Using MLIR Linalg Category Ops for Smarter Compilation

Javed Absar

Principal Engineer, Qualcomm Technologies International, Ltd.

Agenda

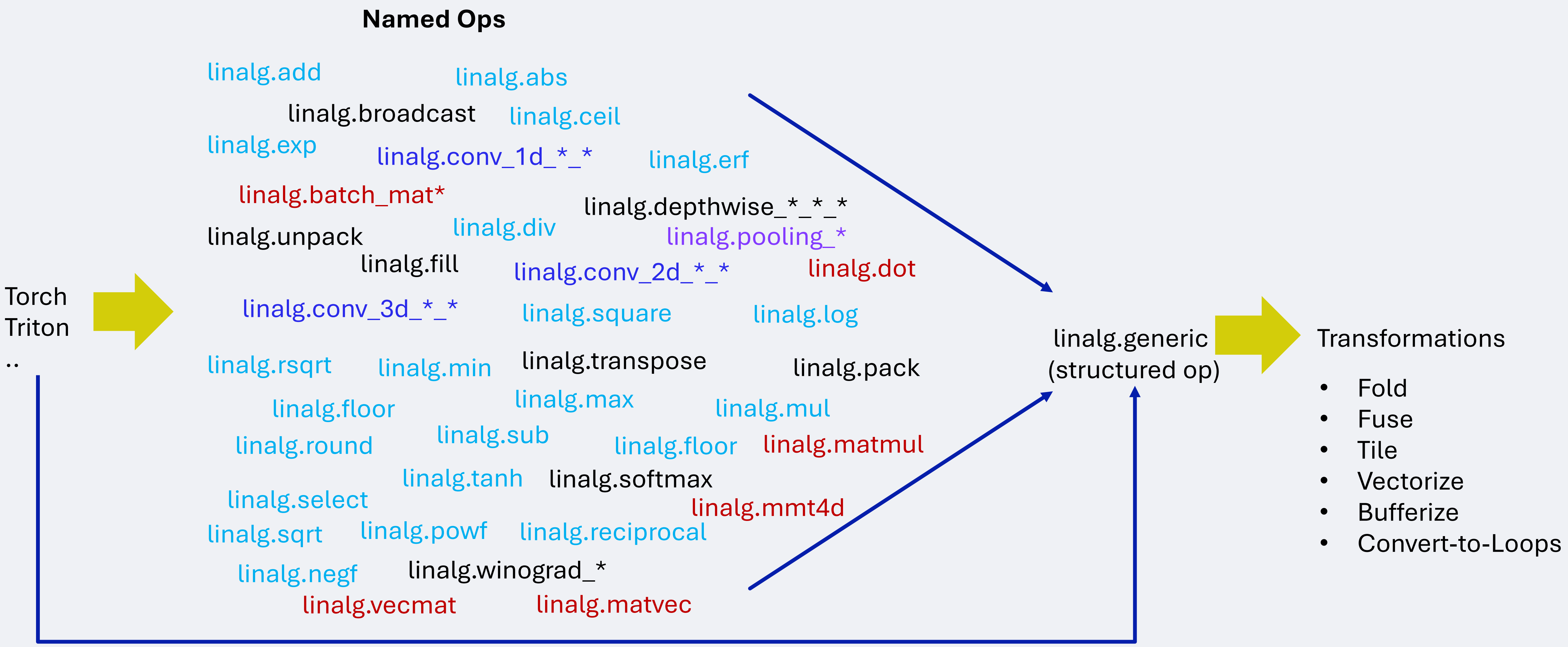
Operators in Linalg Dialect

Categeory Ops – A New Abstraction

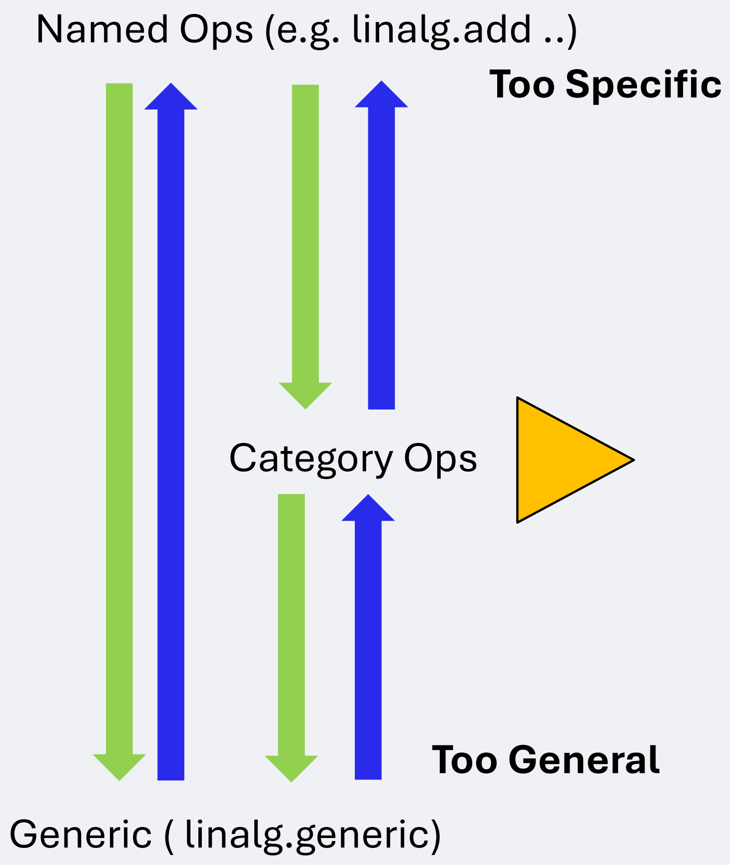
Morphism across Representations

Ease of optimizations

Operators in Linalg Dialect of MLIR

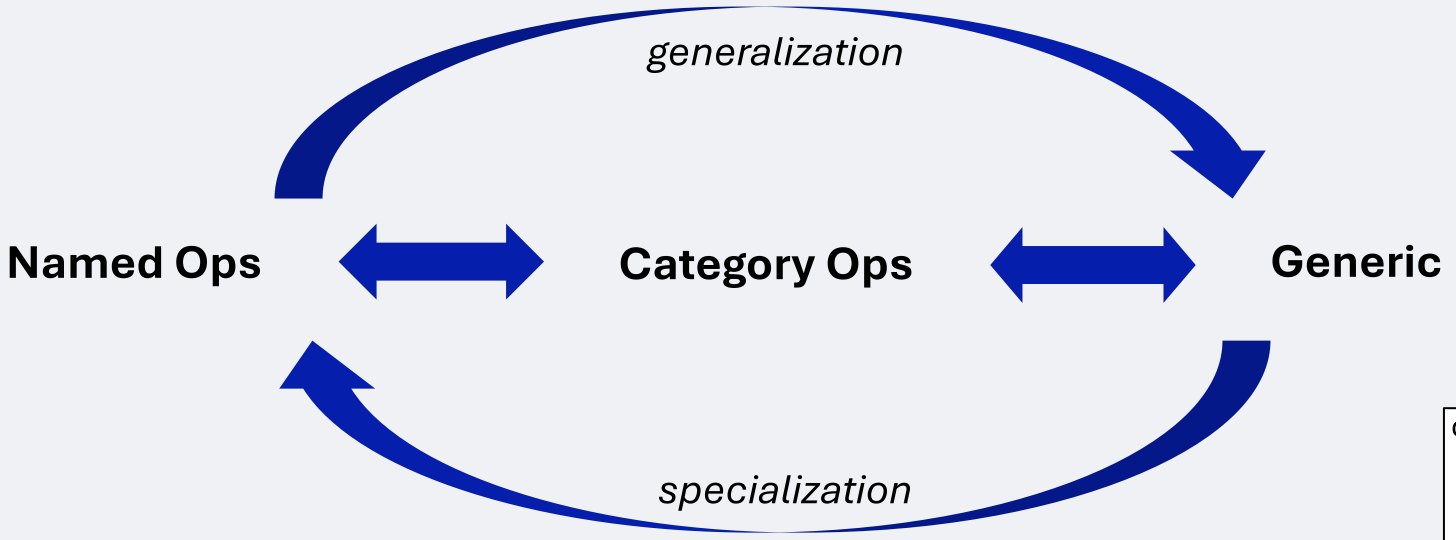


Category Ops - A New Intermediate Abstraction



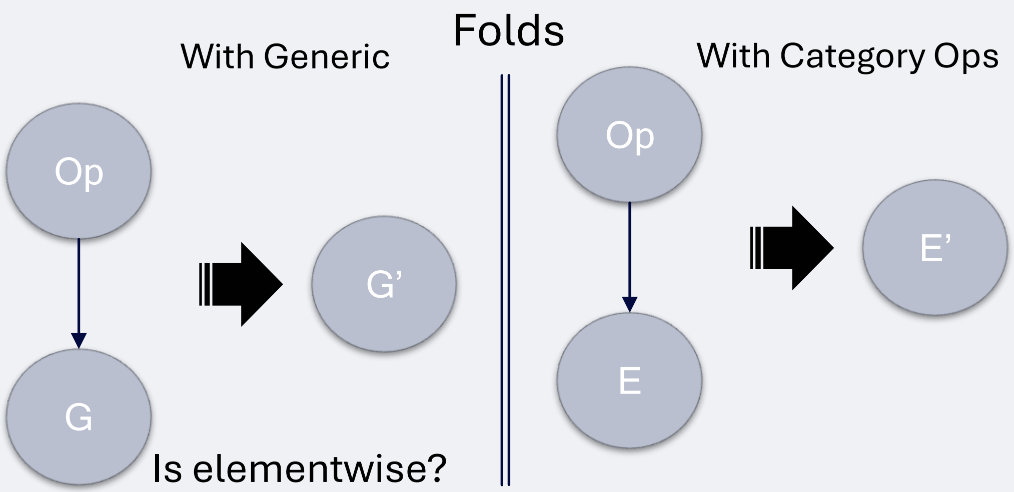

```
generic
 |- contract
 |  |- matmul
 |  |- vecmat
 |  |- matvec
 |  |- batch_...
 |  |- batch_reduce_...
 |  |- quantized_...
 |  |- mmt4d
 |  |- dot
 |- element_wise
 |  |- add
 |  |- max
 |  |- select
 |  |- (all compute unary, binary, ternary)
 |- convolution
 |  |- conv_3d_ncdhw_fcdhw
 |  |- conv_2d_...
 |  |- conv_...
 |  |- depthwise_conv_...
 |- pooling
 |  |- pooling_nchw_max
 |  |- pooling_...
 |- winograd
 |  |- winograd_...
...
```

Morphism across Representations

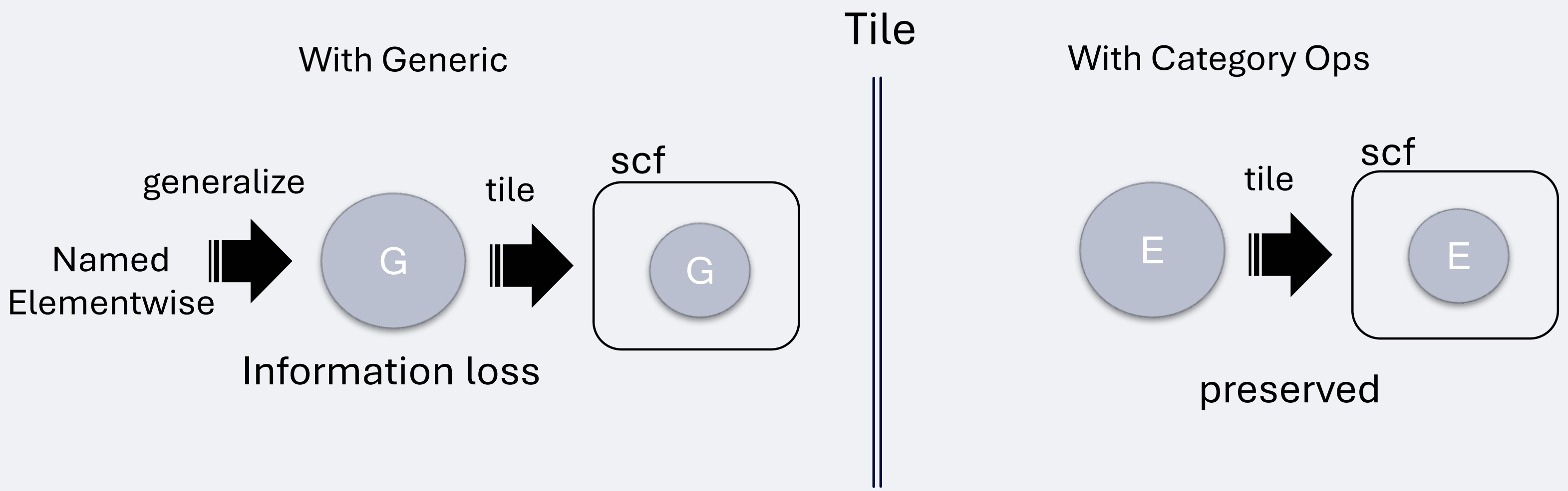



```
categories
|- contract
|  |- ...
|  |- ...
|- element_wise
|  |- ...
|  |- ...
||- convolution
|  |- ..
|  |- ...
|- pooling
|  |- ...
|- ...
```

Ease of Optimizations in the New Abstraction



Ease of Optimizations in the New Abstraction



Example – Morphism ` Named to Generic`

Same computation, different representation

```
func.func @exp(%A : tensor<16x8xf32>, %B : tensor<16x8xf32>) ->  tensor<16x8xf32> {
  %exp = linalg.exp
           ins(%A : tensor<16x8xf32>)
           outs(%B :  tensor<16x8xf32>) -> tensor<16x8xf32>
  return %exp :  tensor<16x8xf32>
}
```

```
$ mlir-opt   -linalg-morph-ops=named-to-generic linalg-morph-category-ops.mlir
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @exp(%arg0: tensor<16x8xf32>,
                 %arg1: tensor<16x8xf32>) -> tensor<16x8xf32> {
    %0 = linalg.generic
           {indexing_maps = [#map, #map],
            iterator_types = ["parallel", "parallel"]}
         ins(%arg0 : tensor<16x8xf32>) outs(%arg1 : tensor<16x8xf32>) {
    ^bb0(%in: f32, %out: f32):
      %1 = math.exp %in : f32
      linalg.yield %1 : f32
    } -> tensor<16x8xf32>
    return %0 : tensor<16x8xf32>
  }
}
```

Example – Morphism `Named to Category`

Same computation, different representation

```
func.func @exp(%A : tensor<16x8xf32>, %B : tensor<16x8xf32>) ->  tensor<16x8xf32> {
  %exp = linalg.exp
           ins(%A : tensor<16x8xf32>)
           outs(%B :  tensor<16x8xf32>) -> tensor<16x8xf32>
  return %exp :  tensor<16x8xf32>
}
```

```
$ mlir-opt -linalg-morph-ops=named-to-category linalg-morph-category-ops.mlir
module {
  func.func @exp(%arg0: tensor<16x8xf32>, %arg1: tensor<16x8xf32>) -> tensor<16x8xf32> {

     %0 = linalg.elementwise kind=#linalg.elementwise_kind<exp>
          ins(%arg0 : tensor<16x8xf32>) outs(%arg1 : tensor<16x8xf32>)
         -> tensor<16x8xf32>
    return %0 : tensor<16x8xf32>
  }
}
```

Example – Folding Transpose, Broadcast

```
identity = affine_map<(d0, d1, d2) -> (d0, d1, d2)>
#map1 = affine_map<(d0, d1, d2) -> (d1, d2)>
func.func @foo(%A: tensor<16xf32>, %B: tensor<8x16x32xf32>) -> tensor<8x16x32xf32> {
  %empty_1 = tensor.empty() : tensor<32x16xf32>
  %empty_2 = tensor.empty() : tensor<16x32xf32>
  %b_A = linalg.broadcast
           ins(%A : tensor<16xf32>)  outs(%empty_1 : tensor<32x16xf32>) dimensions = [0]
  %tb_A = linalg.transpose
           ins(%b_A : tensor<32x16xf32>) outs(%empty_2 : tensor<16x32xf32>) permutation = [1, 0]
  %result = linalg.elementwise
              kind=#linalg.elementwise_kind<exp>
              indexing_maps = [#map1, #identity]
              ins(%tb_A : tensor<16x32xf32>) outs(%B : tensor<8x16x32xf32>) -> tensor<8x16x32xf32>
  return %result : tensor<8x16x32xf32>
}
```

```
#map = affine_map<(d0, d1, d2) -> (d1)>
#map1 = affine_map<(d0, d1, d2) -> (d0, d1, d2)>
module {
  func.func @foo(%arg0: tensor<16xf32>, %arg1: tensor<8x16x32xf32>) -> tensor<8x16x32xf32> {
    %0 = linalg.elementwise
         kind=#linalg.elementwise_kind<exp>
         indexing_maps = [#map, #map1]
         ins(%arg0 : tensor<16xf32>) outs(%arg1 : tensor<8x16x32xf32>) -> tensor<8x16x32xf32>
    return %0 : tensor<8x16x32xf32>
  }
}
```

Conclusion

- Category Ops – a new abstraction in Linalg
- Morphism across the different representations
- Ease and opportunities for optimizations

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog