14th International Workshop on OpenCL and SYCL

IWOCL 2026

ProtoSYCL: A Sample Implementation of a SYCL Compiler for Conformance Test Suite Development

Michael Aziz, Intel

Michael Aziz, Intel. Yvan Labiche, Carleton University.

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

Outline

1. Problem Statement
2. Proposed Solution
3. Implementation
4. Evaluation
5. Conclusion

Problem Statement

SYCL Verification

- Many SYCL implementations in development
- Need to ensure SYCL implemented correctly
- Bugs can exist in many places:
  - Implementation
  - Test cases
  - Specification

Conformance Test Suite (CTS)

- Open source: https://github.com/KhronosGroup/SYCL-CTS
- Comprises 72 test categories
- Includes test plans, extension tests
- Supports three SYCL implementations

Motivation

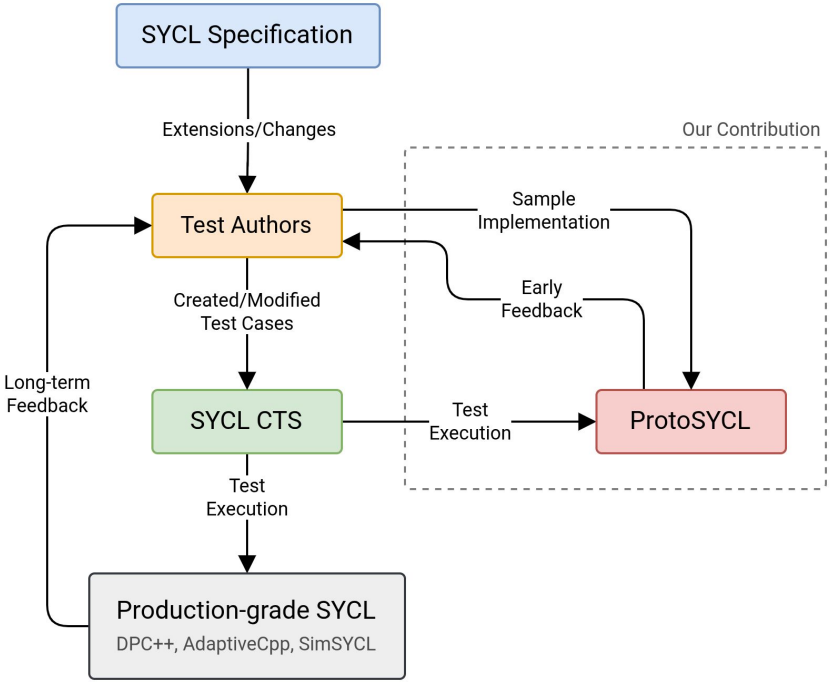
- CTS is invaluable for finding SYCL bugs
- Correctness issues in CTS cause problems for implementers
- Need a way to find CTS bugs
- Requires compiling/linking/executing CTS

❗	The CTS is currently only *compiled* during CI, but not *executed*. This means that passing CI does not imply anything about the quality of your testing logic.

Proposed Solution

ProtoSYCL

- Sample SYCL implementation
- Open source: https://github.com/0x12CC/ProtoSYCL
- Intended to help SYCL testers
- Prioritizes conformance over performance
- Passes 68 of 72 CTS categories



ProtoSYCL

- Supports nearly all SYCL features
- Useful for improving CTS coverage and quality
- Supports sanitizers, debuggers, instrumentation
- Device code can use any C++23 feature

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

int main() {
 sycl::queue q;
 q.parallel_for(10, [](const sycl::item<1> it) {
   const std::size_t idx = it.get_linear_id();
   std::println("Hello from work-item {}!",
idx);
 });
 q.wait();
}
```

Non-portable SYCL code

Implementation

Implementation

- Single-source single compiler pass (SSCP)
- Compiler:
  - Implements kernel/device attributes
  - Clang-based compilation
  - Out-of-tree Clang & LLVM plugins
- Runtime:
  - Implements all other SYCL features
  - Based on C++ concurrency primitives

Compiler

- SYCL attributes require compiler support
- **Problem:** lambda kernel attributes appertain to type
  - KhronosGroup/SYCL-Docs#136
  - Cannot implement using a Clang plugin
- **Solution:** rewrite attributes to appertain to declaration

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

int main() {
 sycl::queue q;
 q.parallel_for<struct ExampleKernel>(
     sycl::nd_range{{12}, {4}},
     [](const sycl::item<1> it) [[sycl::reqd_work_group_size(4)]] {});
 q.wait();
}
```

Example kernel attribute

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

int main() {
 sycl::queue q;
 q.parallel_for<struct ExampleKernel>(
     sycl::nd_range{{12}, {4}},
     [](const sycl::nd_item<1> it)
         __attribute__((annotate("sycl::reqd_work_group_size", 4))) {});
 q.wait();
}
```

Clang input

Compiler

- Attributes automatically rewritten during compilation
- Dynamically interposes Clang's file I/O
- No changes to Clang's parser
- Clang and LLVM plugins associate kernel RTTI with attributes

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

int main() {
 sycl::queue q;
 q.parallel_for<struct ExampleKernel>(
     sycl::nd_range{{12}, {4}},
     [](const sycl::item<1> it) [[sycl::reqd_work_group_size(4)]] {});
 q.wait();
}
```

Example kernel attribute

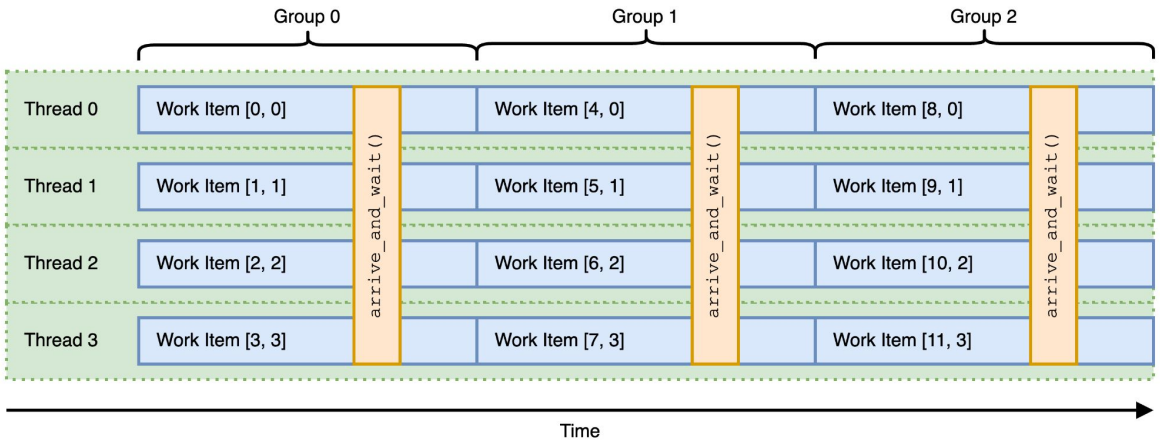
```
#include <sycl/sycl.hpp>

int main() {
 sycl::queue q;
 q.parallel_for<struct ExampleKernel>(
     sycl::nd_range{{12}, {4}},
     [](const sycl::nd_item<1> it)
         __attribute__((annotate("sycl::reqd_work_group_size", 4))) {});
 q.wait();
}
```

Clang input

Runtime

- Directed acyclic graph scheduler for task parallelism
- Thread pool for data parallelism
- Each work item in group runs on a thread



Evaluation

Finding CTS Bugs

1. Compile CTS with sanitizers enabled
2. Analyze each compilation error, test failure, and sanitizer report
3. If root cause is CTS bug, fix it: #1163, #1164, #1165, #1166, #1167, #1168, #1169, #1170

Research Questions

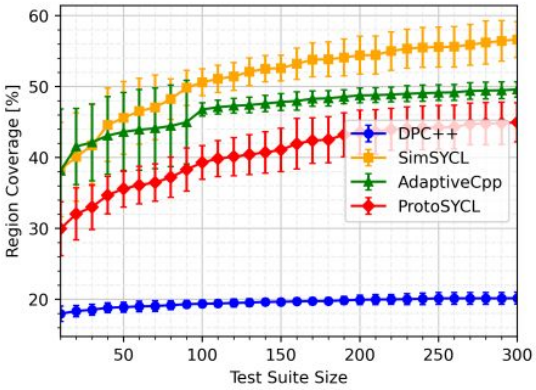
1. Does ProtoSYCL's structural code coverage mirror that of production-grade implementations?

2. Can ProtoSYCL effectively compare and rank different test suites?

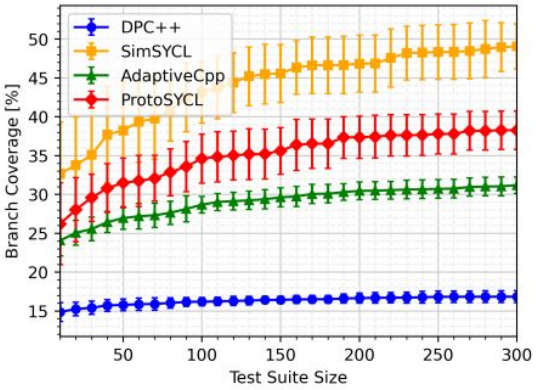
Experimental Evaluation

1. Create test pool of 3000 test cases from CTS
2. Generate 450 random test suites by sampling from pool
3. Run each test suite with different SYCL implementations
4. Analyze coverage measurements

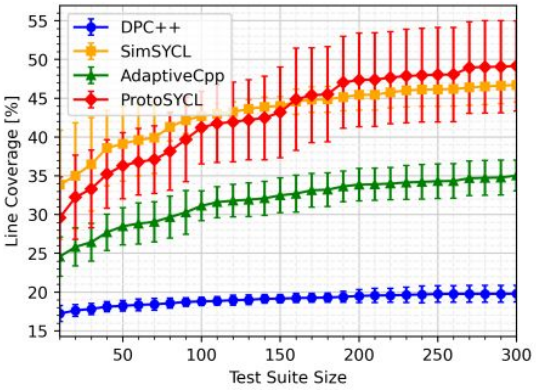
Structural Coverage



(a) Region

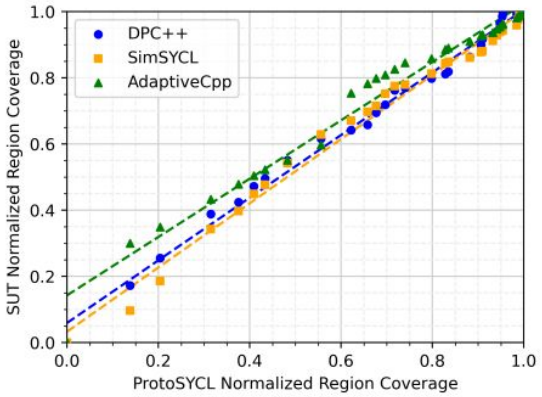


(b) Branch

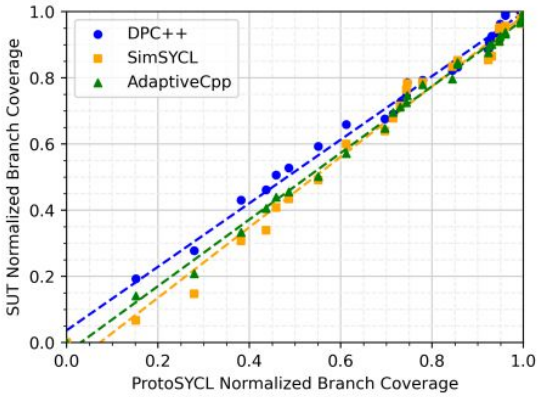


(c) Line

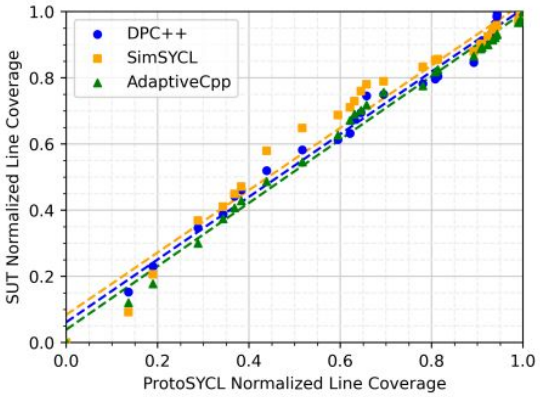
Coverage Correlation



(a) Region

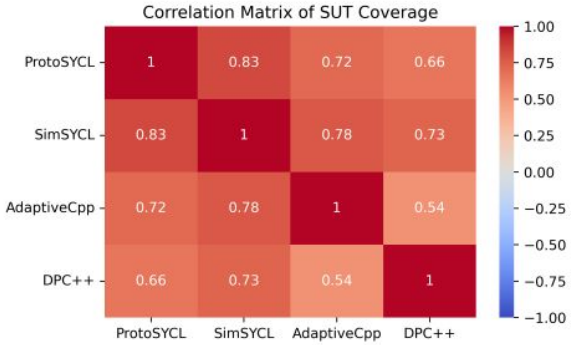


(b) Branch

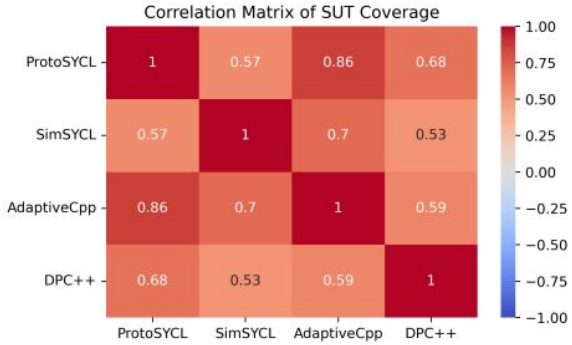


(c) Line

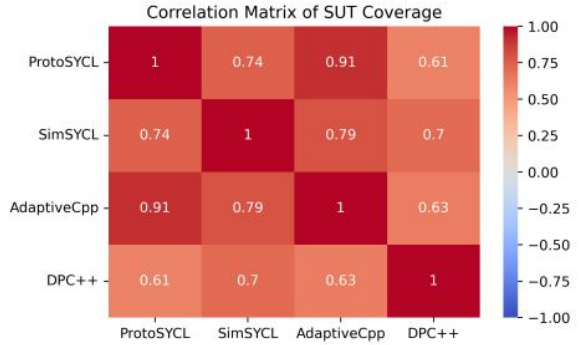
Coverage Correlation



(a) Region

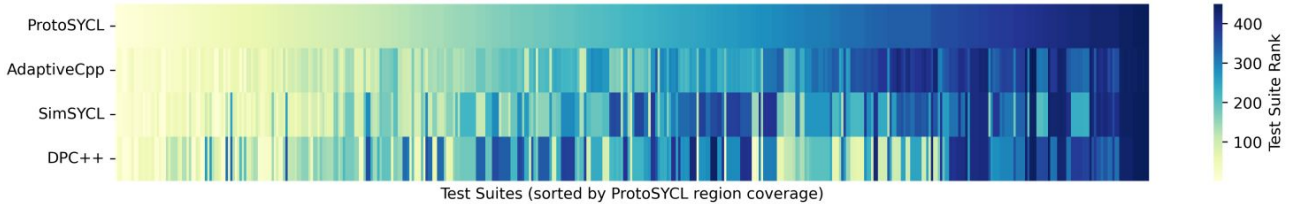


(b) Branch

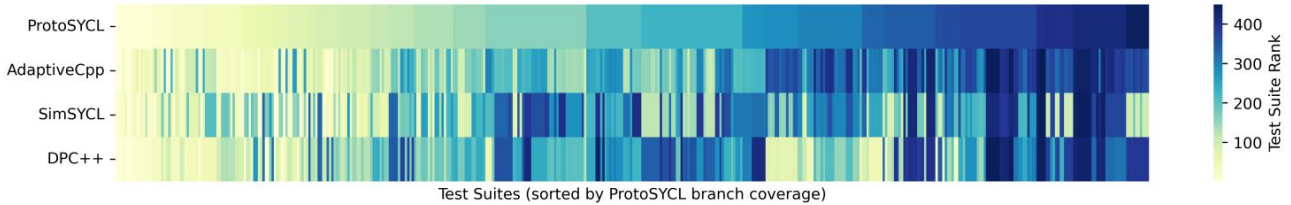


(c) Line

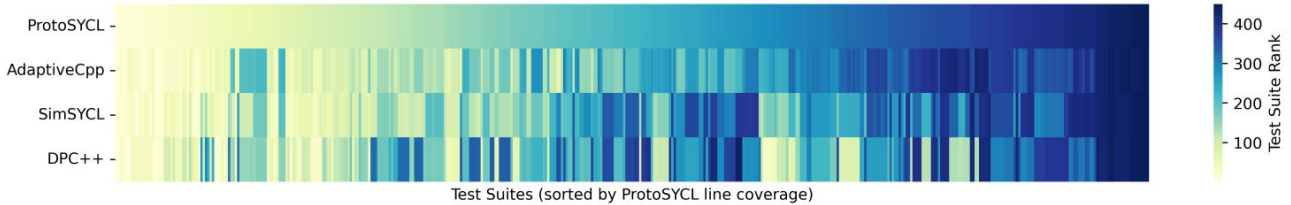
Test Suite Ranking



(a) Region



(b) Branch



(c) Line

Conclusion

Summary

- ProtoSYCL is a sample SYCL implementation
- Useful for test case development, debugging, and analysis
- Coverage analysis suggests it can model real SYCL implementations

Future Work

- Implement remaining SYCL features
  - 4 missing test categories
  - KHR extension tests
- Support SYCL experiments
  - Prototype proposed KHR extensions
  - Provide configurable forward progress guarantees
  - Evaluate test suites using other coverage criteria