SYCL 2020 has released!

Looking for feedback to help prioritize open source implementation

James Brodman, Mike Kinsner, Greg Lueck, John Pennycook, Roland Schulz

Feb 24, 2021

SYCL 2020 has released!

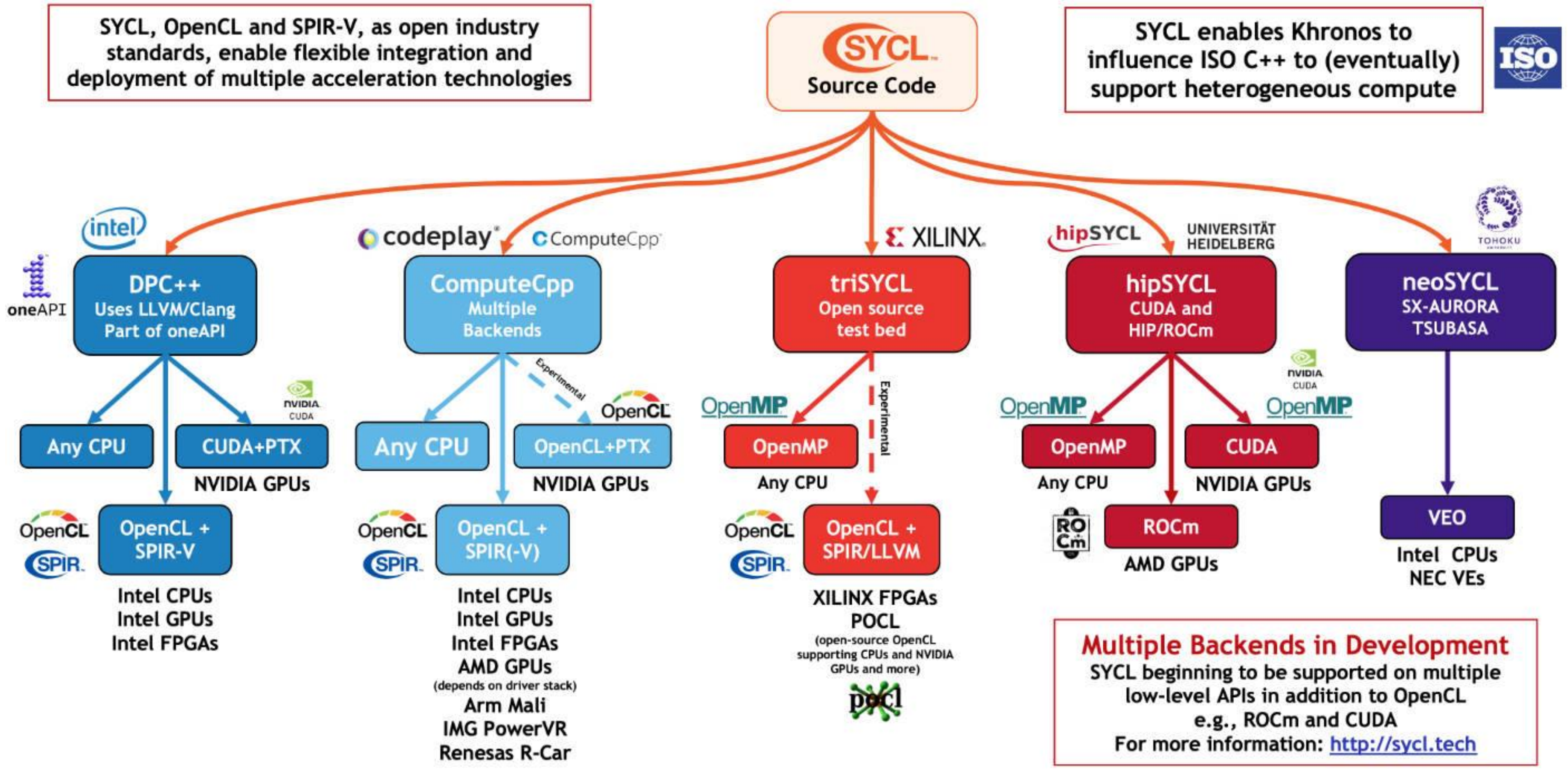
- **Feb 9, 2021:** Public release of SYCL 2020
  - Thanks to everyone from TAB that has helped to steer the direction!
  - Also thanks to the Khronos WG members (many are also in TAB), that have worked hard for years now toward this release

- Numerous DPC++ extensions are now in the core SYCL specification

$$\begin{tikzcd}[column sep=small] \text{DPC++} & = & \text{ISO C++} & + & \text{SYCL} \arrow[ll, bend left] & + & \text{extensions} \arrow[ll, bend left] \end{tikzcd}$$

- **Future:** SYCL 2020 conformance tests to be written + released

The SYCL Ecosystem is Growing


SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

Multiple Backends in Development
SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL e.g., ROCm and CUDA
For more information: http://sycl.tech

Source: https://www.khronos.org/sycl/

Call for Input on Prioritization

- We need your help to prioritize SYCL 2020 features for implementation in the LLVM open source project
  - Based on your anticipated workloads / research / needs

- This deck covers many of features that we think may be of interest
  - Please provide your view on implementation prioritization. Options:
  - Email the TAB DPC++ mailing list
  - Email anyone from the author list of this presentation
  - Open an issue on the DPC++ TAB GitHub

Major Features in SYCL 2020
(does anyone want a more detailed presentation?)

- Based on C++17
- Unified Shared Memory (USM)
- Extension mechanism and device specialization
- atomic_ref
- Reductions
- Math array
- Numerous interface improvements
  (accessor simplifications, class template argument deduction)

Incompletely Implemented Features
(needing prioritization)

Modern (C++20) Atomics

- `sycl::atomic<T>` deprecated in favor of `sycl::atomic_ref<T, Order, Scope, AddressSpace>`.
- Supports shorthand operators (e.g. `+=`) and floating-point types
- Memory order enables more use-cases (e.g. device barriers)
- Memory scope for performance tuning and cross-device atomics

- Prototyped in DPC++ as `sycl::ONEAPI::atomic_ref`, but without support for SYCL 2020 generic address space; requires `multi_ptr`.

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:atomic-references

Sub-groups

- SYCL 2020 replacement for `sub_group` class from DPC++ extension.

- Minor changes from DPC++ version:
  - Namespace change (`sycl::ONEAPI` => `sycl::`)
  - Member variables for generic programming across `group`/`sub_group` classes
  - New `leader()` member function to simplify and future-proof common pattern:
    `if (sg.get_local_id() == 0)` ⇒ `if (sg.leader())`
  - Shuffles and barriers no longer member functions; replaced by free functions

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sub-group-class

Group Algorithms

- SYCL 2020 replacement for group algorithms from DPC++, with new naming convention to distinguish between use-cases.

DPC++ extension

```
sycl::group<1> g = it.get_group();

// Apply algorithm to range [first, last) in memory
auto sum = sycl::ONEAPI::reduce(g, first, last, std::plus<>());

// Apply algorithm directly to variables in private memory
auto sum = sycl::ONEAPI::reduce(g, x, std::plus<>());
```

Same name for both use-cases, in `sycl::ONEAPI::` namespace.

SYCL 2020

```
sycl::group<1> g = it.get_group();

// Apply algorithm to range [first, last) in memory
auto sum = sycl::joint_reduce(g, first, last, std::plus<>());

// Apply algorithm directly to variables in private memory
auto sum = sycl::reduce_over_group(g, x, std::plus<>());
```

Unique name per use-case, in `sycl::` namespace.

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:algorithms

Sub-group Algorithms

- SYCL 2020 replacement for sub-group shuffles from DPC++, enabling direct communication between work-items in the same sub-group.

DPC++ extension

```
sycl::ONEAPI::sub_group sg = it.get_sub_group();

auto a = sg.shuffle_down(x, 1);
auto b = sg.shuffle_up(x, 1);
auto c = sg.shuffle(x, id);
auto d = sg.shuffle_xor(x, mask);
```

Shuffles as member functions.

SYCL 2020

```
sycl::sub_group sg = it.get_sub_group();

auto a = sycl::shift_group_left(sg, x, 1);
auto b = sycl::shift_group_right(sg, x, 1);
auto c = sycl::select_from_group(sg, x, id);
auto d = sycl::permute_group_by_xor(sg, x, mask);
```

Shuffles as free functions.
Names aligned with C++.

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:algorithms

Reductions

```
myQueue.submit([&](handler& cgh) {

  // Input values to reductions are standard accessors (or USM pointers)
  auto inputValues = valuesBuf.get_access<access_mode::read>(cgh);

  // Create temporary objects describing variables with reduction semantics
  auto sumReduction = reduction(sumBuf, cgh, plus<>());
  auto maxReduction = reduction(maxBuf, cgh, maximum<>());

  // parallel_for performs two reduction operations
  cgh.parallel_for(range<1>{1024},
    sumReduction, maxReduction,
    [=](id<1> idx, auto& sum, auto& max) {
      sum += inputValues[idx];
      max.combine(inputValues[idx]);
  });
});
```

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:reduction

Reductions

- Simpler reduction interface, limited to statically sized objects:
  - `reduction(buffer, cgh, plus<>())           => scalar reduction`
  - `reduction(ptr, plus<>())                   => scalar reduction`
  - `reduction(span<int, 16>(ptr), plus<>()) => fixed-size array reduction`

- Property to request automatic initialization of the reduction variable:
  ```
  reduction(ptr, plus<>(),
  property::reduction::initialize_to_identity{})
  ```

- Support for multiple reductions in a single `parallel_for`.

- Support for `parallel_for` without explicit work-group size.

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:reduction oneAPI

Group Mask

- Not SYCL 2020, but an **extension** on top of sub-groups. Exposes the same functionality as `sub_group_ballot` from OpenCL.
- Converts a Boolean condition per work-item into a bitmask:

```
// Assume sub-group size of 8
sycl::sub_group sg = it.get_sub_group();

bool condition = it.get_global_id() % 2;
auto mask = sycl::ext::intel::ballot(sg, condition); // returns 0b01010101
uint32_t set = mask.count();                         // returns 4
uint32_t low = mask.find_low();                      // returns 1
```

- May simplify **complex control** flow (e.g. loops over all active work-items).

https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/GroupMask/SYCL_INTEL_group_mask.asciidoc

Accessor Changes 1/2

- Decrease verbosity
  - Default parameters and CTAD allow omission of template parameter often
  - New “local_accessor”, “host_accessor”, image accessor types
- Improved support for read-only accessors
  - Implicit conversion from read/write accessor to read-only
  - Read-only accessors return “const reference” to data (rather than copy)
- Models ReversibleContainer
  - begin(), end(), cbegin(), cend(), rbegin(), rend(), crbegin(), crend()

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#subsec:accessors

Accessor Changes 2/2

- New “no_init” property (clearer semantics than “discard_write”)
- Fixed confusing method names regarding size in bytes/elements
  - get_size() is replaced by byte_size()

oneAPI

Work-group Local Memory

- This is not SYCL 2020, but an extension on top - may be tweaked
    - Allocate local memory within kernel, not as accessor outside the kernel
    - Want feedback on spec + prioritization

SYCL 2020

```
Q.submit([&](handler& h) {
      auto A = local_accessor<int, 1>(16, h); // 1D local accessor consisting of 16 ints
      h.parallel_for(nd_range{{size}, {16}}, [=](nd_item it) {
        auto l_index = it.get_local_id();
        localIntAcc[l_index] = 42;
}); });
```

Accessor outside kernel

DPC++ extension

```
Q.submit([&](handler& h) {
      h.parallel_for(nd_range{{size}, {16}}, [=](nd_item it) {
        auto ptr = group_local_memory<int[64]>(it.get_group());
        auto& ref = *ptr;
        ref[it.get_local_id()] = 42;
});  });
```

Allocate within kernel (like OpenCL)

Changes to multi_ptr

- New “generic” address space
  - Removes the requirement for deduction rules which our compiler doesn’t have
- Deprecated “constant” address space
- Better type safety with decorated vs. undecorated pointers
  - New template parameter tells whether methods return decorated vs. undecorated pointers
  - Remove implicit conversion to undecorated pointer
  - Removed constructors from undecorated pointers
  - These conversions must now be explicit

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:multiptr

Device Aspects

- Queries features that a device supports

```
device d = /*…*/;

if (d.has(aspect::usm_device_allocations)) {
  // OK to call malloc_device(d)
}
if (d.has(aspect::fp16)) {
  // OK to use "sycl::half" in kernel
}
```

```
enum class aspect {
  cpu,
  gpu,
  accelerator,
  custom,
  emulated,
  host_debuggable,
  fp16,
  fp64,
  atomic64,
  image,
  online_compiler,
  online_linker,
  queue_profiling,
  usm_device_allocations,
  usm_host_allocations,
  usm_atomic_host_allocations,
  usm_shared_allocations,
  usm_atomic_shared_allocations,
  usm_system_allocation
};
```

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:device-aspects

Heterogeneous Device Features

- OK to use features from different devices in same source file

```
device d1 = /*…*/;
device d2 = /*…*/;

if (d1.has(aspect::fp16)) {
  // Define and submit kernel with "sycl::half"
}
if (d2.has(aspect::atomic64)) {
  // Define and submit kernel with 64-bit atomics
}
```

No compilation error even if "d2" does not support "sycl::half"
or if "d1" does not support 64-bit atomics

(Compiler cannot give error from speculative compilation of kernel for a device.)

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:optional-kernel-features

Kernel Bundles

- Replaces “program” class in SYCL 1.2.1
  - Control timing of kernel JIT compilation
  - Specialization constants
  - Kernel introspection (query kernel properties)

https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html#sec:interfaces.bundles

Specialization constants

- Variables which must be set prior to enqueuing a kernel
- This allows an implementation to JIT for a specific value

```
specialization_id<coeff_t> coeff_id; //global scope
cgh.set_specialization_constant<coeff_id>(…); //command group scope
kh.get_specialization_constant<coeff_id>(); //kernel scope
```

Device copyable

- SYCL 1.2.1 required that objects copied (capture and buffer) are standard layout and trivially copyable
- SYCL 2020 **provisional** requires trivially copyable (implemented)
- SYCL 2020 **final** allows users to declare types as device copyable
- The user does this by specializing the trait is_device_copyable
- User guarantees that doing bitwise copy is the same as calling copy constructor/assignment

Miscellaneous API changes

- Kernel attributes go to the right and don’t propagate
  - old attributes are deprecated
- The old “_class”-suffix API classes were removed

sampled_image, unsampled_image

- SYCL 1.2.1’s image class was replaced with two new classes:
  - sampled_image for images that may use samplers
  - unsampled_image for images that may not

Accessor to const T is read-only accessor

- Additional mechanism to create read-only accessor

SYCL
2020

```
buffer<int> B { data, range{ 1024 } };

Q.submit([&](handler& h) {
  accessor rwAcc { B, h };
  accessor readAcc { B, h, read_only };
  accessor<const int, 1> readAcc2{ B, h };
```

Additional mechanism to acquire read-only accessor

sycl::exception error codes, not class hierarchy

- Simplification of exception management

SYCL 1.2.1

```
try{
  // Do some SYCL work
} catch (sycl::runtime_error &e) {
  std::cout << "Caught SYCL runtime exception: " << e.what() << "\n";
} catch (sycl::accessor_error &e) {
  std::cout << "Caught SYCL accessor exception: " << e.what() << "\n";
}
```

Each error is a different class

SYCL 2020

```
try{
  // Do some SYCL work
} catch (sycl::exception &e) {
  if (e.code() == sycl::errc::runtime) {
    std::cout << "Caught SYCL runtime exception: " << e.what() << "\n";
  } else if (e.code() == sycl::errc::accessor) {
    std::cout << "Caught SYCL accessor exception: " << e.what() << "\n";
  }
}
```

Distinguish errors via code now

Implemented Features
(for education / heads up)

Implemented

oneAPI

Kernels must be immutable

- SYCL 2020 clarifies implementation flexibility and kernel mutability
  - Restricts some behavior for implementation freedom, aligned with STL

SYCL 2020

Kernel invocation functions, such as parallel_for, now take kernel functions by const reference. Kernel functions must now have a const-qualified operator(), and are allowed to be copied zero or more times by an implementation. These clarifications allow implementations to have flexibility for specific devices, and define what users should expect with kernel functors. Specifically, kernel functors can not be marked as mutable, and sharing of data between work-items should not be attempted through state stored within a kernel functor.

```
class MyKernel {
 public:
  MyKernel(accessor<int> ptr)
      : ptr_ { ptr } {
  }
  void operator()(item<1> it) const { ptr_[it.get_id()] = 42; }

 private:
  accessor<int> ptr_;
};
```

Function call operator must be const!

```
h.parallel_for(range{16}, [=](nd_item it) ~~mutable~~ {
```

mutable now illegal!

Implemented

oneAPI

marray

- Old vec class is mostly used as a vector in SPMD code
- Was designed for SIMD
- Experimental support for SIMD provided via ESIMD, sycl::vec, or std::simd (in the future)
- For vectors in SPMD code new marray is recommended
- Improved usage
  - Size never contains padding (vec of 3 elements is 4 long)
  - Element access doesn’t require swizzle and write to element is allowed

Implemented

oneAPI

sycl::exception now derives from std::exception

- SYCL 2020 change to be aware of
  - Simplifies exception management in an application – matters for libraries

SYCL 1.2.1

```
try{
  // Do some SYCL work
} catch (sycl::exception &e) {
  std::cout << "Caught SYCL exception: " << e.what() << "\n";
  return 1;
} catch (std::exception &e) {
  std::cout << "Caught std exception: " << e.what() << "\n";
  return 2;
}
```

SYCL exceptions distinct from std:: exceptions

SYCL 2020

```
try{
  // Do some SYCL work
} catch (std::exception &e) {
  // Do something to output or handle any exception
  std::cout << "Caught exception: " << e.what() << "\n";
  return 1;
}
```

Only catch SYCL exceptions if you want to distinguish

Implemented

oneAPI

Default asynchronous exception handler

- Async errors no longer silently ignored if user doesn't handle
  - Calls std::terminate if unhandled async exceptions on queue/context destruction
  - May manifest at program teardown time, but at least not silent

SYCL 1.2.1

```
// Simple asynchronous handler function
auto handle_async_error = [](exception_list elist) {
  for (auto &e : elist) {
    try{ std::rethrow_exception(e); }
    catch ( sycl::exception& e ) {
      std::cout << "ASYNC EXCEPTION!!\n" << e.what() << "\n";
    } } };
int main() {
  queue Q1{ gpu_selector{}, handle_async_error }
  ...
  Q1.throw_asynchronous();
  ...
```

Must define and install async handler to avoid silently lost errors

SYCL 2020

```
int main() {
  {
    queue Q1{ gpu_selector{} };
  }  // Queue destruction invokes std::terminate() if unhandled async exception
  ...
```

Automatically see errors without doing anything

Implemented

oneAPI

sycl::bit_cast

- SYCL 2020 pulls in C++20 bit_cast as sycl::bit_cast
  - Legal/defined way to reinterpret data

SYCL 2020

```
constexpr std::uint64_t u64v2 = 0x3fe9000000000000ull;
constexpr auto f64v2 = sycl::bit_cast<double>(u64v2);
```

C++20 std::bit_cast

Obtain a value of type To by reinterpreting the object representation of from. Every bit in the value representation of the returned To object is equal to the corresponding bit in the object representation of from. The values of padding bits in the returned To object are unspecified.

reinterpret_cast (or equivalent explicit cast) between pointer or reference types shall not be used to reinterpret object representation in most cases because of the type aliasing rule.

- https://en.cppreference.com/w/cpp/numeric/bit_cast

Implemented

oneAPI

queue(context, device, …)

- SYCL 1.2.1’s queue class was missing a constructor that took both an explicit context and an explicit device

- Implemented? Yes

Implemented

Miscellaneous API changes

- Namespace changed from cl::sycl to sycl
  - Currently implemented with cl as inline namespace

Call for Input on Prioritization

- We need your help to prioritize SYCL 2020 features for implementation in the LLVM open source project
  - Based on your anticipated workloads / research / needs

- This deck covers many of features that we think may be of interest
  - Please provide your view on implementation prioritization. Options:
  - Email the TAB DPC++ mailing list
  - Email anyone from the author list of this presentation
  - Open an issue on the DPC++ TAB GitHub