WebAssembly support in MLIR

Ferdinand Lemaire
Jessica Paquette
Luc Forget

Woven By Toyota

Proposed agenda

- Rapid presentation (~15-20’)
  - Motivations for WebAssembly support in MLIR
  - Guided tour
  - Overview of current implementation state
    - Testing
    - Technical challenges
- Technical discussion (~35-40’)
- Path to upstream

Motivations

- WebAssembly (Wasm) : a well specified, portable binary format with built in safety features.
  - Wasm module is executed by *an embedder* (wasm equivalent of JVM) which provides lightweight sandboxing.
  - Wasm ISA offers good control flow integrity by design
  - Despite the name, not specific to web browser execution. Many embedders exists for standalone execution.
  - Many programming languages can be compiled to Wasm binaries.
- Motivation of this work: provide a framework for AOT compilation of Wasm modules to native code, benefiting from the high quality codegen of LLVM.

Who would be interested by a Wasm dialect ?

- People wanting to improve Wasm backends (using the dialect as a target):
  - Researcher wanting to improve Wasm performance, e.g. authors of this paper on similar work, showing they can improve the quality of generated wasm when starting from a higher abstraction level than LLVM IR to Wasm.
  - Wasm backend developer for various programming language (see comment in the RFC).
- People interested in embedder related development
- People wanting to develop analysis / optimisation of Wasm module or work on improving the quality of native code generation from Wasm module in the comfortable setting of MLIR.

Guided tour of current implementation

We have

- Binary Wasm → Wasm Dialect importer
  ```
  mlir-translate --import-wasm XYZ.wasm -o XYZ.mlir
  ```
- Wasm Dialect → usual “core” dialect lowering
  ```
  mlir-convert --raise-wasm XYZ.mlir -o XYZ_lowered.mlir
  ```
- A driver to make it simple to run an end-to-end
  ```
  wasabi -o XYZ.ll XYZ.wasm
  ```

Guided tour of current implementation

- Binary Wasm → Wasm Dialect importer
  Is a regular translation.
  Sources: mlir/lib/Target/Wasm Headers: mlir/include/mlir/Target/Wasm
  Tests: mlir/test/Target/WebAssembly
- Wasm Dialect
  Ops definitions: include/mlir/Dialect/WebAssembly/IR/WebAssemblyOps.td
  SSA representation of Wasm operations
- Lowering to usual set of dialects (arith, cf, math, func, memref…)
  Implemented as one conversion pass.
  Implementation: mlir/lib/Conversion/RaiseWasm/RaiseWasmMLIR.cpp
  Tests: mlir/test/Conversion/RaiseWasm

Guided tour of current implementation

Dialect design:

- Very straightforward, numeric operations looks a lot like their counterparts from `arith` and `math` dialects
- Using symbol visibility mechanism to represent imports and exports
- One “exotic” feature for `wasm.function`:
  - Wasm function arguments have reference semantic, can be written to like local variables.
  - As a result, type T of function input type is mapped to entry block argument of type !wasm<local T>.

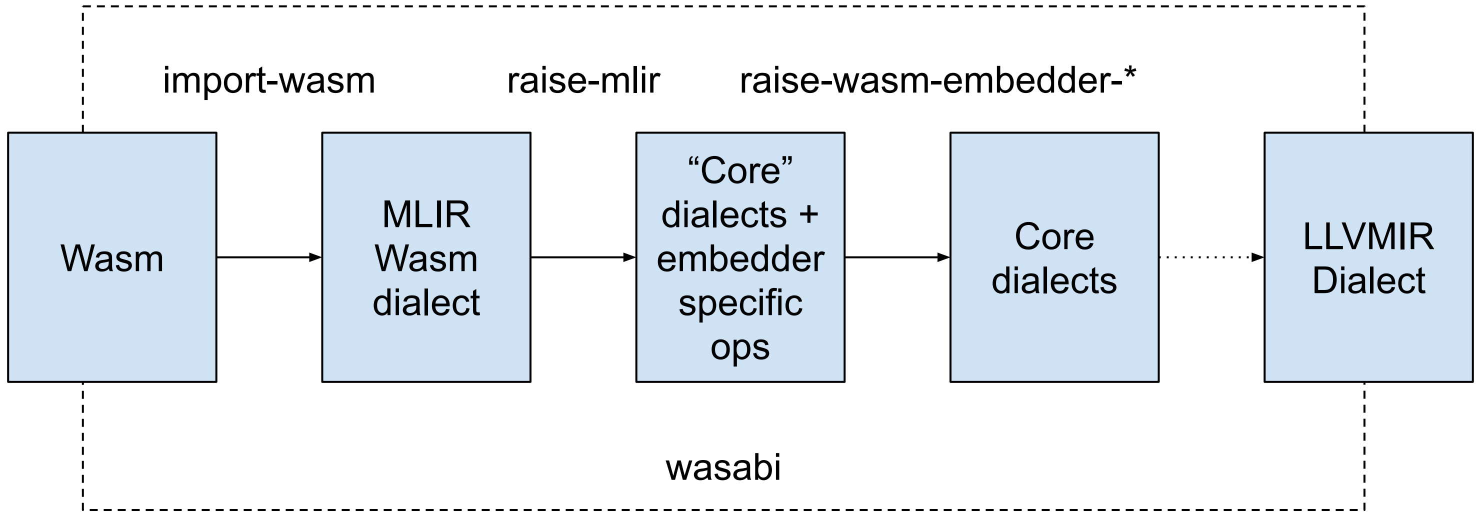
Guided tour of current implementation

Testing:

- Basic testing for each supported import and lowering of op, using llvm-lit.
- Needs a bit more negative testing (in particular for malformed wasm binaries)
- When better coverage, the Wasm conformance test suite could be used for end to end validation.

Technical challenges

- Wasm: stack based virtual machine. MLIR works better with SSA, so the dialect technically represents superset of Wasm.
    - In RFC: proposal of a two dialect architecture with one that would really be one to one mapping to “real” wasm, with the op having only “side effects” to represent the program.
- Handling of the multiple embedders:
    - Some operation are tightly coupled to the embedder: e.g. Trap
    - Once again, got good suggestion for RFC to have a dialect extension for embedder ops an embedder specific lowering.
    - For instance
        - `wasm.trap` → `wasm.wasmtime.trap` → relevant function call during lowering to LLVM IR.
- Versioning:
    - Wasm is supposed to be backward compatible, so issue mostly concerns the translations
- (minor) lack of control on FP rounding mode in some of arith / maths operations + loose specification of some FP corner cases for some ops.


import-wasm
raise-mlir
raise-wasm-embedder-*
Wasm
MLIR Wasm dialect
“Core” dialects + embedder specific ops
Core dialects
LLVMIR Dialect
wasabi

Dialect example: function

```
wasm.func nested @func_1(%arg0: !wasm<local ref to i32>, %arg1: !wasm<local ref to i32>) -> i32 {
  %v0 = wasm.local_get %arg0 : ref to i32
  %v1 = wasm.local_get %arg1 : ref to i32
  %0 = wasm.add %v0 %v1 : i32
  wasm.return %0 : i32
}
```

Dialect example: loop

```
module {
 wasm.func nested @func_0() -> i32 {
   %0 = wasm.local of type i32
   wasm.loop : {
     %1 = wasm.local_get %0 : ref to i32
     %2 = wasm.const 10 : i32
     %3 = wasm.lt_si %1 %2 : i32 -> i32
     wasm.block_return %3 : i32
   }> ^bb1
 ^bb1(%1: i32):  // pred: ^bb0
   wasm.return %1 : i32
 }
}
```

Discussion

Path to upstream