ML Frameworks and Frontends in MLIR

Hot Chips 34

Suraj Sudhir
August 21 2022
ML Framework Frontends

- Environments to define and build ML models
ML Framework Frontends

- Environments to define and build ML models
- Offer a range of capabilities
ML Framework Frontends

- Environments to define and build ML models
- Offer a range of capabilities
- Very dynamic & evolving space
ML Framework Frontends

• Environments to define and build ML models

• Offer a range of capabilities

• Very dynamic & evolving space

• ML compiler / systems design goals:
  • Support multiple frameworks
  • Keep up with their evolution
ML Framework Characteristics

• Expressiveness
  • High-level language capabilities and paradigms
ML Framework Characteristics

• Expressiveness
  • High-level language capabilities and paradigms

• Feature richness
  • Operator sets / libraries
ML Framework Characteristics

• Expressiveness
  • High-level language capabilities and paradigms

• Feature richness
  • Operator sets / libraries

• Infrastructural
  • Training, quantization, optimization/performance.
ML Framework Characteristics

• Expressiveness
  • High-level language capabilities and paradigms

• Feature richness
  • Operator sets / libraries

• Infrastructural
  • Training, quantization, optimization/performance.

• ML compiler ask:
  • All this needs to "just work".
MLIR In ML Frameworks

• Starts with an ML model constructed within a framework.
MLIR In ML Frameworks

- Starts with an ML model constructed within a framework.

- Translators convert serialized model to MLIR form.
MLIR In ML Frameworks

- Starts with an ML model constructed within a framework.
- Translators convert serialized model to MLIR form.
- Enables construction of MLIR based compiler infrastructure.
MLIR In ML Frameworks

- MLIR dialects of multiple frameworks already present.

- TensorFlow and TensorFlow Lite from TensorFlow project

- PyTorch via Torch-MLIR

- JAX

- ONNX via ONNX-MLIR
Framework Consumption in MLIR

- Use case
  - Inference
  - Training
- Data types
  - Floating point
  - Integer
- Quantization
- Other Reqs

Frameworks

Compiler Frontend

Code Gen
Connecting Frameworks and Code Generation

- ‘Reduction’ or ‘waistline’ mid-level dialects.
  - Designed to be compilation friendly.

- Convert frontend ops to mid-level dialect(s).

- Complex ops decomposed into sequence of simpler ones

- Backend code generation paths target reduction dialect(s)
Mid-level MLIR Dialects

- **TOSA** (Tensor Operator Set Architecture) Dialect
  - Specification-based
  - Defines functionality and precision
  - Enables hardware/software codesign.

- **HLO** (High Level Operations) Dialect
  - Input language to XLA compiler at Google
  - Primary output of JAX for compilation.

- **LinAlg** Dialect
  - Powerful codegen oriented dialect
  - Enables tiling, vectorization, bufferization and other capabilities
Case Study: **TOSA**

- Designed at ARM.

- Problem: Frontend dynamism and heterogeneity, needed to stabilize hardware design.

- Goal: Target multiple frontends, stable path to ML accelerators.

- Whole-tensor operator set architecture backed by specification.
  - Defines functionality, precision, quantization.

- **MLIR dialect** implements specification.
TOSA

- TOSA is stable and versioned.

- Defines profiles
  - Base inference, main inference, training

- Conversions from frontends to TOSA
  - TensorFlow, TensorFlow Lite (stable)
  - Torch-MLIR for PyTorch (advanced development)
  - ONNX-MLIR for ONNX (WIP)

- Hardware and software designed to TOSA compliance.
  - TOSA compliant hardware development at ARM.
  - Used within Google’s IREE MLIR compiler, and elsewhere.
Example: Quantized Conv2D

- Input Frontend: TensorFlow Lite
  - Quantized Conv2D + bias addition
  - Fused relu6 activation
  - symmetrically quantized signed 16 bit datatype

```mlir
module attributes {tf_saved_model.semantics, tfl.description = "MLIR Converted.", tfl.schema_version = 3 : i32} {  
func @main(%arg0: tensor<1x32x32x8x!quant.uniform<i16:f32, 6.1037011619191617E-5>>) ->  
(tensor<1x32x32x16x!quant.uniform<i16:f32, 1.8311105668544769E-4>> {  
  %0 = "tfl.pseudo_qconst"() {qtype = tensor<16x1x1x8x!quant.uniform<i8<-127:127>:f32:0, {..zps..}>>, value = dense"..rawdata.."} : tensor<16x1x1x8xi8>} : () -> tensor<16x1x1x8!quant.uniform<i8<-127:127>:f32:0>>
  %1 = "tfl.pseudo_const"() {value = dense<0> : tensor<16xi64>} : () -> tensor<16xi64>
  %2 = "tfl.conv_2d"(%arg0, %0, %1) {dilation_h_factor = 1 : i32, dilation_w_factor = 1 : i32, fused_activation_function = "RELU6", padding = "SAME", stride_h = 1 : i32, stride_w = 1 : i32} : (tensor<1x32x32x8x!quant.uniform<i16:f32, 6.1037011619191617E-5>>, tensor<16x1x1x8x!quant.uniform<i8<-127:127>:f32:0, {..zps..}>>, tensor<16xi64>) -> tensor<1x32x32x16x!quant.uniform<i16:f32, 1.8311105668544769E-4>>
  return %2 : tensor<1x32x32x16x!quant.uniform<i16:f32, 1.8311105668544769E-4>>}
}
Conv2D: Conversion to TOSA

- `tosa.conv2d` consumes input zps, bias added to accumulated result
- `tosa.rescale` + `tosa.clamp` performs output rescaling + activation

```mlir
module attributes {tf_saved_model.semantics, tfl.description = "MLIR Converted.", tfl.schema_version = 3 : i32} {
  func @main(%arg0: tensor<1x32x32x8xi16> ) -> (tensor<1x32x32x16xi16> ) {
    %0 = "tosa.const"() {value = dense<0> : tensor<16xi48>} : () -> tensor<16xi48>
    %1 = "tosa.const"() {value = dense<"..raw.."> : tensor<16x1x1x8xi8>} : () -> tensor<16x1x1x8xi8>
    %2 = "tosa.conv2d"(%arg0, %1, %0) {dilation = [1, 1], pad = [0, 0, 0, 0], quantization_info = {input_zp = 0 : i32, weight_zp = 0 : i32}, stride = [1, 1]} : (tensor<1x32x32x8xi16>, tensor<16x1x1x8xi8>, tensor<16xi48>) -> tensor<1x32x32x16xi48>
    %3 = "tosa.rescale"(%2) {double_round = false, input_zp = 0 : i32, multiplier = [21438 : i32, 18643 : i32, 20949 : i32, 19892 : i32, 18542 : i32, 20624 : i32, 20035 : i32, 21773 : i32, 19670 : i32, 31465 : i32, 18895 : i32, 21587 : i32, 31080 : i32, 19230 : i32, 21345 : i32, 20069 : i32], output_zp = 0 : i32, per_channel = true, scale32 = false, shift = ..shifts..} : (tensor<1x32x32x16xi48>) -> tensor<1x32x32x16xi16>
    %4 = "tosa.clamp"(%3) {max_fp = 0.000000e+00 : f32, max_int = 32767 : i64, min_fp = 0.000000e+00 : f32, min_int = 0 : i64} : (tensor<1x32x32x16xi16>) -> tensor<1x32x32x16xi16>
    return %4 : tensor<1x32x32x16xi16>
  }
}
Example: Complex MatMul

- PyTorch n-dim matrix multiplication
  - `matmul(4x8x16x32, 8x32x17) -> 4x8x16x17`

- Additional artifacts
  - Implicit and explicit broadcasting
  - Shape inference
  - Map to fixed hardware-friendly TOSA 3x3 `matmul`

```mlir
module attributes {torch.debug_module_name = "Matmul"} {
  func @forward(%arg0: !torch.vtensor<[4,8,16,32],f32>, %arg1: !torch.vtensor<[8,32,17],f32>) -> !torch.vtensor<[?,?,?,?],f32> {
    %0 = torch.aten.matmul %arg0, %arg1 : !torch.vtensor<[4,16,32],f32>, !torch.vtensor<[8,32,17],f32> -> !torch.vtensor<[?,?,?,?],f32>!
    return %0 : !torch.vtensor<[?,?,?,?],f32>
  }
}
```
MatMul-ND: Conversion to TOSA

- **Transpose** + **reshape** to canonical form
  - LHS: common x lhs_bcast x reduction
  - RHS: common x reduction x rhs_bcast

```mlir
module attributes {torch.debug_module_name = "Matmul"} {
  func @forward(%arg0: tensor<4x8x16x32xf32>, %arg1: tensor<8x32x17xf32>) -> tensor<?x?x?x?xf32> {
    %0 = "tosa.const"() {value = dense<[1, 2, 0, 3]> : tensor<4xi32>} : () -> tensor<4xi32>
    %1 = "tosa.const"() {value = dense<[1, 0, 2, 3]> : tensor<4xi32>} : () -> tensor<4xi32>
    %2 = "tosa.reshape"(%arg1) {new_shape = [1, 8, 32, 17]} : (tensor<8x32x17xf32>) -> tensor<1x8x32x17xf32>
    %3 = "tosa.transpose"(%arg0, %1) : (tensor<4x8x16x32xf32>, tensor<4xi32>) -> tensor<8x4x16x32xf32>
    %4 = "tosa.reshape"(%3) {new_shape = [8, 64, 32]} : (tensor<8x4x16x32xf32>) -> tensor<8x64x32xf32>
    %5 = "tosa.transpose"(%2, %0) : (tensor<1x8x32x17xf32>, tensor<4xi32>) -> tensor<8x32x1x17xf32>
    %6 = "tosa.reshape"(%5) {new_shape = [8, 32, 17]} : (tensor<8x32x1x17xf32>) -> tensor<8x32x17xf32>
    %7 = "tosa.matmul"(%4, %6) : (tensor<8x64x32xf32>, tensor<8x32x17xf32>) -> tensor<8x64x17xf32>
    %8 = "tosa.reshape"(%7) {new_shape = [8, 4, 16, 17]} : (tensor<8x64x17xf32>) -> tensor<8x4x16x17xf32>
    %9 = "tosa.transpose"(%8, %1) : (tensor<8x4x16x17xf32>, tensor<4xi32>) -> tensor<4x8x16x17xf32>
    %10 = tensor.cast %9 : tensor<4x8x16x17xf32> to tensor<?x?x?xf32>
    return %10 : tensor<?x?x?xf32>
  }
}``
Example: n-Dim Gather

- Input Frontend: TensorFlow
  - GatherND

```mlir
module attributes {tf.versions = {bad_consumers = [], min_consumer = 0 : i32, producer = 1011 : i32}} {
  func @main(%arg0: tensor<1x32x32x8xf32>) -> tensor<3x3x32x8xf32> attributes {tf.entry_function = {control_outputs = "", inputs = "placeholder_0", outputs = "result"} {%
cst = "tf.Const"() {device = "", value = dense<[[[0, 1], [0, 19], [0, 30]], [[0, 10], [0, 7], [0, 9]], [[0, 2], [0, 24], [0, 31]]]} : tensor<3x3x2xi32>}{()} -> tensor<3x3x2xi32>
  %0 = "tf.GatherNd"(%arg0, %cst) {device = ""} : (tensor<1x32x32x8xf32>, tensor<3x3x2xi32>) -> tensor<3x3x32x8xf32>
  return %0 : tensor<3x3x32x8xf32>
}
```
GatherND: Conversion to TOSA

- TensorFlow to TOSA conversion
  - TOSA transpose + reshape + gather

```mlir
module attributes {tf.versions = {bad_consumers = [], min_consumer = 0 : i32, producer = 1011 : i32}} {
  func @main(%arg0: tensor<1x32x32x8xf32>) -> tensor<3x3x32x8xf32> attributes {tf.entry_function = {control_outputs = "", inputs = "placeholder_0", outputs = "result"}} {
    %0 = "tosa.const"() {value = dense<[[32, 1]]> : tensor<1x2xi32>} : () -> tensor<1x2xi32>
    %1 = "tosa.const"() {value = dense<[[0, 1], [0, 19], [0, 30], [0, 10], [0, 7], [0, 2], [0, 24], [0, 31]]> : tensor<9x2xi32>} : () -> tensor<9x2xi32>
    %2 = "tosa.reshape"(%arg0) {new_shape = [1, 32, 256]} : (tensor<1x32x32x8xf32>) -> tensor<1x32x256xf32>
    %3 = "tosa.mul"(%1, %0) {shift = 0 : i32} : (tensor<9x2xi32>, tensor<1x2xi32>) -> tensor<9x2xi32>
    %4 = "tosa.reduce_sum"(%3) {axis = 1 : i64} : (tensor<9x2xi32>) -> tensor<9x1xi32>
    %5 = "tosa.reshape"(%4) {new_shape = [1, 9]} : (tensor<9x1xi32>) -> tensor<1x9xi32>
    %6 = "tosa.gather"(%2, %5) : (tensor<1x32x256xf32>, tensor<1x9xi32>) -> tensor<1x9x256xf32>
    %7 = "tosa.reshape"(%6) {new_shape = [3, 3, 32, 8]} : (tensor<1x9x256xf32>) -> tensor<3x3x32x8xf32>
    return %7 : tensor<3x3x32x8xf32>
  }
}
```
TOSA to Code Gen

- Implemented by Google IREE MLIR team

---

--tosa-to-arith - Lower TOSA to the Arith dialect
--tosa-to-linalg - Lower TOSA to LinAlg on tensors
--tosa-to-linalg-named - Lower TOSA to LinAlg named operations
--tosa-to-scf - Lower TOSA to the SCF dialect
--tosa-to-tensor - Lower TOSA to the Tensor dialect

---

Example:

```mlir
module attributes {tf_saved_model.semantics, tfl.description = "MLIR Converted.", tfl.schema_version = 3 : i32} {
  func.func @main(%arg0: tensor<1x32x32x8xf32>) -> (tensor<1x32x32x8xf32>) {
    %0 = "tosa.max_pool2d"(%arg0) {kernel = [2, 2], pad = [0, 1, 0, 1], stride = [1, 1]} : (tensor<1x32x32x8xf32>) -> tensor<1x32x32x8xf32>
    return %0 : tensor<1x32x32x8xf32>
  }
}
Example: TOSA to LinAlg

$ mlir-opt -pass-pipeline="func.func(tosa-to-linalg-named, tosa-to-linalg)" maxpool.mlir
TOSA: Current Status

- Part of the core MLIR dialect set.

- Significant support infrastructure around dialect.
  - Reference model
  - Large unit testing infrastructure

- Multiple stable frontend consumption paths.
  - Thousands of models run (Arm, Google, elsewhere)

- Hardware development at Arm + elsewhere.

- Collaboration and interest across MLIR ecosystem.
Mid-level IR Design: Reflections from TOSA

• Defining the overall requirement is critical.
  • Close to frontend ? Co-design friendly ? Spec-backed ? Other ?
  • Define principles and/or write a rationale document.
Mid-level IR Design: Reflections from TOSA

• Defining the overall requirement is critical.
  • Close to frontend? Co-design friendly? Spec-backed? Other?
  • Define principles and/or write a rationale document.

• Development and connectivity were significant efforts.
  • Multiple person-years
  • Inter company collaboration - Arm, Google, AMD and more.
Mid-level IR Design: Reflections from TOSA

• Defining the overall requirement is critical.
  • Close to frontend? Co-design friendly? Spec-backed? Other?
  • Define principles and/or write a rationale document.

• Development and connectivity were significant efforts.
  • Multiple person-years
  • Inter company collaboration - Arm, Google, AMD and more.

• What kind of quarterback do you want?
Summary

• ML compiler developers may have to support a range of capabilities present across multiple frameworks.

• There’s a substantial gap in abstraction between framework level ops and backend code generation patterns.

• Choosing or developing an appropriate mid level IR is critical to effectively connect the framework and low level code gen.

• Developers can leverage the experience that went into existing mid level IRs to make the right design choices.
Acknowledgments

• Arm ML Technology and Engineering teams

• Google IREE team

• and the MLIR community
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
ありがとう
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה