DialEgg

Dialect-Agnostic MLIR Optimizer using Equality Saturation with Egglog

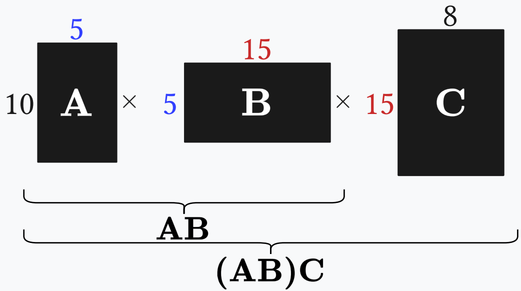
CGO 2025

March 3, 2025

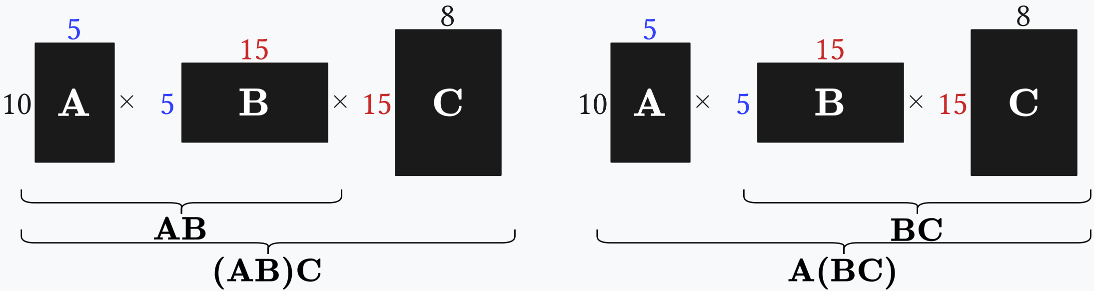
Abd-El-Aziz Zayed and Christophe Dubach

McGill

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

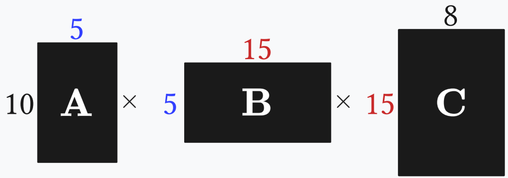


MatMul is Associative: $(AB)\,C = A\,(BC)$



How do we find out which one is more efficient?

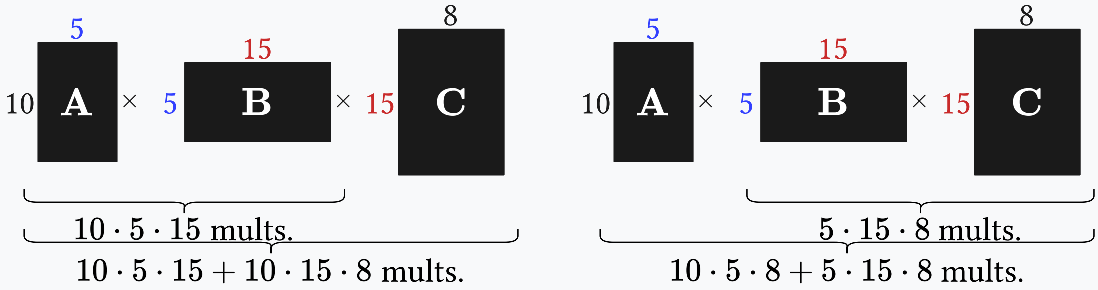
The Order of Operations Matters



$$\underbrace{\underbrace{A \times B}_{10 \cdot 5 \cdot 15 \text{ mults.}} \times C}_{10 \cdot 5 \cdot 15 + 10 \cdot 15 \cdot 8 \text{ mults.}}$$

`cost` **(AB) C** $= 1950$ multiplications.

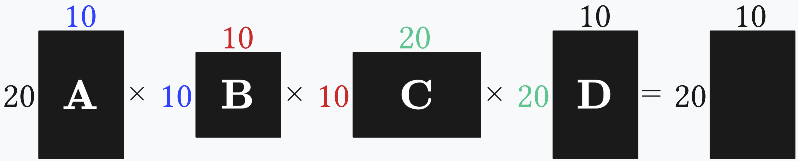
The Order of Operations Matters



`cost` **(AB) C** = 1950 multiplications.

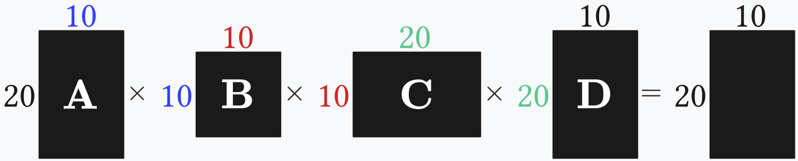
`cost` **A (BC)** = 1000 multiplications.

Cost of 3MM



- cost $((\mathbf{AB})\,\mathbf{C})\,\mathbf{D}$ $= 10,000$ multiplications.
- cost $(\mathbf{A}\,(\mathbf{BC}))\,\mathbf{D}$ $= 10,000$ multiplications.
- cost $\mathbf{A}\,((\mathbf{BC})\,\mathbf{D})$ $= 6,000$ multiplications.
- cost $\mathbf{A}\,(\mathbf{B}\,(\mathbf{CD}))$ $= 5,000$ multiplications.
- cost $(\mathbf{AB})\,(\mathbf{CD})$ $= 6,000$ multiplications.

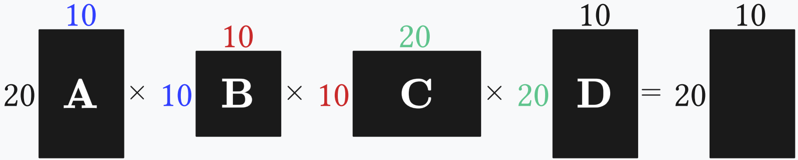
((AB) C) D in MLIR



Why Multi-Level IR?

- Perform high-level operations like matrix multiplication.
- Great for domain-specific optimizations.
- Domain-specific operations are grouped into MLIR dialects.
- MatMul in the linear algebra dialect: `linalg.matmul`.

((AB) C) D in Pseudo-MLIR

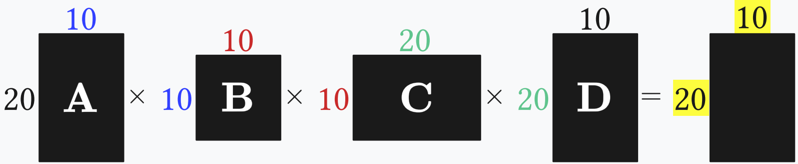



```
func.func @3mm(%A: tensor<20x10xi64>, %B: tensor<10x10xi64>,
               %C: tensor<10x20xi64>, %D: tensor<20x10xi64>)
                -> tensor<20x10xi64> {

    %AB   = linalg.matmul %A   %B -> tensor<20x10xi64>
    %ABC  = linalg.matmul %AB  %C -> tensor<20x20xi64>
    %ABCD = linalg.matmul %ABC %D -> tensor<20x10xi64>

    func.return %ABCD
}
```

((AB) C) D in Pseudo-MLIR

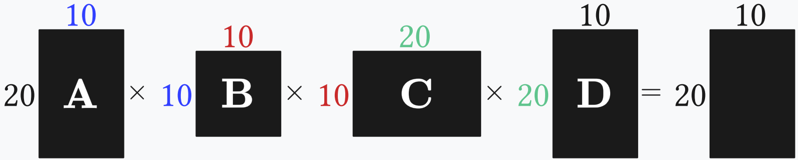



```
func.func @3mm(%A: tensor<20x10xi64>, %B: tensor<10x10xi64>,
               %C: tensor<10x20xi64>, %D: tensor<20x10xi64>)
                -> tensor<20x10xi64> {

    %AB   = linalg.matmul %A   %B -> tensor<20x10xi64>
    %ABC  = linalg.matmul %AB  %C -> tensor<20x20xi64>
    %ABCD = linalg.matmul %ABC %D -> tensor<20x10xi64>

    func.return %ABCD
}
```

((AB) C) D in Pseudo-MLIR




```
func.func @3mm(%A: tensor<20x10xi64>, %B: tensor<10x10xi64>,
               %C: tensor<10x20xi64>, %D: tensor<20x10xi64>)
                -> tensor<20x10xi64> {

    %AB   = linalg.matmul %A   %B -> tensor<20x10xi64>
    %ABC  = linalg.matmul %AB  %C -> tensor<20x20xi64>
    %ABCD = linalg.matmul %ABC %D -> tensor<20x10xi64>

    func.return %ABCD
}
```

Associativity Rewrite in MLIR

How can we express the rewrite **(XY) Z** $\Leftrightarrow$ **X (YZ)** in an MLIR pass?

- >120 lines of C++ code.
- Hard to write and maintain.

Likewise for the cost model.

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 ⇔ 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.

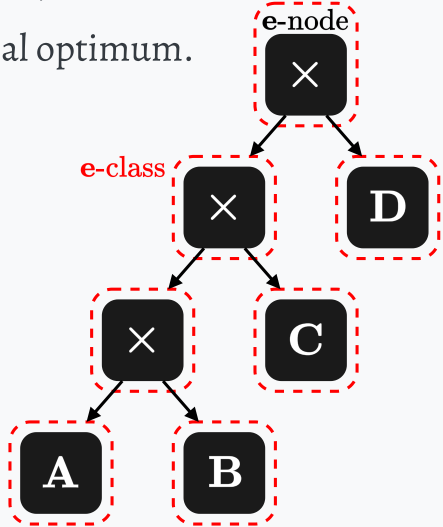
Equality Saturation for ((AB) C) D

Use **e**quivalence graphs to find the global optimum.

Equality Saturation for ((AB) C) D

Use **e**quivalence graphs to find the global optimum.

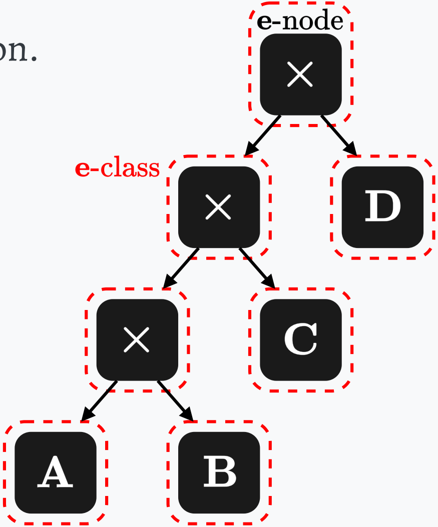
1. Build an **e**-graph of **((AB) C) D**
2. Apply the rewrite rule **(XY) Z** $\Leftrightarrow$ **X (YZ)** to the e-graph until a fixed-point.
3. Extract the optimized expression via the cost model $\texttt{cost}\ \mathbf{XY} =$ $\text{nrows}(\mathbf{X}) \cdot \text{ncols}(\mathbf{X}) \cdot \text{ncols}(\mathbf{Y})$



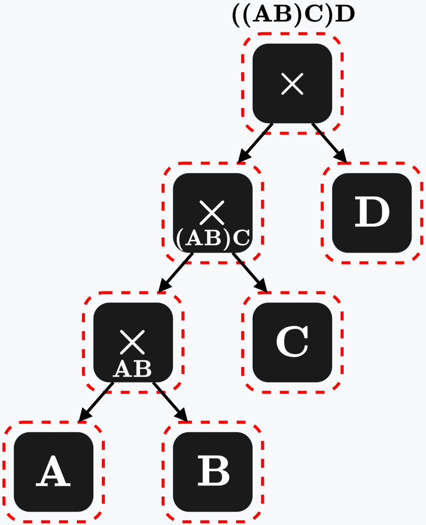
1. Build an e-graph of ((AB) C) D

AST-like representation of the expression.

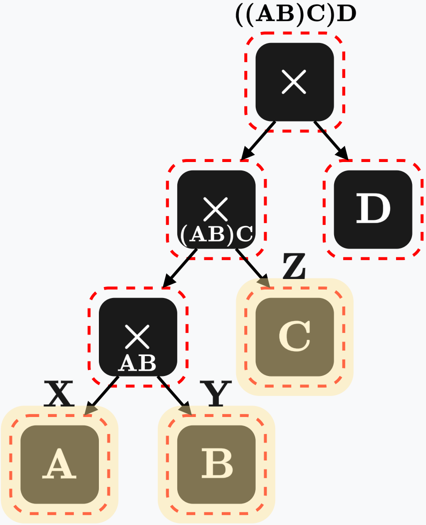
1. **e-node**: Operation node.
2. **e-class**: Set of equivalent e-nodes.
3. **e-graph**: Set of e-classes, capturing equivalent versions of a program compactly.



2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)



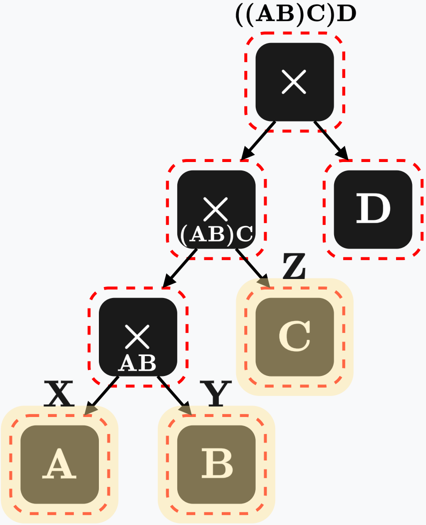
2. Exhaustively Apply the Rewrite $(XY)\,Z \Leftrightarrow X\,(YZ)$

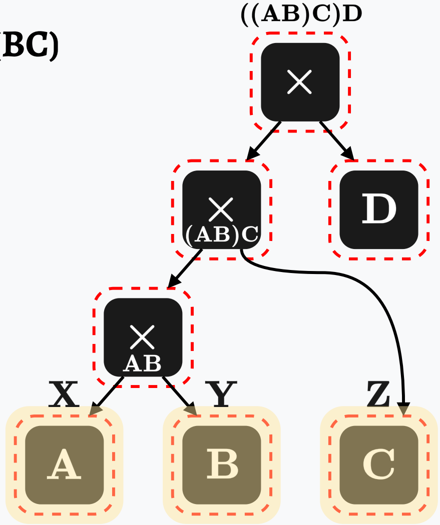


$$(AB)\,C = A\,(BC)$$

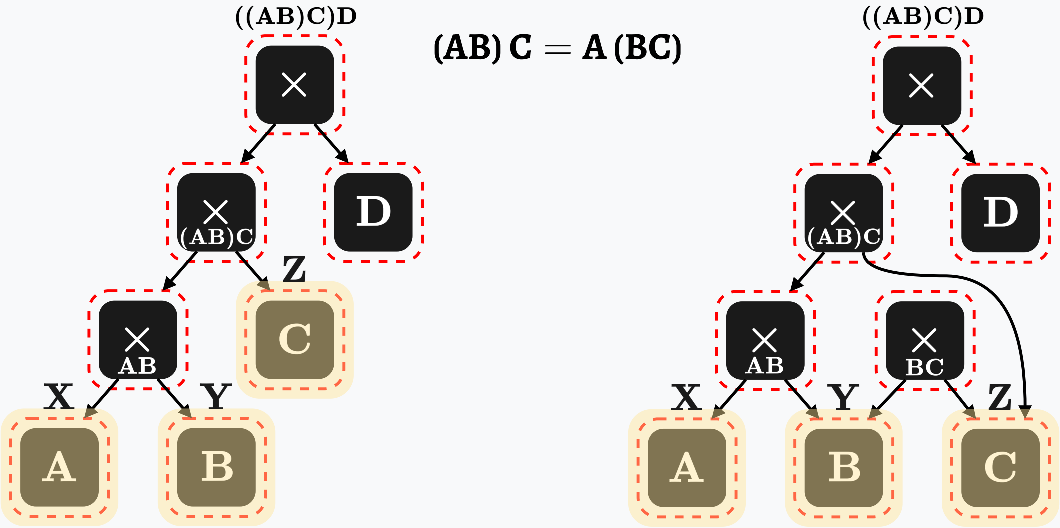
2. Exhaustively Apply the Rewrite $(XY)\,Z \Leftrightarrow X\,(YZ)$

$$(AB)\,C = A\,(BC)$$

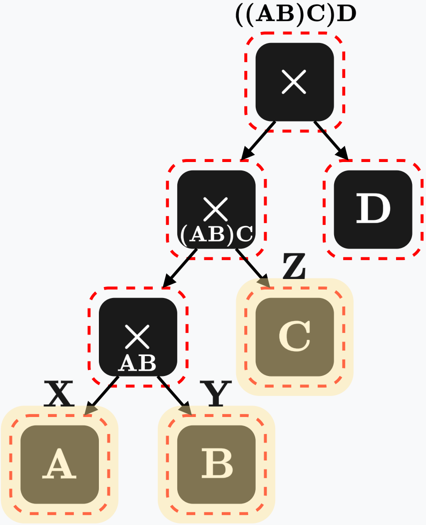




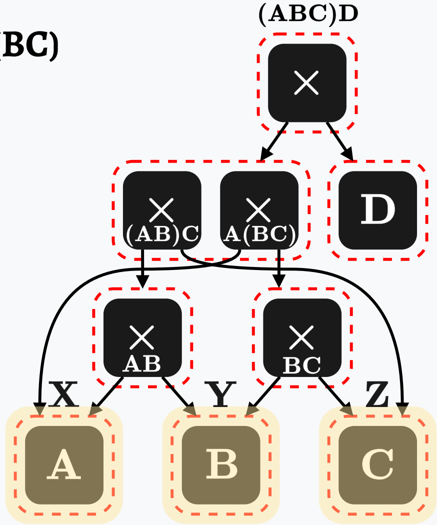
2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)



2. Exhaustively Apply the Rewrite $(XY)\,Z \Leftrightarrow X\,(YZ)$

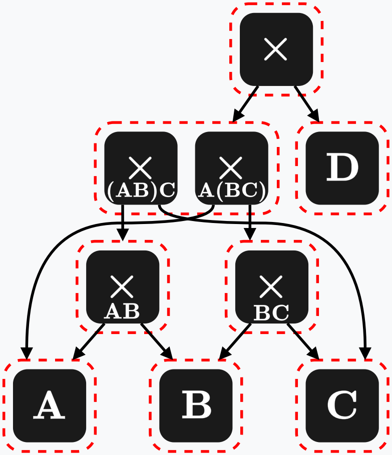


$$(AB)\,C = A\,(BC)$$

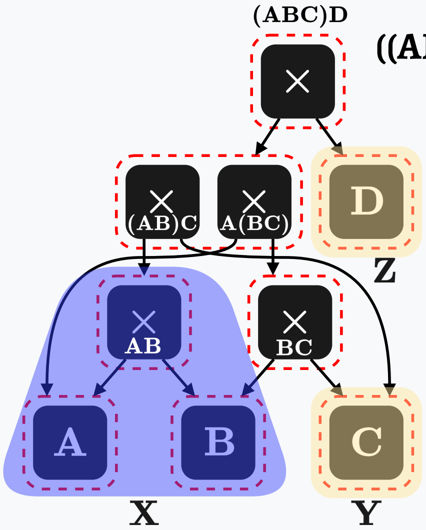


2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)

(ABC)D



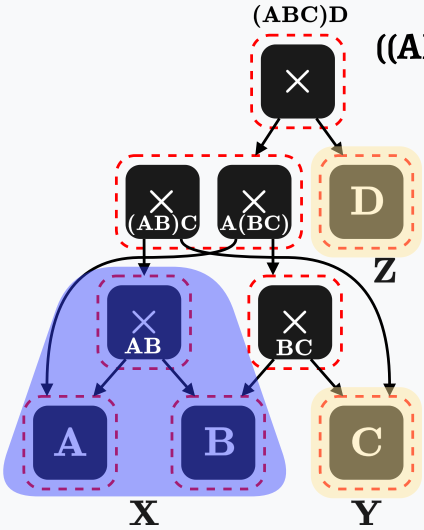
2. Exhaustively Apply the Rewrite (XY) Z $\Leftrightarrow$ X (YZ)

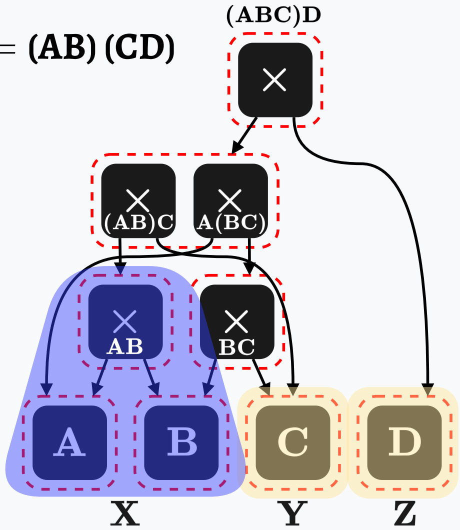


$$((AB)\,C)\;\;D = (AB)\,(CD)$$

2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)

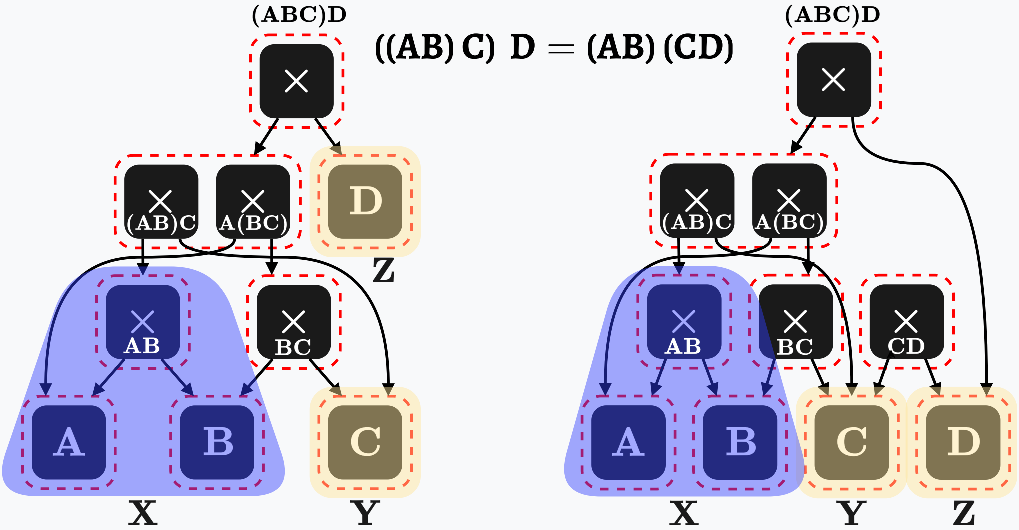
$$((AB)\,C)\;D = (AB)\,(CD)$$





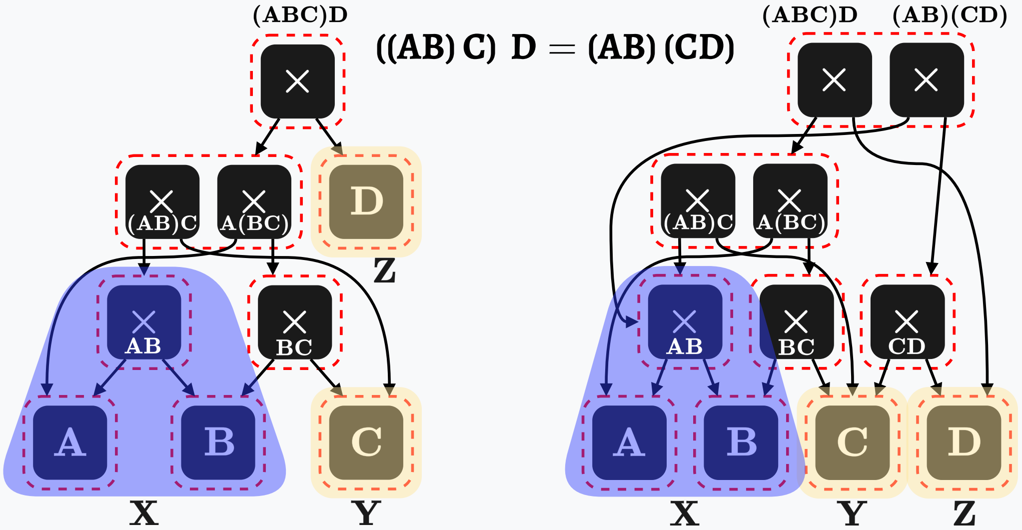
2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)

((AB) C) D = (AB) (CD)

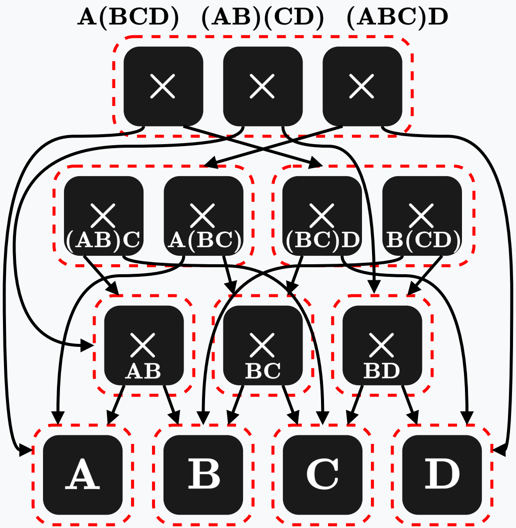


2. Exhaustively Apply the Rewrite (XY) Z ⇔ X (YZ)

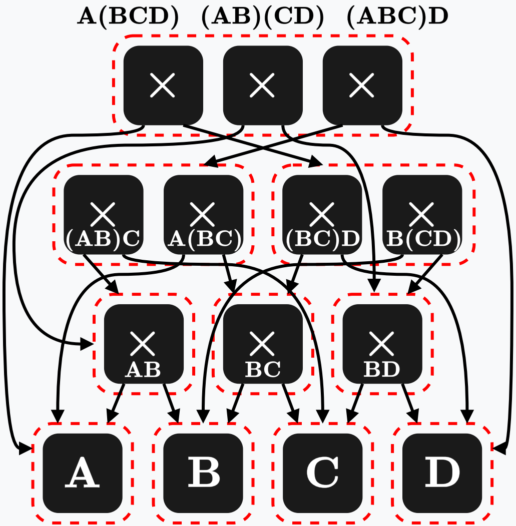
$$((AB)\,C)\;D = (AB)\,(CD)$$



Final E-Graph After Saturation

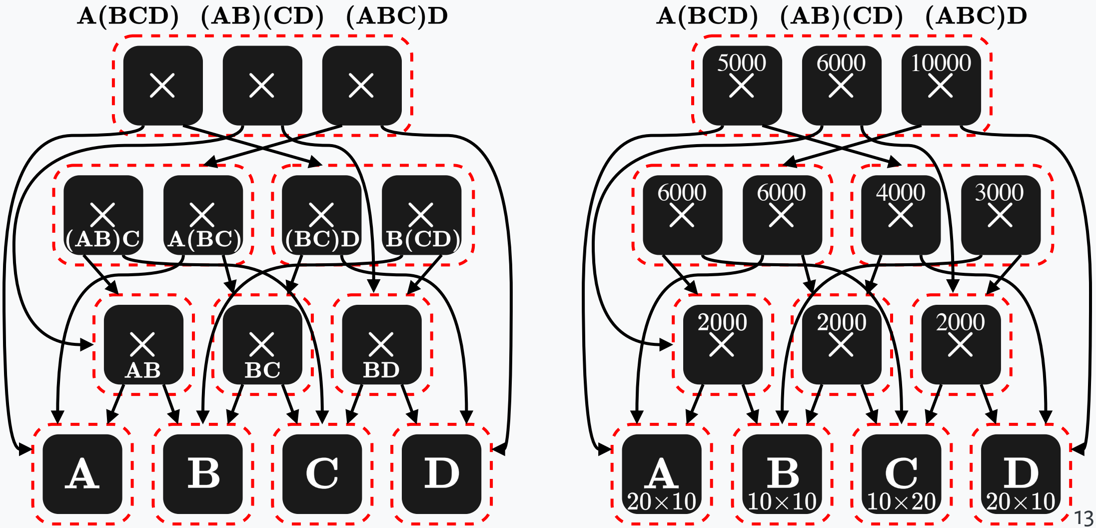


3. Extract the Optimal Expression

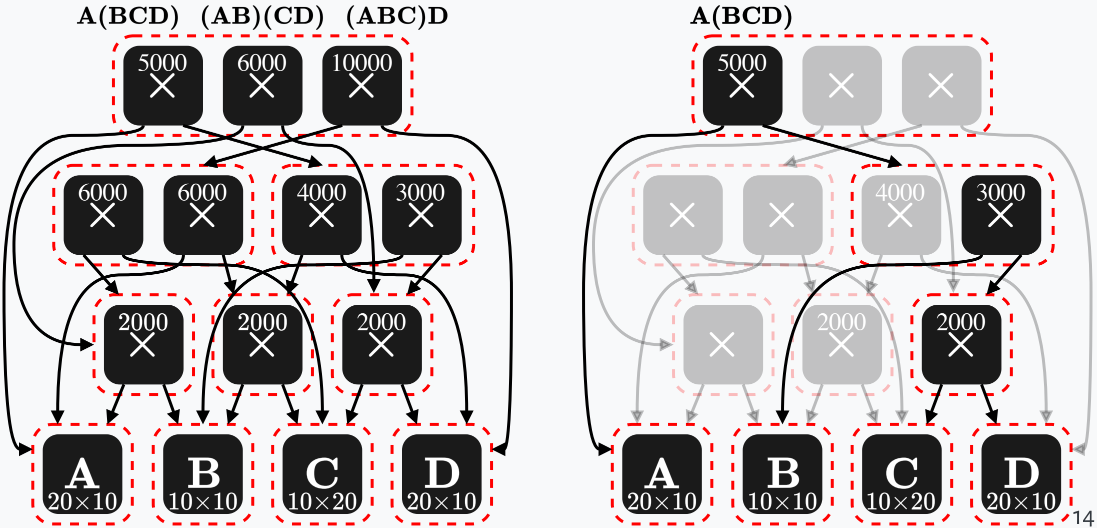


- Root e-class contains equivalent expressions of the program.
- Compute cost of e-nodes in the root e-class.
- Pick the e-node in the root e-class with the lowest cost.

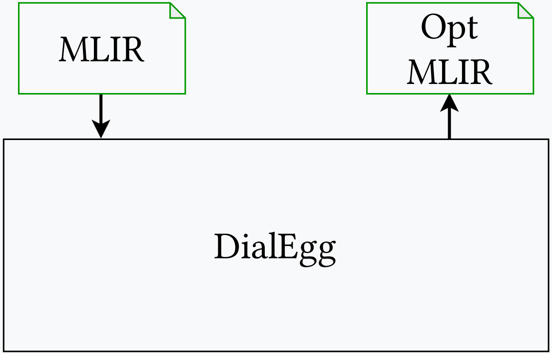
3. Extract the Optimal Expression



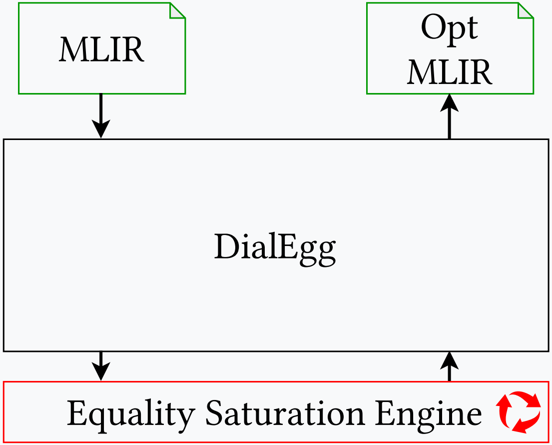
The Optimal Expression is A (B (CD))



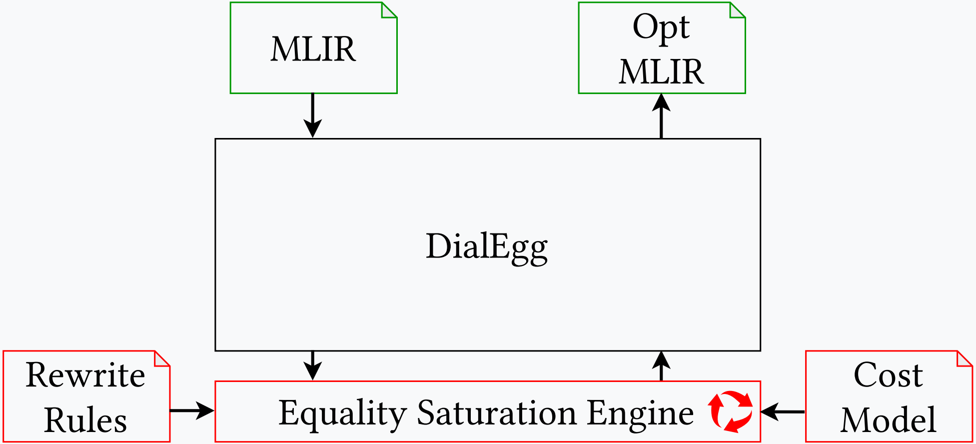
DialEgg



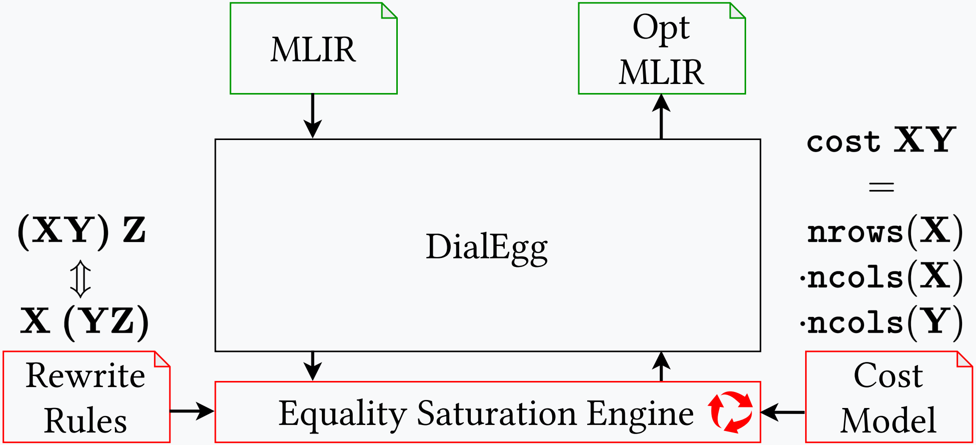
DialEgg



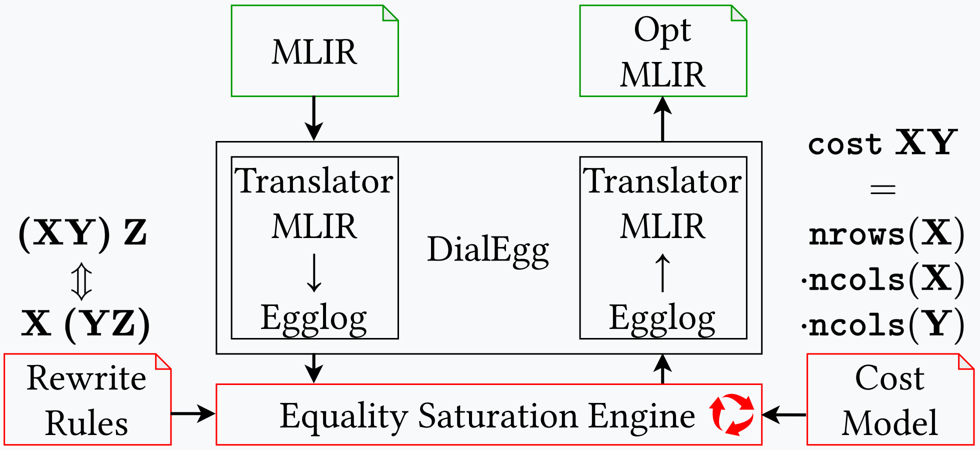
DialEgg


MLIR
Opt MLIR
DialEgg
Rewrite Rules
Equality Saturation Engine
Cost Model

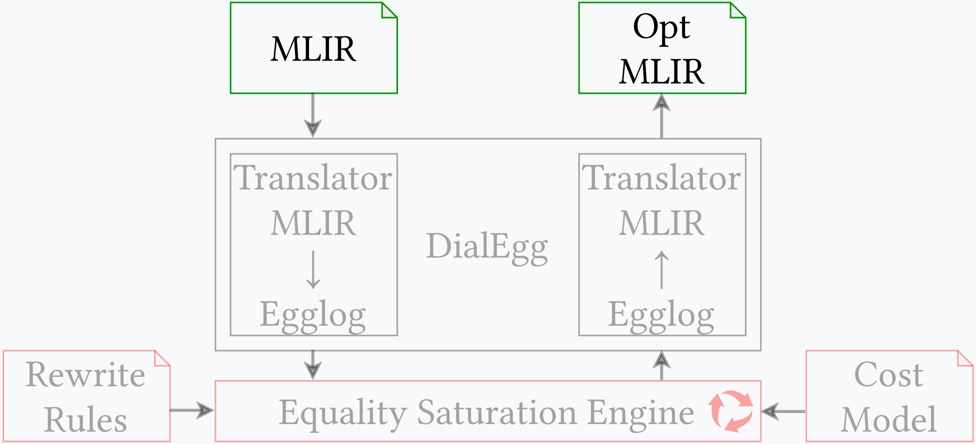
DialEgg


MLIR
Opt MLIR
DialEgg
$(\mathbf{XY})\,\mathbf{Z}$
$\Updownarrow$
$\mathbf{X}\,(\mathbf{YZ})$
Rewrite Rules
Equality Saturation Engine
Cost Model
$\texttt{cost}\ \mathbf{XY} = \texttt{nrows}(\mathbf{X}) \cdot \texttt{ncols}(\mathbf{X}) \cdot \texttt{ncols}(\mathbf{Y})$

DialEgg


MLIR
Opt MLIR
Translator MLIR → Egglog
DialEgg
Translator MLIR ← Egglog
cost XY = nrows(X) ·ncols(X) ·ncols(Y)
(XY) Z ⇕ X (YZ)
Rewrite Rules
Equality Saturation Engine
Cost Model

MLIR


MLIR
Opt MLIR
Translator MLIR → Egglog
DialEgg
Translator MLIR ← Egglog
Rewrite Rules
Equality Saturation Engine
Cost Model

MLIR

Operations are first-class citizens. Each operation has at least:

- A dialect and name that identifies the operation.
- A type that defines the type of the result(s).
- A list of operands.

```
%AmB = linalg.matmul %A %B : tensor<i64>
%ApB = arith.addi %A %B : tensor<i64>
%if  = scf.if ... { scf.yield %A }
           else   { scf.yield %B } : tensor<i64>
```

MLIR

Operations are first-class citizens. Each operation has at least:

- A dialect and name that identifies the operation.
- A type that defines the type of the result(s).
- A list of operands.

```
%AmB = linalg.matmul %A %B : tensor<i64>
%ApB = arith.addi %A %B : tensor<i64>
%if  = scf.if ... { scf.yield %A }
           else   { scf.yield %B } : tensor<i64>
```

MLIR

Operations are first-class citizens. Each operation has at least:

- A dialect and name that identifies the operation.
- A type that defines the type of the result(s).
- A list of operands.

```
%AmB = linalg.matmul %A %B : tensor<i64>
%ApB = arith.addi %A %B : tensor<i64>
%if  = scf.if ... { scf.yield %A }
           else     { scf.yield %B } : tensor<i64>
```

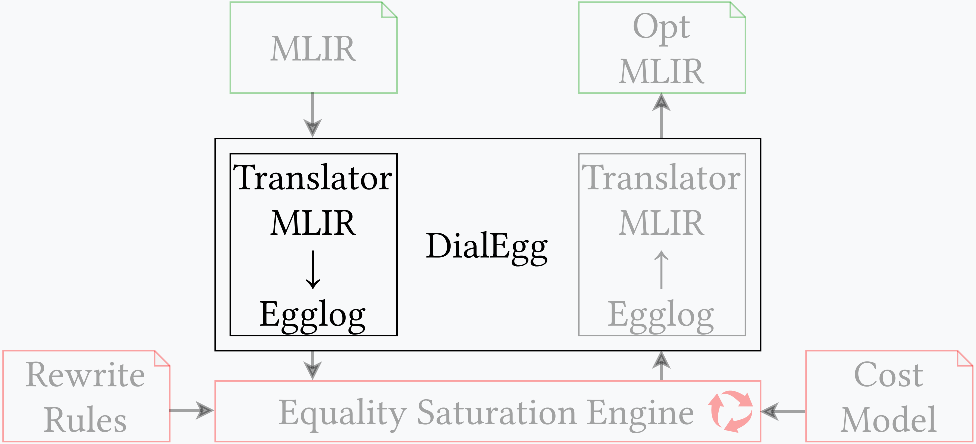
MLIR

Operations are first-class citizens. Each operation has at least:

- A dialect and name that identifies the operation.
- A type that defines the type of the result(s).
- A list of operands.

```
%AmB = linalg.matmul %A %B : tensor<i64>
%ApB = arith.addi %A %B : tensor<i64>
%if  = scf.if ... { scf.yield %A }
           else    { scf.yield %B } : tensor<i64>
```

MLIR → Egglog


MLIR
Opt MLIR
Translator MLIR ↓ Egglog
DialEgg
Translator MLIR ↑ Egglog
Rewrite Rules
Equality Saturation Engine
Cost Model

MLIR → Egglog: Types

DialEgg provides most built-in MLIR types in Egglog.

```
(datatype Type
    (F32)
    (I64)
    (Tensor IntVec Type)
    (OpaqueType String String)
    ...
)
```

MLIR → Egglog: Types

DialEgg provides most built-in MLIR types in Egglog.

```
(datatype Type
    (F32)
    (I64)
    (Tensor IntVec Type)
    (OpaqueType String String)
    ...
)
```

Tensor Type in Egglog

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

MLIR → Egglog: Operations

Operations are translated piece by piece to Egglog.

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

MLIR → Egglog: Operations

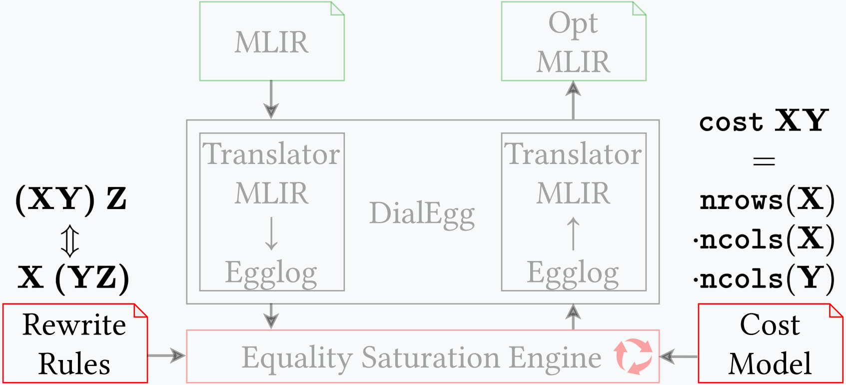
Operations are translated piece by piece to Egglog.

```
(datatype Op
    (Value i64 Type) ; (id, type)
    (arith_const i64 Type) ; simplified constant
    (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
 ((= ?LHS (linalg_matmul (linalg_matmul ?X ?Y ?XY_t) ?Z ?XYZ_t))
  (= ?b (nrows type-of ?Y))
  (= ?d (ncols type-of ?Z))
  (= ?XYZ_t (Tensor ?_ ?t)))

 ((let YZ_t (Tensor (vec-of ?b ?d) ?t))
  (union ?LHS
    (linalg_matmul ?X (linalg_matmul ?Y ?Z ?YZ_t) ?XYZ_t)))
)
```

MM Associativity Rewrite Rule in Egglog

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

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

```
(rule
 ((= ?LHS (linalg_matmul (linalg_matmul ?X ?Y ?XY_t) ?Z ?XYZ_t))
  (= ?b (nrows type-of ?Y))
  (= ?d (ncols type-of ?Z))
  (= ?XYZ_t (Tensor ?_ ?t)))

 ((let YZ_t (Tensor (vec-of ?b ?d) ?t))
  (union ?LHS
    (linalg_matmul ?X (linalg_matmul ?Y ?Z ?YZ_t) ?XYZ_t)))
)
```

MM Associativity Rewrite Rule in Egglog

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

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

```
(rule
 ((= ?LHS (linalg_matmul (linalg_matmul ?X ?Y ?XY_t) ?Z ?XYZ_t))
  (= ?b (nrows type-of ?Y))
  (= ?d (ncols type-of ?Z))
  (= ?XYZ_t (Tensor ?_ ?t)))

 ((let YZ_t (Tensor (vec-of ?b ?d) ?t))
  (union ?LHS
    (linalg_matmul ?X (linalg_matmul ?Y ?Z ?YZ_t) ?XYZ_t)))
)
```

MM Associativity Cost Model in Egglog

`cost XY`

where **X**: $a \times b$ and **Y**: $b \times c$

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

  ((cost (linalg_matmul ?X ?Y ?XY_t) (* (* ?a ?b) ?c)))
)
```

MM Associativity Cost Model in Egglog

cost **XY**

where **X**: $a \times b$ and **Y**: $b \times c$

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

  ((cost (linalg_matmul ?X ?Y ?XY_t) (* (* ?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

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

```
(rule
  ((= ?lhs (math_powf ?x (arith_const ?n ?t) ?t))
   (>= ?n 1))

  ((union ?lhs
     (arith_mulf ?x (math_powf ?x (arith_const (- ?n 1) ?t)) ?t))
  )
)
```

DialEgg

- More case studies are available in the paper:
  - Image conversion: rewrites with computation.
  - Vector normalization: rewrites matching on MLIR attributes.
- DialEgg is open-source.
- I look forward to see how you use DialEgg.

azizzayed.com

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_addf ?x ?y ?a ?t) (arith_addf ?y ?x ?a ?t))
(rewrite (arith_mulf ?x ?y ?a ?t) (arith_mulf ?y ?x ?a ?t))
(rewrite ; (x + y) + z = x + (y + z)
  (arith_addf (arith_addf ?x ?y ?a ?t) ?z ?a ?t)
  (arith_addf ?x (arith_addf ?y ?z ?a ?t) ?a ?t))
(rewrite ; (x * y) * z = x * (y * z)
  (arith_mulf (arith_mulf ?x ?y ?a ?t) ?z ?a ?t)
  (arith_mulf ?x (arith_mulf ?y ?z ?a ?t) ?a ?t))

(rewrite (arith_mulf ?x ; x * 1 = x
    (arith_constant (NamedAttr "value" (FloatAttr 1.0 ?t)) ?t)
  ?a ?t) ?x)
(rewrite (math_powf ?x ; x^0 = 1
  (arith_constant (NamedAttr "value" (FloatAttr 0.0 ?t)) ?t) ?a ?t)
  (arith_constant (NamedAttr "value" (FloatAttr 1.0 ?t)) ?t))
```

Poly Rewrite Rules

```
(rewrite ; mx + nx = x(m + n)
  (arith_addf (arith_mulf ?m ?x ?a ?t) (arith_mulf ?n ?x ?a ?t)
      ?a ?t)
  (arith_mulf ?x (arith_addf ?m ?n ?a ?t) ?a ?t))

(rule ((= ?lhs (math_powf ?x ; x^n = x * x^(n - 1)
    (arith_constant (NamedAttr "value" (FloatAttr ?n ? t)) ?t)
  ?a ?t)) (>= ?n 1.0))
 ((union ?lhs
    (arith_mulf ?x (math_powf ?x
     (arith_constant
      (NamedAttr "value"(FloatAttr (- ?n 1.0) ?t))
      ?t)
     ?a ?t) ?a ?t)
  ))
)
```

Compile Time and Scalability

Bench.	**#Rules**	**#Ops**	**MLIR $\downarrow\uparrow$ Egglog**	**Egglog**	**Saturat.**	**Canon.**
Img Conv	1	29	0.4ms	14.6ms	<0.1ms	0.2ms
Vec Norm	1	44	0.5ms	21.6ms	<0.1ms	0.2ms
Poly	8	26	0.5ms	18.9ms	2ms	0.2ms
3MM	5	8	0.3ms	8.7ms	1ms	0.1ms
10MM	5	22	0.5ms	14.4ms	4ms	0.1ms
20MM	5	42	1.0ms	41.3ms	23ms	0.2ms
40MM	5	82	1.8ms	296.2ms	235ms	0.3ms
80MM	5	162	7.3ms	4939.3ms	3732ms	0.6ms

Qualitative Analysis