DialEgg

Dialect-Agnostic MLIR Optimizer using Egglog

EGRAPHS Community Meeting

August 21, 2025

Abd-El-Aziz Zayed and Christophe Dubach

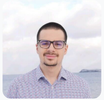
McGill

Equality Saturation in the CASL Lab

PI Christophe Dubach

Jonathan Van der Cruysse
Latent idiom recognition with a minimalist IR using EqSat

Tzung-Han Juang
EqSat for hardware resource sharing

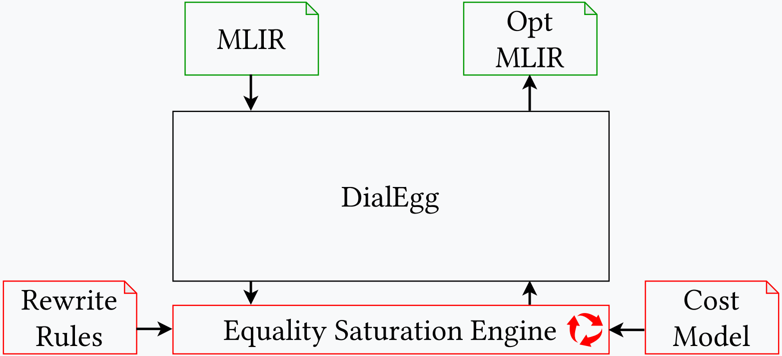


Adam Musa
Driving EqSat with RL

Aziz Zayed (me)
EqSat and egraphs in modern compilers like MLIR

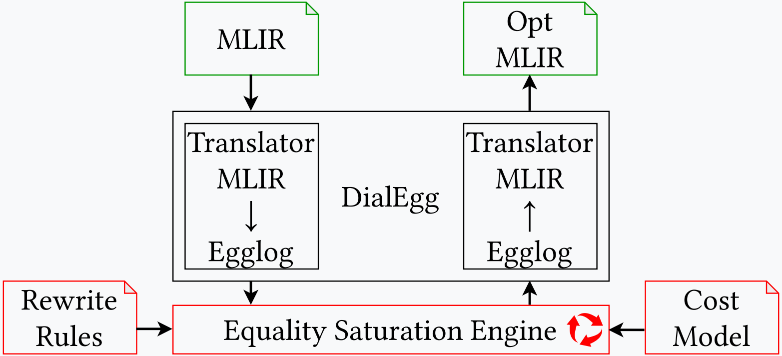
DialEgg

The first successful attempt to integrate equality saturation with MLIR across abstraction layers.

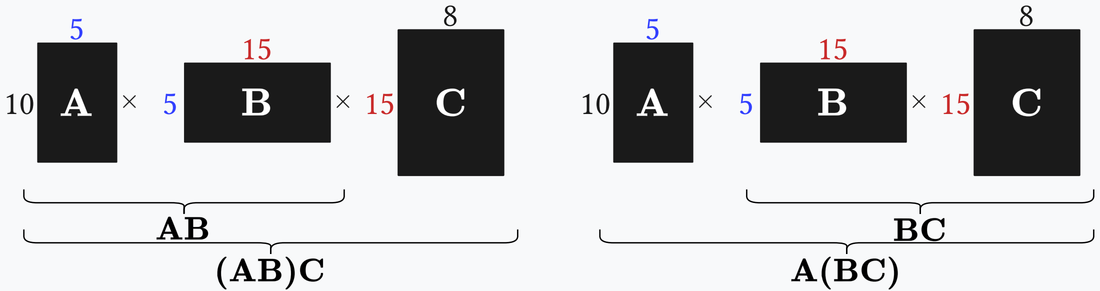


DialEgg

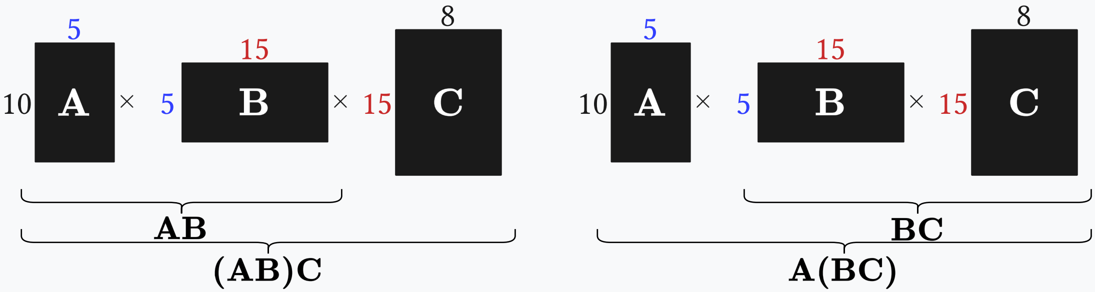
The first successful attempt to integrate equality saturation with MLIR across abstraction layers.



MatMul is Associative: $(AB)\,C = A\,(BC)$

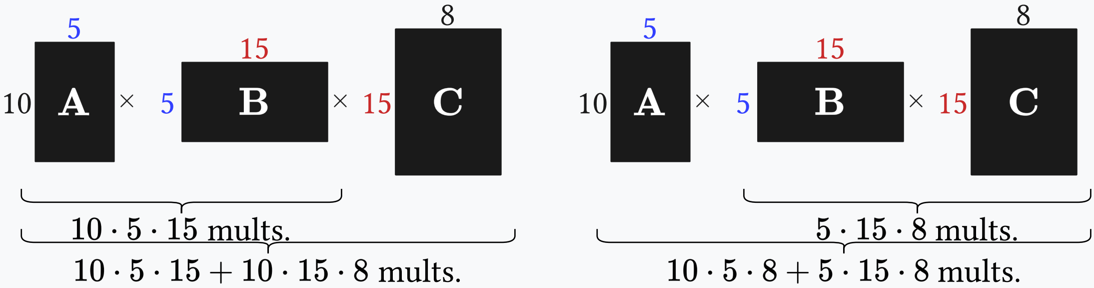


MatMul is Associative: $(AB)\,C = A\,(BC)$



How do we find out which one is more efficient?

The Order of Operations Matters



$\texttt{cost}\ (\mathbf{AB})\,\mathbf{C} = 1950$ mults.

$\texttt{cost}\ \mathbf{A}\,(\mathbf{BC}) = 1000$ mults.

Associativity Rewrite

How we want to express the rewrite $(\mathbf{XY})\,\mathbf{Z} \Leftrightarrow \mathbf{X}\,(\mathbf{YZ})$:

```
cost (MatMul X Y) = (nrows X) * (ncols X) * (ncols Y)
rewrite (MatMul (MatMul X Y) Z) <=> (MatMul X (MatMul Y Z))
```

Repeatedly apply the rewrite to the AST until a fixed-point is reached.

Associativity Rewrite

How we want to express the rewrite $\mathbf{(XY)\,Z} \Leftrightarrow \mathbf{X\,(YZ)}$:

```
cost (MatMul X Y) = (nrows X) * (ncols X) * (ncols Y)
rewrite (MatMul (MatMul X Y) Z) <=> (MatMul X (MatMul Y Z))
```

Repeatedly apply the rewrite to the AST until a fixed-point is reached.

Associativity Rewrite

How we want to express the rewrite **(XY) Z** $\Leftrightarrow$ **X (YZ)**:

```
cost (MatMul X Y) = (nrows X) * (ncols X) * (ncols Y)
rewrite (MatMul (MatMul X Y) Z) <=> (MatMul X (MatMul Y Z))
```

Repeatedly apply the rewrite to the AST until a fixed-point is reached.

Associativity Rewrite in MLIR

```
class MMAssociateRewritePattern : public OpRewritePattern<MatmulOp> {
public:
    MMAssociateRewritePattern(MLIRContext* context) : OpRewritePattern<MatmulOp>(context) {}

    LogicalResult matchAndRewrite(MatmulOp op, PatternRewriter& rewriter) const override {
        Location loc = op.getLoc();
        Value lhs = op.getOperand(0);
        Value rhs = op.getOperand(1);
        Operation* lhsOp = lhs.getDefiningOp();
        Operation* rhsOp = rhs.getDefiningOp();

        Value x, y, z;
        bool is_x_yz = false, is_xy_z = false;
        if (lhsOp != nullptr && isa<MatmulOp>(lhsOp)) {
            is_xy_z = true;
            x = lhsOp->getOperand(0);  // a*b
            y = lhsOp->getOperand(1);  // b*c
            z = rhs;                   // c*d
        } else if (rhsOp != nullptr && isa<MatmulOp>(rhsOp)) {
            is_x_yz = true;
            x = lhs;                   // a*b
            y = rhsOp->getOperand(0);  // b*c
            z = rhsOp->getOperand(1);  // c*d
        } else {
            return failure();  // this is just a normal matmul (XY)
        }
```

Pattern matcher

Associativity Rewrite in MLIR

```
class MMAssociateRewritePattern : public OpRewritePattern<MatmulOp> {
public:
    MMAssociateRewritePattern(MLIRContext* context) : OpRewritePattern<MatmulOp>(context) {}
```

```
RankedTensorType xType = cast<RankedTensorType>(x.getType());
RankedTensorType yType = cast<RankedTensorType>(y.getType());
RankedTensorType zType = cast<RankedTensorType>(z.getType());
Type opType = xType.getElementType();

if (xType.getNumDynamicDims() > 0 || yType.getNumDynamicDims() > 0 || zType.getNumDynamicDims() > 0) {
    return failure();
}

int64_t a = xType.getShape()[0];
int64_t b = xType.getShape()[1];
int64_t c = yType.getShape()[1];
int64_t d = zType.getShape()[1];

int64_t x_yzCost = b * c * d + a * b * d;
int64_t xy_zCost = a * b * c + a * c * d;
```

Cost function

```
    } else {
        return failure(); // this is just a normal matmul (XY)
    }
```

Pattern matcher

Associativity Rewrite in MLIR

```cpp
class MMAssociateRewritePattern : public OpRewritePattern<MatmulOp> {
public:
        if (xy_zCost < x_yzCost && is_x_yz) {  // (XY)Z is better than X(YZ)
            Type xyType = RankedTensorType::get({a, c}, opType);
            Value xyInit = rewriter.create<EmptyOp>(loc, xyType, ValueRange {});
            MatmulOp xy = rewriter.create<MatmulOp>(loc, xyType, ValueRange {x, y}, xyInit);
            Type xyzType = RankedTensorType::get({a, d}, opType);
            Value xyzInit = rewriter.create<EmptyOp>(loc, xyzType, ValueRange {});
            MatmulOp xyz = rewriter.create<MatmulOp>(loc, xyzType, ValueRange {xy.getResult(0), z}, xyzInit);
            rewriter.replaceOp(op, xyz.getResult(0));
        } else if (x_yzCost < xy_zCost && is_xy_z) {  // X(YZ) is better than (XY)Z
            Type yzType = RankedTensorType::get({b, d}, opType);
            Value yzInit = rewriter.create<EmptyOp>(loc, yzType, ValueRange {});
            MatmulOp yz = rewriter.create<MatmulOp>(loc, yzType, ValueRange {y, z}, yzInit);
            Type xyzType = RankedTensorType::get({a, d}, opType);
            Value xyzInit = rewriter.create<EmptyOp>(loc, xyzType, ValueRange {});
            MatmulOp xyz = rewriter.create<MatmulOp>(loc, xyzType, ValueRange {x, yz.getResult(0)}, xyzInit);
            rewriter.replaceOp(op, xyz.getResult(0));
        }  // if equal, just keep the original

        return success();
    }
};
```

Rewriter

Associativity Rewrite in MLIR

```
class MMAssociateRewritePattern : public OpRewritePattern<MatmulOp> {
public:
```

```
    if (xy_zCost < x_yzCost && is_x_yz) {  // (XY)Z is better than X(YZ)
```

```
class MMAssociatePass : public PassWrapper<MMAssociatePass, OperationPass<FuncOp>> {
public:
    StringRef getArgument() const override { return "mm-assoc"; }
    StringRef getDescription() const override { return "Associates matmul operations."; }

    void runOnOperation() override {
        FuncOp func = getOperation();
        MLIRContext* context = &getContext();
        RewritePatternSet patterns(context);
        patterns.add<MMAssociateRewritePattern>(context);

        GreedyRewriteConfig config = GreedyRewriteConfig()
                                     .setStrictness(GreedyRewriteStrictness::ExistingAndNewOps)
                                     .setUseTopDownTraversal();
        if (failed(applyPatternsAndFoldGreedily(func, std::move(patterns), config))) {
            signalPassFailure();
        }
    }
};
```

Pass

Associativity Rewrite in Egglog with DialEgg

```
(rule [(= A (Value ?id ?t))] [(set (type-of A) ?t)])
(rule [(= A (linalg_matmul ?x ?y ?o ?t))] [(set (type-of A) ?t)])

(function nrows (Type) i64)
(function ncols (Type) i64)
(rule [(= ?t (RankedTensor ?shape ?tp))]
    [(set (nrows ?t) (vec-get ?shape 0))
     (set (ncols ?t) (vec-get ?shape 1))])

(rule ; cost[XY] = nrows(X) * ncols(X) * ncols(Y)
    [(= xy linalg_matmul ?x ?y ?t)]
    [(cost xy (* (* (nrows (type-of ?x)) (ncols (type-of ?x))) (ncols (type-of ?y))))])

(rule ; (xy)z -> x(yz)
    [(= xy (linalg_matmul ?x ?y (RankedTensor ?_ ?t)))
     (= xy_z (linalg_matmul xy ?z ?xyz_t))]

    [(let yz (linalg_matmul ?y ?z (RankedTensor (vec-of (nrows (type-of ?y)) (ncols (type-of ?z))) ?t)))
     (let x_yz (linalg_matmul ?x yz ?xyz_t))
     (union xy_z x_yz)])
```

Associativity Rewrite in Egglog with DialEgg

Cost function

```
(rule [(= A (Value ?id ?t))] [(set (type-of A) ?t)])
(rule [(= A (linalg_matmul ?x ?y ?o ?t))] [(set (type-of A) ?t)])

(function nrows (Type) i64)
(function ncols (Type) i64)
(rule [(= ?t (RankedTensor ?shape ?tp))]
    [(set (nrows ?t) (vec-get ?shape 0))
     (set (ncols ?t) (vec-get ?shape 1))])

(rule ; cost[XY] = nrows(X) * ncols(X) * ncols(Y)
    [(= xy linalg_matmul ?x ?y ?t)]
    [(cost xy (* (* (nrows (type-of ?x)) (ncols (type-of ?x))) (ncols (type-of ?y))))])

(rule ; (xy)z -> x(yz)
    [(= xy (linalg_matmul ?x ?y (RankedTensor ?_ ?t)))
     (= xy_z (linalg_matmul xy ?z ?xyz_t))]

    [(let yz (linalg_matmul ?y ?z (RankedTensor (vec-of (nrows (type-of ?y)) (ncols (type-of ?z))) ?t)))
     (let x_yz (linalg_matmul ?x yz ?xyz_t))
     (union xy_z x_yz)])
```

Associativity Rewrite in Egglog with DialEgg

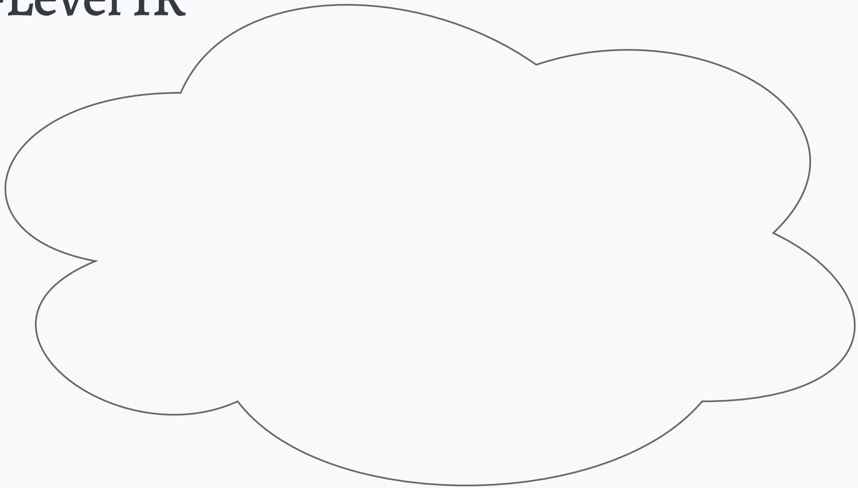
Cost function

```
(rule [(= A (Value ?id ?t))] [(set (type-of A) ?t)])
(rule [(= A (linalg_matmul ?x ?y ?o ?t))] [(set (type-of A) ?t)])

(function nrows (Type) i64)
(function ncols (Type) i64)
(rule [(= ?t (RankedTensor ?shape ?tp))]
    [(set (nrows ?t) (vec-get ?shape 0))
     (set (ncols ?t) (vec-get ?shape 1))])

(rule ; cost[XY] = nrows(X) * ncols(X) * ncols(Y)
    [(= xy linalg_matmul ?x ?y ?t)]
    [(cost xy (* (* (nrows (type-of ?x)) (ncols (type-of ?x))) (ncols (type-of ?y))))])
```

Rewrite rule

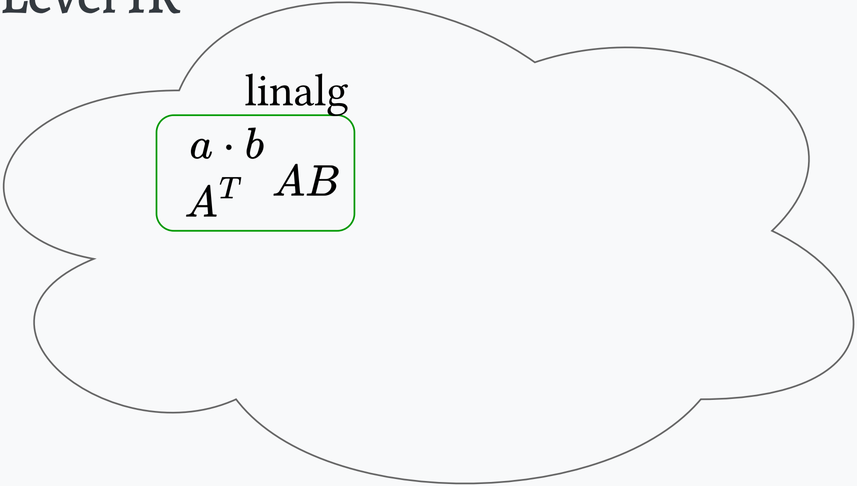
```
(rule ; (xy)z -> x(yz)
    [(= xy (linalg_matmul ?x ?y (RankedTensor ?_ ?t)))
     (= xy_z (linalg_matmul xy ?z ?xyz_t))]

    [(let yz (linalg_matmul ?y ?z (RankedTensor (vec-of (nrows (type-of ?y)) (ncols (type-of ?z))) ?t)))
     (let x_yz (linalg_matmul ?x yz ?xyz_t))
     (union xy_z x_yz)])
```

Multi-Level IR



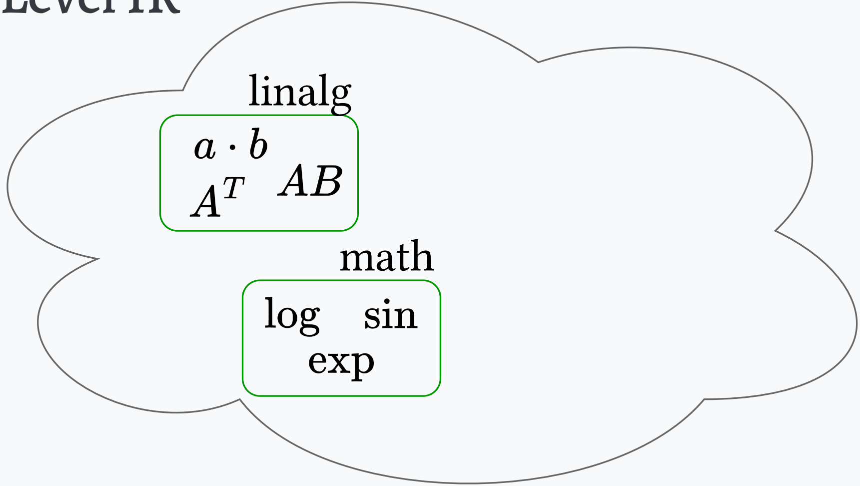
How do we represent many domains and abstraction levels in one IR?

Multi-Level IR



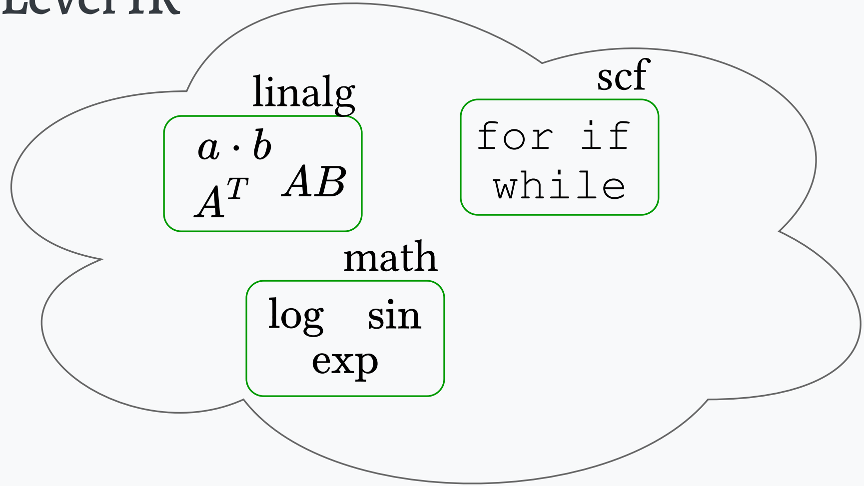
How do we represent many domains and abstraction levels in one IR?

Multi-Level IR



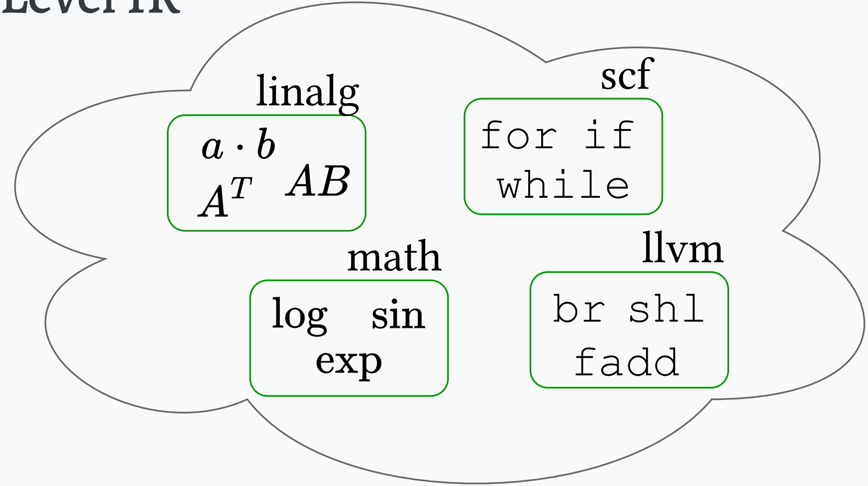
How do we represent many domains and abstraction levels in one IR?

Multi-Level IR



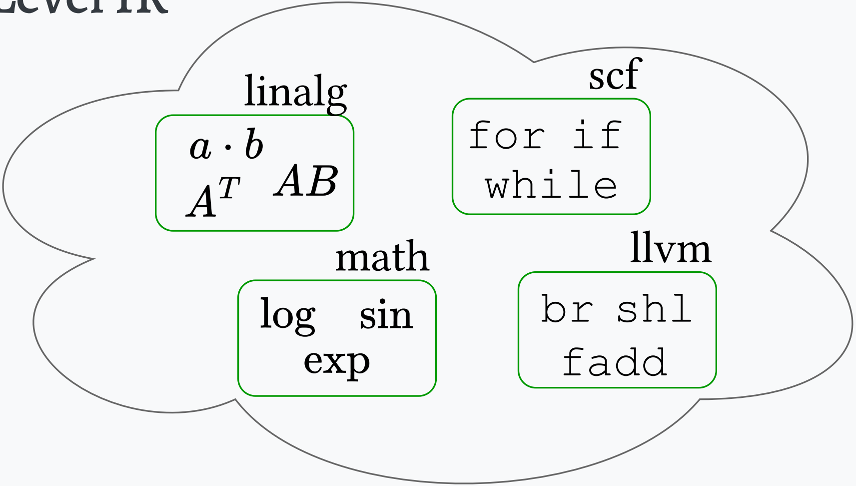
How do we represent many domains and abstraction levels in one IR?

Multi-Level IR



Abstraction layer ≈ IR level ≈ MLIR dialect.

Multi-Level IR



And many more: `func`, `arith`, `gpu`, etc.

MLIR Operations

- Secret sauce:
  recursive operation
  structure

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

Function to print a vector of size 5.

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

Function to print a vector of size 5.

```
func.func {sym="tensor_print",
           type=tensor<5xi64> -> ()}
{

}
```

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

Function to print a vector of size 5.

```
func.func {sym="tensor_print",
           type=tensor<5xi64> -> ()}
{
  %c0 = arith.constant {value=0}: index
  %c5 = arith.constant {value=5}: index


}
```

MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

Function to print a vector of size 5.

```
func.func {sym="tensor_print",
           type=tensor<5xi64> -> ()}
{
  %c0 = arith.constant {value=0}: index
  %c5 = arith.constant {value=5}: index
  scf.for %i = %c0 to %c5
  {
  }
}
```

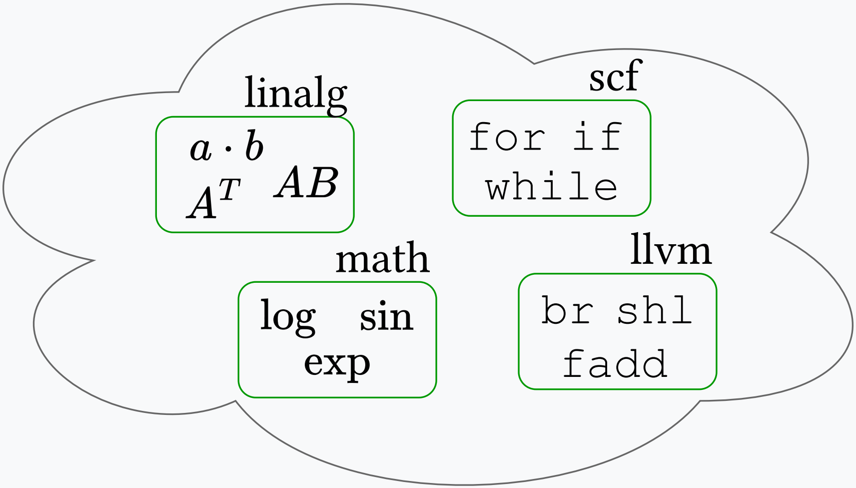
MLIR Operations

- Secret sauce: recursive operation structure
  - Operands and results
  - Attributes
  - Regions
    - Blocks (more ops!)

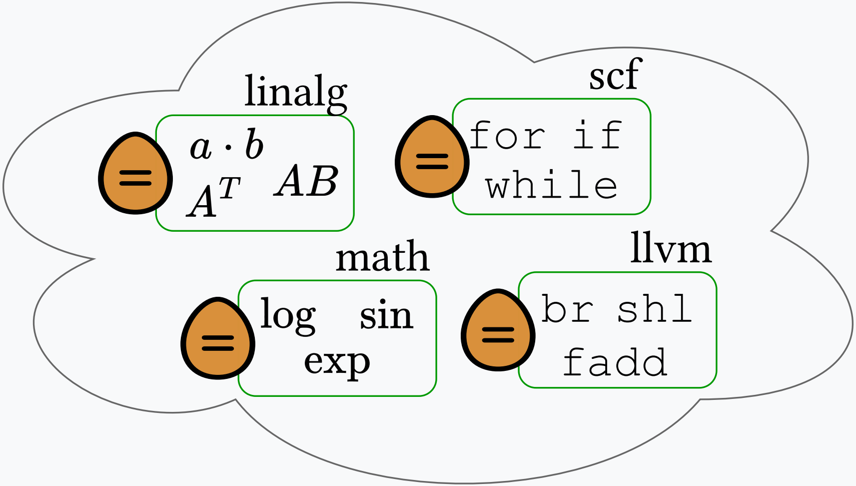
Function to print a vector of size 5.

```
func.func {sym="tensor_print",
           type=tensor<5xi64> -> ()}
{
 %c0 = arith.constant {value=0}: index
 %c5 = arith.constant {value=5}: index
 scf.for %i = %c0 to %c5
 {
  %e = tensor.extract %t[%i]: tensor<5xi64>
  func.call(%e){callee="print_i64"}: (i64)->()
 }
}
```

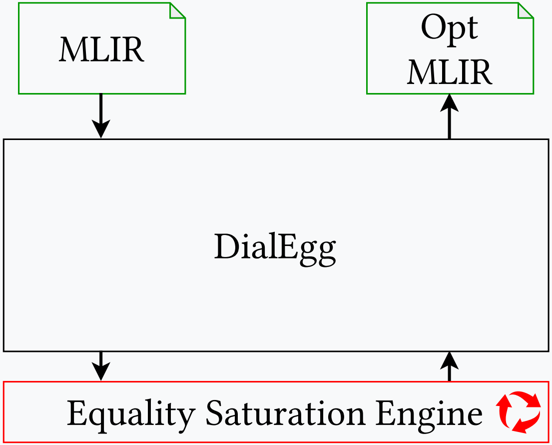
DialEgg: A Dialect-Agnostic MLIR Optimizer



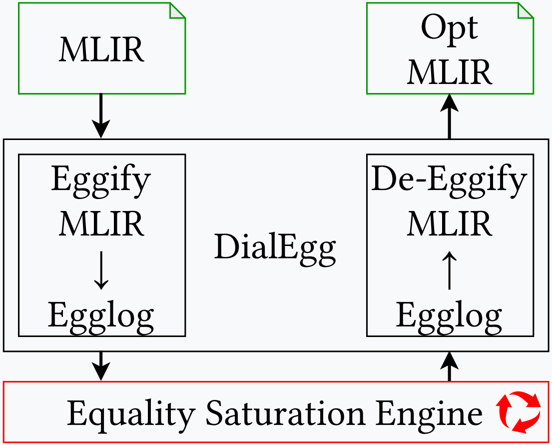
DialEgg: A Dialect-Agnostic MLIR Optimizer



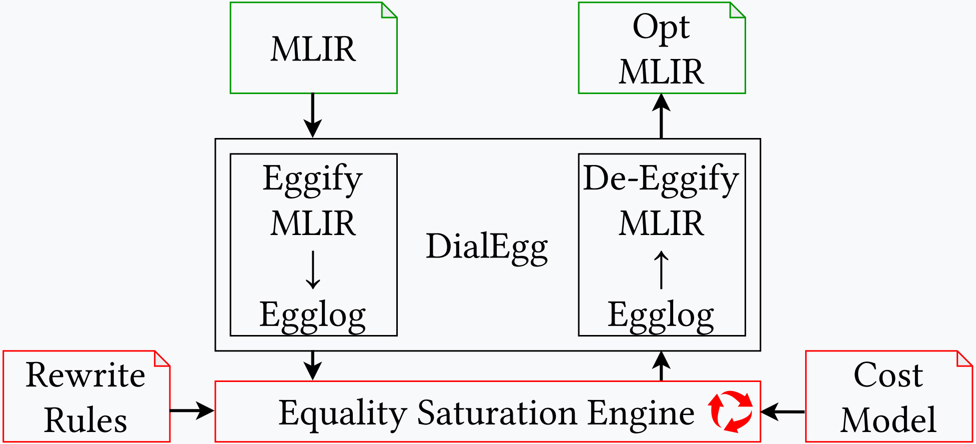
DialEgg



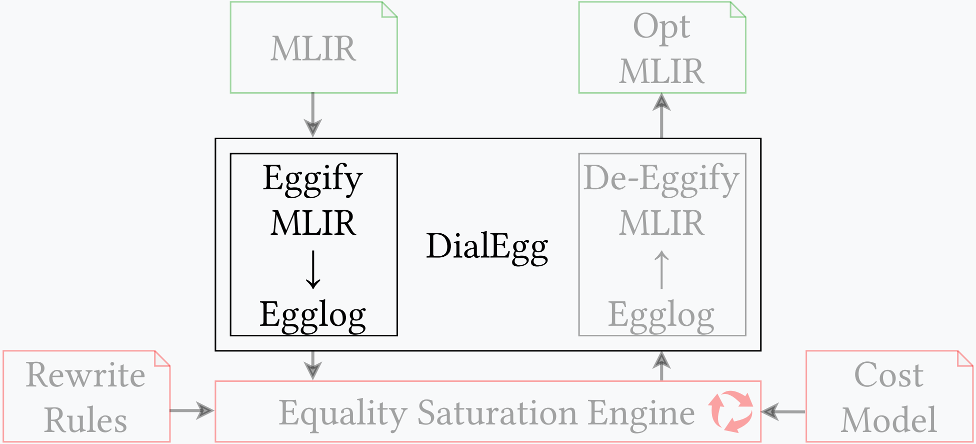
DialEgg


MLIR
Opt
MLIR
Eggify
MLIR
Egglog
DialEgg
De-Eggify
MLIR
Egglog
Equality Saturation Engine

DialEgg


MLIR
Opt MLIR
Eggify MLIR
Egglog
DialEgg
De-Eggify MLIR
Egglog
Rewrite Rules
Equality Saturation Engine
Cost Model

MLIR → Egglog


MLIR
Opt MLIR
Eggify MLIR ↓ Egglog
DialEgg
De-Eggify MLIR ↑ Egglog
Rewrite Rules
Equality Saturation Engine
Cost Model

MLIR → Egglog: Types

MLIR types are eggified to variants of the Type ADT in Egglog.

```
(datatype Type
    (F32)
    (I64)
    (Tensor IntVec Type)
    (OpaqueType String String) ; (type, value)
    ...
)
```

MLIR → Egglog: Types

MLIR types are eggified to variants of the Type ADT in Egglog.

```
(datatype Type
    (F32)
    (I64)
    (Tensor IntVec Type)
    (OpaqueType String String) ; (type, value)
    ...
)
```

Tensor Type in Egglog

```
(Tensor (vec-of 2 3) (I64)) ; tensor<2x3xi64>
```

MLIR → Egglog: Attributes

MLIR attributes are eggified to variants of the `Attr` ADT in Egglog.

```
(datatype Attr
    (IAttr i64 Type)
    (DenseIntElementsAttr IntVec Type)
    (OpaqueAttr String String) ; (name, value)
    ...
)
(datatype NamedAttrT (NamedAttr String Attr))
```

MLIR → Egglog: Attributes

MLIR attributes are eggified to variants of the `Attr` ADT in Egglog.

```
(datatype Attr
    (IAttr i64 Type)
    (DenseIntElementsAttr IntVec Type)
    (OpaqueAttr String String) ; (name, value)
    ...
)
(datatype NamedAttrT (NamedAttr String Attr))
```

Constant operation in MLIR

```
arith.const {value=5: i64} : i64
```

```
(NamedAttr "value" (IAttr 5 (I64)))
```

MLIR → Egglog: Operations

Operations are eggified piece by piece to variants of the `Op` ADT.

```
(datatype Op
    (Value i64 Type) ; (id, type)
    (arith_const NamedAttrT Type)
    (arith_addi Op Op Type)
    (linalg_matmul Op Op Type)
    (scf_if Op Region Region Type)
    ...
)
```

MLIR → Egglog: Operations

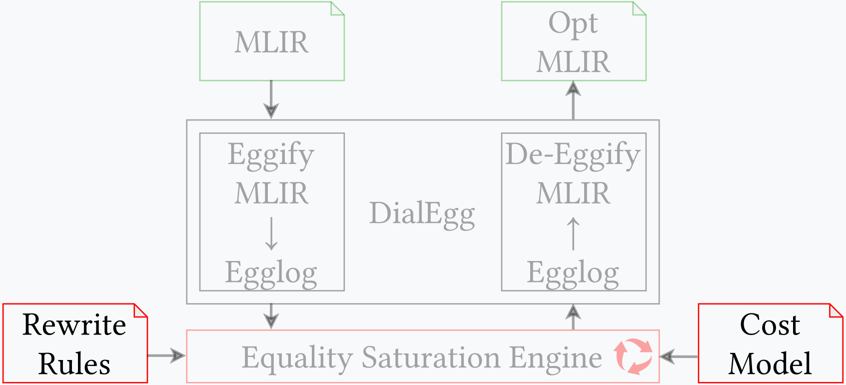
Operations are eggified piece by piece to variants of the `Op` ADT.

```
(datatype Op
    (Value i64 Type) ; (id, type)
    (arith_const NamedAttrT Type)
    (arith_addi Op Op Type)
    (linalg_matmul Op Op Type)
    (scf_if Op Region Region Type)
    ...
)
```

MatMul Operation in Egglog

```
(linalg_matmul A B (Tensor (vec-of 20 10) (I64)))
```

MM Associativity Rewrite Rule in Egglog



MM Associativity Rewrite Rule in Egglog

$$(\mathbf{XY})\,\mathbf{Z} \Rightarrow \mathbf{X}\,(\mathbf{YZ})$$

where $\mathbf{X}$: $a \times b$, $\mathbf{Y}$: $b \times c$, and $\mathbf{Z}$: $c \times d$

```
(rule
 [(= XY (linalg_matmul ?X ?Y (Tensor ?_ ?t)))
  (= XY_Z (linalg_matmul XY ?Z ?XYZ_t))
  (= b (nrows type-of ?Y))
  (= d (ncols type-of ?Z))]

 [(let YZ (linalg_matmul ?Y ?Z (Tensor (vec-of b d) ?t)))
  (let X_YZ (linalg_matmul ?X YZ ?XYZ_t))
  (union XY_Z X_YZ)])
)
```

MM Associativity Rewrite Rule in Egglog

$$(\mathbf{XY})\,\mathbf{Z} \Rightarrow \mathbf{X}\,(\mathbf{YZ})$$

where $\mathbf{X}$: $a \times b$, $\mathbf{Y}$: $b \times c$, and $\mathbf{Z}$: $c \times d$

```
(rule
 [(= XY (linalg_matmul ?X ?Y (Tensor ?_ ?t)))
  (= XY_Z (linalg_matmul XY ?Z ?XYZ_t))
  (= b (nrows type-of ?Y))
  (= d (ncols type-of ?Z))]

 [(let YZ (linalg_matmul ?Y ?Z (Tensor (vec-of b d) ?t)))
  (let X_YZ (linalg_matmul ?X YZ ?XYZ_t))
  (union XY_Z X_YZ)])
)
```

MM Associativity Cost Model in Egglog

$$\texttt{cost}\ \mathbf{XY} = abc$$

where $\mathbf{X}$: $a \times b$ and $\mathbf{Y}$: $b \times c$

```
(rule
  [(= XY (linalg_matmul ?X ?Y ?XY_t))
   (= a (nrows type-of ?X))
   (= b (ncols type-of ?X))
   (= c (ncols type-of ?Y))]

  [(cost XY (* (* a b) c))]
)
```

MM Associativity Cost Model in Egglog

$$\texttt{cost}\ \mathbf{XY} = abc$$
$$\text{where } \mathbf{X}\text{: } a \times b \text{ and } \mathbf{Y}\text{: } b \times c$$

```
(rule
  [(= XY (linalg_matmul ?X ?Y ?XY_t))
   (= a (nrows type-of ?X))
   (= b (ncols type-of ?X))
   (= c (ncols type-of ?Y))]

  [(cost XY (* (* a b) c))]
)
```

Optimized 3MM

```
func.func @3mm(%A,%B,%C,%D) {
 %AB  =linalg.matmul %A   %B
 %ABC =linalg.matmul %AB  %C
 %ABCD=linalg.matmul %ABC %D
 func.return %ABCD
}
```

$\texttt{cost}\ ((\mathbf{AB})\ \mathbf{C})\ \mathbf{D} = 10,000$ mults.

Optimized 3MM

```
func.func @3mm(%A,%B,%C,%D) {
 %AB  =linalg.matmul %A   %B
 %ABC =linalg.matmul %AB  %C
 %ABCD=linalg.matmul %ABC %D
 func.return %ABCD
}
```

$\texttt{cost}\ ((\mathbf{AB})\,\mathbf{C})\,\mathbf{D} = 10,000$ mults.

```
func.func @3mm(%A,%B,%C,%D) {
 %CD  =linalg.matmul %C %D
 %BCD =linalg.matmul %B %CD
 %ABCD=linalg.matmul %A %BCD
 func.return %ABCD
}
```

$\texttt{cost}\ \mathbf{A}\,(\mathbf{B}\,(\mathbf{CD})) = 5,000$ mults.

Case Study: Polynomial Evaluation

Reduce from $O(n^2)$ to $O(n)$ multiplications using Horner's method.

$$
\begin{aligned}
P(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\
&= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n)))
\end{aligned}
$$

Case Study: Polynomial Evaluation

Reduce from $O(n^2)$ to $O(n)$ multiplications using Horner's method.

$$
\begin{aligned}
P(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\
&= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n)))
\end{aligned}
$$

- Exponentiation: $\underline{x^0 \Rightarrow 1}$ and $\underline{x^n \Leftrightarrow x \cdot x^{n-1}}$

Case Study: Polynomial Evaluation

Reduce from $O(n^2)$ to $O(n)$ multiplications using Horner's method.

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n))) \end{aligned}$$

- Exponentiation: $\underline{x^0 \Rightarrow 1}$ and $\underline{x^n \Leftrightarrow x \cdot x^{n-1}}$
- Commutativity: $\underline{x + y \Leftrightarrow y + x}$ and $\underline{x \cdot y \Leftrightarrow y \cdot x}$
- Associativity: $\underline{(x + y) + z \Leftrightarrow x + (y + z)}$ and $\underline{(x \cdot y) \cdot z \Leftrightarrow x \cdot (y \cdot z)}$
- Distributivity: $\underline{x \cdot (y + z) \Leftrightarrow x \cdot y + x \cdot z}$
- Identity: $\underline{x \cdot 1 \Rightarrow x}$

Case Study: Polynomial Evaluation

- Exponentiation: $x^n \Leftrightarrow x \cdot x^{n-1}$

```
(rule
 [(= n1 (arith_const (NamedAttr "value" (IAttr ?n)) ?t))
  (= x1 (math_ipowi ?x n1 ?t)) ; x^n
  (>= ?n 1)]

 [(let n2 (arith_const (NamedAttr "value" (IAttr (- ?n 1))) ?t))
  (let x2 (math_ipowi ?x n2 ?t)) ; x^{n-1}
  (let x3 (arith_muli ?x x2 ?t)) ; x * x^{n-1}
  (union x1 x3)]
)
```

Compile Time and Scalability

Bench.	#Rules	#Ops	MLIR ↓↑ Egglog	Egglog	Saturation
Img Conv	1	29	0.4ms	14.6ms	<0.1ms
Vec Norm	1	44	0.5ms	21.6ms	<0.1ms
Poly	8	26	0.5ms	18.9ms	2ms
3MM	5	8	0.3ms	8.7ms	1ms
10MM	5	22	0.5ms	14.4ms	4ms
20MM	5	42	1.0ms	41.3ms	23ms
40MM	5	82	1.8ms	296.2ms	235ms
80MM	5	162	7.3ms	4939.3ms	3732ms

DialEgg

- More case studies in the paper:
  - Image conversion: rewrites with computation.
  - Vector normalization: rewrites matching on MLIR attributes.

azizzayed.com

DialEgg paper

MM Associativity Helpers

```
(rule ((= ?A (Value ?id ?t))) ((set (type-of ?A) ?t)))
(rule ((= ?A (linalg_matmul ?x ?y ?t))) ((set (type-of ?A) ?t)))

(function nrows (Type) i64)
(function ncols (Type) i64)
(rule ((= ?t (RankedTensor ?dims ?tp)))
      ((set (nrows ?t) (vec-get ?dims 0))
       (set (ncols ?t) (vec-get ?dims 1))))

(rule ((linalg_matmul ?x ?y ?t)
       (= a (nrows (type-of ?x)))
       (= b (ncols (type-of ?x)))
       (= c (ncols (type-of ?y))))

    ((cost (linalg_matmul ?x ?y ?t) (* (* a b) c))))
```

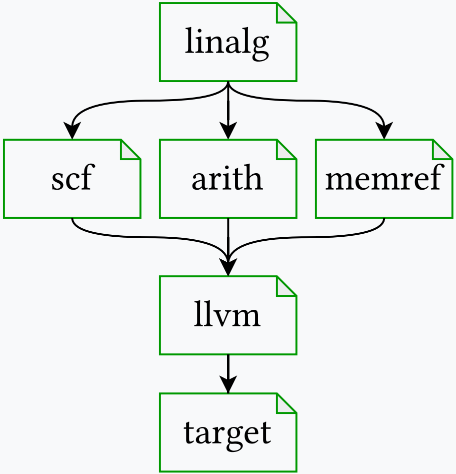
Poly Rewrite Rules

```
(rewrite (arith_addi ?x ?y ?a ?t) (arith_addi ?y ?x ?a ?t))
(rewrite (arith_muli ?x ?y ?a ?t) (arith_muli ?y ?x ?a ?t))
(rewrite ; (x + y) + z = x + (y + z)
  (arith_addi (arith_addi ?x ?y ?a ?t) ?z ?a ?t)
  (arith_addi ?x (arith_addi ?y ?z ?a ?t) ?a ?t))
(rewrite ; (x * y) * z = x * (y * z)
  (arith_muli (arith_muli ?x ?y ?a ?t) ?z ?a ?t)
  (arith_muli ?x (arith_muli ?y ?z ?a ?t) ?a ?t))

(rewrite (arith_muli ?x ; x * 1 = x
    (arith_const (NamedAttr "value" (IAttr 1.0 ?t)) ?t)
  ?a ?t) ?x)
(rewrite (math_ipowi ?x ; x^0 = 1
  (arith_const (NamedAttr "value" (IAttr 0.0 ?t)) ?t) ?a ?t)
  (arith_const (NamedAttr "value" (IAttr 1.0 ?t)) ?t))
```

Poly Rewrite Rules

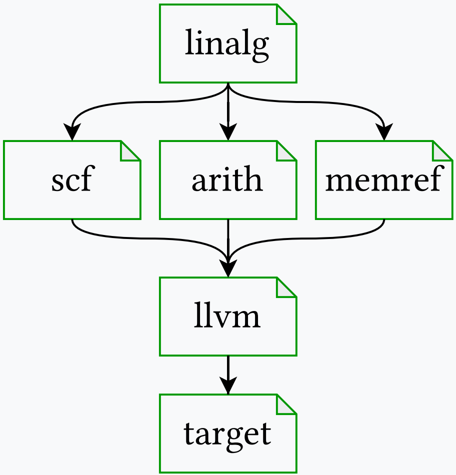
```
(rewrite ; mx + nx = x(m + n)
  (arith_addi (arith_muli ?m ?x ?a ?t) (arith_muli ?n ?x ?a ?t)
      ?a ?t)
  (arith_muli ?x (arith_addi ?m ?n ?a ?t) ?a ?t))

(rule
 [(= n1 (arith_const (NamedAttr "value" (IAttr ?n)) ?t))
  (= x1 (math_ipowi ?x n1 ?t)) ; x^n
  (>= ?n 1)]

 [(let n2 (arith_const (NamedAttr "value" (IAttr (- ?n 1))) ?t))
  (let x2 (math_ipowi ?x n2 ?t)) ; x^{n-1}
  (let x3 (arith_muli ?x x2 ?t)) ; x * x^{n-1}
  (union x1 x3)]
)
```

MLIR Lowering and Code Generation



Lower to a lower level of abstraction.

MLIR Lowering and Code Generation



Lower to a lower level of abstraction.

```
// X: tensor<5x10xi64>
// Y: tensor<10x3xi64>
%XY = linalg.matmul %X %Y
```

MLIR Lowering and Code Generation

```
// X: tensor<5x10xi64>
// Y: tensor<10x3xi64>
%XY = linalg.matmul %X %Y
```

MLIR Lowering and Code Generation

```
// X: tensor<5x10xi64>
// Y: tensor<10x3xi64>
%XY = linalg.matmul %X %Y
```

MatMul with scf.for

```
%XY = memref.alloc(): memref<5x3xi64>
scf.for %i = %0 to %5 {
  scf.for %j = %0 to %3 {
    scf.for %k = %0 to %10 {
      %X_ik = memref.load %X[%i, %k]
      %Y_kj = memref.load %Y[%k, %j]
      %XY_ij = memref.load %XY[%i, %j]
      %a = arith.muli %X_ik, %Y_kj
      %b = arith.addi %XY_ij, %a
      memref.store %b, %XY[%i, %j]
    }
  }
}
```