7-15 Personal Research

Author: 洪祐鈞

Abstract

Since I had Gastroenteritis and some private family stuff to deal with, it's just a quick summary what I've discovered.

All IR dump

refer to the how to debug, we might be able to print all MLIR conversion.

```
./onnx-mlir /home/sylvex/mnist_export/mnist_model.onnx -mlir-print-ir-before-all &> log3.txt
```

```
// -----// IR Dump Before (anonymous
namespace)::DecomposeONNXToONNXPass (decompose-onnx) //----- //
func.func @main_graph(%arg0: tensor<1x1x28x28xf32> {onnx.name =
"x.1"}) -> (tensor<1x10xf32> {onnx.name = "19"}) {
%0 = onnx.Constant dense<[-0.0159200802, 0.357616782, 2.3570277E-4,
...]> : tensor<32xf32>
%1 = onnx.Constant
dense<"0x2F9C9F3ED8E10A3F24140B3F0EE8F7BDD999DE3C4D04633E4653A..."> :
tensor<32x1x3x3xf32>
...

%8 = "onnx.Conv"(%arg0, %1, %0) {auto_pad = "NOTSET", dilations = [1,
1], group = 1 : si64, kernel_shape = [3, 3], onnx_node_name =
"/conv1/Conv", pads = [1, 1, 1, 1], strides = [1, 1]} :
(tensor<1x1x28x28xf32>, tensor<32x1x3x3xf32>, tensor<32xf32>) ->
tensor<1x32x28x28xf32>
%9 = "onnx.Relu"(%8) {onnx_node_name = "/relu/Relu"} :
(tensor<1x32x28x28xf32>) -> tensor<1x32x28x28xf32>
%10 = "onnx.MaxPoolSingleOut"(%9) {auto_pad = "NOTSET", ceil_mode = 0
: si64, dilations = [1, 1], kernel_shape = [2, 2], onnx_node_name =
"/pool/MaxPool", pads = [0, 0, 0, 0], storage_order = 0 : si64,
strides = [2, 2]} : (tensor<1x32x28x28xf32>) ->
tensor<1x32x14x14xf32>
%11 = "onnx.Conv"(%10, %3, %2) {auto_pad = "NOTSET", dilations = [1,
1], group = 1 : si64, kernel_shape = [3, 3], onnx_node_name =
"/conv2/Conv", pads = [1, 1, 1, 1], strides = [1, 1]} :
(tensor<1x32x14x14xf32>, tensor<64x32x3x3xf32>, tensor<64xf32>) ->
tensor<1x64x14x14xf32>
...

%17 = "onnx.Relu"(%16) {onnx_node_name = "/Relu_1"} :
(tensor<1x128xf32>) -> tensor<1x128xf32>
%18 = "onnx.Gemm"(%17, %7, %6) {alpha = 1.000000e+00 : f32, beta =
1.000000e+00 : f32, onnx_node_name = "/fc2/Gemm", transA = 0 : si64,
transB = 1 : si64} : (tensor<1x128xf32>, tensor<10x128xf32>,
tensor<10xf32>) -> tensor<1x10xf32>
onnx.Return %18 : tensor<1x10xf32>

// -----// IR Dump Before (anonymous
namespace)::RecomposeONNXToONNXPass (recompose-onnx) //----- //
func.func @main_graph(%arg0: tensor<1x1x28x28xf32> {onnx.name =
"x.1"}) -> (tensor<1x10xf32> {onnx.name = "19"}) {
...

}
```

For the IR Dump we can know some of the details:

1. Once Onnx is imported, the first IR Dump (mlir dump), the model weight value would be kept in to dense array object.
2. The dense array is denoted as [multidimension x type]
3. The higher dimension is encoded in hexadecimal raw data.

> Now the first goal we encounter is how do we extract value from the dense array object.

Possible way to extract the value from raw data:

discussion

From the log we have the following IR passes:

```
./onnx-mlir /home/sylvex/mnist_export/mnist_model.onnx -mlir-print-ir-before-all 2>&1 | grep Dump > log4.txt
```

```
// from here is onnx dialect
(anonymous namespace)::DecomposeONNXToONNXPass (decompose-onnx)
(anonymous namespace)::RecomposeONNXToONNXPass (recompose-onnx)
(anonymous namespace)::ONNXHybridTransformPass (onnx-hybrid-
transform)
(anonymous namespace)::ConvOptONNXToONNXPass (conv-opt-onnx)
(anonymous namespace)::ONNXHybridTransformPass (onnx-hybrid-
transform)
(anonymous namespace)::SimplifyShapeRelatedOpsPass (simplify-shape-
related-ops-onnx)
(anonymous namespace)::ConstPropONNXToONNXPass (constprop-onnx)
onnx_mlir::(anonymous namespace)::ShapeInferencePass (shape-
inference)
Canonicalizer (canonicalize)
(anonymous namespace)::ConstPropONNXToONNXPass (constprop-onnx)
onnx_mlir::(anonymous namespace)::ShapeInferencePass (shape-
inference)
Canonicalizer (canonicalize)
(anonymous namespace)::ConstPropONNXToONNXPass (constprop-onnx)
onnx_mlir::(anonymous namespace)::ShapeInferencePass (shape-
inference)
Canonicalizer (canonicalize)
(anonymous namespace)::ONNXHybridTransformPass (onnx-hybrid-
transform)
onnx_mlir::(anonymous namespace)::StandardFuncReturnPass (standard-
func-return)
SymbolDCE (symbol-dce)
onnx_mlir::(anonymous namespace)::ScrubDisposablePass (scrub-
disposable)
(anonymous namespace)::SetONNXNodeNamePass (set-onnx-node-name)
onnx_mlir::InstrumentPass (instrument)
CSE (cse)
(anonymous namespace)::ONNXPreKrnlVerifyPass (onnx-pre-krnl-verify)
onnx_mlir::FrontendToKrnlLoweringPass (convert-onnx-to-krnl)

// from here is krnl dialect
Canonicalizer (canonicalize)
onnx_mlir::krnl::ConvertKrnlToAffinePass (convert-krnl-to-affine)
CSE (cse)

// from here we have affine
ConvertVectorToSCF (convert-vector-to-scf)

// from here we have scf
ConvertAffineToStandard (lower-affine)
(anonymous namespace)::LowerKrnlRegionPass (lower-krnl-region)
BufferLoopHoisting (buffer-loop-hoisting)
BufferDeallocation (buffer-deallocation)
FoldMemRefAliasOps (fold-memref-alias-ops)
onnx_mlir::krnl::ConvertKrnlToLLVMPass (convert-krnl-to-llvm)

// now we got the llvm dialect ir
ReconcileUnrealizedCasts (reconcile-unrealized-casts)
Canonicalizer (canonicalize)
```

From the log I can discovered few things:

1. The dense array are kept the same, until lower to llvm dialect ir.
2. Repetitive pass exist.
3. The instructions or operations we might care about like: `%31 = arith.mulf %29, %30 : f32` emerge when it convert to krnl dialect.

We know for a fact:

- onnx dialect is just a collection of onnx ops represent in mlir.
- krnl dialect is higher level of affine dialect. from the paper:

  ```
  krnl dialect provides a representation that is suitable for loop
  optimizations, which is able to carry out affine transformations
  such as tile, skew, and permutation easily. It plays as an
  intermediate dialect for efficiently lowering the onnx dialect
  into low-level dialects (e.g., affine, std and llvm)
  ```

- once we can lower the krnl dialect to affine dialect, the existing affine and llvm optimization can be applied.

> Now the second goal is knowing how does the onnx dialect converted krnl dialect, since it disassble all the onnx ops to loop representation.

MISC:

1. Using GDB to debug MLIR project is impratical.
2. The compiled tool: `<project_name>-mlir` executable only deal with mlir part, the lexer, parser seemed not handled.
3. I still have no idea how the onnx model get parsed to the onnx-dialect.

Future Goals:

1. knowing how do we extract value from the dense array object
2. knowing how does the onnx dialect converted krnl dialect