An Introduction to Tensor Tiling in MLIR

Kunwar Grover
Mahesh Ravishanker

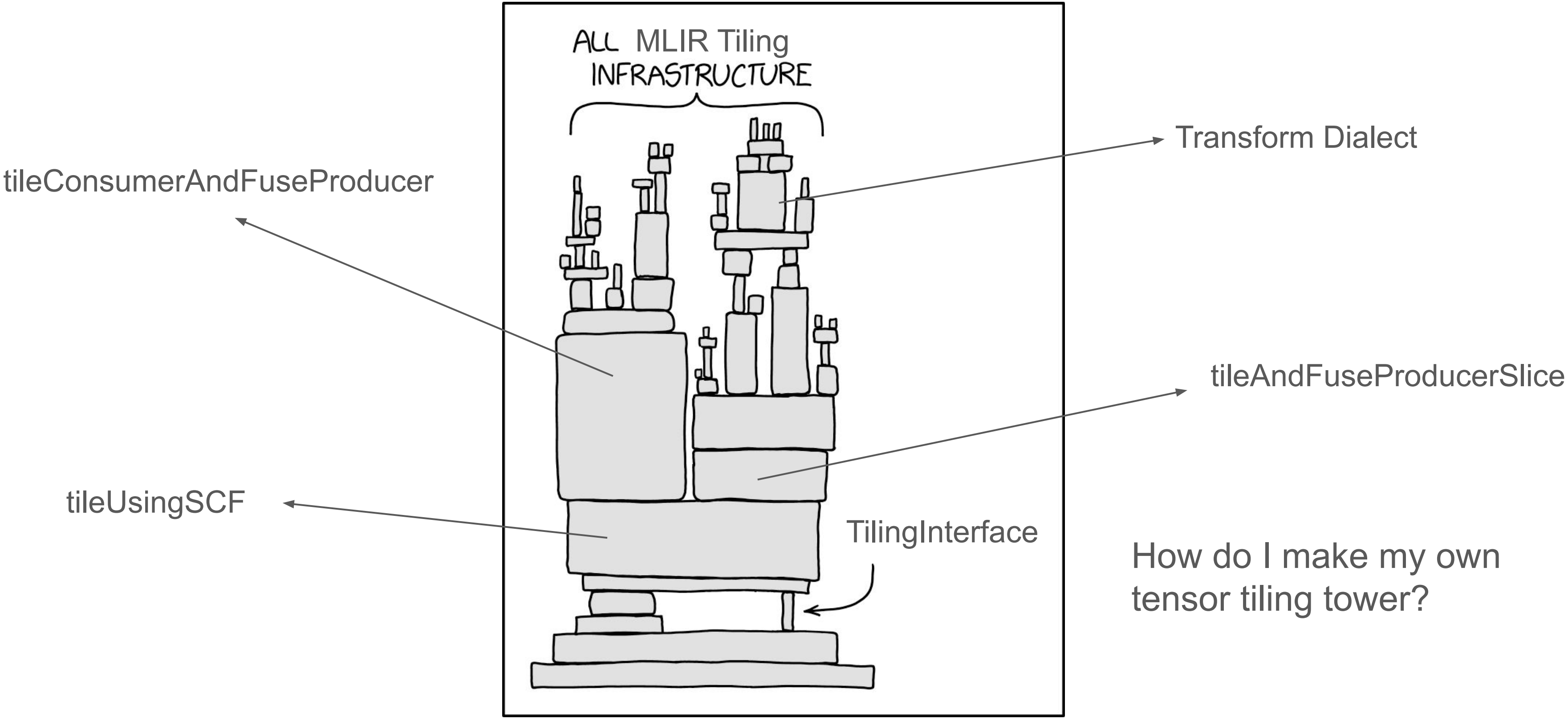
Why This Tutorial?

People frequently ask me:

- How do I build my own tensor compiler using upstream dialects?
- Why isn't there a simple pass to tile tensor operations?

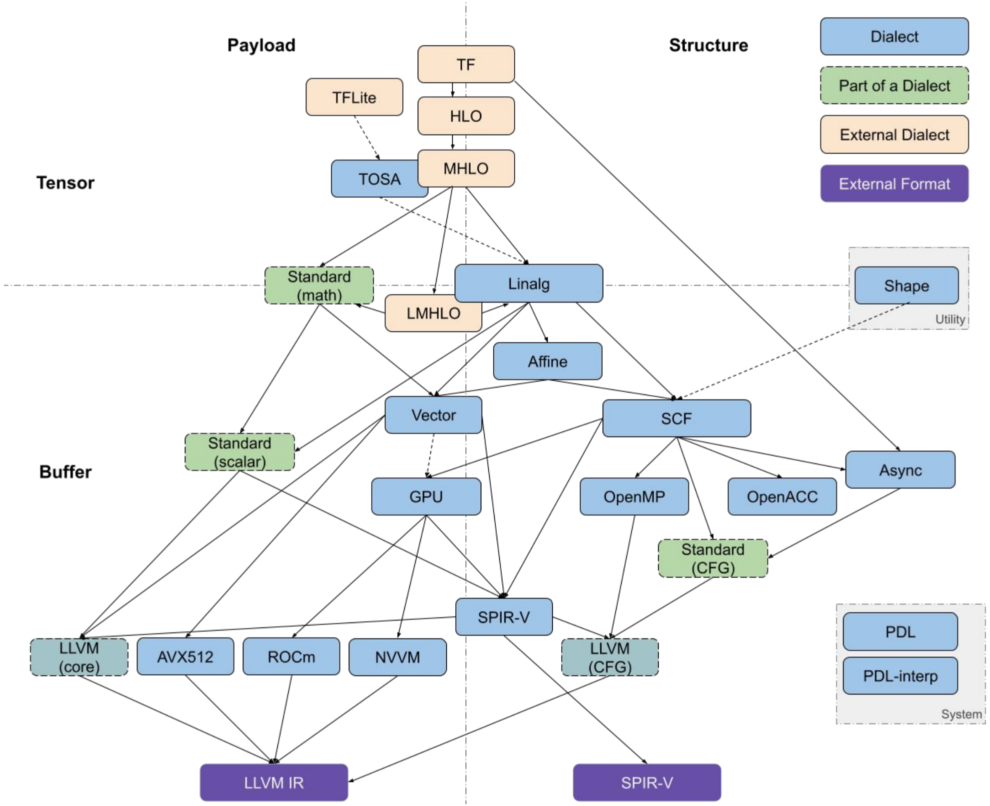
160k+ LoC (C++)

A 100 foot view of Tensor Tiling in MLIR

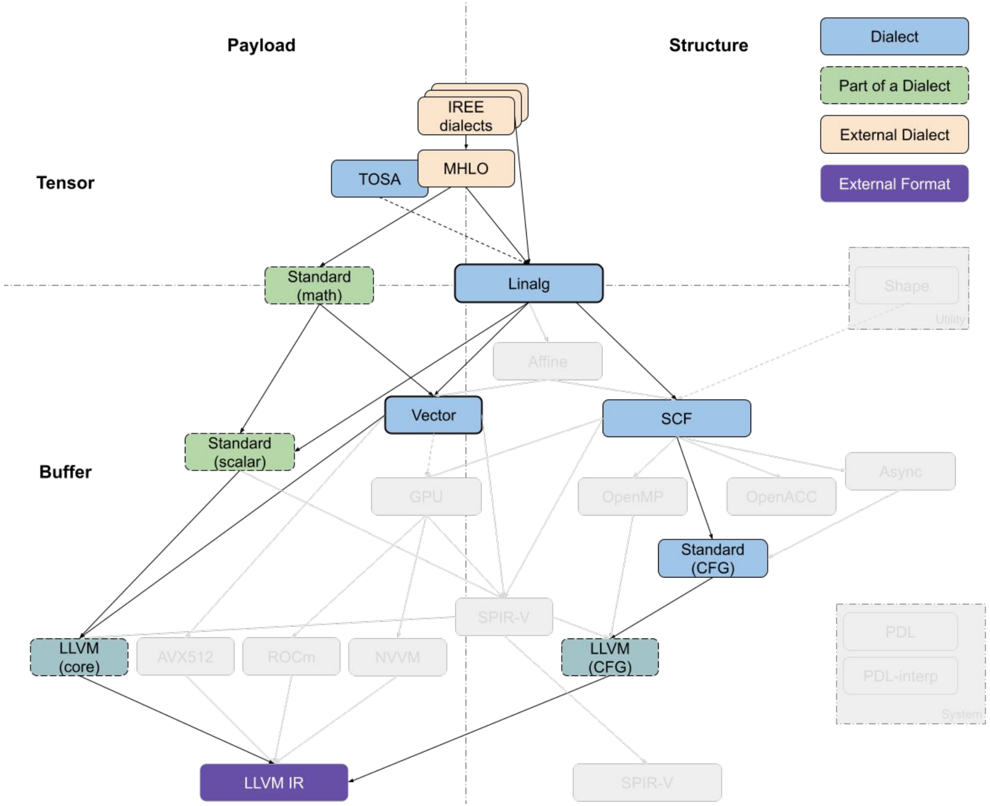


How do I make my own tensor tiling tower?

What will we cover?



What will we cover?

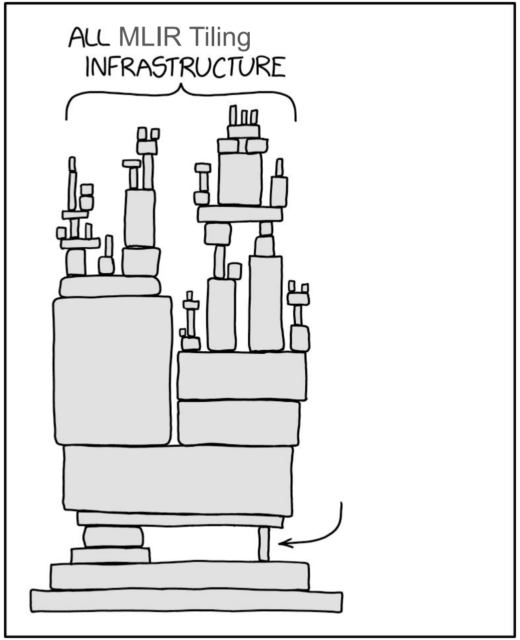


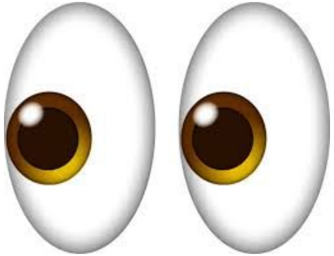
This Tutorial

1. Observe
2. Understand
3. Build
4. Extend

This Tutorial

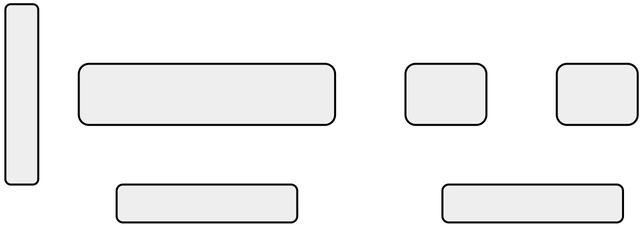
1. **Observe**
2. Understand
3. Build
4. Extend





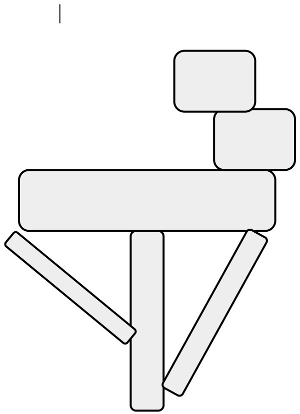
This Tutorial

1. Observe
2. **Understand**
3. Build
4. Extend



This Tutorial

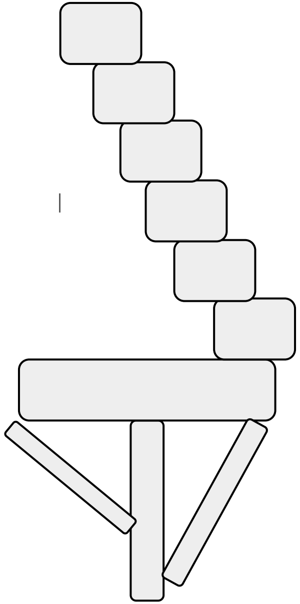
1. Observe
2. Understand
3. **Build**
4. Extend



~400 LoC

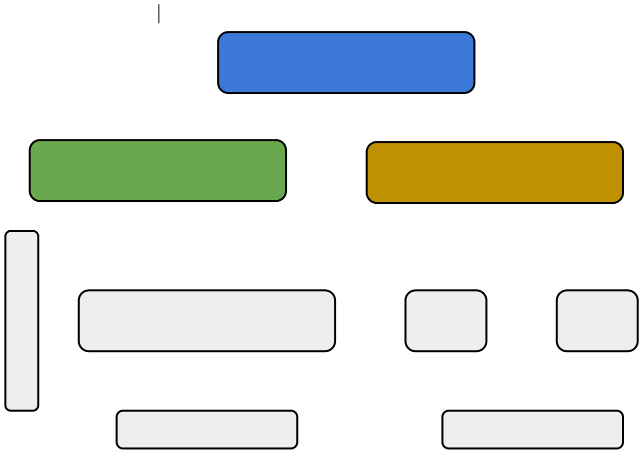
This Tutorial

1. Observe
2. Understand
3. Build
4. **Extend**

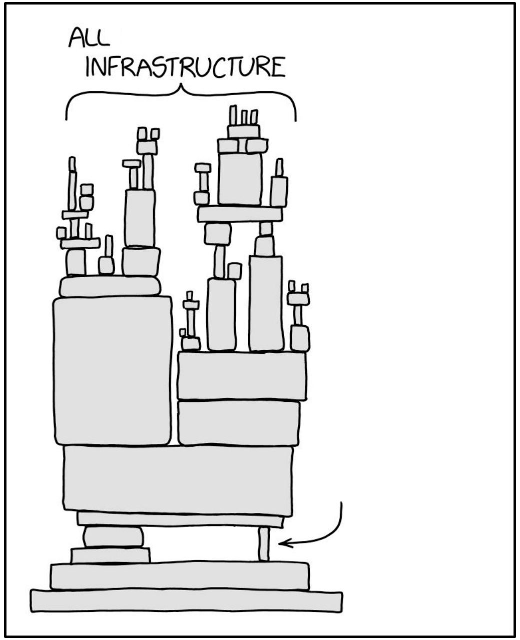


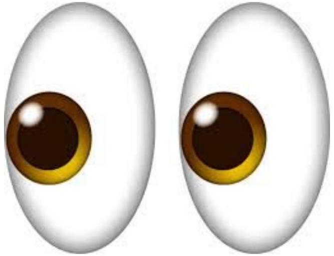
This Tutorial

1. Observe
2. Understand
3. Build
4. Extend
5. **Advanced**



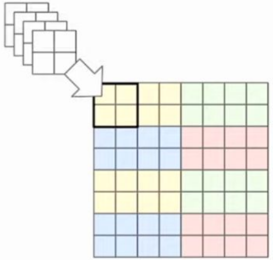
Observe





Fusion into Loops

```
linalg.generic
scf.forall (%i, %j) in (2, 4) {
  linalg.generic {
    indexing_maps = ...,
    iterator_types = ...,
  } ins(memref<4x2xf32>, memref<4x2xf32>, f32)
   outs(memref<4x2xf32>) {
     ...
  }
}
```



```
transform.structured.fuse_into_containing_op
  %structured into %loop
```

Similar to `compute_at` as long as the loop has been materialized.

Google Research

2023 EURO LLVM DEVELOPERS' MEETING

Tutorial: Controllable Transformations in MLIR

Alex Zinenko

Replicating Halide For Convolutions

```
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
for n
  for y
    for x
      for c
        conv[n, y, x, c] = bias[c]
for n
  for y
    for x
      for c
        for rz
          for ry
            for rx
              conv[n, y, x, c] += filter[rx, rz, ry, c] * input[n, y+rz, x+ry, rx]
for n
  for y
    for x
      for c
        relu[n, y, x, c] = max(0, conv[n, y, x, c])
```

Source IR: Conv2d + ReLU

Objective: Fuse Broadcast, Conv and ReLU, and target the Conv to a good tile size for the hardware

https://mlir.llvm.org/docs/Tutorials/transform/ChH/

Replicating Halide For Convolutions

```
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
for n
  for y
    for x
      for c
        conv[n, y, x, c] = bias[c]
        for rz
          for ry
            for rx
              conv[n, y, x, c] += filter[rx, rz, ry, c] * input[n, y+rz, x+ry, rx]
        relu[n, y, x, c] = max(0, conv[n, y, x, c])
```

Source IR: Conv2d + ReLU

Objective: Fuse Broadcast, Conv and ReLU, and target the Conv to a good tile size for the hardware

https://mlir.llvm.org/docs/Tutorials/transform/ChH/

Transform Dialect

```
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

 ...

}
```

Transform Dialect: Describe transformations on operations

Transform Dialect

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) …

  "scf.forall.yield" %conv_tile
}
%relu = linalg.elementwise ...
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  %tiled_relu, %forall =
  transform.structured.tile_using_forall %conv
                                        // n  x  y  c
                                tile_sizes [1, 1, 5, 64]


}
```

Transform Dialect: Describe transformations on operations

Replicating Halide For Convolutions: Parallel Tiling

```
%init = linalg.broadcast ...
%conv = linalg.conv2d ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %conv_tile = tensor.extract_slice %conv ...

  %relu_tile = linalg.elementwise ins(%conv_tile, 0)

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  %tiled_relu, %forall =
  transform.structured.tile_using_forall %relu
                                        // n  x  y  c
                              tile_sizes [1, 1, 5, 64]

}
```

Replicating Halide For Convolutions: Parallel Tiling

```
%init = linalg.broadcast ...
%conv = linalg.conv2d ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %conv_tile = tensor.extract_slice %conv ...

  %relu_tile = linalg.elementwise ins(%conv_tile, 0)

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %conv into %forall

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %bias into %forall

}
```

Replicating Halide For Convolutions: Parallel Tiling

```
%init = linalg.broadcast ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile)
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %conv into %forall

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %bias into %forall

}
```

Replicating Halide For Convolutions: Parallel Tiling

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile = linalg.conv2d ...
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %conv into %forall

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %bias into %forall

}
```

Replicating Halide For Convolutions: Reduction Tiling

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %red_fill, %red_conv, %combining, %forloops =
  transform.structured.tile_reduction_using_for %conv3
                                  //  n  x  y  c rz ry rx
                            tile_sizes [0, 0, 0, 0, 1, 1, 1]

}
```

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile = linalg.conv2d ...
  %relu_tile = linalg.elementwise ...

  “scf.forall.yield” %relu_tile
}
```

Replicating Halide For Convolutions: Reduction Tiling

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %red_fill, %red_conv, %combining, %forloops =
  transform.structured.tile_reduction_using_for %conv3
                                            //  n  x  y  c rz ry rx
                                   tile_sizes [0, 0, 0, 0, 1, 1, 1]

}
```

Replicating Halide For Convolutions: Loop Structure

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
for n
  for y
    for x
      for c
        conv[n, y, x, c] = bias[c]
        for rz
          for ry
            for rx
              conv[n, y, x, c] += filter[rx, rz, ry, c] * input[n, y+rz, x+ry, rx]
        relu[n, y, x, c] = max(0, conv[n, y, x, c])
```

Replicating Halide For Convolutions: Vectorization

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  transform.structured.vectorize_children_and_apply_patterns %func

}
```

Replicating Halide For Convolutions: Vectorization

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %bias_tile = vector.transfer_read %bias ...
  %init_tile = vector.broadcast %bias_tile
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = vector.transfer_read %filter ...
        %input_subtile = vector.transfer_read %input ...
        %conv_mul = arith.mulf ... : vector<...>
        %conv_sum = arith.addf ... : vector<...>
        scf.yield %conv_sm
      }
    }
  }
  %relu_tile = arith.maxnumf %conv_tile ... : vector<...>

  "scf.forall.yield" %relu_tile
}
```

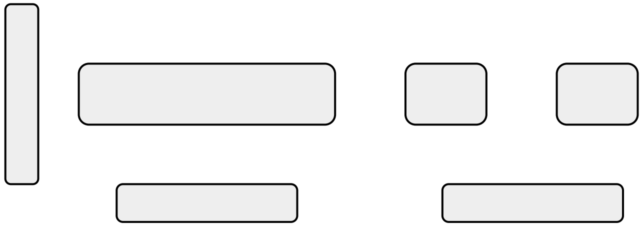
```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  transform.structured.vectorize_children_and_apply_patterns %func

}
```

Replicating Halide For Convolutions: Bufferization

```
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %bias_tile = vector.transfer_read %bias ...
  %init_tile = vector.broadcast %bias_tile
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = vector.transfer_read %filter ...
        %input_subtile = vector.transfer_read %input ...
        %conv_mul = arith.mulf ... : vector<...>
        %conv_sum = arith.addf ... : vector<...>
        scf.yield %conv_sm
      }
    }
  }
  %relu_tile = arith.maxnumf %conv_tile ... : vector<...>
  vector.transfer_write %relu_tile, %relu ...
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  transform.bufferization.one_shot_bufferize %func {
    bufferize_function_boundaries = true,
    function_boundary_type_conversion = 1 : i32
  }


}
```

Understand



Main Observations

- Other than tiling, rest of the pipeline is fixed
- Getting the loop structure right is important
- Loop structure is built using Tiling and Fusing

Under The Hood: TilingInterface

```
%init = linalg.broadcast ...
%conv = linalg.conv2d ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %conv_tile = tensor.extract_slice %conv ...

  %relu_tile = linalg.elementwise ins(%conv_tile, 0)

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  %tiled_relu, %forall =
  transform.structured.tile_using_forall %relu
                                        // n  x  y  c
                                tile_sizes [1, 1, 5, 64]

}
```

Tiling an Operation: `tileUsingSCF`

Under The Hood: Tiling By Fusion

```
%init = linalg.broadcast ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile)
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %conv into %forall

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %bias into %forall

}
```

Tiling by Fusion:

```
tileAndFuseProduceSlice
tileAndFuseConsumerSlice
```

Under The Hood: Tiling By Fusion

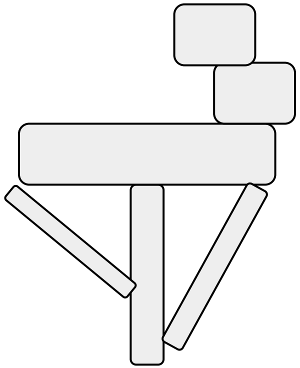
```
%init = linalg.broadcast ...
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile)
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
transform.sequence {
^bb0(%bias, %conv, %relu):

  ...

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %conv into %forall

  %tiled_conv, %forall2 =
  transform.structured.fuse_into_containing_op %bias into %forall

}
```

Tiling by Fusion:

```
tileAndFuseProduceSlice
tileAndFuseConsumerSlice
```

Build



https://github.com/Groverkss/tinytile

TinyTile: Greedy TileAndFuse

```
%init = linalg.broadcast ins(%bias) ...

// tile conv using scf.forall:
//                n  x  y  c   rz    rx    ry
// tile_sizes = [1, 1, 5, 64, None, None, None]
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...

%relu = linalg.elementwise ins(%conv, 0) ...
```

```
LogicalResult tileAndFuse(TilingInterface op,
                          ArrayRef<int64_t> tileSizes) {
  ...
  // Control how to tile the operation.
  scf::SCFTilingOptions tilingOptions;
  tilingOptions.setTileSizes(tileSizes);

  // Tile the operation.
  scf::tileUsingSCF(rewriter, tilingOp, tilingOptions);
  ...
}
```

TinyTile: Greedy TileAndFuse

```
%init = linalg.broadcast ins(%bias) ...

// tile conv using scf.forall:
//                n  x  y  c   rz    rx    ry
// tile_sizes = [1, 1, 5, 64, None, None, None]
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...

%relu = linalg.elementwise ins(%conv, 0) ...
```

↓

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) ...

  "scf.forall.yield" %conv_tile
}
%relu = linalg.elementwise ...
```

```
LogicalResult tileAndFuse(TilingInterface op,
                          ArrayRef<int64_t> tileSizes) {
  ...
  // Control how to tile the operation.
  scf::SCFTilingOptions tilingOptions;
  tilingOptions.setTileSizes(tileSizes);

  // Tile the operation.
  scf::tileUsingSCF(rewriter, tilingOp, tilingOptions);
  ...
}
```

TinyTile: Greedy TileAndFuse

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) ...

  “scf.forall.yield” %conv_tile
}
%relu = linalg.elementwise ...
```

```
LogicalResult tileAndFuse(TilingInterface op
                             ArrayRef<int64_t> tileSizes) {
  // Tile the operation
  ...

  while (true) {
    // Get a candidate slice.
    Operation *candidate = ...

    // Fuse producer.
    if (isa<tensor::ExtractSliceOp>(candidate)) {
      scf::tileAndFuseProducerOfSlice(rewriter, candidate, loops);
    }

    // Fuse consumer.
    if (isa<tensor::InsertSliceOp,
            tensor::ParallelInsertSliceOp>(candidate)) {
      scf::tileAndFuseConsumerOfSlice(rewriter, candidate, loops);
    }

  }
  ...
}
```

TinyTile: Greedy TileAndFuse

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) ...

  "scf.forall.yield" %conv_tile
}
%relu = linalg.elementwise ...
```

↓

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile = linalg.conv2d ...
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
LogicalResult tileAndFuse(TilingInterface op
                          ArrayRef<int64_t> tileSizes) {
  // Tile the operation
  ...

  while (true) {
    // Get a candidate slice.
    Operation *candidate = ...

    // Fuse producer.
    if (isa<tensor::ExtractSliceOp>(candidate)) {
      scf::tileAndFuseProducerOfSlice(rewriter, candidate, loops);
    }

    // Fuse consumer.
    if (isa<tensor::InsertSliceOp,
            tensor::ParallelInsertSliceOp>(candidate)) {
      scf::tileAndFuseConsumerOfSlice(rewriter, candidate, loops);
    }

  }
  ...
}
```

TinyTile: Tracking Slices

```
%init = linalg.broadcast ins(%bias) ...

// tile conv using scf.forall:
//                 n  x  y  c   rz    rx    ry
// tile_sizes = [1, 1, 5, 64, None, None, None]
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...

%relu = linalg.elementwise ins(%conv, 0) ...
```

↓

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) ...

  "scf.forall.yield" %conv_tile
}
%relu = linalg.elementwise ...
```

```
struct SliceListener : public RewriterBase::Listener {
  void notifyOperationInserted(Operation* op,
                               OpBuilder::InsertPoint) override {
    if (isa<tensor::ExtractSliceOp,
            tensor::InsertSliceOp,
            tensor::ParallelInsertSliceOp>(op)) {
      candidates.push_back(op);
    }
  }
  std::deque<Operation*> candidates;
};
```

TinyTile: Tracking Slices

```
%init = linalg.broadcast ins(%bias) ...

// tile conv using scf.forall:
//                n  x  y  c   rz    rx    ry
// tile_sizes = [1, 1, 5, 64, None, None, None]
%conv = linalg.conv2d ins(%filter, %input) outs(%init) …

%relu = linalg.elementwise ins(%conv, 0) ...
```

```
%init = linalg.broadcast ...
%conv =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %init_tile = tensor.extract_slice %init ...

  %conv_tile = linalg.conv2d ins(%filter_tile, %input_tile)
                             outs(%init_tile) …

  “scf.forall.yield” %conv_tile
}
%relu = linalg.elementwise ...
```

```
struct SliceListener : public RewriterBase::Listener {
  void notifyOperationInserted(Operation* op,
                               OpBuilder::InsertPoint) override {
    if (isa<tensor::ExtractSliceOp,
            tensor::InsertSliceOp,
            tensor::ParallelInsertSliceOp>(op)) {
      candidates.push_back(op);
    }
  }
  std::deque<Operation*> candidates;
};

LogicalResult tileAndFuse(TilingInterface op
                          ArrayRef<int64_t> tileSizes) {
  SliceListener listener;
  rewriter.setListener(&listener);
  ...
  while (!candidates.empty()) {
    // Get a candidate slice.
    Operation *candidate = listener.candidates.front();
    listener.candidates.pop_front()
    ...
  }
  ...
}
```

TinyTile: Multiple Tiling Levels

```
%init = linalg.broadcast ins(%bias) ...

// tile using scf.forall:
// tile_sizes = [1, 1, 5, 64, 0, 0, 0]

// tile using scf.for
// tile_sizes = [0, 0, 0, 0, 1, 1, 1]

%conv = linalg.conv2d {
          lowering_config = {
            parallel  = [1, 1, 5, 64, 0, 0, 0],
            reduction = [0, 0, 0, 0,  1, 1, 1]
          }
        } ins(%filter, %input) outs(%init) …

%relu = linalg.elementwise ins(%conv, 0) ...
```

```
LogicalResult tileAndFuse(TilingInterface op) {
  ...
  // Control how to tile the operation.
  scf::SCFTilingOptions tilingOptions;

  SmallVector<int64_t> tileSizes = getTileSizes(op);
  tilingOptions.setTileSizes(tileSizes);

}
```

TinyTile: Multiple Tiling Levels

```
%init = linalg.broadcast ins(%bias) ...

// tile using scf.forall:
// tile_sizes = [1, 1, 5, 64, 0, 0, 0]

// tile using scf.for
// tile_sizes = [0, 0, 0, 0, 1, 1, 1]

%conv = linalg.conv2d {
          lowering_config = {
            parallel  = [1, 1, 5, 64, 0, 0, 0],
            reduction = [0, 0, 0, 0,  1, 1, 1]
          }
        } ins(%filter, %input) outs(%init) ...

%relu = linalg.elementwise ins(%conv, 0) ...
```

```
LogicalResult tileAndFuse(TilingInterface op) {
  ...
  // Control how to tile the operation.
  scf::SCFTilingOptions tilingOptions;

  SmallVector<int64_t> tileSizes = getTileSizes(op);
  tilingOptions.setTileSizes(tileSizes);

  if (tilingLevel == tutorial::TilingLevel::Parallel) {
    tilingOptions.setLoopType(
                  scf::SCFTilingOptions::LoopType::ForallOp);
  } else {
    tilingOptions.setLoopType(
                  scf::SCFTilingOptions::LoopType::ForOp);
  }
  ...
}
```

TinyTile: Pass Pipeline

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
void createPassPipeline(PassManager &pm) {

  // Parallel tiling using scf.forall
  {
    tutorial::TutorialTileAndFuseOptions options;
    options.tilingLevel = tutorial::TilingLevel::Parallel;
    pm.addPass(tutorial::createTutorialTileAndFuse(options));
  }

  // Reduction tiling using scf.for
  {
    tutorial::TutorialTileAndFuseOptions options;
    options.tilingLevel = tutorial::TilingLevel::Reduction;
    pm.addPass(tutorial::createTutorialTileAndFuse(options));
  }

  ...
}
```

TinyTile: Vectorization

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %bias_tile = vector.transfer_read %bias ...
  %init_tile = vector.broadcast %bias_tile
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = vector.transfer_read %filter ...
        %input_subtile = vector.transfer_read %input ...
        %conv_mul = arith.mulf ... : vector<...>
        %conv_sum = arith.addf ... : vector<...>
        scf.yield %conv_sm
      }
    }
  }
  %relu_tile = arith.maxnumf %conv_tile ... : vector<...>

  "scf.forall.yield" %relu_tile
}
```

```
// Vectorization
{
  pm.addPass(createLinalgGeneralizeNamedOpsPass());
  pm.addPass(tutorial::createTutorialVectorization());
  // Cleanup
  pm.addPass(createCanonicalizerPass());
  pm.addPass(createCSEPass());
  pm.addPass(tensor::createFoldTensorSubsetOpsPass());
}
```

TinyTile: Vectorization

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %bias_tile = vector.transfer_read %bias ...
  %init_tile = vector.broadcast %bias_tile
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = vector.transfer_read %filter ...
        %input_subtile = vector.transfer_read %input ...
        %conv_mul = arith.mulf ... : vector<...>
        %conv_sum = arith.addf ... : vector<...>
        scf.yield %conv_sm
      }
    }
  }
  %relu_tile = arith.maxnumf %conv_tile ... : vector<...>

  "scf.forall.yield" %relu_tile
}
```

```
// Vectorization
{
  pm.addPass(createLinalgGeneralizeNamedOpsPass());
  pm.addPass(tutorial::createTutorialVectorization());
  // Cleanup
  pm.addPass(createCanonicalizerPass());
  pm.addPass(createCSEPass());
  pm.addPass(tensor::createFoldTensorSubsetOpsPass());
}
```

```
SmallVector<linalg::GenericOp> candidates;
  funcOp.walk([&](linalg::GenericOp op) {
    candidates.push_back(op);
  });

  for (linalg::GenericOp candidate : candidates) {
    (void)linalg::vectorize(rewriter, candidate);
  }
```

TinyTile: Bufferization

```
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %bias_tile = vector.transfer_read %bias ...
  %init_tile = vector.broadcast %bias_tile
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = vector.transfer_read %filter ...
        %input_subtile = vector.transfer_read %input ...
        %conv_mul = arith.mulf ... : vector<...>
        %conv_sum = arith.addf ... : vector<...>
        scf.yield %conv_sm
      }
    }
  }
  %relu_tile = arith.maxnumf %conv_tile ... : vector<...>
  vector.transfer_write %relu_tile, %relu ...
}
```

```
// Bufferization
{
  bufferization::OneShotBufferizePassOptions options;
  options.bufferizeFunctionBoundaries = true;
  options.functionBoundaryTypeConversion =
      bufferization::LayoutMapOption::IdentityLayoutMap;
  pm.addPass(bufferization::createOneShotBufferizePass(options));
  pm.addPass(createCanonicalizerPass());
  pm.addPass(createCSEPass());
  pm.addPass(memref::createFoldMemRefAliasOpsPass());
}
```

TinyTile: A Matmul/Convolution Compiler!

```
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  “scf.forall.yield” %relu_tile
}
```

TinyTile: A Matmul/Convolution Compiler!

```
%packed_input = linalg.pack %input
%packed_filter = linalg.pack %filter
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%packed_filter, %packed_input)
                      outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile ...
        %packed_filter = linalg.pack %filter_subtile ...
        %packed_input  = linalg.pack %input_subtile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

TinyTile: A Matmul/Convolution Compiler!

```
%packed_input = linalg.pack %input
%packed_filter = linalg.pack %filter
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%packed_filter, %packed_input)
                      outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) …
%actv = linalg.elementwise ins(%relu, %scale) …
```

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile …
        %packed_filter = linalg.pack %filter_subtile ...
        %packed_input  = linalg.pack %input_subtile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise …
  %scale_tile = tensor.extract_slice %scale ...
  %actv_tile = linalg.elementwise ...

  “scf.forall.yield” %relu_tile
}
```

TinyTile: Controlling Tiling Scheme

```
func.func @kernel (...) -> ...
attributes { tiling_transform_spec = "__halide" } {
  ...
}

transform.named_sequence @__halide(...) {
  ...
}
```

TinyTile: Controlling Tiling Scheme

```
func.func @kernel (...) -> ...
attributes { tiling_transform_spec = "__halide" } {
  ...
}

transform.named_sequence @__halide(...) {
  ...
}
```

```
// Get transform entry point.
auto entryPoint =
  funcOp->getAttrOfType<StringAttr>("transform_tiling_spec");

// Create a dynamic pass pipeline to run.
OpPassManager modulePassManager(ModuleOp::getOperationName());
transform::InterpreterPassOptions options;
options.entryPoint = entryPoint.str();
modulePassManager.addPass(transform::createInterpreterPass(options));

// Run pipeline on the module.
runPipeline(modulePassManager, moduleOp);
```

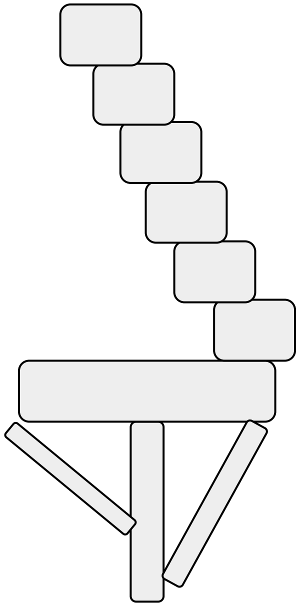
TinyTile: Controlling Tiling Scheme

```
func.func @kernel (...) -> ...
attributes { tiling_transform_spec = "__halide" } {
  ...
}

transform.named_sequence @__halide(...) {
  ...
}
```

```
// Get transform entry point.
auto entryPoint =
  funcOp->getAttrOfType<StringAttr>("transform_tiling_spec");

// Create a dynamic pass pipeline to run.
OpPassManager modulePassManager(ModuleOp::getOperationName());
transform::InterpreterPassOptions options;
options.entryPoint = entryPoint.str();
modulePassManager.addPass(transform::createInterpreterPass(options));

// Run pipeline on the module.
runPipeline(modulePassManager, moduleOp);
```

```
// Apply required transform spec.
pm.addPass(tutorial::createTutorialApplyTilingSpec());

...
// Parallel Tiling
...
// Reduction Tiling
...
```

Extend



A New Op? : TilingInterface

```
%deqi = tutorial.dequant %input, %scale
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %deqi) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                             "dequant",
  [AllTypesMatch]> {

  let arguments = (
    RankedTensorType:$input,
    RankedTensorType:%scale
  );

  Let results = (
    RankedTensorType:$output
  );

}
```

TilingInterface: Tiling

```
%deqi = tutorial.dequant %input, %scale
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                             "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    [



]>]>
```

TilingInterface: `getIterationDomain`

```
for row in range(M):
  for col in range(N):
    deqi[row][col] = dequant(input[row][col], scale[row][col])
```

```
Iteration Domain: 0 <= row < M, 0 <= col < N
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                               "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",




]>]>
```

TilingInterface: `getIterationDomain`

```
for row in range(M):
  for col in range(N):
    deqi[row][col] = dequant(input[row][col], scale[row][col])

Iteration Domain: 0 <= row < M, 0 <= col < N
```

```
  int64_t rank = getInputType().getRank();

  SmallVector<OpFoldResult> sizes =
    tensor::getMixedSizes(getInput());

  SmallVector<Range> loopBounds(rank);
  for (auto dim : llvm::seq<int64_t>(rank)) {
    loopBounds[dim].offset = 0;
    loopBounds[dim].size = sizes[dim];
    loopBounds[dim].stride = 1;
  }

  return loopBounds;
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                              "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",




]>]>
```

TilingInterface: `getLoopIteratorTypes`

```
for row in range(M):
  for col in range(N):
    deqi[row][col] = dequant(input[row][col], scale[row][col])
```

Both loops are parallel

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                             "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",



]>]>
```

TilingInterface: getLoopIteratorTypes

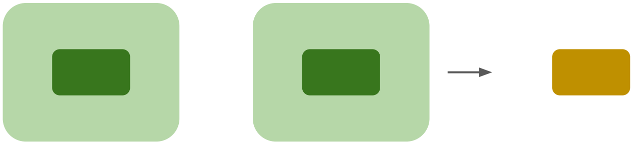
```
for row in range(M):
  for col in range(N):
    deqi[row][col] = dequant(input[row][col], scale[row][col])
```

Both loops are parallel

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                              "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",



]>]>
```

```
int64_t rank = getInput().getType().getRank();
return SmallVector<utils::IteratorType>(rank,
                                        utils::IteratorType::parallel);
```

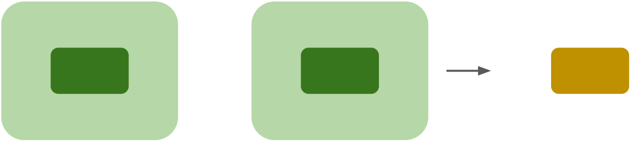
TilingInterface: `getTiledImplementation`



Given: iteration domain
Objective: generate code to compute output tile

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                            "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",


]>]>
```

TilingInterface: `getTiledImplementation`

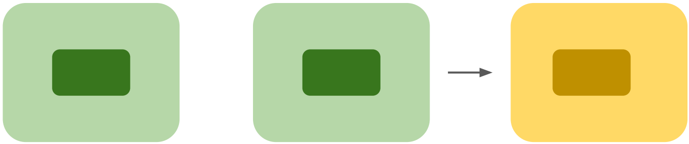


```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                                "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",


]>]>
```

Given: iteration domain
Objective: generate code to compute output tile

```
auto inputTile = b.create<tensor::ExtractSliceOp>(getInput(),
                                                  offsets, sizes);
auto scaleTile = b.create<tensor::ExtractSliceOp>(getScale(),
                                                  offsets, sizes);

Type resultType = inputTile.getResultType();
Operation *tiledOp =
    mlir::clone(b, getOperation(), {resultType}, {inputTile, scaleTile});
```

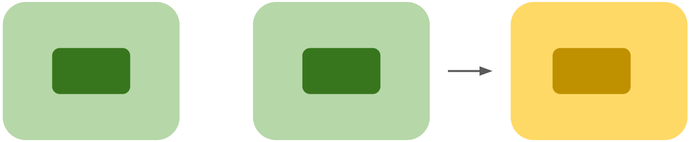
TilingInterface: `getResultTilePosition`



Given: iteration domain
Objective: offset, size of result tiles

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                             "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

]>]>
```

TilingInterface: `getResultTilePosition`



```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                            "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

]>]>
```

Given: iteration domain
Objective: offset, size of result tiles

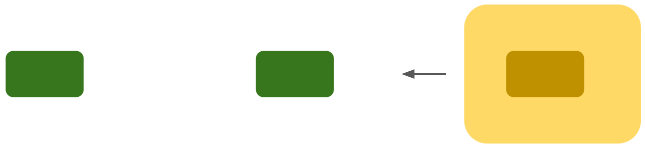
```
resultOffsets = offsets;
resultSizes   = sizes;
```

TilingInterface: Producer Fusion

```
%deqi = tutorial.dequant %input, %scale
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %deqi ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile = linalg.conv2d ...
  %relu_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                                  "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     "getResultTilePosition"

     // producer fusion



]>]>
```

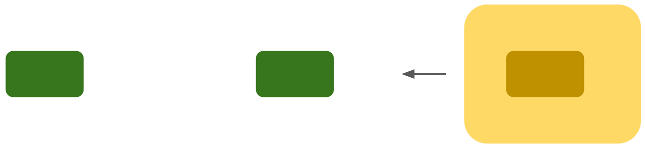
TilingInterface: `getIterationDomainFromResultTile`



Given: offsets, size of result tiles
Objective: iteration domain

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                               "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”


]>]>
```

TilingInterface: `getIterationDomainFromResultTile`

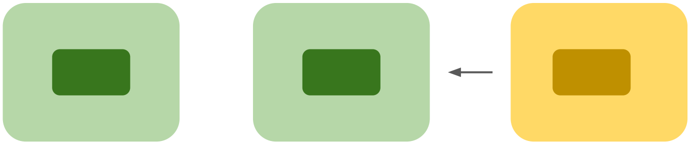


Given: offsets, size of result tiles
Objective: iteration domain

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                            "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”


]>]>
```

```
iterDomainOffsets = offsets;
iterDomainSizes   = sizes;
```

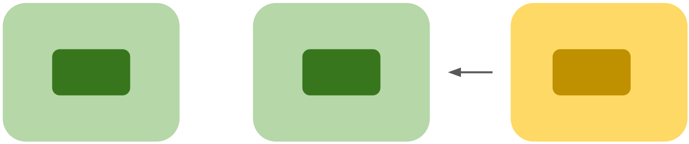
TilingInterface: generateResultTileValue



Given: offsets, size of result tiles
Objective: generate code to compute result tile

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                              "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

]>]>
```

TilingInterface: generateResultTileValue



Given: offsets, size of result tiles
Objective: generate code to compute result tile

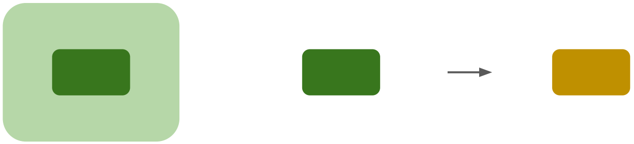
```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                            "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

]>]>
```

```
SmallVector<OpFoldResult> mappedOffsets;
SmallVector<OpFoldResult> mappedSizes;
getIterationDomainTileFromResultTile(offsets, sizes,
                                     mappedOffsets, mappedSizes)
return getTiledImplementation(mappedOffsets, mappedSizes);
```

TilingInterface: Consumer Fusion

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                                 "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

     // consumer fusion



]>]>
```

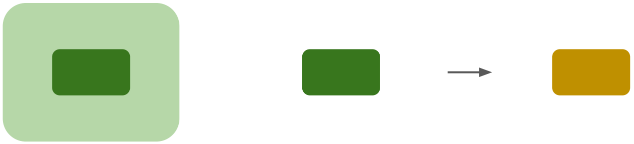
TilingInterface: `getIterationDomainTileFromOperand`



Given: offsets, size of an input tile
Objective: iteration domain

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                                "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     "getResultTilePosition"

     // producer fusion
     "getIterationDomainFromResultTile",
     "generateResultTileValue"

     // consumer fusion
     "getIterationDomainTileFromOperandTile"


]>]>
```

TilingInterface: `getIterationDomainTileFromOperand`

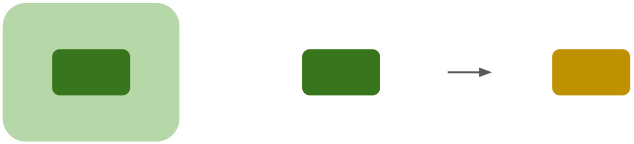


Given: offsets, size of an input tile
Objective: iteration domain

```
iterDomainOffsets = llvm::to_vector(offsets);
iterDomainSizes = llvm::to_vector(sizes);
```

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                            "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

     // consumer fusion
     “getIterationDomainTileFromOperandTile”


]>]>
```

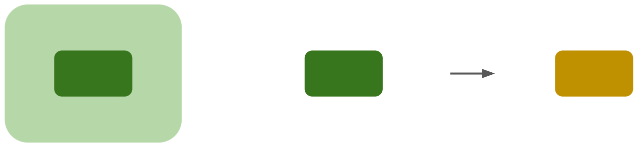
TilingInterface: `getTiledImplementationFromOperand`



Given: offsets, size of an input tile
Objective: generate code to compute result tile

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                              "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

     // consumer fusion
     “getIterationDomainTileFromOperandTile”
     “getTiledImplementationFromOperandTile”

]>]>
```

TilingInterface: `getTiledImplementationFromOperand`



Given: offsets, size of an input tile
Objective: generate code to compute result tile

```
def Tutorial_DequantOp : Op<Tutorial_Dialect,
                                             "dequant",

  [DeclareOpInterfaceMethods<TilingInterface,

    ["getIterationDomain",
     "getLoopIteratorTypes",
     "getTiledImplementation",
     “getResultTilePosition”

     // producer fusion
     “getIterationDomainFromResultTile”,
     “generateResultTileValue”

     // consumer fusion
     “getIterationDomainTileFromOperandTile”
     “getTiledImplementationFromOperandTile”

]>]>
```

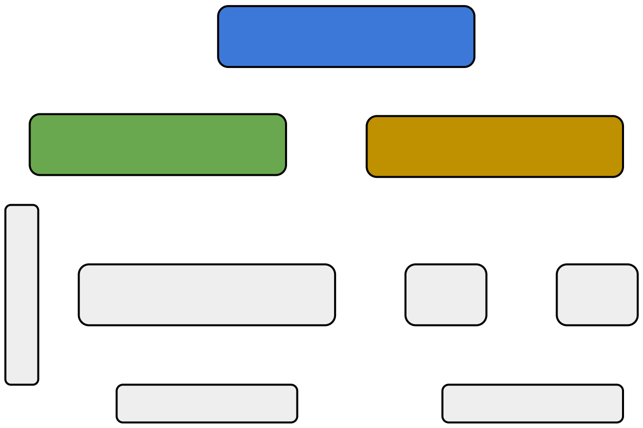
```
SmallVector<OpFoldResult> mappedOffsets;
SmallVector<OpFoldResult> mappedSizes;
getIterationDomainTileFromOperandTile(offsets, sizes,
                                   mappedOffsets, mappedSizes)
return getTiledImplementation(mappedOffsets, mappedSizes);
```

TinyTile: Extended

```
%deqi = tutorial.dequant %input, %scale
%packed_input = linalg.pack %deqi
%packed_filter = linalg.pack %filter
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%packed_filter, %packed_input)
                      outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) …
%actv = linalg.elementwise ins(%relu, %scale) …
```

```
%relu =
scf.forall (%n, %x, %y, %c) in (.../1, .../1, .../5, .../64) {
  %filter_tile = tensor.extract_slice %filter ...
  %input_tile = tensor.extract_slice %input ...
  %scale_tile = tensor.extract_slice %scale ...
  %bias_tile = tensor.extract_slice %bias ...

  %init_tile = linalg.broadcast ins(%bias_tile)
  %conv_tile =
  scf.for %rz ... {
    scf.for %ry ... {
      scf.for %rx ... {
        %filter_subtile = tensor.extract_slice %filter_tile ...
        %input_subtile = tensor.extract_slice %input_tile …
        %scale_subtile = tensor.extract_slice %scale_tile …
        %packed_deqi = tutorial.dequant ...
        %packed_filter = linalg.pack %filter_subtile ...
        %packed_input  = linalg.pack %input_subtile ...
        %conv_subtile = linalg.conv2d ...
        scf.yield %conv_subtile
      }
    }
  }
  %relu_tile = linalg.elementwise …
  %scale_tile = tensor.extract_slice %scale ...
  %actv_tile = linalg.elementwise ...

  "scf.forall.yield" %relu_tile
}
```

Advanced



Tiling on GPUs

```
scf.forall (%warp_x, %warp_y) in (2, 2) {

...

} { mapping = [#gpu.warp<x>, #gpu.warp<y>] }
```

Tiling on GPUs

```
%init = linalg.broadcast ins(%bias) ...
%conv = linalg.conv2d ins(%filter, %input) outs(%init) ...
%relu = linalg.elementwise ins(%conv, 0) ...
```

```
scf::SCFTilingOptions options;
options.setMapping(...)

...

tileUsingSCF(op, options);
```

Tiling on GPUs

```
%init = linalg.broadcast ins(%bias) ...

%conv =
scf.forall (%warp_x, %warp_y) in (2, 2) {
  %filter_slice = tensor.extract_slice %filter
  %input_slice = tensor.extract_slice %input
  %init_slice = tensor.extract_slice %init

  %conv_tile = linalg.conv2d ...

  "scf.forall_yield" %conv_tile
} { mapping = [#gpu.warp<x>, #gpu.warp<y>] }

%relu = linalg.elementwise ins(%conv, 0) ...
```

```
scf::SCFTilingOptions options;
options.setMapping(...)

...

tileUsingSCF(op, options);
```

Tiling on GPUs

```
%init = linalg.broadcast ins(%bias) ...

%relu =
scf.forall (%warp_x, %warp_y) in (2, 2) {
  %filter_slice = tensor.extract_slice %filter
  %input_slice = tensor.extract_slice %input
  %init_slice = tensor.extract_slice %init

  %conv_tile = linalg.conv2d …
  %relu_tile = linalg.elementwise …

  "scf.forall_yield" %conv_tile
} { mapping = [#gpu.warp<x>, #gpu.warp<y>] }
```

Applying Greedy Tile And Fuse

Tiling on GPUs

Applying Greedy Tile And Fuse

```
%relu =
scf.forall (%warp_x, %warp_y) in (2, 2) {
  %filter_slice = tensor.extract_slice %filter
  %input_slice = tensor.extract_slice %input
  %bias_slice = tensor.extract_slice %bias

  %init_tile = linalg.broadcast ...
  %conv_tile = linalg.conv2d ...
  %relu_tile = linalg.elementwise ...

  "scf.forall_yield" %conv_tile
} { mapping = [#gpu.warp<x>, #gpu.warp<y>] }
```

Reduction Tiling

```
enum class ReductionTilingStrategy {

  // [reduction] -> [reduction1, reduction2]
  // -> loop[reduction1] { [reduction2] }
  FullReduction,

  // [reduction] -> [reduction1, parallel2]
  // -> loop[reduction1] { [parallel2] }; merge[reduction1]
  PartialReductionOuterReduction,

  // [reduction] -> [parallel1, reduction2]
  // -> loop[parallel1] { [reduction2] }; merge[parallel1]
  PartialReductionOuterParallel

};
```

Reduction Tiling : Split-K

```
%sum = linalg.sum ins(%input)
                  outs(%init)
```

```
scf::SCFTilingOptions options;
options.setReductionTilingStrategy(
  PartialReductionOuterParallel
);

...

tileUsingSCF(op, options);
```

Reduction Tiling : Split-K

```
%partial_sum =
scf.forall (...) in (...) {
  %input_tile = tensor.extract_slice %input
  %sum_tile = linalg.sum ins(%input)
                         outs(%init)
  “scf.forall_yield” %sum_tile
}

%sum = linalg.sum %partial_sum
```

```
scf::SCFTilingOptions options;
options.setReductionTilingStrategy(
  PartialReductionOuterParallel
);

...

tileUsingSCF(op, options);
```

Thank You!

https://github.com/Groverkss/tinytile