ComputeCpp: SYCL 2020 with Address Space Inference

Peter Žužek, ComputeCpp Product Owner

Khronos SYCL Webinar, December 7th 2021

Overview

- Address Space Inference
- Status of SYCL 2020 implementation
- ComputeCpp extensions

ComputeCpp 2.8 available

ComputeCpp™

https://developer.codeplay.com

- Explicit USM fully featured
- SYCL 2020 features including unnamed lambdas and reductions

https://developer.codeplay.com/products/computecpp/ce/guides/release-notes

Address Space Inference

Address spaces in SYCL

- *5.9. Address-space deduction in SYCL 2020*
- Address spaces mark pointers in disjoint memory
  - Not part of standard C++
    - *__global, __local, __private, __constant, …*
  - Hardware can benefit from address spaces when it comes to power efficiency
- How to determine address spaces
  - Explicit address spaces as template arguments
    - multi_ptr, accessors
  - SYCL user code should avoid writing them
  - Not many pointers are annotated with an address space, what to do with them?
    - Generic address space vs. compiler determining address spaces

Generic address space

- Some compilers place unannotated pointers into generic address space
  - No special qualifier needed
  - Additional instructions to resolve the pointer
- Drawbacks
  - Runtime overhead
  - Target device must support this
    - Not in OpenCL 1.2
    - Embedded and automotive devices usually don’t
    - Need well-defined address spaces

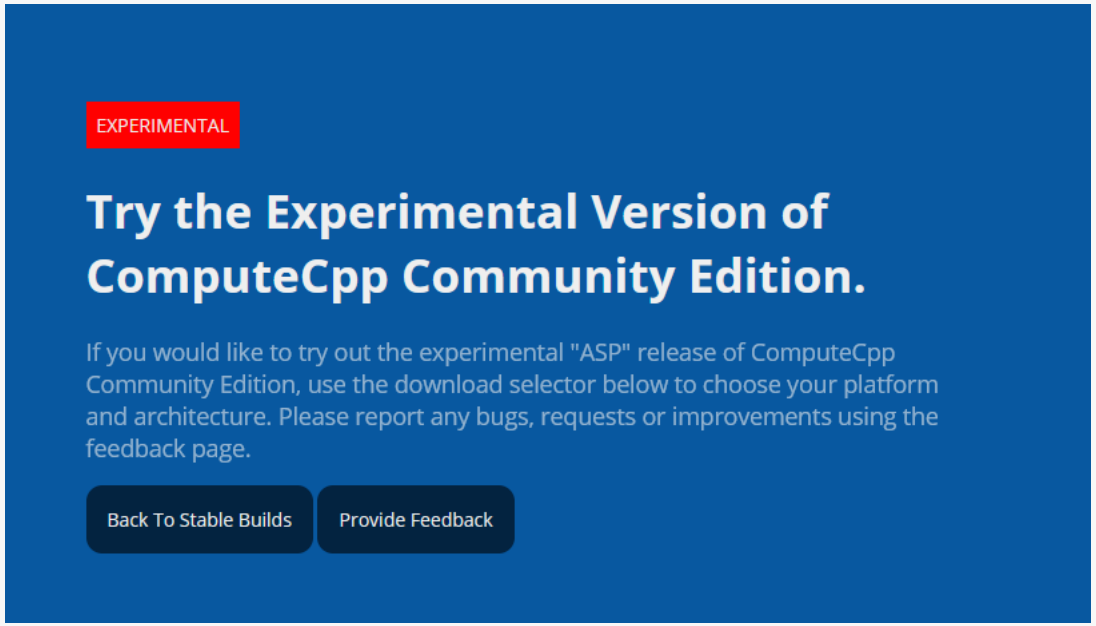
Inferred address space in current compute++

- Device compiler determines exact address spaces
  - Treat address spaces as a C++ language extension
    - Address space is a type qualifier
  - Use initial values from some types and conditions
  - Deduce address space for unannotated pointers from initial values
- Lets developers target a wider range of more power efficient hardware
- Drawbacks
  - Requires modifications to Clang
  - Interferes with C++ semantic analysis
    - Creates inconsistencies in template deduction and instantiation in user code

Inference instead of deduction

- Run extra LLVM pass
  - When address space is determined (solved) for a type, propagate it back to all types related to the solved one
    - Type unification based on usage
    - Can detect mismatches
  - Does not interfere with C++ type system
- Unlocks more optimization opportunities compared to generics
  - Alias analysis
  - Interprocedural (optimizations don’t affect result)
- Some breaking changes compared to current compiler
  - More in line with other SYCL implementations

Experimental Device Compiler



https://developer.codeplay.com/experimental

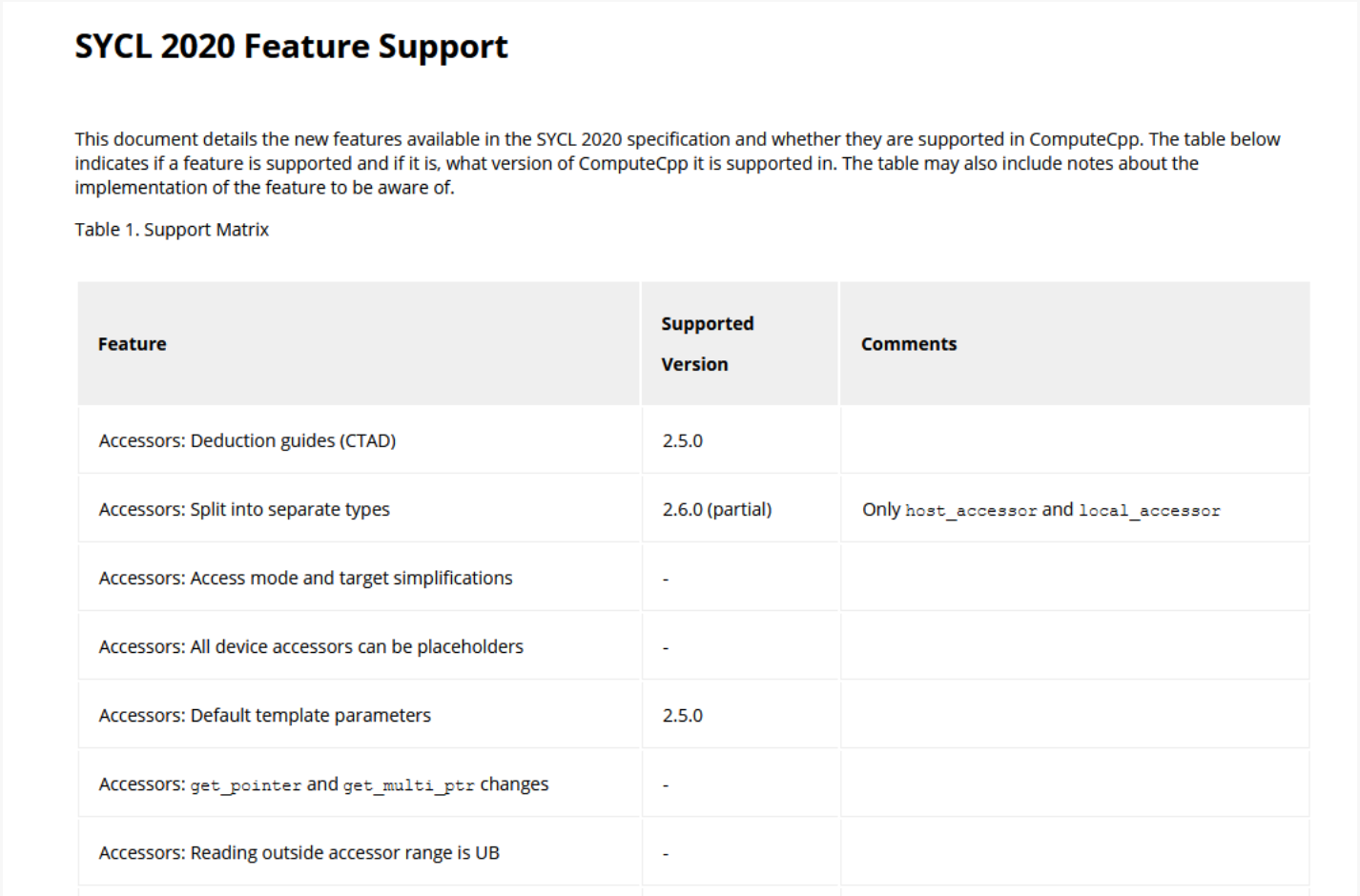
- Implements new Address Space Inference
- Will follow LLVM upstream closely
- Will become default compiler
- ComputeCpp 2.8 ships with almost fully functional experimental compiler
  - No support for hierarchical parallelism yet

Still experimental, please try it and report any issues

Status of SYCL 2020 implementation

Feature Support Matrix

- Up to date matrix
- Specifies version information
- Early access to features



SYCL 2020 Feature Support

This document details the new features available in the SYCL 2020 specification and whether they are supported in ComputeCpp. The table below indicates if a feature is supported and if it is, what version of ComputeCpp it is supported in. The table may also include notes about the implementation of the feature to be aware of.

Table 1. Support Matrix

Feature	Supported Version	Comments
Accessors: Deduction guides (CTAD)	2.5.0	
Accessors: Split into separate types	2.6.0 (partial)	Only `host_accessor` and `local_accessor`
Accessors: Access mode and target simplifications	-	
Accessors: All device accessors can be placeholders	-	
Accessors: Default template parameters	2.5.0	
Accessors: `get_pointer` and `get_multi_ptr` changes	-	
Accessors: Reading outside accessor range is UB	-	

https://developer.codeplay.com/sycl2020

Exclusive Features: Experimental Compiler

- Unified Shared Memory
  - Used to require usm_wrapper class, no longer needed
    - Most other functionality available since 2.2
  - Relies heavily on Address Space Inference
- Unnamed kernel lambdas
  - Host compiler must support __builtin_sycl_unique_stable_name
    - Must match device compiler behavior
    - Works best with –sycl-driver switch

```
auto ptr = sycl::malloc_device<float>(count, testQueue);
testQueue.submit([&](handler& cgh) {
  cgh.parallel_for(range{count}, [=](auto itemID) {
    auto i = itemID.get_linear_id();
    ptr[i] = static_cast<float>(i);         }); });
```

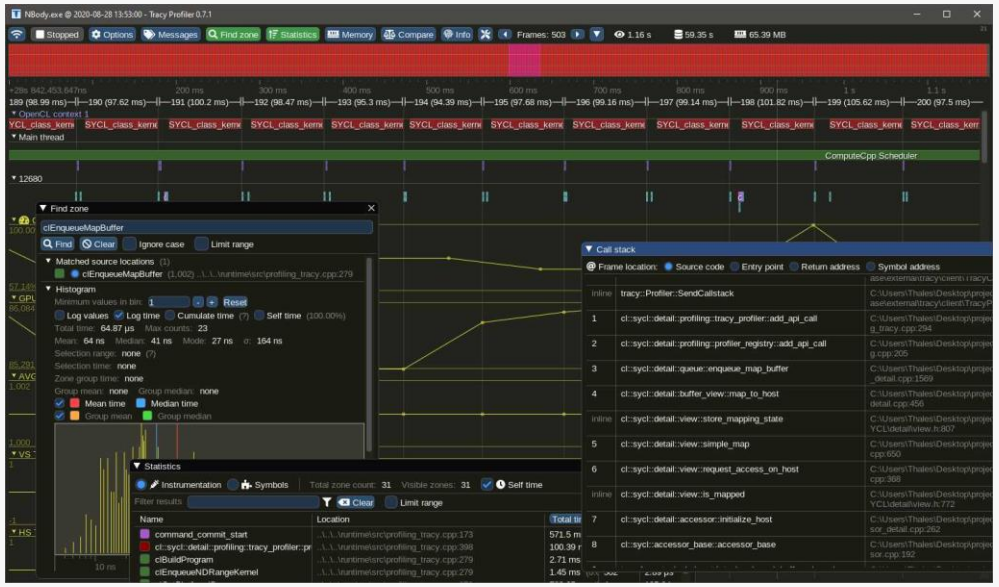
Other big-ticket items

- Reductions
- Backend generalization
  - Host task accessors
- CTAD
- Simplified accessors
  - Deduction guides, separate types, default parameters
- New device selector API
- Error codes
- Accessors as reversible containers
- accessor::operator[](0) into start of range
- In-order queues
- Host-only for some features
  - atomic_ref, subgroups, group algorithms

```
testQueue.submit([&](sycl::handler& cgh) {
  cgh.parallel_for<kernelName>(
      range<1>(dataSize),
      // Reduction objects, to perform summation
      reduction(ptrOutput, binaryOpA{}, propList),
      reduction(bufOutput, cgh, binaryOpB{}, propList),
      [=](id<1> idx, auto& sumA, auto& sumB) {
        size_t i = idx.get(0);
        sumA.combine(ptrAT[i] * ptrBT[i]);
        sumB.combine(ptrAU[i] * ptrBU[i]);
      });
});
```

ComputeCpp extensions

ComputeCpp Profiling Support



Tracy Profiler Output

- JSON and Tracy output
- Support performance counters
  - Intel MDAPI
  - ARM HWCPipe
- Tuning via configuration file
- `codeplay::profiling::profiling_zone`

SYCL extensions

- constexpr id and range classes
  - Enabled by default in C++14 and higher
- COMPUTECPP_EXT_READ_ACC_CONST_PTR
  - Mark pointers of read-only accessors as const
- Command group batching
  - Don’t flush queue on submit
- host_access
  - How device data can be accessed by the host, if at all
- DMA and on-chip memory
  - Global address space, but separated from main memory

Extra capabilities

- Custom instructions
  - `[[computecpp::builtin]]`
- Buffer creation policies
- Offline kernel compilation
- Multiple device targets in single binary
- Checking accessor bounds
- Host device
- Configuration file
- Kernel performance inspector
- computecpp_info

```
********************************************************************************

ComputeCpp Info (RC 2.8.0 2021/12/06)

SYCL 1.2.1 revision 3

********************************************************************************


Device Info:

Discovered 1 devices matching:
  platform    : <any>
  device type : <any>

--------------------------------------------------------------------------------
Device 0:

  Device is supported                     : UNTESTED - Untested OS
  Bitcode targets                         : spir64 spirv64
  CL_DEVICE_NAME                          : Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
  CL_DEVICE_VENDOR                        : Intel(R) Corporation
  CL_DRIVER_VERSION                       : 2020.10.6.0.04
  CL_DEVICE_TYPE                          : CL_DEVICE_TYPE_CPU

********************************************************************************
```

Download ComputeCpp Today

https://developer.codeplay.com

https://developer.codeplay.com/experimental

- Download the latest releases
- The brand new compiler is available at the experimental link
- Remember to give us your feedback

sycl@codeplay.com

We're Hiring!

codeplay.com/careers/

codeplay

Enabling AI to be Open, Safe & Accessible to All

@codeplaysoft

info@codeplay.com

codeplay.com

Template instantiation changes

```
template <typename T>
void act(T *t) {
  t[0] = 1.0;
};
template<>
void act(float __attribute__((opencl_global))* t) {
  t[0] = 2.0;
}
...
// buf points to `float value`
accessor acc{buf, cgh};
cgh.single_task<class kernel>([=]() {
  // get() returns `__global float*`
  float* as_pointer = acc.get_multi_ptr().get();
  act(as_pointer);
});
...
std::cout << "value: " << value << "\n";
```

- Current compiler prints 2.0
  - `as_pointer` deduced to `__global float*`
- Experimental compiler prints 1.0
  - Generic template selected before inference happens