EmitC

Recent Improvements and Future Developments

October 12, 2023

Marius Brehler, Simon Camphausen

Fraunhofer IML

Outline

- The EmitC Dialect
- Users and Use-Cases
- Recent Improvements
- Future Developments

The EmitC Dialect

- EmitC is an MLIR dialect to emit C and C++ code.
- The dialect was initialy presented with https://reviews.llvm.org/D76571.

- Dialect was upstreamed on June 19, 2021.
- Emitter was upstreamed on September 2, 2021.

- The initially upstreamed dialect consisted of:
    - `emitc.apply`
    - `emitc.call`
    - `emitc.constant`
    - `emitc.include`
    - `#emitc.opaque`
    - `!emitc.opaque`

EmitC - CallOp

operation ::=

`emitc.call` $callee `(` $operands `)` attr-dict `:` functional-type($operands, results)

- The call operation represents a C++ function call.

Example:

```
func.func @f(%arg0: i64, %arg1: i64) -> i64 {
 %0 = "emitc.call"(%arg0,%arg1) {callee = "foo"} : (i64, i64) -> i64
 return %0 : i64
}


func.func @f(%arg0: i64, %arg1: i64) -> i64 {
 %0 = emitc.call "foo" (%arg0,%arg1) {} : ( i64, i64) -> i64
 return %0 : i64
}
```

```
int64_t f(int64_t v1, int64_t v2) {
 int64_t v3 = foo(v1, v2);
 return v3;
}
```

Further Operations and Types

- Further operations and types were added:

- `emitc.cast`
- `emitc.variable`
- `!emitc.ptr`

Example:

```
// Cast from `int32_t` to `float`

%0 = emitc.cast %arg0: i32 to f32


// Cast from `void` to `int32_t` pointer
%1 = emitc.cast %arg1 : !emitc.ptr<!emitc.opaque<"void">> to !emitc.ptr<i32>
```

Users and Use-Cases

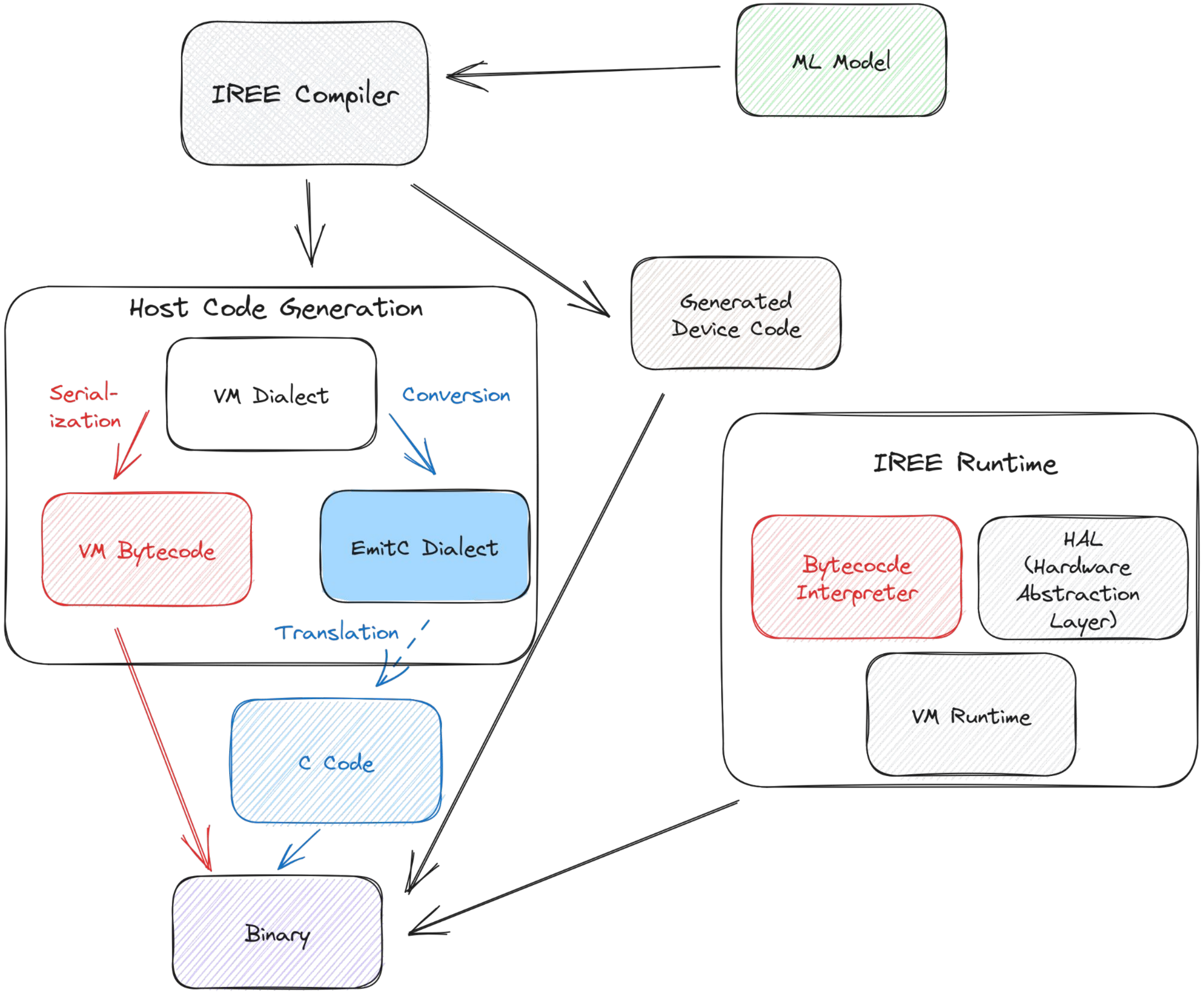
- **IREE** and especially **TinyIREE**
- (ETH Zurich’s) End-to-End Toolchain for Fully Homomorphic Encryption
- **CIRCT**
- **Kokkos** emitter

- TOSA/StableHLO to C++ (**MLIR-EmitC**)
  - Google’s Machine Learning Guided Compiler Optimizations Framework (**MLGO**)

- MLIR/C Interfacing

IREE

- VM Bytecode is replaced by generated C code
- Bytecode Interpreter is no longer needed
- Results in smaller executables
- Currently still requires a custom emitter



CIRCT & Kokkos Emitter

Circuit IR Compilers and Tools

- Contains the SystemC dialect and an ExportSystemC emitter

- SystemC dialect imports types from EmitC
- The ExportSystemC emitter contains a emitter for EmitC patterns

```
systemc.module @emitcEmission () {
 systemc.ctor {
   %0 = "emitc.constant"() {value = #emitc.opaque<"5">
        : !emitc.opaque<"int">} : () ->
          !emitc.opaque<"int">
   %five = systemc.cpp.variable %0
             : !emitc.opaque<"int">
…
```

CIRCT & Kokkos Emitter

Circuit IR Compilers and Tools

- Contains the SystemC dialect and an ExportSystemC emitter

- SystemC dialect imports types from EmitC
- The ExportSystemC emitter contains a emitter for EmitC patterns

```
systemc.module @emitcEmission () {
 systemc.ctor {
   %0 = "emitc.constant"() {value = #emitc.opaque<"5">
          : !emitc.opaque<"int">} : () ->
            !emitc.opaque<"int">
   %five = systemc.cpp.variable %0
               : !emitc.opaque<"int">
…
```

Kokkos emitter

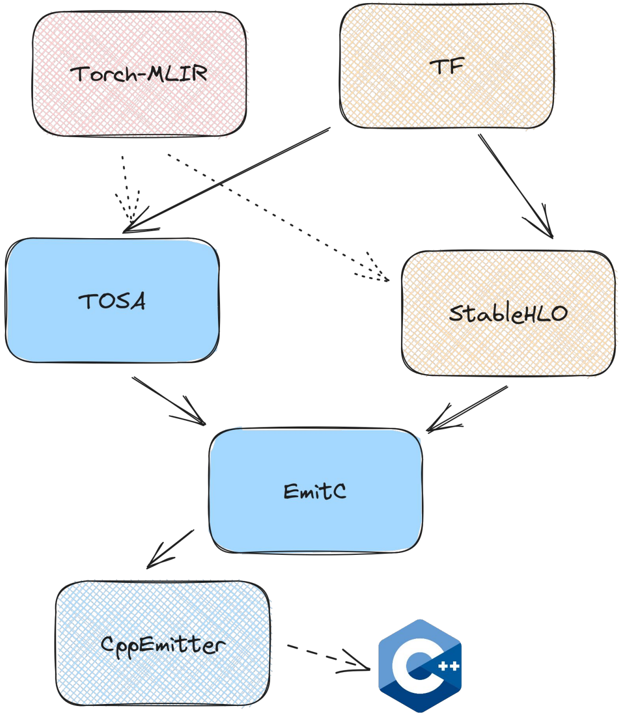
- Allows to compile Python programs to Kokkos C++ source code
- Used MLIR's existing C++ emitter as a starting point
- A higher-level set of dialects was chosen

MLIR	Kokkos
`memref.store %50 %A[%1]`	`A(i) = 50;`

`%0 = memref.alloc() :`
`memref<100xf32>`
	`View<float[100]> v("v");`

`%a = math.sqrt %b`	`float a = Kokkos::sqrt(b);`
`%a = arith.subf %b, %c`	`float a = b - c;`

TOSA / StableHLO to C++



- Conversions from TOSA / StableHLO to EmitC are available at https://github.com/iml130/mlir-emitc

- Provides a header-only C++ reference implementation

- Allows to translate MobileNetV2 to a C++

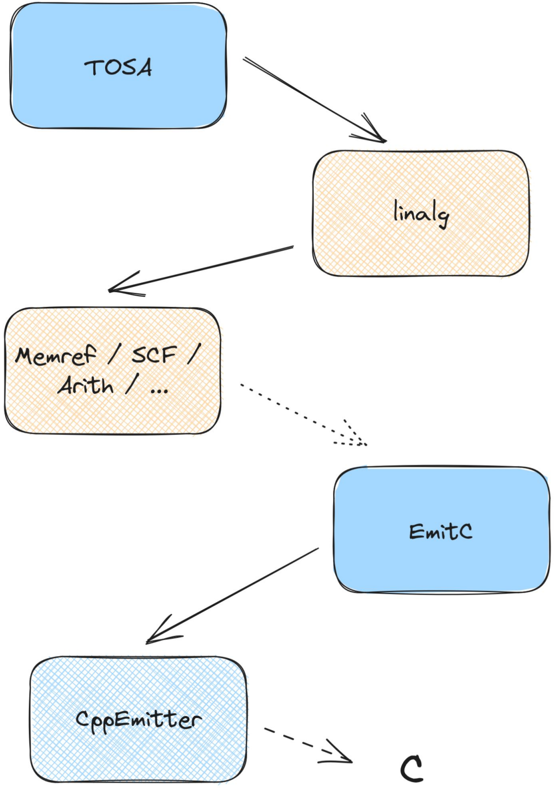
- Used by Google’s Machine Learning Guided Compiler Optimizations Framework (MLGO) (https://github.com/google/ml-compiler-opt) to remove direct dependency to TensorFlow

MLIR / C Interfacing

- There are use-cases where code can or should not be compiled with LLVM
    - Generate C or C++ instead

- Can be realized via MLIR's conversion framework

- This requires further operations to represent C / C++ constructs, currently not supported by the upstream EmitC dialect



Recent Improvements

- Arithmetic Operations
  - `emitc.add`
  - `emitc.div`
  - `emitc.mul`
  - `emitc.rem`
  - `emitc.sub`
- `emitc.cmp`, supports
  - equal to
  - not equal to
  - less than
  - less than or equal
  - greater than
  - greater than or equal
  - three-way-comparison

- `emitc.literal`

- `emitc.assign`
- `emitc.if`
- `emitc.for`
- `emitc.yield`

Future Developments

- Further operations and types
  - `emitc.array`
  - `emitc.struct` (type / definition)
  - `emitc.array`
  - `emitc.func`

- Support for const type qualifiers
- Preprocessor directives
- Function declarations for mutually recursive functions

- Verifiers (for C99, …)

Conclusion

- The development is need-driven!
- If you have other use-cases and further requirements, let us know.
- Contributions are highly appreciated!