Leveraging MLIR for Better SYCL Compilation

Víctor Pérez-Carrasco*, Ettoire Tiotto†, Whitney Tsang†, Lukas Sommer*, Victor Lomüller*, James Brodman†, Mehdi Gol†

Codeplay Software Ltd.*

Intel Corporation†

The SYCL Platform

- Single source, high-level standard C++ programming model targeting a range of heterogeneous platforms
- Different implementations:
  - DPC++ (Intel)
  - ComputeCpp (Codeplay)
  - HighSCL (Heidelberg University)
  -...

An MLIR-Based SYCL Compiler

- Extensible framework to build compilers
- C/C++ lacks a proper MLIR front-end
- Using polygeist (incomplete) for device code
- Exploring raising from LLVM IR for host code
- Allowing code for different targets in a single module thanks to its nested structure
- Using higher-level abstractions for higher-level optimizations

Goal of Project: 1-pass MLIR-Based Compiler

- Keep host and device code as close as possible
- Device code in GPU module
- Embed the SYCL API semantic into MLIR
- Optimize SYCL specific patterns
- Split into different modules when lowering to LLVM

Dialects: Capturing Higher-Level Semantics

C++ to target using clang + LLVM
- C++ to target using clang + Polygeist + MLIR + SYCL + LLVM

SYCL to MLIR Translation

Reduction Loop in SYCL

#include <sycl/sycl.hpp>

void initit( { std::vector<long> a();
  sycl::queue q;
  for (sycl::accessor buf[a], cgh, sycl::write_only, sycl::no_init);
  cgh.parallel_for(a.size(),
  [=](sycl::id<1> i) {
    for (size_t k = 0; k < M; ++k)
      acc[a] = expm(k);
  });
}

Scalor Replacement on Reduction Loops

For(size_t k = 0; k < M; ++k)
acc[i] = expm(k);

Calls to SYCL runtime functions obfuscate program semantics when the code is lowered to LLVM IR (making difficult to recognize the opportunity). In MLIR memory semantics of the dialect operations can be easily modeled.

Next Steps

- Work on host code generation
  - Proper C++ compiler? Looking into some projects:
    - Clang/IR
    - Polygeist (currently used for device code)
  - Raising from LLVM IR (current approach for host code)
  - Connection with other dialects
  - Optimize host code to avoid frequent calls to SYCL runtime functions

Project Status

- Attributes, types and operations to represent common SYCL constructs:
  - attributes: address, address_space, alignment
  - types: id, item, nd_item
  - container: accessor, vec
  - reduction: minimum, maximum
  - constructor
  - global_id, local_id
  - accessor, subscript
  - > 60 % SYCL test-suite success rate
  - No host-side representation
  - Parallel for: call-site representation, buffer, queue, etc.
  - Host-side lowering is an open question:
    - In lack of a capable C++ front-end, current approach involves.

Performance gains for a subset of SYCL Bench workloads due to scalar replacement on reduction loops / LICM.

Performance gains for a subset of SYCL Bench workloads due to scalar replacement on reduction loops / LICM.