ShapeTransform Op in onnx-mlir

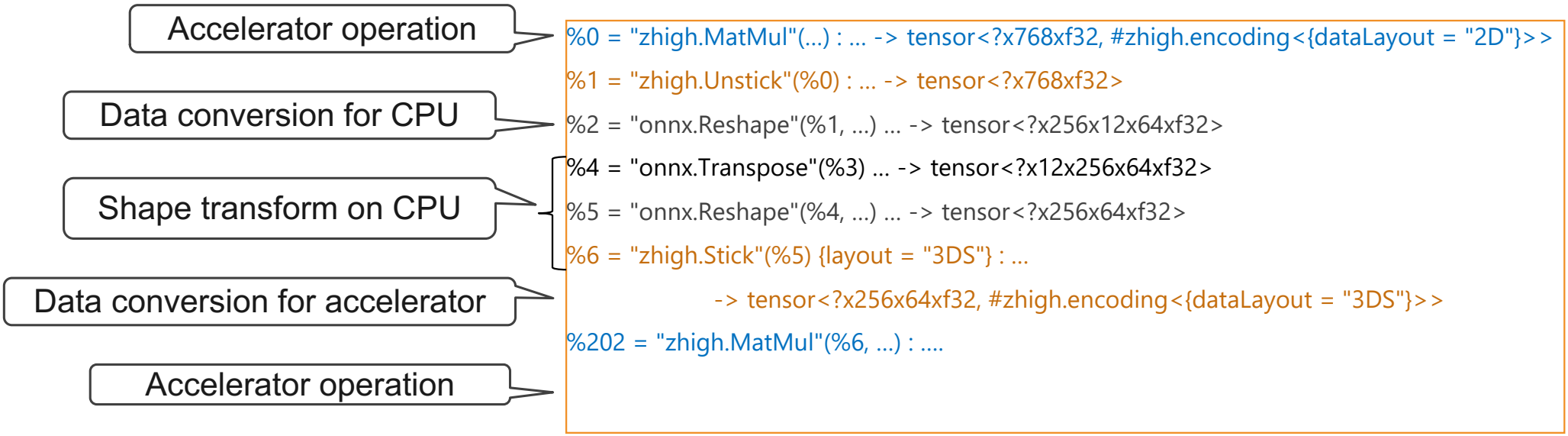
ONNX-MLIR weekly meeting

March 14th, 2023

Tung D. Le, IBM Research - Tokyo

Motivation

- MatMul-Transpose-MatMul pattern in BertSquad
  - How can we fuse all data transformation between accelerator operations?



Introduce onnx.ShapeTransform: PR#2022

```
#reshape = affine_map(d0, d1) -> (d0/32, d0%32, d1/64, d1%64)
%0       = onnx.ShapeTransform(%arg0) {index_map = #reshape} :
                    (tensor<128x128xf32>) -> tensor<4x32x2x64xf32>
```

- onnx.**ShapeTransform** to transform a tensor using an affine map
  - The affine map must be bijective
    - to derive dimensions (or upper bounds) of the output buffer
  - It will be materialized into affine.for/affine.load/affine.store via krnl dialect
  - Currently implementation only supports static output dimensions

```
%alloc = memref.alloc() {alignment = 16 : i64} : memref<4x32x2x64xf32>
affine.for %arg1 = 0 to 128 {
  affine.for %arg2 = 0 to 128 {
    %0 = affine.load %arg0[%arg1, %arg2] : memref< 128x128xf32 >
    affine.store %0, %alloc[%arg1 / 32, %arg1 % 32, %arg2 / 64, %arg2 % 64] : memref<4x32x2x64xf32>
  }
}
```

Introduce onnx.ShapeTransform

- Composable
  - by composing affine maps

```
#transpose = affine_map<(d0, d1, d2, d3) -> (d2, d0, d1, d3)>
#reshape   = affine_map<(d0, d1) -> (d0 / 32, d0 % 32, d1 / 64, d1 % 64)>

%0         = “onnx.ShapeTransform”(%arg0) {index_map = #reshape} :
                          (tensor<128x128xf32>) -> tensor<4x32x2x64xf32>
%1         = “onnx.ShapeTransform”(%0) {index_map = #transpose} :
                          (tensor<4x32x2x64xf32>) -> tensor<2x4x32x64xf32>
```

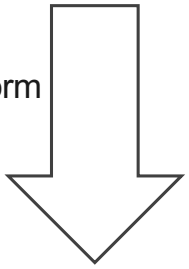
```
#reshape_transpose = affine_map<(d0, d1) -> (d1 / 64, d0 / 32, d0 % 32, d1 % 64)>
                %0 = ”onnx.ShapeTransform"(%arg0) {index_map = #reshape_transpose} :
                                  (tensor<128x128xf32>) -> tensor<2x4x32x64xf32>
```

Apply onnx.ShapeTransform to Bertsquad

```
%1 = "zhigh.Unstick"(%0)  : tensor<1024x768xf32, #encoding<{dataLayout = "2D"}>> -> tensor<1024x768xf32>
%2 = "onnx.Reshape"(%1, ...) ... -> tensor<4x256x12x64xf32>
%3 = "onnx.Transpose"(%2 ... -> tensor<4x12x256x64xf32>
%4 = "onnx.Reshape"(%3, ...) ... -> tensor<48x256x64xf32>
%5 = "zhigh.Stick"(%4) {layout = "3DS"} : tensor<48x256x64xf32> -> tensor<48x256x64xf32, #encoding<{dataLayout = "3DS"}>>
```

```
(d0, d1)[s0, s1] -> (d0/s0, d0%s0, d1/s1, d1%s1)
(d0, d1, d2, d3) -> (d0, d2, d1, d3)
(d0, d1, d2, d3)[s0] -> (d0*s0 + d1, d2, d3)
```

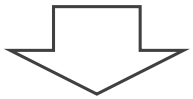
- Replace Reshape, Transpose by ShapeTransform
- Compose ShapeTransform



```
#map = affine_map<(d0, d1) -> ((d0 / 256) * 12 + d1 / 64, d0 % 256, d1 / 64)>
%1 = "zhigh.Unstick"(%0) : tensor<1024x768xf32, #encoding<{dataLayout = "2D"}>> -> tensor<1024x768xf32>
%2 = "onnx.ShapeTransform"(%1) {index_map = #map} : tensor<48x256x64xf32> -> tensor<48x256x64xf32>
%3 = "zhigh.Stick"(%2) {layout = "3DS"} : -> tensor<48x256x64xf32> -> tensor<48x256x64xf32, #encoding<{dataLayout = "3DS"}>>
```

Apply onnx.ShapeTransform to Bertsquad

```
#map = affine_map<(d0, d1) -> ((d0 / 256) * 12 + d1 / 64, d0 % 256, d1 / 64)>
%1 = "zhigh.Unstick"(%0) : tensor<1024x768xf32, #encoding<{dataLayout = ”2D"}> -> tensor<1024x768xf32>
%2 = ”onnx.ShapeTransform"(%1) {index_map = #map} : tensor<48x256x64xf32> -> tensor<48x256x64xf32>
%3 = "zhigh.Stick"(%2) {layout = "3DS"} : -> tensor<48x256x64xf32> -> tensor<48x256x64xf32, #encoding<{dataLayout = "3DS"}>>
```



```
#map = affine_map<(d0, d1) -> ((d0 / 256) * 12 + d1 / 64, d0 % 256, d1 / 64)>
%1 = ”onnx.ShapeTransform"(%0) {index_map = #map} :
        tensor<1024x768xf32, #encoding<{dataLayout = ”2D"}>>
    -> tensor<48x256x64xf32, #encoding<{dataLayout = "3DS"}>>
```

Conclusion

- onnx.ShapeTransform can be used for
  - onnx.Reshape
  - onnx.Transpose
  - onnx.Squeeze
  - onnx.Unsqueeze

- Future work
  - onnx.ShapeTransform for dynamic dimensions
    - How to derive upper bounds from affine_map? It is quite complicated or non-trivial in current MLIR.
  - Transformation for onnx.Concat, onnx.Split, onnx.Slice, etc.
    - affine_map is injective