Sharding Framework Design for Device Mesh

Chengji Yao

ByteDance AI Compiler Team

Motivation

Could you exact two key single-word terms from the following title:

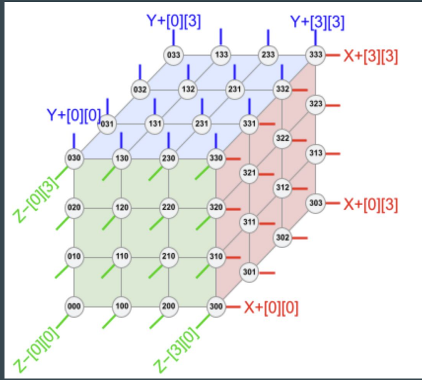
Sharding Framework Design for Device Mesh

Sure. The two key single-word terms I would exact from the title "Sharding Framework Design for Device Mesh" are:

- Sharding
- Mesh

Motivation

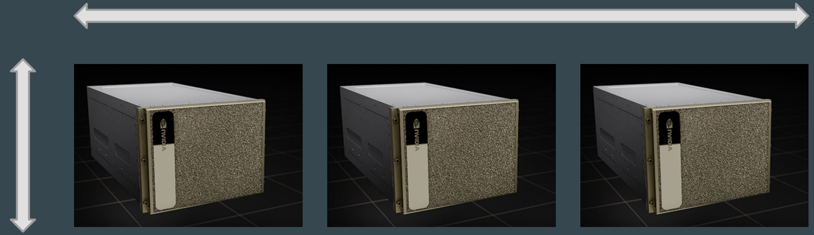
- ❖ Sharding infrastructure is imperative for LLM
- ❖ To represent mesh-like cluster
- ❖ To represent distributed tensor
- ❖ To represent communications on mesh cluster



TPU pod

axis 1

axis 0



DGX * N

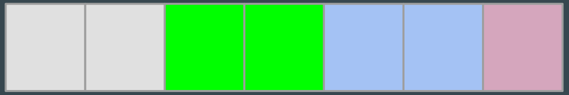
Agenda

- Mesh Dialect
  - Mesh definition
  - mesh.cluster op
  - MeshShardingAttr
  - mesh.annotate op
  - ccl ops in mesh
  - other mesh ops
- Typical Workflow in this Sharding Framework
  - sharding propagation
  - sharding materialization
  - Mesh CCL optimization
  - analysis and sharding mutations (optional)
  - sharding partition

Mesh Definition

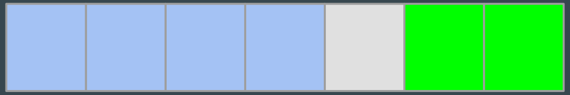
- Not to express the connection of devices in physical cluster
- Is a tool to express sharding in a simplified yet constrained manner.

1D tensor with 7 elements sharded by
1D mesh cluster with 4 devices



Only one sharding result

1D tensor with 7 elements sharded by
a general cluster with 4 devices



Sharding result is arbitrary

Different colors means different devices

mesh.cluster

Is a SymbolOp placed within a ModuleOp

Mesh with 2 axes

```
mesh.cluster @mesh0(rank = 2, dim_sizes = [4, 8])
```

Number of devices along axis 0 is 4
Number of devices along axis 1 is 8
Total device number is 32

```
mesh.cluster @mesh1(rank = 3, dim_sizes = [0, 4])
```

Numbers of devices along axis 0 and 2 are dynamic
Number of devices along axis 1 is 4
Total device number is dynamic

MeshShardingAttr

- An attribute type used to convert a standard tensor into a distributed one.
- Employed in the encoding of a RankedTensorType or used in mesh.annotate
- Not designed to be accurate
- Currently, contains a SymbolAttr and ArrayAttr (array of int array)

@mesh0 has 4 devices with ids:

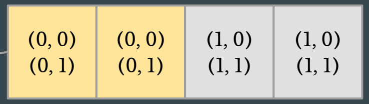
(0, 0), (0, 1)
(1, 0), (1, 1)

Axis 0

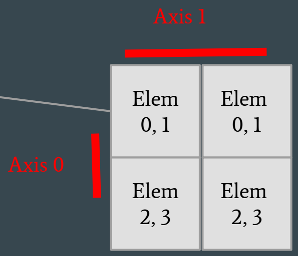
Axis 1

```
mesh.cluster @mesh0(rank = 2, dim_sizes = [2, 2])

func.func @foo(...) -> (...) {

  … = … : tensor<4xf32, #mesh.shard<@mesh0, [[0]]>

}
```

@mesh0



Tensor View



Device View

Sharded along axis 0 and replicated along axis 1

MeshShardingAttr

```
mesh.cluster @mesh1(rank = 3, dim_sizes = [2, 2, 2])

func.func @bar(...) -> (...)

     attributes { mesh_cluster = @mesh1 } {


  … = … : tensor<4x8xf32, #mesh.shard<[[2, 0], [], [1]]>

}
```

Use the default mesh defined in the func op's attrs

Sharded the 1st dim along axis 2 and 0, the order matters. Partial-sum along axis 1

mesh.annotate

Alternative way for holding MeshShardingAttr.

Useful because the encoding in RankedTensorType might be discarded during some transforms.

```
%0 = op0 ...
%1 = mesh.annotate %0 {#mesh.shard<[[0], [1]]>,
          required = false, as_result = true} :
          tensor<2x5xf32> -> tensor<2x5xf32>
%2 = op1(%1) : ...
```

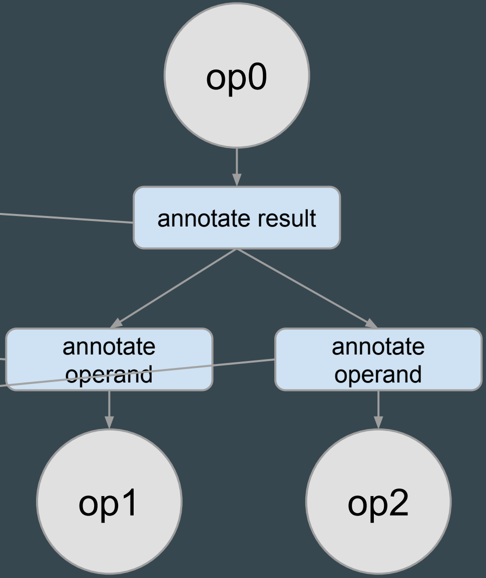
Hold MeshShardingAttr

true: mandated
false: used as hint

Operand and result
are standard tensor

mesh.annotate

```
%0 = op0 ...
%1 = mesh.annotate %0 {#mesh.shard<[[0], [1]]>, required = true,
        as_result = true} : tensor<2x5xf32> -> tensor<2x5xf32>
%2 = mesh.annotate %1 {#mesh.shard<[[0]]>, required = true,
        as_result = false} : tensor<2x5xf32> -> tensor<2x5xf32>
%3 = op1(%2) : ...
%4 = mesh.annotate %1 {#mesh.shard<[[1]]>, required = true,
        as_result = false} : tensor<2x5xf32> -> tensor<2x5xf32>
%5 = op2(%4) : ...
```



CCL ops in mesh

- Use mesh axis to specify groups instead of using device ids.
- Ease the comparing of replica groups between ccl ops.

All-gather will be applied along mesh axis 1 on tensor dim 1

Operands and results are distributed tensor

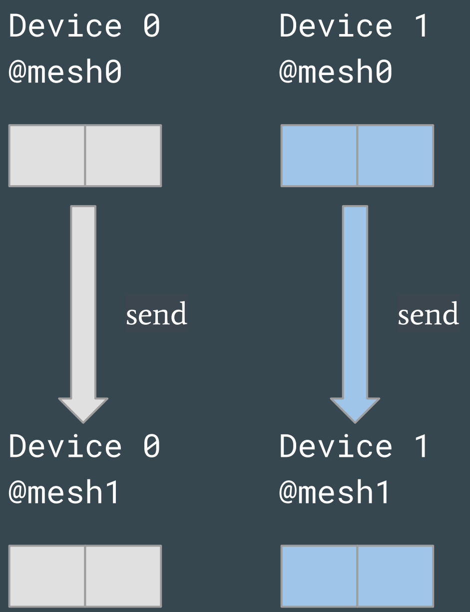
```
%1 = mesh.all_gather %0 {mesh_axis = [[], [1]]} :
  tensor<2x4xf32, #mesh.shard<[[], [1, 0]]>> ->
  tensor<2x4xf32, #mesh.shard<[[], [0]]>>
```

mesh.all_reduce, mesh.reduce_scatter, mesh.all_to_all are similar

CCL ops in mesh

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])
mesh.cluster @mesh1(rank = 1, dim_sizes = [2])


%1 = mesh.collective_permute %0 :
  tensor<4xf32, #mesh.shard<@mesh0, [[0]]>> ->
  tensor<4xf32, #mesh.shard<@mesh1, [[0]]>>
```

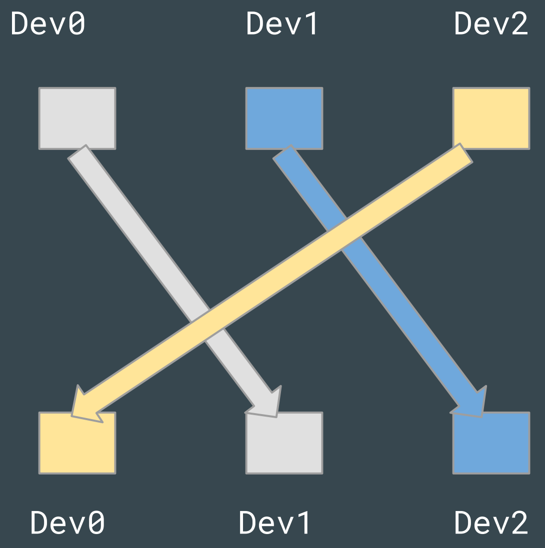
Useful in pipeline parallel



CCL ops in mesh

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [3])

%1 = mesh.collective_permute %0 {mesh_axis = 0} :
  tensor<3xf32, #mesh.shard<@mesh0, [[0]]>> ->
  tensor<3xf32, #mesh.shard<@mesh0, [[0]]>>
```

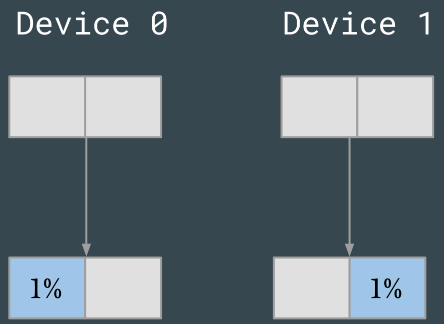
Useful for halo exchange of conv op sharding



CCL ops in mesh

Extract a smaller tensor locally from a larger one.

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])


%1 = mesh.local_split %0 {sharding = [[0]]} :
    tensor<2xf32, #mesh.shard<[]>> ->
    tensor<2xf32, #mesh.shard<[[0]]>>
```




```
mesh.cluster @mesh1(rank = 2, dim_sizes = [2, 2])

%1 = mesh.local_split %0 {sharding = [[], [0]]} :
    tensor<2x4xf32, #mesh.shard<[[1], []]>> ->
    tensor<2x4xf32, #mesh.shard<[[1], [0]]>>
```

The operand could also be an already sharded tensor

Other Mesh Ops

`mesh.idx` ← Get the index of current device along specified mesh axis.

`mesh.size` ← Get the device number along specified mesh axis.

`mesh.sub_cluster` ← extract a sub-cluster from a given mesh.

Intermediate Consolidation

```
%0 = ... : tensor<2x4xf32>

%1 = mesh.annotate %0 {#mesh.shard<[[], [0, 1]]>, as_result =
true} : tensor<2x4xf32> -> tensor<2x4xf32>

%2 = mesh.annotate %1 {#mesh.shard<[[]], [0]]>, as_result =
false} : tensor<2x4xf32> -> tensor<2x4xf32>

... = "use"(%2) : ...
```

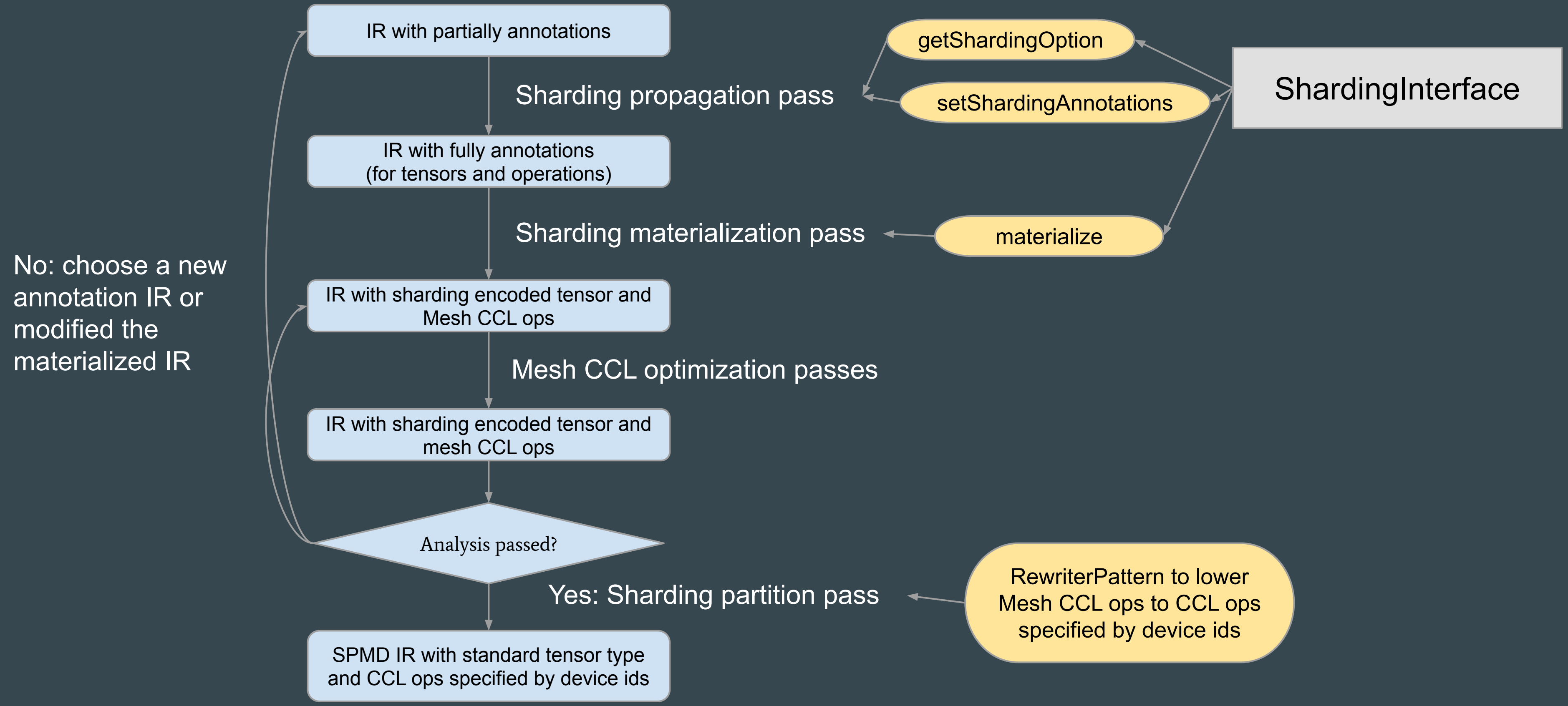
Materialization

```
%0 = ... : tensor<2x4xf32, #mesh.shard<[[], [0]]>>

%1 = mesh.all_gather %0 {mesh_axis = [[], [1]]} :

  tensor<2x4xf32, #mesh.shard<[[], [0, 1]]>> ->

  tensor<2x4xf32, #mesh.shard<[[], [0]]>>

... = "use"(%1) : ...
```

Agenda

- Mesh Dialect
  - mesh.cluster op
  - MeshShardingAttr
  - mesh.annotate op
  - ccl ops in mesh
  - other mesh ops
- Typical Workflow in this Sharding Framework
  - sharding propagation
  - sharding materialization
  - Mesh CCL optimization
  - analysis and sharding mutations (optional)
  - sharding partition

Flow Chart


IR with partially annotations
Sharding propagation pass
getShardingOption
setShardingAnnotations
ShardingInterface
IR with fully annotations
(for tensors and operations)
Sharding materialization pass
materialize
No: choose a new annotation IR or modified the materialized IR
IR with sharding encoded tensor and Mesh CCL ops
Mesh CCL optimization passes
IR with sharding encoded tensor and mesh CCL ops
Analysis passed?
Yes: Sharding partition pass
RewriterPattern to lower Mesh CCL ops to CCL ops specified by device ids
SPMD IR with standard tensor type and CCL ops specified by device ids

Before ShardingInterface

- ShardingIteratorType Enum
  - Similar to IteratorType enum used for tiling interface
  - Currently has 3 values:
    - parallel
    - reduction_sum
    - invalid
- ShardingOption
  - Used as an additional attr in an op
  - Similar to tiling option used for tiling interface
  - Specify mesh axis for each loop's sharding
  - Describe the sharding of an op more precisely than MeshShardingAttr

```
%1, %loops = transform.structured.tile %0 [0, 8, 0] :
…
```

```
%0 = "mhlo.dot"(%lhs, %rhs) {sharding = [[], [], [0]]} :
    (tensor<2x4xf32>, tensor<4x8xf32>) -> tensor<2x8xf32>
```

ShardingInterface methods

- getLoopIteratorTypes
- getIndexingMaps
- getShardingOption (default implementation)
- setShardingAnnotations (default implementation)
- materialize (default implementation)

Get the sharding option of the op from certain annotated operands/results

Not to yield the optimal sharding result
Logic as simple as possible

Complete the non-annotated operands/result with given sharding option

- convert mesh annotate op to distributed tensor
- communication op will be added as necessary

Sharding Propagation

Complete the sharding annotation for all the operations and function arguments.

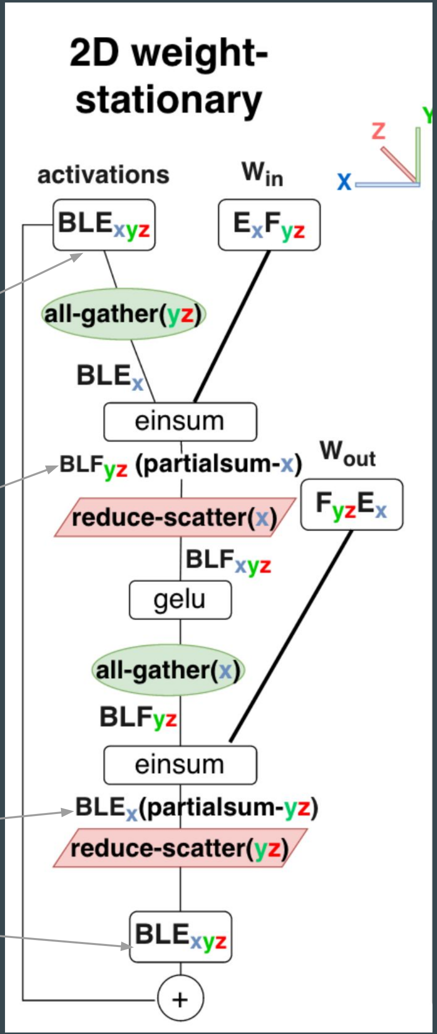
Use [Efficiently Scaling Transformer Inference]()
Fig 2(b) as example

Annotate [[], [], [0, 1, 2]]

Annotate [[], [], [1, 2], [0]]

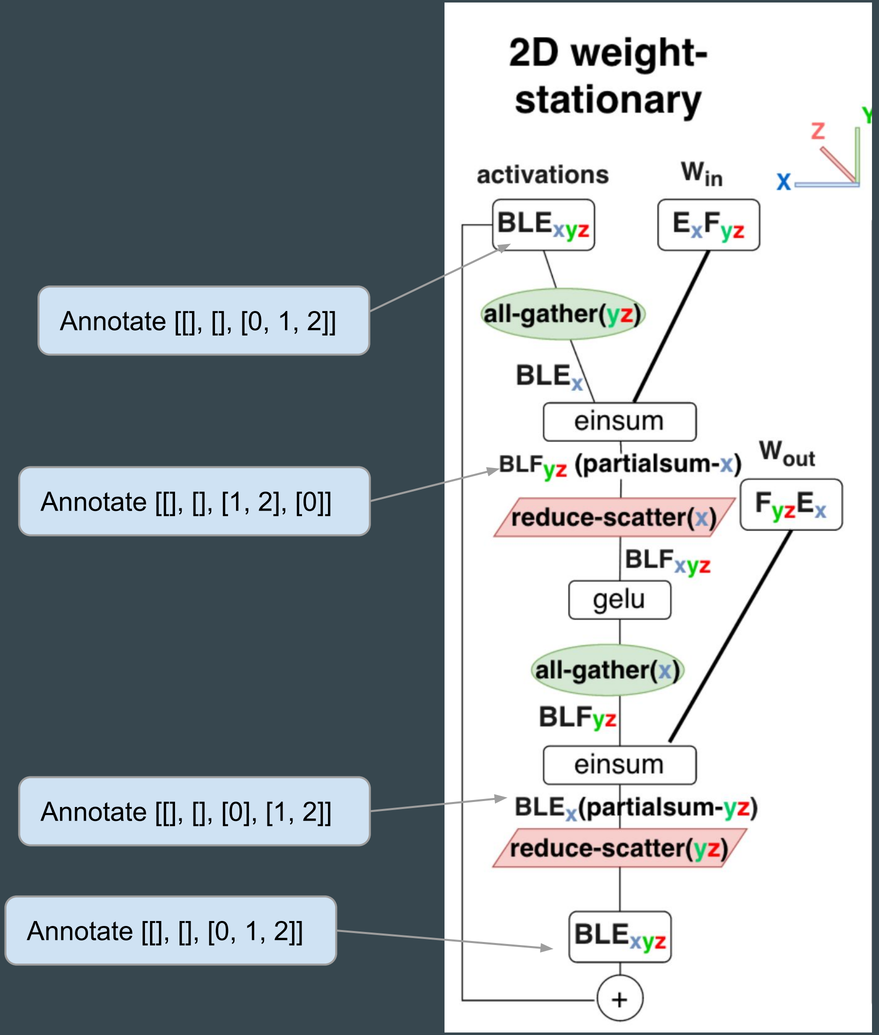
Annotate [[], [], [0], [1, 2]]

Annotate [[], [], [0, 1, 2]]



Sharding Propagation

```
mesh.cluster @mesh0(rank = 3, dim_sizes = [2, 2, 2])


func.func @mlp_2d_weight_stationary(%arg0: tensor<2x4x8xf32>,
                                    %arg1: tensor<8x32xf32>,
                                    %arg2: tensor<32x8xf32>) ->
                    tensor<2x4x8xf32> attributes {mesh_cluster = @mesh0} {
  %0 = mesh.annotate %arg0 {#mesh.shard<[[], [], [0, 1, 2]]>} : tensor<2x4x8xf32>
  %1 = "mhlo.dot_general"(%0, %arg1) {...} :
    (tensor<2x4x8xf32>, tensor<8x32xf32>) -> tensor<2x4x32xf32>
  %2 = mesh.annotate %1 {#mesh.shard<[[], [], [1, 2], [0]]>} : tensor<2x4x32xf32>
  %3 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
  %4 = mhlo.maximum %2, %3 : tensor<2x4x32xf32>
  %5 = "mhlo.dot_general"(%4, %arg2) {...} :
    (tensor<2x4x32xf32>, tensor<32x8xf32>) -> tensor<2x4x8xf32>
  %6 = mesh.annotate %5 {#mesh.shard<[[], [], [0], [1, 2]]>} : tensor<2x4x8xf32>
  %7 = mesh.annotate %6 {as_result = false, #mesh.shard<[[], [], [0, 1, 2]]>} : …
  return %7 : tensor<2x4x8xf32>
}
```



Sharding Propagation

```
mesh.cluster @mesh0(rank = 3, dim_sizes = [2, 2, 2])


func.func @mlp_2d_weight_stationary(%arg0: tensor<2x4x8xf32>,
                                    %arg1: tensor<8x32xf32>,
                                    %arg2: tensor<32x8xf32>) ->
                    tensor<2x4x8xf32> attributes {mesh_cluster = @mesh0} {
  %0 = mesh.annotate %arg0 {#mesh.shard<[[], [], [0, 1, 2]]>} : tensor<2x4x8xf32>
  %1 = "mhlo.dot_general"(%0, %arg1) {..., sharding = [[], [], [1, 2], [0]]} :
    (tensor<2x4x8xf32>, tensor<8x32xf32>) -> tensor<2x4x32xf32>
  %2 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
  %3 = mhlo.maximum %1, %2 : tensor<2x4x32xf32>
  %4 = "mhlo.dot_general"(%3, %arg2) {..., sharding = [[], [], [0], [1, 2]]} :
    (tensor<2x4x32xf32>, tensor<32x8xf32>) -> tensor<2x4x8xf32>
  %5 = mesh.annotate %4 {as_result = false, #mesh.shard<[[], [], [0, 1, 2]]>} : …
  return %5 : tensor<2x4x8xf32>
}
```

(B, L, F) = (B, L, E) @ (E, F)

loop types: [parallel parallel parallel reduction_sum]

indexing maps:
(B, L, F, E) -> (B, L, E)
(B, L, F, E) -> (E, F)
(B, L, F, E) -> (B, L, F)

Sharding Propagation

```
func.func @mlp_2d_weight_stationary(...) -> tensor<2x4x8xf32> {
  %0 = mesh.annotate %arg1 {as_result = false,#mesh.shard<[[0], [1, 2]]>} : tensor<8x32xf32>
  %1 = mesh.annotate %arg2 {as_result = false,#mesh.shard<[[1, 2], [0]]>} : tensor<32x8xf32>
  %2 = mesh.annotate %arg0 {#mesh.shard<[[], [], [0, 1, 2]]} : tensor<2x4x8xf32>
  %3 = mesh.annotate %2 {as_result = false,#mesh.shard<[[], [], [0]]>} : tensor<2x4x8xf32>
  %4 = "mhlo.dot_general"(%3, %0) {..., sharding = [[], [], [1, 2], [0]]} : …
  %5 = mesh.annotate %4 {#mesh.shard<[[], [], [1, 2], [0]]>} : tensor<2x4x32xf32>
  %6 = mesh.annotate %5 {as_result = false,#mesh.shard<[[], [], [1, 2]]>} : tensor<2x4x32xf32>
  %7 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
  %8 = mesh.annotate %7 {required = false,#mesh.shard<[]>} : tensor<2x4x32xf32>
  %9 = mesh.annotate %8 {as_result = false,#mesh.shard<[[], [], [1, 2]]>} : tensor<2x4x32xf32>
  %10 = mhlo.maximum %6, %9 {sharding = [[], [], [1, 2]]} : tensor<2x4x32xf32>
  %11 = mesh.annotate %10 {required = false, #mesh.shard<[[], [], [1, 2]]>} : tensor<2x4x32xf32>
  %12 = mesh.annotate %11 {as_result = false,#mesh.shard<[[], [], [1, 2]]>} : tensor<2x4x32xf32>
  %13 = "mhlo.dot_general"(%12, %1) {..., sharding = [[], [], [0], [1, 2]]} : …
  %14 = mesh.annotate %13 {#mesh.shard<[[], [], [0], [1, 2]]>} : tensor<2x4x8xf32>
  %15 = mesh.annotate %14 {as_result = false,#mesh.shard<[[], [], [0, 1, 2]]>} : tensor<2x4x8xf32>
  return %15 : tensor<2x4x8xf32>
}
```

Both of the partial sharding IR will result in the same result IR

All the operands and results are annotated

Sharding option is added in the op

Sharding Materialization

- Erase mesh.annotate op
- Convert the standard tensor to distributed tensors
- Insert concrete communication between and within operations.

Created from annotate op

Created from different sharding

```
func.func @mlp_2d_weight_stationary(%arg0: tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>, %arg1: tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>,
                   %arg2: tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> attributes {mesh_cluster = @mesh0} {
    %0 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
    %1 = mesh.all_gather %arg0 {mesh_axis = [[], [], [1, 2]]} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> ->
                                                                                     tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>
    %2 = "mhlo.dot_general"(%1, %arg1) {..., sharding = [[], [], [1, 2], [0]]} : (tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>, tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>)
                                                                                    -> tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>>
    %3 = mesh.all_reduce %2 {mesh_axis = [0], reduction = "sum"} : tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>> -> tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>
    %4 = mesh.local_split %0 {sharding = [[], [], [1, 2]]} : tensor<2x4x32xf32> -> tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>
    %5 = mhlo.maximum %3, %4 {sharding = [[], [], [1, 2]]} : tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>
    %6 = "mhlo.dot_general"(%5, %arg2) {..., sharding = [[], [], [0], [1, 2]]} : (tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>,
        tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>>
    %7 = mesh.reduce_scatter %6 {mesh_axis = [1, 2], reduction = "sum", tensor_axis = 2 : i64} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>> ->
                                                                                                    tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    return %7 : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
  }
```

Mesh CCL Optimization

E.g. All-Reduce decompose & All-Gather move down

```
func.func @mlp_2d_weight_stationary(%arg0: tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>, %arg1: tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>,
        %arg2: tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> attributes {mesh_cluster = @mesh0} {
    %0 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
    %1 = mesh.all_gather %arg0 {mesh_axis = [[], [], [1, 2]]} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> ->
                                                                 tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>
    %2 = "mhlo.dot_general"(%1, %arg1) {..., sharding = [[], [], [1, 2], [0]]} : (tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>, tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>)
                                                                                  -> tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>>
    %3 = mesh.reduce_scatter %2 {mesh_axis = [0], reduction = "sum", tensor_axis = 1 : i64} : tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>> ->
                                                                                               tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %4 = mesh.local_split %0 : tensor<2x4x32xf32> -> tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %5 = mhlo.maximum %3, %4 {sharding = [[], [], [0, 1, 2]]} : tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %6 = mesh.all_gather %5 {mesh_axis = [[], [], [0]]} : tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>> ->
                                                          tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>
    %7 = "mhlo.dot_general"(%6, %arg2) {..., sharding = [[], [], [0], [1, 2]]} : (tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>,
                tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>>
    %8 = mesh.reduce_scatter %7 {mesh_axis = [1, 2], reduction = "sum", tensor_axis = 2 : i64} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>> ->
                                                                                                  tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    return %8 : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
}
```

Sharding Analysis

E.g.

- Estimate memory usage
- Get redundant computation
- Get communication volume
- Estimate performance gain from communication / computation overlap

Sharding Analysis

Estimate memory usage

Get Communication Volume

No redundant computation

```
func.func @mlp_2d_weight_stationary(%arg0: tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>, %arg1: tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>,
        %arg2: tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> attributes {mesh_cluster = @mesh0} {
    %0 = mhlo.constant dense<0.000000e+00> : tensor<2x4x32xf32>
    %1 = mesh.all_gather %arg0 {mesh_axis = [[], [], [1, 2]]} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>> ->
                                                                tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>
    %2 = "mhlo.dot_general"(%1, %arg1) {..., sharding = [[], [], [1, 2], [0]]} : (tensor<2x4x8xf32, #mesh.shard<[[], [], [0]]>>, tensor<8x32xf32, #mesh.shard<[[0], [1, 2]]>>)
                                                                                -> tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>>
    %3 = mesh.reduce_scatter %2 {mesh_axis = [0], reduction = "sum", tensor_axis = 1 : i64} : tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2], [0]]>> ->
                                                                                             tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %4 = mesh.local_split %0 : tensor<2x4x32xf32> -> tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %5 = mhlo.maximum %3, %4 {sharding = [[], [], [0, 1, 2]]} : tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    %6 = mesh.all_gather %5 {mesh_axis = [[], [], [0]]} : tensor<2x4x32xf32, #mesh.shard<[[], [], [0, 1, 2]]>> ->
                                                          tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>
    %7 = "mhlo.dot_general"(%6, %arg2) {..., sharding = [[], [], [0], [1, 2]]} : (tensor<2x4x32xf32, #mesh.shard<[[], [], [1, 2]]>>,
             tensor<32x8xf32, #mesh.shard<[[1, 2], [0]]>>) -> tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>>
    %8 = mesh.reduce_scatter %7 {mesh_axis = [1, 2], reduction = "sum", tensor_axis = 2 : i64} : tensor<2x4x8xf32, #mesh.shard<[[], [], [0], [1, 2]]>> ->
                                                                                                 tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
    return %8 : tensor<2x4x8xf32, #mesh.shard<[[], [], [0, 1, 2]]>>
}
```

Sharding Partition

- Create the SPMD IR
- Converts distributed tensors into standard tensors with smaller shape for each device
- Converts mesh CCL operations into more defined CCL ops with device IDs

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])

Func.func @foo(%arg0: tensor<8xf32, #mesh.shard<[[0]]>>) -> ()
          attributes { mesh_cluster = @mesh0 } {
  "use"(%arg0) ...
  ...
}
```

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])

Func.func @foo(%arg0: tensor<8xf32>) -> ()
    attributes { mesh_cluster = @mesh0 } {
  %idx = mesh.idx(0)
  %c4 = arith.constant 4 : i64
  %start = arith.muli %idx, %c4 : i64
  %arg0_slice = "mhlo.dynamic_slice"(%arg0, %start) {
    slice_sizes = dense<[4]> : tensor<1xi64>
  } : (tensor<8xf32>, i64) -> tensor<4xf32>
  "use"(%arg0_slice) ...
  ...
}
```

Sharding Partition

- Alternative of partition result if the physical tensor is already sharded

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])

func.func @foo(%arg0: tensor<8xf32, #mesh.shard<[[0]]>>) -> ()
          attributes { mesh_cluster = @mesh0 } {
  "use"(%arg0) ...
  ...
}
```

```
mesh.cluster @mesh0(rank = 1, dim_sizes = [2])

func.func(%arg0: tensor<4xf32>) -> ()
        attributes { mesh_cluster = @mesh0 } {
  "use"(%arg0) ...
  ...
}
```

Questions