Google

How to Write Optimization Passes in the HEIR Compiler

Jeremy Kun
Google
FHE.org Toronto 2024 Conference
2024-03-24

Agenda

01 HEIR pitch

02 MLIR overview

03 Declarative rewrite rules (local optimizations)

04 Dataflow analyses & global optimizations

05 Resources

01

HEIR Pitch

Using the MLIR framework

We're building a compiler toolchain

Every FHE scheme, fast polynomial math, optimizations, noise models, ...

with all the established ideas in FHE

Focus on your novel idea, not compiler infra

to accelerate FHE research

Benchmarking, hardware targets, frontends (C++, Python, Tensorflow)

and lower the barrier to production FHE

Working group!

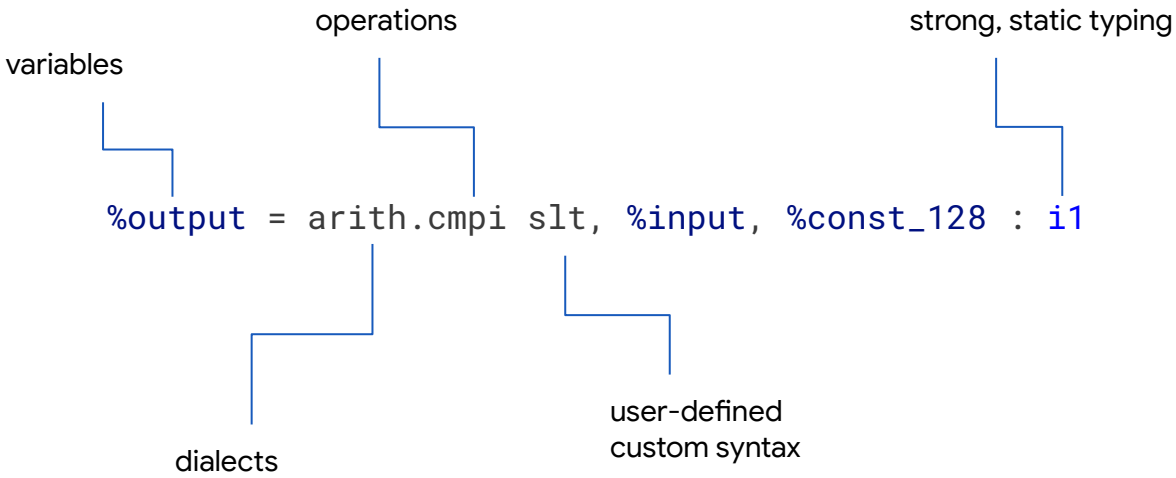
- heir.dev
- Open design meeting every 2 weeks (next: 2024-04-04)
- Holistic view: see FHE.org meetup talk (2023-10-19)
- Active projects:
  - TFHE/CGGI -> FPT FPGA
  - BGV -> ASICs (BASALISC and HERACLES via OpenFHE)
  - Porting features from other compilers like HECO
  - FHE schemes -> polynomial arithmetic
  - C++, Python, TensorFlow front-ends

📅 Meeting calendar 📅

02

MLIR Overview

HEIR: Homomorphic Encryption Intermediate Representation
MLIR: Multi-Level Intermediate Representation



MLIR: Multi Level Intermediate Representation

```
func.func @dist(%arg0: tensor<4xi16>, %arg1: tensor<4xi16>) -> i16 {
  %c0 = arith.constant 0 : index
  %c0_si16 = arith.constant 0 : i16
  %0 = affine.for %arg2 = 0 to 4 iter_args(%arg6 = %c0_si16) -> i16 {
    %1 = tensor.extract %arg0[%arg2]
    %2 = tensor.extract %arg1[%arg2]
    %3 = arith.subi %1, %2
    %4 = arith.muli %3, %3
    %5 = arith.addi %arg3, %4
    affine.yield %5
  }
  func.return %0 : i16
}
```

- Dialects coexist
- DSL for defining custom dialects
- Common framework for writing optimization passes, lowerings

Defining new dialects/ops/types

- Want to model SIMD computations
  - Reuse as much of existing MLIR as possible
  - Arithmetic FHE computation model
  - Scheme-agnostic
- Goal: reproduce rotation alignment optimizations from HECO
  - *HECO: Fully Homomorphic Encryption Compiler*

    Viand-Jattke-Haller-Hithnawi USENIX Security ‘23

The tensor_ext dialect

Tablegen definition (slightly abbreviated, for original see heir/pull/478)

```
def TensorExt_Dialect : Dialect {
  let name = "tensor_ext";
  let cppNamespace = "::mlir::heir::tensor_ext";
  let dependentDialects = ["tensor::TensorDialect"];
}

def TensorExt_RotateOp : Op<TensorExt_Dialect, ...> {
  let mnemonic = "rotate";
  let summary = "Rotate a tensor some number of indices left.";
  let arguments = (ins AnyTensor:$tensor, SignlessIntegerLike:$shift);
  let results = (outs AnyTensor:$output);
  let assemblyFormat = "operands attr-dict `:` qualified(type($tensor)) `,` type($shift)";
}
```

Example MLIR

```
func.func @test_rotate(%0: tensor<16xi32>) -> tensor<16xi32> {
  %c1 = arith.constant 1 : i32
  %1 = tensor_ext.rotate %0, %c1 : tensor<16xi32>, i32
  return %1 : tensor<16xi32>
}
```

The secret dialect

Abbreviated TableGen dialect definitions

```
def Secret_Dialect : Dialect {
  let name = "secret";
  let cppNamespace = "::mlir::heir::secret";
}

def Secret : TypeDef<Secret_Dialect, "Secret"> {
  let mnemonic = “secret”;
  let summary = "A secret value";
  let parameters = (ins "Type":$valueType);
  let assemblyFormat = "`<` $valueType `>`";
}

def Secret_GenericOp : Op<Secret_Dialect, "generic",
  traits = [SingleBlock, SingleBlockImplicitTerminator<"YieldOp">]> {
  let summary = "Lift a plaintext computation to operate on secrets.";
  let arguments = (ins Variadic<AnyType>:$inputs);
  let results = (outs Variadic<AnyType>:$results);
  let regions = (region AnyRegion:$region);
  let hasCustomAssemblyFormat = 1;
}
```

Example MLIR

```
%Z = secret.generic ins(%X, %Y :
  !secret.secret<i32>, !secret.secret<i32>) {
    ^bb0(%x: i32, %y: i32):
      %d = arith.addi %x, %y: i32
      secret.yield %d : i32
    } -> (!secret.secret<i32>)
```

Example pass pipeline

```
heir-opt --secretize=entry-function=dist --wrap-generic --canonicalize --full-loop-unroll --canonicalize
  --cse --insert-rotate --canonicalize --cse
```

Input

```
func.func @dist(%arg0: tensor<4xi16>,
                %arg1: tensor<4xi16>) -> i16 {
  %c0 = arith.constant 0 : index
  %c0_si16 = arith.constant 0 : i16
  %0 = affine.for %arg2 = 0 to 4
       iter_args(%arg3 = %c0_si16) -> i16 {
    %1 = tensor.extract %arg0[%arg2]
    %2 = tensor.extract %arg1[%arg2]
    %3 = arith.subi %1, %2
    %4 = arith.muli %3, %3
    %5 = arith.addi %arg3, %4
    affine.yield %5
  }
  return %0

  %output = arith.cmpi slt, %input, %const_128
}
```

Output

```
func.func @dist(
    %arg0: !secret.secret<tensor<4xi16>>,
    %arg1: !secret.secret<tensor<4xi16>>) -> !secret.secret<i16> {
  %c0_i16 = arith.constant 0 : i16
  %0 = secret.generic ins(%arg0, %arg1 :
       !secret.secret<tensor<4xi16>>, !secret.secret<tensor<4xi16>>) {
  ^bb0(%arg2: tensor<4xi16>, %arg3: tensor<4xi16>):
    %1 = affine.for %arg4 = 0 to 4 iter_args(%arg5 = %c0_i16) -> (i16) {
      %1 = tensor.extract %arg0[%arg4]
      %2 = tensor.extract %arg1[%arg4]
      %3 = arith.subi %1, %2
      %4 = arith.muli %3, %3
      %5 = arith.addi %arg5, %4
      affine.yield %5 : i16
    }
    secret.yield %1 : i16
  } -> !secret.secret<i16>
  return %0 : !secret.secret<i16>
}
```

Example pass pipeline

```
heir-opt --secretize=entry-function=dist --wrap-generic --canonicalize --full-loop-unroll --canonicalize
--cse --insert-rotate --canonicalize --cse
```

Input

```
func.func @dist(
    %arg0: !secret.secret<tensor<4xi16>>,
    %arg1: !secret.secret<tensor<4xi16>>) -> !secret.secret<i16> {
  %c0_i16 = arith.constant 0 : i16
  %0 = secret.generic ins(%arg0, %arg1 :
       !secret.secret<tensor<4xi16>>, !secret.secret<tensor<4xi16>>) {
  ^bb0(%arg2: tensor<4xi16>, %arg3: tensor<4xi16>):
    %1 = affine.for %arg4 = 0 to 4 iter_args(%arg5 = %c0_i16) -> (i16) {
      %1 = tensor.extract %arg0[%arg4]
      %2 = tensor.extract %arg1[%arg4]
      %3 = arith.subi %1, %2
      %4 = arith.muli %3, %3
      %5 = arith.addi %arg5, %4
      affine.yield %5 : i16
    }
    secret.yield %1 : i16
  } -> !secret.secret<i16>
  return %0 : !secret.secret<i16>
}
```

Output

```
func.func @dist(%arg0: !secret.secret<tensor<4xi16>>, %arg1:
!secret.secret<tensor<4xi16>>) -> !secret.secret<i16> {
  %c0 = arith.constant 0 : index
  %c1 = arith.constant 1 : index
  %c2 = arith.constant 2 : index
  %c3 = arith.constant 3 : index
  %0 = secret.generic ins(%arg0, %arg1 :
       !secret.secret<tensor<4xi16>>, !secret.secret<tensor<4xi16>>) {
  ^bb0(%arg2: tensor<4xi16>, %arg3: tensor<4xi16>):
    %extracted = tensor.extract %arg2[%c0] : tensor<4xi16>
    %extracted_0 = tensor.extract %arg3[%c0] : tensor<4xi16>
    %1 = arith.subi %extracted, %extracted_0 : i16
    %2 = arith.muli %1, %1 : i16
    %extracted_1 = tensor.extract %arg2[%c1] : tensor<4xi16>
    %extracted_2 = tensor.extract %arg3[%c1] : tensor<4xi16>
    %3 = arith.subi %extracted_1, %extracted_2 : i16
    %4 = arith.muli %3, %3 : i16
    ...
    secret.yield %11 : i16
  } -> !secret.secret<i16>
  return %0 : !secret.secret<i16>
}
```

Example pass pipeline

```
heir-opt --secretize=entry-function=dist --wrap-generic --canonicalize --full-loop-unroll --canonicalize
--cse --insert-rotate --canonicalize --cse
```

Input

```
func.func @dist(%arg0: !secret.secret<tensor<4xi16>>, %arg1:
!secret.secret<tensor<4xi16>>) -> !secret.secret<i16> {
  %c0 = arith.constant 0 : index
  %c1 = arith.constant 1 : index
  %c2 = arith.constant 2 : index
  %c3 = arith.constant 3 : index
  %0 = secret.generic ins(%arg0, %arg1 :
      !secret.secret<tensor<4xi16>>, !secret.secret<tensor<4xi16>>) {
  ^bb0(%arg2: tensor<4xi16>, %arg3: tensor<4xi16>):
    %extracted = tensor.extract %arg2[%c0] : tensor<4xi16>
    %extracted_0 = tensor.extract %arg3[%c0] : tensor<4xi16>
    %1 = arith.subi %extracted, %extracted_0 : i16
    %2 = arith.muli %1, %1 : i16
    %extracted_1 = tensor.extract %arg2[%c1] : tensor<4xi16>
    %extracted_2 = tensor.extract %arg3[%c1] : tensor<4xi16>
    %3 = arith.subi %extracted_1, %extracted_2 : i16
    %4 = arith.muli %3, %3 : i16
    ...
    secret.yield %11 : i16
  } -> !secret.secret<i16>
  return %0 : !secret.secret<i16>
}
```

Output

```
func.func @dist(%arg0: !secret.secret<tensor<4xi16>>, %arg1:
!secret.secret<tensor<4xi16>>) -> !secret.secret<i16> {
  %c2 = arith.constant 2 : index
  %c3 = arith.constant 3 : index
  %0 = secret.generic ins(%arg0, %arg1 :
!secret.secret<tensor<4xi16>>, !secret.secret<tensor<4xi16>>) {
  ^bb0(%arg2: tensor<4xi16>, %arg3: tensor<4xi16>):
    %1 = arith.subi %arg2, %arg3 : tensor<4xi16>
    %2 = arith.muli %1, %1 : tensor<4xi16>
    %3 = tensor_ext.rotate %2, %c3 : tensor<4xi16>, index
    %4 = arith.addi %3, %2 : tensor<4xi16>
    %5 = tensor_ext.rotate %4, %c2 : tensor<4xi16>, index
    %6 = arith.addi %5, %3 : tensor<4xi16>
    %7 = arith.addi %6, %2 : tensor<4xi16>
    %extracted = tensor.extract %7[%c3] : tensor<4xi16>
    secret.yield %extracted : i16
  } -> !secret.secret<i16>
  return %0 : !secret.secret<i16>
}
```

Example pass pipeline

```
heir-opt --secretize=entry-function=hamming --wrap-generic --canonicalize --full-loop-unroll --canonicalize
--cse --insert-rotate --canonicalize --cse ... --secret-to-bgv --bgv-specific-opt --bgv-to-openfhe --etc
```

You could go on!

But let's focus on `--insert-rotate`

03

Declarative Rewrite Rules

Simple patterns (canonicalize for `tensor_ext.rotate`)

```
// rotate %t, 0 -> %t
def DropZeroRotation : Pat<
  (TensorExt_RotateOp $tensor, (ConstantLikeMatcher APIntAttr:$c0)),
  (replaceWithValue $tensor),
  [(IsZero $c0)]
>;


// %0 = rotate %t, x
// %1 = rotate %0, y
// ---> rotate %t (x+y)
def CombineSequentialRotates : Pat<
  (TensorExt_RotateOp
    (TensorExt_RotateOp $tensor, (Arith_ConstantOp:$xOp APIntAttr:$x)),
    (Arith_ConstantOp:$yOp APIntAttr:$y)),
  (TensorExt_RotateOp $tensor, (Arith_AddIOp $xOp, $yOp)),
  []
>;
```

Added to --canonicalize

Simple patterns (canonicalize for `tensor_ext.rotate`)

```
// Reorder matching rotations with arith ops
foreach ArithOp = [Arith_AddIOp, Arith_SubIOp, Arith_MulIOp] in {
  def FactorParallelRotationsThroughOp_#ArithOp : Pat<
    (ArithOp
      (TensorExt_RotateOp $t1, $i),
      (TensorExt_RotateOp $t2, $i)),
    (TensorExt_RotateOp (ArithOp $t1, $t2), $i)
  >;
}
```

Greedy rewrite engine *repeatedly* applies patterns to a fixpoint

This pattern may put rotations next to each other, so other patterns can combine/eliminate them

More complex pattern (`insert-rotate`)

```
foreach InnerOp = [Arith_AddIOp, Arith_SubIOp, Arith_MulIOp] in {
  foreach OuterOp = [Arith_AddIOp, Arith_SubIOp, Arith_MulIOp] in {
    // (add (add (rotate t1 i1) (rotate t2 i2)) (rotate t3 i3))
    def AlignRotations_LeftAssociated_Inner_#InnerOp#_Outer_#OuterOp : Pattern<
      (OuterOp
        (InnerOp (TensorExt_RotateOp $t1, $i1), (TensorExt_RotateOp $t2, $i2)),
        (TensorExt_RotateOp $t3, $i3)),
      [
        (TensorExt_RotateOp:$r1 $t1, (Arith_SubIOp $i1, $i3)),
        (TensorExt_RotateOp:$r2 $t2, (Arith_SubIOp $i2, $i3)),
        (InnerOp:$addResult $r1, $r2),
        (OuterOp:$output $addResult, $t3),
        // Preserve the original rotation and let cse/canonicalization clean it
up
        (TensorExt_RotateOp $output, $i3),
      ]
    >;
  }
}
```

Using patterns in a pass (`insert-rotate`)

```
// in Tablegen
def InsertRotate : Pass<"insert-rotate"> {
  let summary = "Vectorize arithmetic FHE operations using HECO-style heuristics";
  let description = [{...}];
}

// In C++
#include "include/Dialect/TensorExt/Transforms/InsertRotate.cpp.inc"


struct InsertRotate : impl::InsertRotateBase<InsertRotate> {
  using InsertRotateBase::InsertRotateBase;

  void runOnOperation() override {
    MLIRContext *context = &getContext();
    RewritePatternSet patterns(context);
    populateWithGenerated(patterns);
    (void)applyPatternsAndFoldGreedily(getOperation(), std::move(patterns));
  }
};
```

04

Dataflow Analyses and Global Optimizations

When local rewrite rules aren’t enough

What if the rotation we want to align to is far away in the IR?

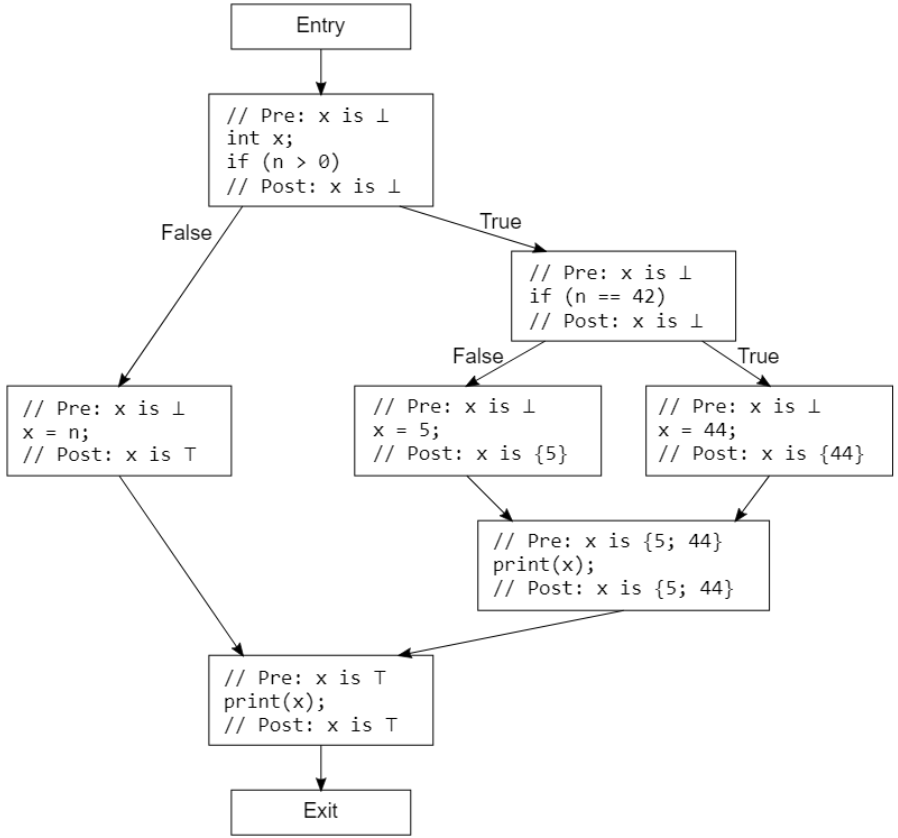
Two options:

1. Implement an **analysis pass** to compute & attach “target slot” to each program value
2. Fall back to arbitrary C++ code

Framework for common analyses: **dataflow analysis**

Dataflow Analysis

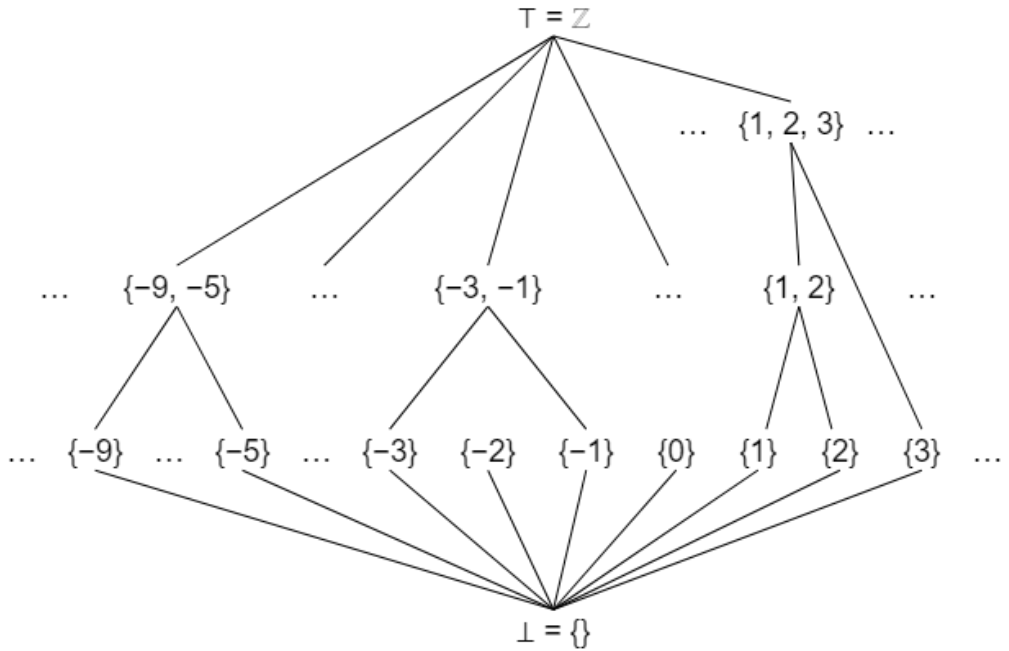
```
void Example(int n) {
  int x;          // x is ⊥
  if (n > 0) {
    if (n == 42) {
       x = 44;    // x is {44}
    } else {
       x = 5;     // x is {5}
    }
    print(x);     // x is {44; 5}
  } else {
    x = n;        // x is ⊤
  }
  print(x);       // x is ⊤
}
```



Source: https://clang.llvm.org/docs/DataFlowAnalysisIntro.html

Dataflow Analysis

```
void Example(int n) {
  int x;          // x is ⊥
  if (n > 0) {
    if (n == 42) {
       x = 44;    // x is {44}
    } else {
       x = 5;     // x is {5}
    }
    print(x);     // x is {44; 5}
  } else {
    x = n;        // x is ⊤
  }
  print(x);       // x is ⊤
}
```



Source: https://clang.llvm.org/docs/DataFlowAnalysisIntro.html

Dataflow Analysis

To define dataflow analysis:

- Define lattice data structure
- Implement op propagation rules
- Cf. https://github.com/google/heir/pull/526

```
class TargetSlot {
 private:
  std::optional<int64_t> value;
  static TargetSlot join(
      const TargetSlot &lhs, const TargetSlot &rhs) {
    return TargetSlot{
       lhs.getValue() < rhs.getValue()
          ? lhs.getValue() : rhs.getValue()};
  }
};

class TargetSlotLattice : public dataflow::Lattice<TargetSlot> {
 public:
  using Lattice::Lattice;
};

class TargetSlotAnalysis
    : public dataflow::SparseBackwardDataFlowAnalysis<TargetSlotLattice> {
  // Given lattice values of op results, set input operand lattice values.
  void visitOperation(...);
};
```

```
void TargetSlotAnalysis::visitOperation(Operation *op,
    ArrayRef<TargetSlotLattice *> operands,
    ArrayRef<const TargetSlotLattice *> results) {
  llvm::TypeSwitch<Operation &>(*op)
      .Case<tensor::InsertOp>([&](auto insertOp) {
        int64_t insertIndex =
           getConstExtractionIndex<tensor::InsertOp>(insertOp);
        TargetSlotLattice *insertedValueLattice = operands[0];
        TargetSlot newSlot = TargetSlot{insertIndex};
        ChangeResult changed = insertedValueLattice->join(newSlot);
        propagateIfChanged(insertedValueLattice, changed);
      })
      ...
      });
}
```

Using the Dataflow Analysis

```
foreach ArithOp = [Arith_AddIOp, Arith_SubIOp, Arith_MulIOp] in {
  def InsertRotations_#ArithOp : Pattern<
    (ArithOp:$arithOp
      (Tensor_ExtractOp $t1, $i1),
      (Tensor_ExtractOp $t2, $i2)),
    [
      (TensorExt_RotateOp:$r1 $t1,
          (Arith_SubIOp $i1, (Arith_ConstantOp (GetTargetSlotAttr $arithOp)))),
      (TensorExt_RotateOp:$r2 $t2,
          (Arith_SubIOp $i2, (Arith_ConstantOp (GetTargetSlotAttr $arithOp)))),
      (ArithOp:$opResult $r1, $r2),
      (Tensor_ExtractOp
        $opResult, (Arith_ConstantOp (GetTargetSlotAttr $arithOp))),
    ]
  >;
}
```

```
// Get the target_slot attribute from an op, or else zero
def GetTargetSlotAttr : NativeCodeCall<
      "$0.getOwner()->hasAttr(\"target_slot\")"
      " ? llvm::cast<mlir::IntegerAttr>($0.getOwner()->getAttr(\"target_slot\"))"
      " : $_builder.getIndexAttr(0)">;
```

Must inject C++ code, + populate the attributes with the analysis in C++.
Not hard, but not slide-worthy. See https://github.com/google/heir/pull/530

Global Optimization (`yosys-optimizer`)

Input:

```
func.func @add_one(%in: !secret.secret<i8>) -> (!secret.secret<i8>) {
  %one = arith.constant 1 : i8
  %1 = secret.generic ins(%in, %one: !secret.secret<i8>, i8) {
      ^bb0(%IN: i8, %ONE: i8) :
          %2 = arith.addi %IN, %ONE : i8
          secret.yield %2 : i8
      } -> (!secret.secret<i8>)
  return %1 : !secret.secret<i8>
}
```

```
// In C++
struct YosysOptimizer :
impl::YosystOptimizerBase<YosysOpimizer> {
  using YosystOptimizerBase::YosystOptimizerBase;

  void runOnOperation() override {
    // build Yosys input, execute yosys, parse output
    ...
  }
};
```

See YosysOptimizer.cpp

Output:

```
func.func @add_one(%arg0: !secret.secret<i8>) -> !secret.secret<i8> {
  %c0 = arith.constant 0 : index
  ...
  %c7 = arith.constant 7 : index
  %0 = secret.cast %arg0 : !secret.secret<i8> to !secret.secret<memref<8xi1>>
  %1 = secret.generic ins(%0 : !secret.secret<memref<8xi1>>) {
  ^bb0(%arg1: memref<8xi1>):
    %3 = memref.load %arg1[%c1] : memref<8xi1>
    %4 = memref.load %arg1[%c0] : memref<8xi1>
    %5 = comb.truth_table %false, %3, %4 -> 6 : ui8
    %6 = comb.truth_table %false, %false, %4 -> 1 : ui8
    %7 = memref.load %arg1[%c2] : memref<8xi1>
    %8 = comb.truth_table %7, %3, %4 -> 120 : ui8
    %9 = comb.truth_table %7, %3, %4 -> 128 : ui8
    ...
    %20 = memref.load %arg1[%c7] : memref<8xi1>
    %21 = comb.truth_table %false, %20, %19 -> 6 : ui8
    %alloc = memref.alloc() : memref<8xi1>
    memref.store %6, %alloc[%c0] : memref<8xi1>
    ...
    memref.store %21, %alloc[%c7] : memref<8xi1>
    secret.yield %alloc : memref<8xi1>
  } -> !secret.secret<memref<8xi1>>
  %2 = secret.cast %1 : !secret.secret<memref<8xi1>> to !secret.secret<i8>
  return %2 : !secret.secret<i8>
}
```

05

Resources

MLIR For Beginners (github.com/j2kun/mlir-tutorial)

A series of articles on the MLIR framework for building compilers.

1. Build System (Getting Started)
2. Running and Testing a Lowering
3. Writing Our First Pass
4. Using Tablegen for Passes
5. Defining a New Dialect
6. Using Traits
7. Folders and Constant Propagation
8. Verifiers
9. Canonicalizers and Declarative Rewrite Patterns
10. Dialect Conversion
11. Lowering through LLVM
12. A Global Optimization and Dataflow Analysis
13. More to come!

github.com/google/heir

heir.dev/community/

Join us!