GmlSt : Tiling and Fusion for XLA Next

@frgossen
OpenXLA Community Meeting
10/18/2022

Tiling and fusion

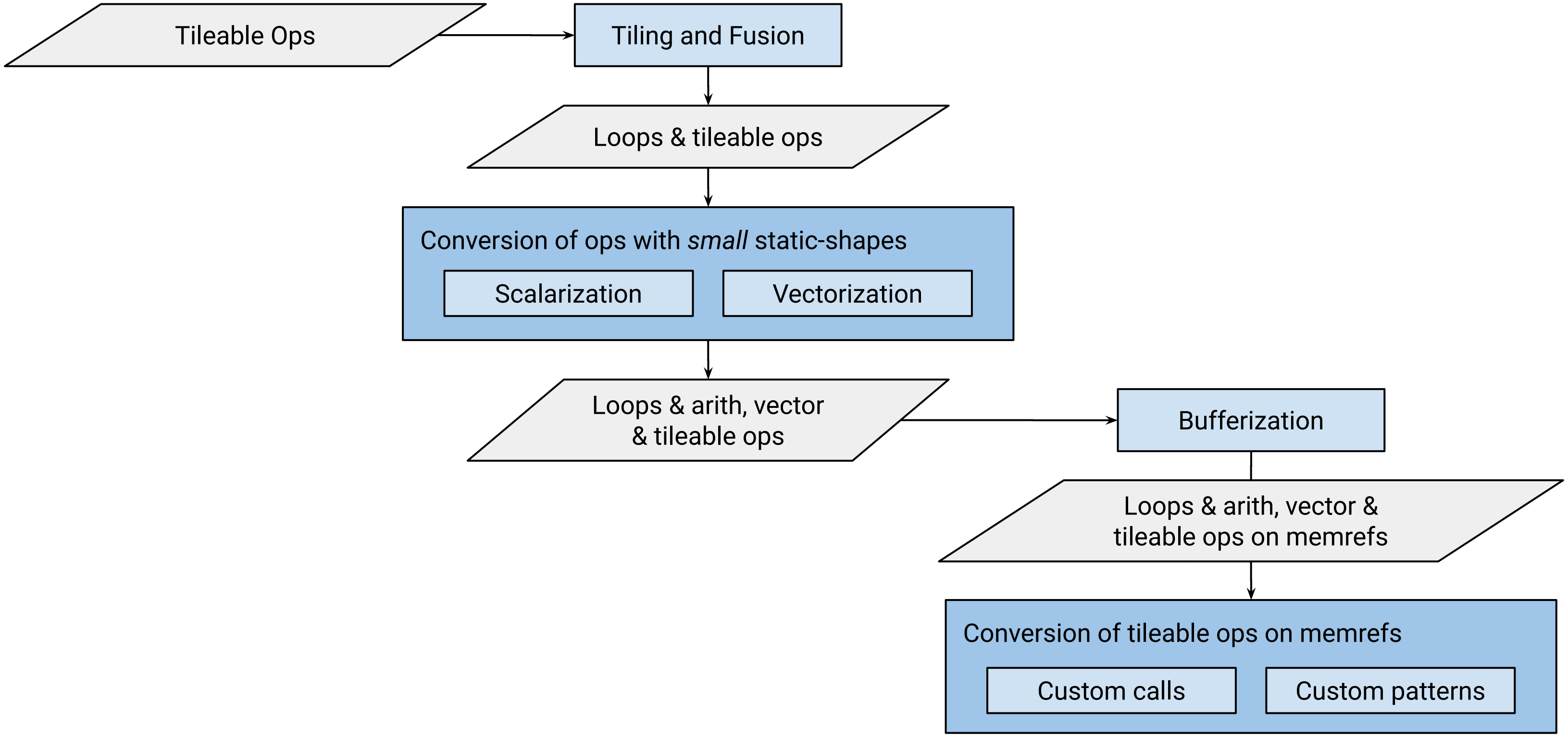
Goal. Break operation down to tiles to …

- … make better use of the cache
- … map nicely to device-specific concepts,
  e.g. blocks/warps (GPU), vector instr. (CPU)

Tiling. Break an operation down to subsets/tiles and process them separately.

Fusion. Pull producers of the inputs for the loop into a tiled implementation.

Pipeline Overview



GmlSt Design Goals

- **Nested tiling.** Tile to multiple levels, e.g. blocks, warps, threads.
- **Dynamism.** Tile & fuse dynamic ops, e.g. bcast, concat, gather, and scatter.
- **Compose** with existing ops and tiling, e.g. fuse into existing loops.
- **Subsets.** Be ready for subset types beyond tiles.

GmlSt Characteristics

- Works on destination-style IR

```
%init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
%ab = linalg.map "add" ins(%a, %b) outs(%init)
```

- Separates **subset computation** and **subset materialization**

```
%space = gml_st.space [1024, 512] : !gml_st.tile<1024x512>
%t0 = gml_st.tile %space [%i, %j] [32, 8] [1, 1] : !gml_st.tile<32x8>
%lhs_sub = gml_st.materialize %lhs[%t0] : tensor<32x8xf32>
```

- Pack subsets and pass them as one value
  - More readable IR (subjective), flexible wrt. future subset types

Examples

Examples

- **Tiling and fusing element-wise ops**
- **Concat**
- Reduction
- Nested tiling

Tile & Fuse Element-wise

Tiling and Fusing Element-wise Ops I

```
func.func @cwise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>, %c: tensor<?x?xf32>)
    -> tensor<?x?xf32> {

  %ab = mhlo.add %a, %b : tensor<?x?xf32>
  %abc = mhlo.multiply %ab, %c : tensor<?x?xf32>
  func.return %abc
}
```

Tiling and Fusing Element-wise Ops II

```
func.func @cwise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>, %c: tensor<?x?xf32>)
    -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  %abc = linalg.map "mul" ins(%ab, %c) outs(%init)
  func.return %abc
}
```

Tiling and Fusing Element-wise Ops III

```
func.func @cwise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>, %c: tensor<?x?xf32>)
    -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  %abc = gml_st.parallel (%i, %j) = (%c0, %c0) to (%d0, %d1) step (%c4, %c8) {
    %space = gml_st.space [%d0, %d1] : !gml_st.tile<?x?>
    %tile = gml_st.tile %space [%i, %j] [%td0, %td1] [1, 1] : !gml_st.tile<?x?>

    %ab_sub = gml_st.materialize %ab[%tile] : tensor<?x?xf32>
    %c_sub = gml_st.materialize %c[%tile] : tensor<?x?xf32>
    %init_sub = gml_st.materialize %init[%tile] : tensor<?x?xf32>

    %abc_sub = linalg.map "mul" ins(%ab_sub, %c_sub) outs(%init_sub)
    gml_st.set_yield %abc_sub into %init[%tile]
  } : tensor<?x?xf32>
  func.return %abc
}
```

Tiling and Fusing Element-wise Ops IV

```
func.func @cwise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>, %c: tensor<?x?xf32>)
    -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>

  %abc = gml_st.parallel (%i, %j) = (%c0, %c0) to (%d0, %d1) step (%c4, %c8) {
    %space = gml_st.space [%d0, %d1] : !gml_st.tile<?x?>
    %tile = gml_st.tile %space [%i, %j] [%td0, %td1] [1, 1] !gml_st.tile<?x?>
    %a_sub = gml_st.materialize %a[%tile] : tensor<?x?xf32>
    %b_sub = gml_st.materialize %b[%tile] : tensor<?x?xf32>
    %c_sub = gml_st.materialize %c[%tile] : tensor<?x?xf32>
    %init_sub = gml_st.materialize %init[%tile] : tensor<?x?xf32>
    %ab_sub = linalg.map "add" ins(%a_sub, %b_sub) outs(%init_sub)
    %abc_sub = linalg.map "mul" ins(%ab_sub, %c_sub) outs(%init_sub)
    gml_st.set_yield %abc_sub into %init[%tile]
  } : tensor<?x?xf32>
  func.return %abc
}
```

Tile Concat

Materialize Ops are an Entry Point to Fusion

- Allows to fuse w/o the presence of a specific loop
  - Concat can be tiled to a switch statement
  - Allows for different loop kinds

Concat I

```
func.func @concat(%a : tensor<?x?xi32>, %b : tensor<?x?xi32>) -> tensor<?x?xi32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>

  %concat = thlo.concatenate ins(%ab, %a) outs(%init : tensor<?x?xi32>) { dimension = 1 }

  func.return %concat
}
```

Concat II

```
func.func @concat(%a : tensor<?x?xi32>, %b : tensor<?x?xi32>) -> tensor<?x?xi32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>

  %concat = gml_st.parallel (%i, %j) = (%c0, %c0) to (%d0, %d1) step (%c4, %c1) {

    %concat_sub = scf.if %subset_in_a {
      %space = gml_st.space [%d0, %abd1]
      %tile = gml_st.tile %space [%i, %j] [%td0, 1] [1, 1]
      %a_sub = gml_st.materialize %ab[%tile] : tensor<?x?xf32>
      scf.yield %a_sub
    } else {

      …
```

Concat III

```
func.func @concat(%a : tensor<?x?xi32>, %b : tensor<?x?xi32>) -> tensor<?x?xi32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xi32>

  %concat = gml_st.parallel (%i, %j) = (%c0, %c0) to (%d0, %d1) step (%c4, %c1) {

    %concat_sub = scf.if %subset_in_a {
      %space = gml_st.space [%d0, %abd1]
      %tile = gml_st.tile %space [%i, %j] [%td0, 1] [1, 1]
      %a_sub = gml_st.materialize %ab[%tile] : tensor<?x?xf32>
      scf.yield %a_sub
    } else {

      …
```

Tile Reductions

Tile Reductions

Reductions can be tiled to

- `gml_st.parallel` with an accumulator or to
- `gml_st.for` as a sequential implementation.

Reduction I

```
func.func @reduce(%arg: tensor<32x16xf32>) -> tensor<32xf32> {
  %init = linalg.init_tensor [32] : tensor<32xf32>
  %fill = linalg.fill ins(%c0) outs(%init) : tensor<32xf32>
  %sum = linalg.reduce "sum" reduction_dim = 1 ins(%arg) outs(%fill)
  func.return %sum
}
```

Reduction II

```
func.func @reduce(%arg: tensor<32x16xf32>) -> tensor<32xf32> {
  %init = linalg.init_tensor [32] : tensor<32xf32>
  %fill = linalg.fill ins(%c0) outs(%init) : tensor<32xf32>
  %sum = gml_st.parallel (%i, %j) = (%c0, %c0) to (%c32, %c16) step (%c4, %c8) {
    %s0 = gml_st.space [32, 16] : !gml_st.tile<32x16>
    %t0 = gml_st.tile %s0 [%i, %j] [%t0d0, %t0d1] [1, 1] : !gml_st.tile<?x?>
    %arg_sub = gml_st.materialize %arg[%t0] : !gml_st.tile<4x8>
    %s1 = gml_st.space [32] : !gml_st.tile<32>
    %t1 = gml_st.tile %s1 [%i] [%t1d0] [1] : !gml_st.tile<4>
    %fill_sub = gml_st.materialize %fill[%t1] : tensor<4xf32>
    %sum_sub = linalg.reduce "sum" reduction_dim = 1 ins(%arg_sub) outs(%fill_sub)
    gml_st.set_yield %sum_sub into %fill[%t1] acc (%a, %b: tensor<4xf32>) {
      %ab = linalg.binary "add" ins(%a) outs(%b)
      gml_st.yield %ab
    }
  } : tensor<32xf32>
  func.return %sum
}
```

Nested Tiling

Nested Tiling

Repeated tiling of one operation

Nested Tiling I

```
func.func @ewise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>) -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
  %ab = linalg.map "add" ins(%a, %b) outs(%init)
  func.return %ab
}
```

Nested Tiling II

```
func.func @ewise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>) -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
  %ab = gml_st.parallel (%i, %j) = (%c0, %c0) to (%0, %1) step (%c128, %c512) {
    %space = gml_st.space [%d0, %d1] : !gml_st.tile<?x?>
    %t0 = gml_st.tile %space [%i, %j] [%t0d0, %t0d1] [1, 1] : !gml_st.tile<?x?>



    %a_sub = gml_st.materialize %a[%t0] : tensor<?x?xf32>
    %b_sub = gml_st.materialize %b[%t0] : tensor<?x?xf32>
    %init_sub = gml_st.materialize %init[%t0] : tensor<?x?xf32>
    %ab_sub = linalg.map "add" ins(%a_sub, %b_sub) outs(%init_sub)


    gml_st.set_yield %ab_sub into %init[%t0]
  } : tensor<?x?xf32>
  func.return %ab
}
```

Nested Tiling III

```
func.func @ewise(%a: tensor<?x?xf32>, %b: tensor<?x?xf32>) -> tensor<?x?xf32> {
  %init = linalg.init_tensor [%d0, %d1] : tensor<?x?xf32>
  %ab = gml_st.parallel (%i, %j) = (%c0, %c0) to (%0, %1) step (%c128, %c512) {
    %space = gml_st.space [%d0, %d1] : !gml_st.tile<?x?>
    %t0 = gml_st.tile %space [%i, %j] [%d0, %d1] [1, 1] : !gml_st.tile<?x?>
    %init_sub = gml_st.materialize %init[%t0] : tensor<?x?xf32>
    %ab_sub = gml_st.parallel (%k, %l) = (%c0, %c0) to (%d0, %d1) step (%c4, %c8) {
      %t1 = gml_st.tile %t0 [%k, %l] [%t1d0, %t1d1] [1, 1] : !gml_st.tile<?x?>
      %a_sub_sub = gml_st.materialize %a[%t1] : tensor<?x?xf32>
      %b_sub_sub = gml_st.materialize %b[%t1] : tensor<?x?xf32>
      %init_sub_sub = gml_st.materialize %init_sub[%t1] : tensor<?x?xf32>
      %ab_sub_sub = linalg.map "add" ins(%a_sub_sub, %b_sub_sub) outs(%init_sub_sub)
      gml_st.set_yield %ab_sub_sub into %init_sub[%t1]
    } : tensor<?x?xf32>
    gml_st.set_yield %ab_sub into %init[%t0]
  } : tensor<?x?xf32>
  func.return %ab
}
```

Questions?

Pipeline Overview