IWOCL 2024

The 12th International Workshop on OpenCL and SYCL

SimSYCL: A SYCL Implementation Targeting Development, Debugging, Simulation and Conformance

Fabian Knorr, University of Innsbruck

Peter Thoman, Fabian Knorr (University of Innsbruck), Luigi Crisci (University of Salerno)

Motivation

- Testing and debugging a SYCL program requires access to accelerator hardware
- SYCL programs are often not portable between GPU vendors
- Implementations do not typically enforce requirements of the kernel API
- Distributed-memory, asynchronous, parallel execution is difficult to debug

Goal: A developer-focused CPU-only SYCL implementation with simulation capabilities.

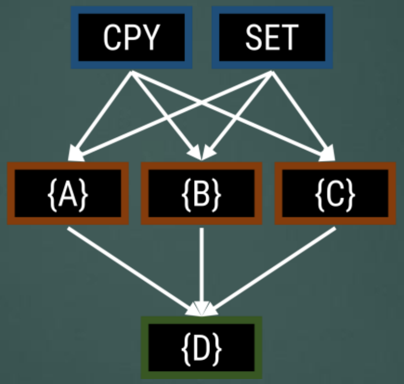
SimSYCL in the Ecosystem

SimSYCL's simulation and verification capabilities helps in quick development of correct and portable SYCL applications.

Debugger-Friendly Synchronous Execution


```
sycl::queue q;
auto cpy = // CPY
  q.memcpy(bufA, bufB, sz);
auto set = // SET
  q.memset(bufC, 0, sz);
q.wait();
auto a = // {A}
  q.single_task([]() { });
auto b = // {B}
  q.single_task([]() { });
// {C}
q.wait();
// {D}
```

Asynchronous SYCL DAG



SimSYCL synchronous execution

Few limitations:

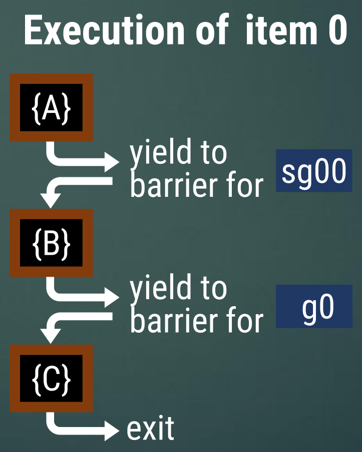
- Kernels can’t wait for live host accessors to go out of scope
- Shared-Memory communication between user-space and kernels is forbidden

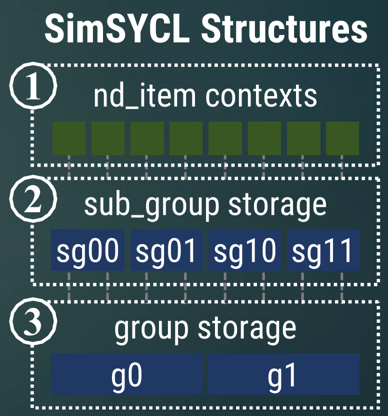
Executing ND-range kernels

In order for work items to meet at group-collective operations (barrier, reduce, …) while keeping local variables intact, a sequential schedule must be able to switch between stacks.

SimSYCL uses *boost.context* to maintain an execution context for each item in a group.

```
sycl::queue().submit([](sycl::handler &cgh) {
  const auto range = sycl::nd_range<1>{8, 4};
  cgh.parallel_for(range, [](auto itm) {
    const auto &g = itm.get_group();
    const auto &sg = itm.get_sub_group();
    // {A}
    sycl::group_barrier(sg);
    // {B}
    sycl::group_barrier(g);
    // {C}
  });
});
```





Verification of SYCL host code

- Strict adherence to the SYCL specification and avoiding any non-standard interfaces will identify non-conformant user code
- Runtime checking of invariants that would negatively impact performance in typical production-grade SYCL implementations
- Full compatibility with AddressSanitizer (even in kernel code!)

Run-time verification in kernel code


```
sycl::queue q;
q.submit([](sycl::handler& cgh) {
    cgh.parallel_for(sycl::nd_range<1>(2,2),
                     [=](sycl::nd_item<1> item) {
        auto id = item.get_global_id(0);
        if(id == 0) {
            sycl::group_barrier(item.get_group());
        }
    });
});
```

Undefined Behavior: All work-items must converge on the group barrier

```
SimSYCL check failed: id_equivalent
  at simsycl/group_operation_impl.cc:37:5
group operation id mismatch:
  group recorded operation "barrier", but work item 1 is trying to perform "exit"
```

Rigorous Concept Checking with C++20

SimSYCL anticipates the switch to C++20 with a concept-based SYCL interface.

```
template<typename T>
concept SyclFloat = std::is_same_v<T, float>
        || std::is_same_v<T, double>
        || std::is_same_v<T, sycl::half>;

template<typename T>
concept GenFloat = SyclFloat<T> || (
    (Swizzle<T> || Vec<T> || MArray<T>)
    && SyclFloat<typename T::element_type>);

template<GenFloat T1, GenFloat T2>
requires(std::same_as<T1,T2> || MatchingVec<T1,T2>)
auto max(T1 x, T2 y) { ... }
```

Officially supported compilers are GCC 11, Clang 17, and MSVC 14.

Inversion of Device Capabilities

1. Specify platforms, devices and capabilities via SimSYCL API or a JSON system definition

```
"devices": {
  "SimSYCL virtual GPU": {
    "device_type": "gpu",
    "max_work_item_sizes<1>": [1024],
    "max_work_item_sizes<2>": [1024, 1024],
    "max_work_item_sizes<3>": [64, 1024, 1024],
    "local_mem_size": 65536,
    "global_mem_size": 8589934592,
    "sub_group_sizes": [32],
    ...
  }
}
```

2. Device enumeration, memory capacities, (sub-) group sizes, and device-info queries are simulated accordingly

```
sycl::device d;
size_t lm_size = d.get_info<
    sycl::info::device::local_mem_size>();
assert(lm_size == 65536);
```

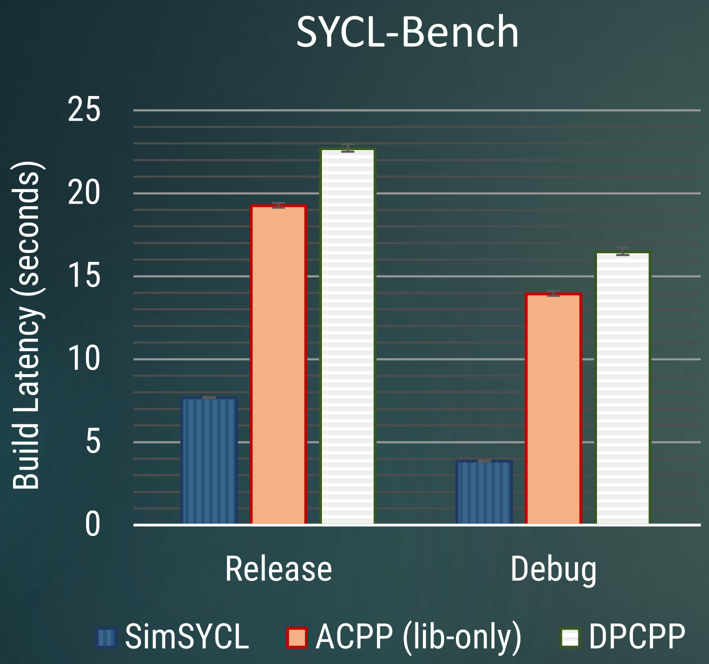
An Executable Specification

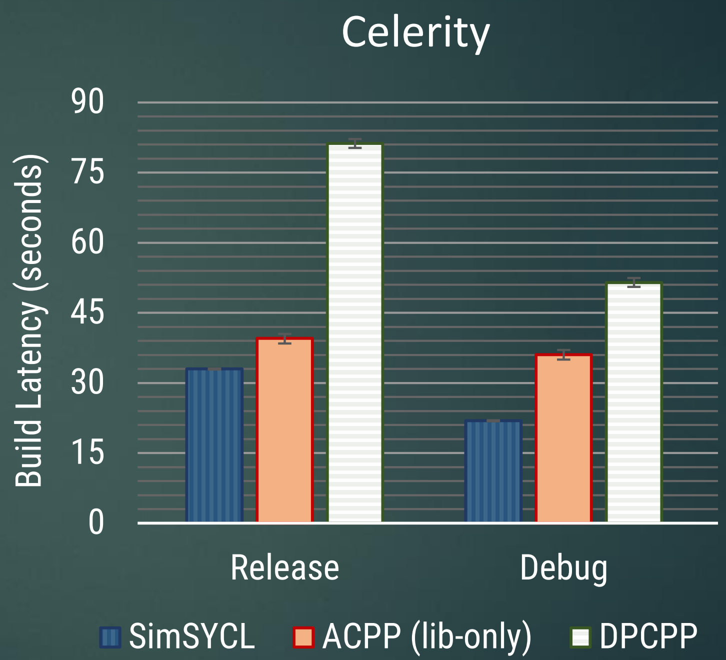
The simplified execution model allows SimSYCL to become the smallest possible conformant implementation and qualifies it as a testing ground for new SYCL features.

There are few SYCL features that SimSYCL cannot support:

- Asynchronicity between the user's application thread and kernels or host tasks
- Attributes like `[[sycl::reqd_sub_group_size]]` (require compiler support)
- Queries on kernel properties like `sycl::is_compatible()`

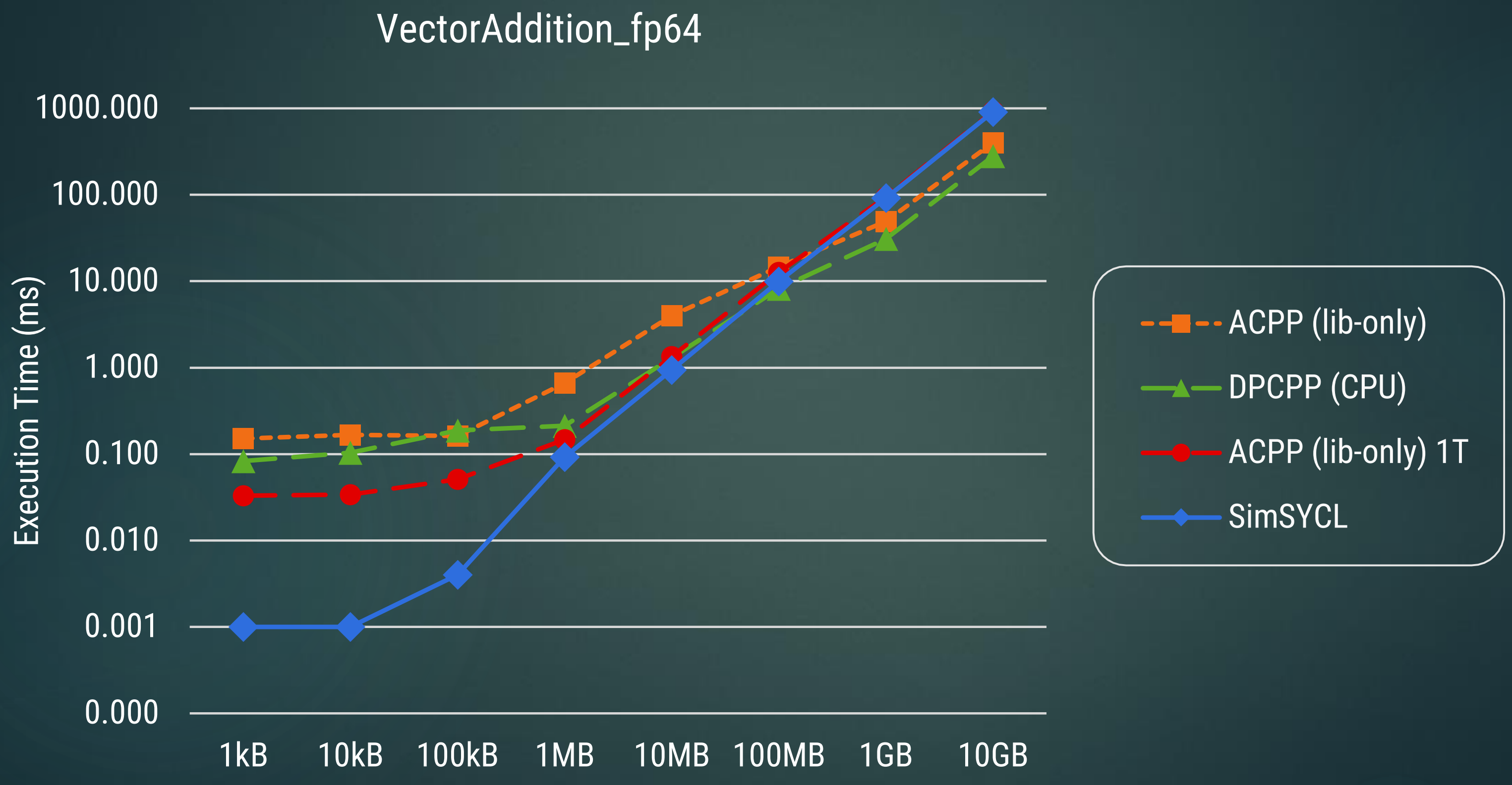
Improved Edit-Compile-Debug Cycle



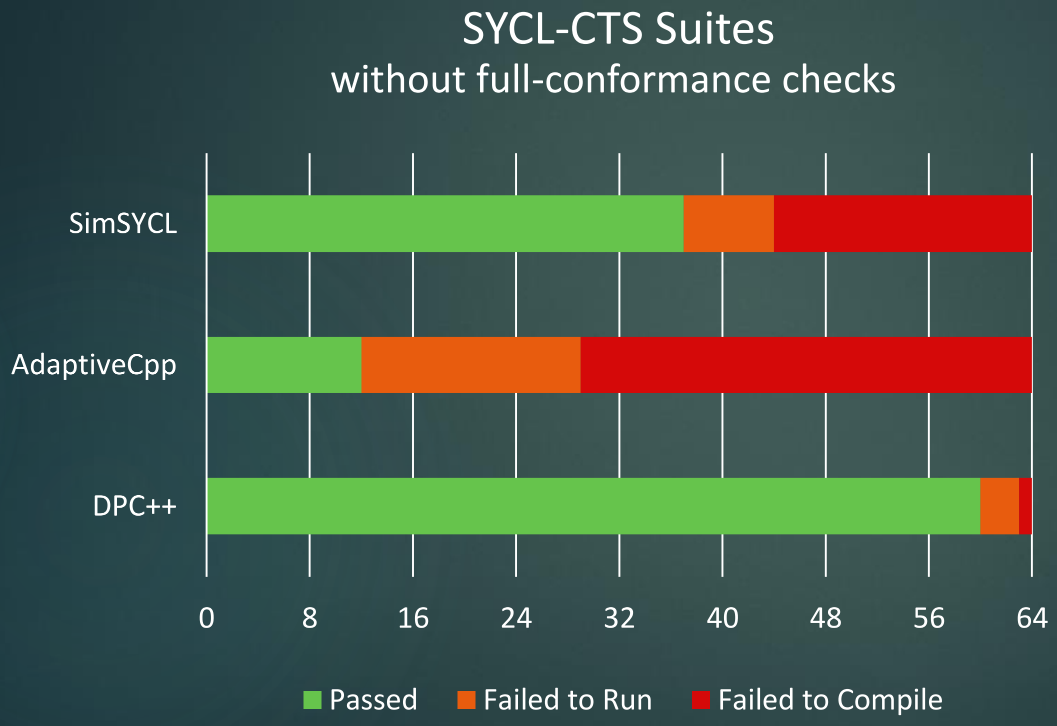


on dual AMD EPYC 7763, 1TB DDR4-3200 RAM, ninja, Clang 17.0.6, ld.mold, Ubuntu 22.04

Runtime Benchmarks – Simple Kernels



SYCL-CTS Conformance



Revision aa0762ef

Revision 3952b468
OpenMP backend

Revision 25c3666d
OpenCL CPU backend

SimSYCL

Try it today!

https://github.com/celerity/SimSYCL