate, 87
host
Platforms, contexts, devices and queues, 82
host_buffer
Data access and storage in SYCL, 34
host_image
Data access and storage in SYCL, 34
host_unified_memory
Platforms, contexts, devices and queues, 79

id
cl::sycl::id, 108

image
Data access and storage in SYCL, 33
image2d_max_height
Platforms, contexts, devices and queues, 78
image2d_max_width
Platforms, contexts, devices and queues, 78
image3d_mas_depth
Platforms, contexts, devices and queues, 78
image3d_max_height
Platforms, contexts, devices and queues, 78
image3d_max_widht
Platforms, contexts, devices and queues, 78
image_array
Data access and storage in SYCL, 34
image_max_array_size
Platforms, contexts, devices and queues, 78
image_max_buffer_size
Platforms, contexts, devices and queues, 78
image_support
Platforms, contexts, devices and queues, 78
implementation
cl::sycl::buffer, 32
include/CL/sycl.hpp, 157
include/CL/sycl/access.hpp, 158, 159
include/CL/sycl/accessor.hpp, 162, 163
include/CL/sycl/accessor/detail/accessor.hpp, 160
include/CL/sycl/address_space.hpp, 170, 171
include/CL/sycl/address_space/detail/address_↩
space.hpp, 164, 166
include/CL/sycl/buffer.hpp, 175
include/CL/sycl/buffer/detail/buffer.hpp, 173
include/CL/sycl/buffer/detail/buffer_base.hpp, 179, 180
include/CL/sycl/buffer/detail/buffer_customer.hpp, 181
include/CL/sycl/buffer_allocator.hpp, 183, 184
include/CL/sycl/command_group/detail/task.hpp, 184, 185
include/CL/sycl/context.hpp, 186, 187
include/CL/sycl/detail/array_tuple_helpers.hpp, 189, 190
include/CL/sycl/detail/debug.hpp, 192, 193
include/CL/sycl/detail/default_classes.hpp, 195
include/CL/sycl/detail/global_config.hpp, 197, 198
include/CL/sycl/detail/linear_id.hpp, 199
include/CL/sycl/detail/small_array.hpp, 200
include/CL/sycl/detail/unimplemented.hpp, 204
include/CL/sycl/device.hpp, 205, 206
include/CL/sycl/device_selector.hpp, 211
include/CL/sycl/error_handler.hpp, 213, 214
include/CL/sycl/exception.hpp, 214, 215
include/CL/sycl/group.hpp, 216, 217
include/CL/sycl/handler.hpp, 219, 220
include/CL/sycl/handler_event.hpp, 223, 224
include/CL/sycl/id.hpp, 224, 225
include/CL/sycl/image.hpp, 226
include/CL/sycl/info/param_traits.hpp, 227, 228
include/CL/sycl/item.hpp, 229
include/CL/sycl/nd_item.hpp, 231
include/CL/sycl/nd_range.hpp, 234
include/CL/sycl/parallelism.hpp, 240, 241
include/CL/sycl/parallelism/detail/parallelism.hpp, 235, 237
include/CL/sycl/platform.hpp, 241, 242
include/CL/sycl/queue.hpp, 244, 245
include/CL/sycl/range.hpp, 249, 250
include/CL/sycl/vec.hpp, 250, 251
inf_nan
Platforms, contexts, devices and queues, 82
is_accelerator
cl::sycl::device, 59
is_available
Platforms, contexts, devices and queues, 79
is_compiler_available
Platforms, contexts, devices and queues, 79
is_cpu
cl::sycl::device, 60
is_gpu
cl::sycl::device, 60
is_host
cl::sycl::context, 56
cl::sycl::device, 60
cl::sycl::platform, 71
cl::sycl::queue, 75
is_linker_available
Platforms, contexts, devices and queues, 79
is_write_access
cl::sycl::detail::accessor, 24
item
cl::sycl::item, 110

L1_cache
Platforms, contexts, devices and queues, 81
L2_cache
Platforms, contexts, devices and queues, 80, 81
L3_cache
Platforms, contexts, devices and queues, 80, 81
L4_cache
Platforms, contexts, devices and queues, 80, 81
last_buffer_customer
cl::sycl::detail::buffer_base, 148
linear_id
Some helpers for the implementation, 97
local
Data access and storage in SYCL, 33
Dealing with OpenCL address spaces, 47
Platforms, contexts, devices and queues, 83
local_address_space
Dealing with OpenCL address spaces, 49
local_index
cl::sycl::nd_item, 118
local_mem_size
Platforms, contexts, devices and queues, 79
local_mem_type
Platforms, contexts, devices and queues, 78, 83
local_range
cl::sycl::nd_range, 121
local_space
Data access and storage in SYCL, 33

lock
cl::sycl::detail::buffer_base, 147

make_id
Expressing parallelism through kernels, 124
make_multi
Dealing with OpenCL address spaces, 49
make_range
Expressing parallelism through kernels, 124, 125
max_clock_frequency
Platforms, contexts, devices and queues, 78
max_compute_units
Platforms, contexts, devices and queues, 78
max_constant_args
Platforms, contexts, devices and queues, 78
max_constant_buffer_size
Platforms, contexts, devices and queues, 78
max_mem_alloc_size
Platforms, contexts, devices and queues, 78
max_parameter_size
Platforms, contexts, devices and queues, 78
max_read_image_args
Platforms, contexts, devices and queues, 78
max_samplers
Platforms, contexts, devices and queues, 78
max_work_group_size
Platforms, contexts, devices and queues, 78
max_work_item_dimensions
Platforms, contexts, devices and queues, 78
max_work_item_sizes
Platforms, contexts, devices and queues, 78
max_write_image_args
Platforms, contexts, devices and queues, 78
mem_base_addr_align
Platforms, contexts, devices and queues, 78
mode
Data access and storage in SYCL, 33
multi_ptr
Dealing with OpenCL address spaces, 47
mutex_class
cl::sycl, 138

ND_range
cl::sycl::nd_item, 118
name
Platforms, contexts, devices and queues, 79, 83
native_vector_witdth_char
Platforms, contexts, devices and queues, 78
native_vector_witdth_double
Platforms, contexts, devices and queues, 78
native_vector_witdth_float
Platforms, contexts, devices and queues, 78
native_vector_witdth_half
Platforms, contexts, devices and queues, 78
native_vector_witdth_int
Platforms, contexts, devices and queues, 78
native_vector_witdth_long_long
Platforms, contexts, devices and queues, 78
native_vector_witdth_short
Platforms, contexts, devices and queues, 78
nd_item
cl::sycl::nd_item, 113
nd_range
cl::sycl::nd_range, 120
ndr
cl::sycl::group, 108
next_generation
cl::sycl::detail::buffer_customer, 151
next_partitionable
Platforms, contexts, devices and queues, 80
no_partition
Platforms, contexts, devices and queues, 81
none
Platforms, contexts, devices and queues, 83
notify_ready
cl::sycl::detail::buffer_customer, 149
num_devices
Platforms, contexts, devices and queues, 77
numa
Platforms, contexts, devices and queues, 80, 81

offset
cl::sycl::item, 112
cl::sycl::nd_range, 121
opencl_type
cl::sycl::detail::address_space_base, 44
cl::sycl::detail::address_space_object, 41
cl::sycl::detail::address_space_variable, 45
opencl_version
Platforms, contexts, devices and queues, 79
operator BasicType
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 95
operator FinalType
cl::sycl::detail::small_array, 93
operator opencl_type &
cl::sycl::detail::address_space_object, 42
cl::sycl::detail::address_space_variable, 46
operator()
cl::sycl::cpu_selector, 63
cl::sycl::default_selector, 62
cl::sycl::device_selector, 61
cl::sycl::gpu_selector, 62
cl::sycl::host_selector, 64
operator[]
cl::sycl::detail::accessor, 24, 25
cl::sycl::group, 107
cl::sycl::item, 111

parallel_OpenMP_for_iterate
cl::sycl::detail::parallel_OpenMP_for_iterate, 122
parallel_for
cl::sycl::handler, 65, 66
Expressing parallelism through kernels, 125
parallel_for_global_offset
Expressing parallelism through kernels, 126
parallel_for_iterate
cl::sycl::detail::parallel_for_iterate, 122

cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >, 123
parallel_for_work_group
cl::sycl::handler, 66
parallel_for_work_item
Expressing parallelism through kernels, 127
parallel_for_workgroup
Expressing parallelism through kernels, 127
parallel_for_workitem
Expressing parallelism through kernels, 127
param_traits.hpp
TRISYCL_INFO_PARAM_TRAITS, 227
TRISYCL_INFO_PARAM_TRAITS_ANY_T, 228
parent_device
Platforms, contexts, devices and queues, 79
partition_affinity_domain
Platforms, contexts, devices and queues, 79
partition_affinity_domain_next_partitionable
Platforms, contexts, devices and queues, 81
partition_by_affinity_domain
Platforms, contexts, devices and queues, 81
partition_by_counts
Platforms, contexts, devices and queues, 81
partition_equally
Platforms, contexts, devices and queues, 81
partition_max_sub_devices
Platforms, contexts, devices and queues, 79
partition_properties
Platforms, contexts, devices and queues, 79
partition_type
Platforms, contexts, devices and queues, 79
platform
cl::sycl::platform, 69
Platforms, contexts, devices and queues, 79, 83
Platforms, contexts, devices and queues, 51
accelerator, 82
address_bits, 78
all, 82
built_in_kernels, 79
context, 77, 84
correctly_rounded_divide_sqrt, 82
cpu, 82
custom, 82
defaults, 82
denorm, 82
device, 77, 84
device_affinity_domain, 80
device_execution_capabilities, 80
device_partition_property, 81
device_partition_type, 81
device_type, 78, 81
device_version, 79
double_fp_config, 78
driver_version, 79
endian_little, 79
error_correction_support, 79
exec_kernel, 81
exec_native_kernel, 81
execution_capabilities, 79
extensions, 79, 83
fma, 82
fp_config, 82
gl_interop, 77
global, 83
global_mem_cache_line_size, 78
global_mem_cache_size, 78
global_mem_cache_type, 78, 82
global_mem_size, 78
gpu, 82
host, 82
host_unified_memory, 79
image2d_max_height, 78
image2d_max_width, 78
image3d_mas_depth, 78
image3d_max_height, 78
image3d_max_widht, 78
image_max_array_size, 78
image_max_buffer_size, 78
image_support, 78
inf_nan, 82
is_available, 79
is_compiler_available, 79
is_linker_available, 79
L1_cache, 81
L2_cache, 80, 81
L3_cache, 80, 81
L4_cache, 80, 81
local, 83
local_mem_size, 79
local_mem_type, 78, 83
max_clock_frequency, 78
max_compute_units, 78
max_constant_args, 78
max_constant_buffer_size, 78
max_mem_alloc_size, 78
max_parameter_size, 78
max_read_image_args, 78
max_samplers, 78
max_work_group_size, 78
max_work_item_dimensions, 78
max_work_item_sizes, 78
max_write_image_args, 78
mem_base_addr_align, 78
name, 79, 83
native_vector_witdth_char, 78
native_vector_witdth_double, 78
native_vector_witdth_float, 78
native_vector_witdth_half, 78
native_vector_witdth_int, 78
native_vector_witdth_long_long, 78
native_vector_witdth_short, 78
next_partitionable, 80
no_partition, 81
none, 83
num_devices, 77
numa, 80, 81

opencl_version, 79
parent_device, 79
partition_affinity_domain, 79
partition_affinity_domain_next_partitionable, 81
partition_by_affinity_domain, 81
partition_by_counts, 81
partition_equally, 81
partition_max_sub_devices, 79
partition_properties, 79
partition_type, 79
platform, 79, 83
preferred_interop_user_sync, 79
preferred_vector_width_char, 78
preferred_vector_width_double, 78
preferred_vector_width_float, 78
preferred_vector_width_half, 78
preferred_vector_width_int, 78
preferred_vector_width_long_long, 78
preferred_vector_width_short, 78
printf_buffer_size, 79
profile, 79, 83
profiling_timer_resolution, 79
properties, 84
queue, 84
queue_properties, 79
read_only, 83
reference_count, 77, 79, 84
round_to_inf, 82
round_to_nearest, 82
round_to_zero, 82
single_fp_config, 78
soft_float, 82
unsupported, 80, 81
vendor, 79, 83
vendor_id, 78
version, 83
write_only, 83

preferred_interop_user_sync
Platforms, contexts, devices and queues, 79

preferred_vector_width_char
Platforms, contexts, devices and queues, 78

preferred_vector_width_double
Platforms, contexts, devices and queues, 78

preferred_vector_width_float
Platforms, contexts, devices and queues, 78

preferred_vector_width_half
Platforms, contexts, devices and queues, 78

preferred_vector_width_int
Platforms, contexts, devices and queues, 78

preferred_vector_width_long_long
Platforms, contexts, devices and queues, 78

preferred_vector_width_short
Platforms, contexts, devices and queues, 78

printf_buffer_size
Platforms, contexts, devices and queues, 79

priv
Dealing with OpenCL address spaces, 49

private_address_space
Dealing with OpenCL address spaces, 49

profile
Platforms, contexts, devices and queues, 79, 83

profiling_timer_resolution
Platforms, contexts, devices and queues, 79

properties
Platforms, contexts, devices and queues, 84

protect_buffer
cl::sycl::detail::buffer_base, 148

queue
cl::sycl::queue, 72–74
Platforms, contexts, devices and queues, 84

queue_profiling
cl::sycl::info, 143

queue_properties
Platforms, contexts, devices and queues, 79

read
Data access and storage in SYCL, 33

read_only
cl::sycl::detail::buffer_base, 148
Platforms, contexts, devices and queues, 83

read_write
Data access and storage in SYCL, 33

ready_cv
cl::sycl::detail::buffer_customer, 151

ready_mutex
cl::sycl::detail::buffer_customer, 151

ready_to_use
cl::sycl::detail::buffer_customer, 151

reference
cl::sycl::accessor, 27
cl::sycl::buffer, 31

reference_count
Platforms, contexts, devices and queues, 77, 79, 84

release
cl::sycl::detail::buffer_customer, 150

release_buffers
cl::sycl::detail::task, 154

released_cv
cl::sycl::detail::buffer_customer, 151

released_mutex
cl::sycl::detail::buffer_customer, 152

report_error
cl::sycl::error_handler, 98
cl::sycl::trisycl::default_error_handler, 99

round_to_inf
Platforms, contexts, devices and queues, 82

round_to_nearest
Platforms, contexts, devices and queues, 82

round_to_zero
Platforms, contexts, devices and queues, 82

schedule
cl::sycl::detail::task, 154

select_device
cl::sycl::device_selector, 61

set

cl::sycl::item, 111
set_arg
cl::sycl::handler, 66, 67
set_global
cl::sycl::nd_item, 118
set_last_buffer_customer
cl::sycl::detail::buffer_base, 147
set_local
cl::sycl::nd_item, 118
set_next_generation
cl::sycl::detail::buffer_customer, 150
shared_ptr_class
cl::sycl, 138
single_fp_config
Platforms, contexts, devices and queues, 78
single_task
cl::sycl::handler, 67
small_array
cl::sycl::detail::small_array, 92, 93
small_array_123
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 95
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 96
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 96
soft_float
Platforms, contexts, devices and queues, 82
Some helpers for the implementation, 91
linear_id, 97
TRISYCL_BOOST_OPERATOR_VECTOR_OP, 97
string_class
cl::sycl, 138
submit
cl::sycl::queue, 76
super
cl::sycl::detail::address_space_array, 39
cl::sycl::detail::address_space_fundamental, 40
cl::sycl::detail::address_space_ptr, 42
cl::sycl::detail::address_space_variable, 45

TRISYCL_ASYNC
global_config.hpp, 198
TRISYCL_BOOST_OPERATOR_VECTOR_OP
Some helpers for the implementation, 97
TRISYCL_DEFINE_VEC_TYPE
Vector types in SYCL, 132
TRISYCL_DEFINE_VEC_TYPE_SIZE
Vector types in SYCL, 132
TRISYCL_DUMP
debug.hpp, 193
TRISYCL_DUMP_T
debug.hpp, 193
TRISYCL_INFO_PARAM_TRAITS
param_traits.hpp, 227
TRISYCL_INFO_PARAM_TRAITS_ANY_T
param_traits.hpp, 228
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET
handler.hpp, 220
TRISYCL_parallel_for_functor_GLOBAL
cl::sycl::handler, 68
handler.hpp, 219
target
Data access and storage in SYCL, 33
throw_asynchronous
cl::sycl::queue, 76
tuple_to_array
Helpers to do array and tuple conversion, 87
tuple_to_array_iterate
Helpers to do array and tuple conversion, 87
type
cl::sycl::detail::address_space_base, 44
cl::sycl::detail::opencl_type, 36
cl::sycl::detail::opencl_type< T, constant_address_space >, 37
cl::sycl::detail::opencl_type< T, generic_address_space >, 37
cl::sycl::detail::opencl_type< T, global_address_space >, 37
cl::sycl::detail::opencl_type< T, local_address_space >, 38
cl::sycl::detail::opencl_type< T, private_address_space >, 38

unimplemented
Debugging and tracing support, 90
unique_ptr_class
cl::sycl, 138
unsupported
Platforms, contexts, devices and queues, 80, 81
user_number
cl::sycl::detail::buffer_customer, 152

value_type
cl::sycl::accessor, 27
cl::sycl::buffer, 31
cl::sycl::detail::accessor, 23
cl::sycl::detail::buffer, 28
variable
cl::sycl::detail::address_space_variable, 46
vec
cl::sycl::vec, 131
Vector types in SYCL, 130
TRISYCL_DEFINE_VEC_TYPE, 132
TRISYCL_DEFINE_VEC_TYPE_SIZE, 132
vector_class
cl::sycl, 138
vendor
Platforms, contexts, devices and queues, 79, 83
vendor_id
Platforms, contexts, devices and queues, 78
version
Platforms, contexts, devices and queues, 83

wait
cl::sycl::detail::buffer_base, 147
cl::sycl::detail::buffer_customer, 150

cl::sycl::queue, 76
wait_and_throw
cl::sycl::queue, 77
wait_released
cl::sycl::detail::buffer_customer, 151
weak_ptr_class
cl::sycl, 138
writable_array_view_type
cl::sycl::detail::accessor, 23
write
Data access and storage in SYCL, 33
write_access
cl::sycl::detail::buffer_customer, 152
write_only
Platforms, contexts, devices and queues, 83