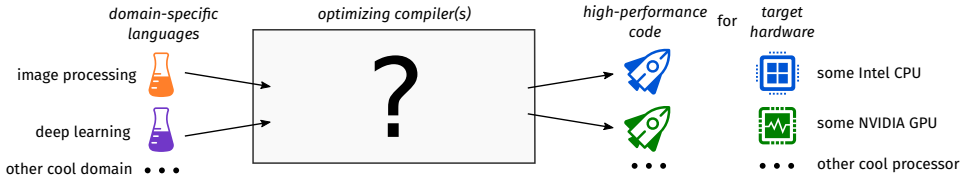
Sketch-Guided Equality Saturation

Thomas Kœhler Phil Trinder Michel Steuwer

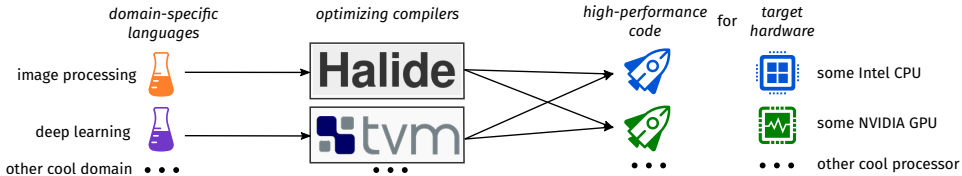
Shameless PLUG Seminar — February 2022

The Optimizing Compiler Challenge



+ convenient, hardware agnostic programming
+ high-performance execution

Domain-Specific Compilers



- fixed set of abstractions and optimizations
- need to design and maintain multiple compilers

Optimization decisions are encoded into a *schedule*

Schedules for Optimization

Halide algorithm: *what to compute*

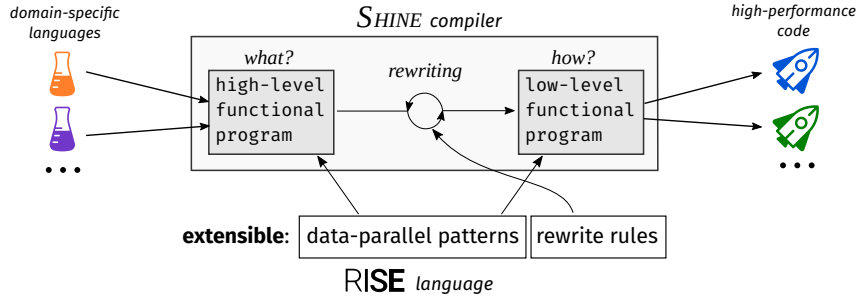
```
blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;
```

Halide schedule: *how to optimize*

```
blur_y.tile(x, y, xi, yi, 256, 32)
    .vectorize(xi, 8).parallel(y);
blur_x.compute_at(blur_y, x).vectorize(x, 8);
```

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines". In: *Acm Sigplan Notices* (2013)

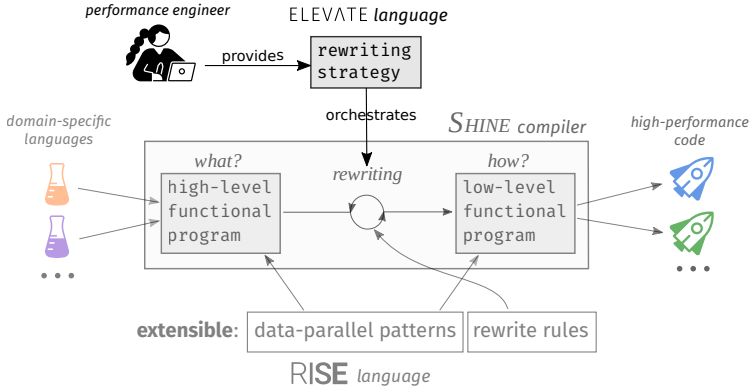
A Domain-Extensible Compiler



\+ extensible set of abstractions and optimizations

Optimization decisions are encoded via rewriting

Rewriting Strategies for Optimization



Bastian Hagedorn, Johannes Lenfers, Thomas Koehler, Xueying Qin, Sergei Gorlatch, and Michel Steuwer. "Achieving high-performance the functional way: a functional pearl on expressing high-performance optimizations as rewrite strategies". In: ICFP (2020)

Rewriting Strategies vs Schedules

Advantages of rewriting strategies:

- 5 principles: separate concerns, facilitate reuse, enable composition, allow reasoning, explicit by default
- in practice, rewriting strategies are a competitive alternative to schedules

  matrix multiplication and corner detection case studies

Rewriting Strategies vs Schedules

Advantages of rewriting strategies:

- 5 principles: separate concerns, facilitate reuse, enable composition, allow reasoning, explicit by default
- in practice, rewriting strategies are a competitive alternative to schedules

  matrix multiplication and corner detection case studies

But … both are difficult to write!

optimization strategy = *schedule* or *rewriting strategy*

Optimization Strategies are Difficult to Write

```
// This schedule fuses the depthwise conv into the pointwise
// conv. The results of the depthwise conv are computed inside
// the outer of the two pointwise reduction loops.

Var xi, yi, di, dii, xii, yii;
RVar ro, ri;

// The pointwise convolution kernel. Produces a 4x4 tile of output.
Func(output)
    .tile({d, x, y}, {di, xi, yi}, {16, 4, 4})
    .tile({di, xi, yi}, {dii, xii, yii}, {1, 2, 2})
    .gpu_threads(di, xi, yi)
    .fuse(y, b, b)
    .gpu_blocks(d, x, b)
    .unroll(xii)
    .unroll(yii)
    .unroll(dii);

pointwise_convolved.compute_at(output, di)
    .reorder(x, y, d)
    .unroll(x)
    .unroll(y)
    .unroll(d)
    .update()
    .unroll(x)
    .unroll(y)
    .unroll(d)
    .split(rc, ro, ri, 4)
    .reorder(ri, x, y, d, ro)
    .unroll(ri);

// We're going to call in() on depthwise_convolved twice.
// The first will be to give it a wrapper to do the
// accumulation in registers before writing the result to
// shared. The second will be staging the loads from
// shared into registers. We write them in reverse order
// below:
```

```
// We can do 4-wide vectorized loads from shared memory if
// we unroll the reduction loop by a factor of four above
// and stage the loads from the depthwise_convolved
// output.

depthwise_convolved.in()
    .in()
    .compute_at(pointwise_convolved, x)
    .bound_extent(d, 4)
    .vectorize(d)
    .unroll(x)
    .unroll(y);

// The depthwise convolution kernel. Produces a 4x4 tile
// of intermediate state, storing the result in shared.
depthwise_convolved.in()
    .compute_at(output, d)
    .tile({d, x, y}, {di, xi, yi}, {32, 4, 4}, TailStrategy::RoundUp)
    .tile({di, xi, yi}, {dii, xii, yii}, {2, 2, 2})
    .gpu_threads(di, xi, yi)
    .unroll(xii)
    .unroll(yii)
    .unroll(dii);

depthwise_convolved
    .compute_at(depthwise_convolved.in(), di)
    .unroll(x)
    .unroll(y)
    .unroll(d)
    .update()
    .reorder(d, x, y, rx, ry, rd)
    .unroll(x)
    .unroll(y)
    .unroll(d);
```

~70 lines of schedule with comments for a single algorithm, and a single hardware target (Halide)

Optimization Strategies are Difficult to Write

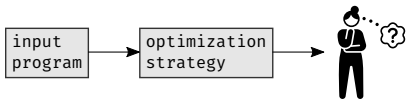
```
73    //scalastyle:off
74    def normForReorder(implicit ev: Traversable[Rise]): Strategy[Rise] =
75      (splitBeforeMap `@` topDown[Rise]) `;;`
76      (fuseReduceMap `@` topDown[Rise]) `;;`
77      (fuseReduceMap `@` topDown[Rise]) `;;` RNF()
78
79    @strategy def reorder(l: List[Int])(implicit ev: Traversable[Rise]): Strategy[Rise] = normForReorder `;` (reorderRec(l) `@` topDown[Rise])
80
81    @strategy def reorderRec(l: List[Int])(implicit ev: Traversable[Rise]): Strategy[Rise] = e => {
82
83      def freduce(s: Strategy[Rise]): Strategy[Rise] =
84        function(function(argumentOf(reduceSeq.primitive, body(body(s)))))
85      def freduceX(s: Strategy[Rise]): Strategy[Rise] =
86        argument(function(function(argumentOf(reduceSeq.primitive, body(body(s))))))
87      def stepDown(s: Strategy[Rise]): Strategy[Rise] = freduceX(s) <+ freduce(s) <+ fmap(s)
88
89      val isFullyAppliedReduceSeq: Strategy[Rise] = isApplied(isApplied(isApplied(isReduceSeq))) <+
90        argument(isApplied(isApplied(isApplied(isReduceSeq)))) // weird reduce special case
91      val isFullyAppliedMap: Strategy[Rise] = isApplied(isApplied(isMap))
92
93      // pos = how far nested is the reduction?
94      def moveReductionUp(pos: Int): Strategy[Rise] = {
95        if (pos <= 1) id
96        else
97          applyNTimes(pos-2)(stepDown)(function(liftReduce)) `;` DFNF() `;` RNF() `;`  moveReductionUp(pos-1)
98      }
99
100     l match {
101       // nothing to reorder, go further down
102       case x :: xs if x == 1 => (stepDown(reorderRec(xs.map(y => if (y>x) y-1 else y ))))(e)
103       // work to do
104       case pos :: xs => (
105         (applyNTimes(pos-1)(stepDown)(isFullyAppliedReduceSeq) `;`
106           // move reduction and normalize the AST
107           moveReductionUp(pos) `;`
108           // recurse and continue reordering
109           stepDown(reorderRec(xs.map(y => if (y>pos) y-1 else y )))) <+
110           // other case: is it a map?
111           applyNTimes(pos-1)(stepDown)(isFullyAppliedMap) `;`
112           basic.fail
113         )(e)
114       case Nil => id(e)
115       case _ => Failure(reorderRec(l))
116     }
117   }
118 }
```

~40 lines of ELEVATE strategy to define loop reordering for matrix multiplication

Optimization Strategies are Difficult to Write

- to write: which sequence of transformations lead to an optimized program?

- to read: what is the resulting program? what are the intermediate programs?



Transformed program is hidden state that needs to be reasoned about

Reasoning about Strategies with Program Shapes

```
1  //function definitions for KWZ group
2  K(x, y, c) = E(x, y) + E(x+1, y) + E(x+2, y)
3  H(x, y) = E(x, y) + 4
4  W(x, y) = K(x, y, 0) + K(x, y, 1) + K(x, y, 2) + 2 * H(x, y)
5  Z(x, y) = W(x, y-2) + W(x, y-1) + W(x, y) +  W(x, y+1) +  W(x, y+2)
6
schedule
7  //group schedule
8  //start with the output of the group
9  Z.compute_root()
10  //tile the loop
11  .split(x, x_o, x_i, 4)
12  .split(y, y_o, y_i, 4)
13  .reorder(x_i,y_i,x_o,y_o);
14  //assign Vars to threads
15  .gpu_threads(x_i,y_i)
16  .gpu_blocks(y_o,y_o);
17 W.compute_at(Z,x_o)
18   //optimize the member stage
19   .reorder(x, y)
20   .gpu_threads(x, y);
21  //nested fusion should be allowed
22 K.compute_at(W, x)
23   .unroll(c);
```

(a) Definitions and Example GPU Schedule of Group KWZ: Computation of K has been moved at the block (innermost inter-tile level) of the output Z and W has been interleaved inside the thread level that computes W.

optimized program shape

```
1  //produce Z
2  <CUDA>gpu_block y_o
3   <CUDA>gpu_block x_o
4    allocate __shared__ W[4*8]
5    //produce W
6    <CUDA>gpu_thread W.y_i
7     <CUDA>gpu_thread W.x_i
8      //produce K
9      unrolled K.c
10      K(...) = (...)
11      //consume K
12      W(...) = (...)
13    //consume W
14    <CUDA>gpu_thread y_i
15     <CUDA>gpu_thread x_i
16      Z(...) = (...)
```

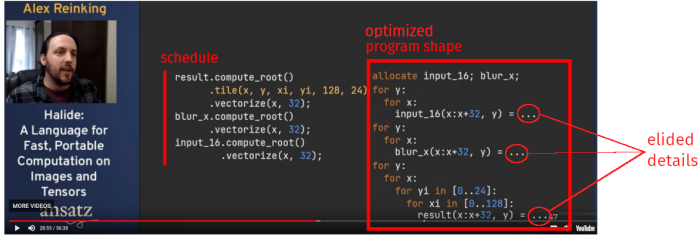
elided details

(b) Equivalent pseudo-CUDA loop nest for segment KWZ: Allocation of stage W is moved to the shared memory. A single kernel is launched for the whole segment. Values of K are computed as needed per pixel of W and stored in registers until consumption.

Savvas Sioutas, Sander Stuijk, Twan Basten, Henk Corporaal, and Lou Somers. “Schedule synthesis for halide pipelines on gpus”. In: *TACO* (2020)

Reasoning about Strategies with Program Shapes

Not only found in papers … but also in talks:



Reasoning about Strategies with Program Shapes

In, ELEVATE we define the `fuseOperators` strategy transforming the Harris program (listing 3) into a pipeline over image lines:

```
map(grayLine) ▷ slide(3,1) ▷
map(sobelLine) ▷ slide(3,1) ▷ map(coarsityLine)
```

elided details

The ELEVATE strategy `splitPipeline(32);parallel` has the same effect, producing a program that slides over $p+4$ lines of input with step p to compute chunks of size p in parallel:

```
slide(p+4, p) ▷ mapGlobal(
  map(grayLine) ▷ slide(3,1) ▷
  map(sobelLine) ▷ slide(3,1) ▷
  map(coarsityLine)
) ▷ join
```

The ELEVATE strategy `circularBufferStages` has the same effect, producing a program with the shape:

```
slide(p+4, p) ▷ mapGlobal(
  circularBuffer(global, 3, grayLine) ▷
  circularBuffer(global, 3, sobelLine) ▷
  mapSeq(coarsityLine)
) ▷ join
```

Thomas Koehler and Michel Steuwer. "Towards a Domain-Extensible Compiler: Optimizing an Image Processing Pipeline on Mobile CPUs". In: *CGO*. 2021

Sketches to Formalize Program Shapes

sketches = program patterns that leave details unspecified

$$S \;::=\; ? \;\mid\; F(S, .., S) \;\mid\; \mathit{contains}(S)$$

$$\begin{aligned} R(?) &= T = \{F(t_1, .., t_n)\} \\ R(F(s_1, .., s_n)) &= \{F(t_1, .., t_n) \mid t_i \in R(s_i)\} \\ R(\mathit{contains}(s)) &= R(s) \cup \{F(t_1, .., t_n) \mid \exists t_i \in R(\mathit{contains}(s))\} \end{aligned}$$

Basic sketch grammar (top) and the terms it represents (bottom)

Sketches to Formalize Program Shapes

sketches = program patterns that leave details unspecified

$$S \;::=\; ? \;\mid\; F(S, .., S) \;\mid\; \mathit{contains}(S)$$

```
map(grayLine)  ▷ slide(3, 1) ▷
map(sobelLine) ▷ slide(3, 1) ▷ map(coarsityLine)
```

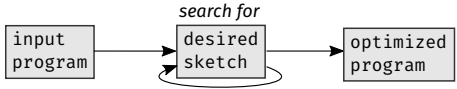
```
contains(map(?) ▷ slide(3, 1) ▷
         map(?) ▷ slide(3, 1) ▷ map(?))
```

Example informal program shape (middle) and possible sketch (bottom)

Sketches for Optimization

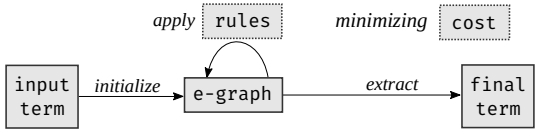
Why not use sketches to specify optimization goals?

- instead of writing optimization strategy, write desired program shape:



- instead of reading optimization strategy, read program shape:

Automated Search?



Equality Saturation

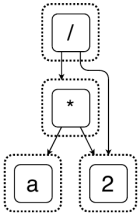
- An e-graph efficiently represents a large set of equivalent programs.
- The e-graph is grown by applying all possible rewrite rules in a purely additive way.
- After growing the e-graph, the best program found is extracted.

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. "Equality saturation: a new approach to optimization". In: *POPL*. 2009

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. "egg: fast and extensible equality saturation". In: POPL (2021)

E-Graph Example

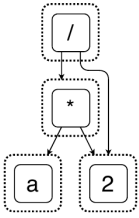
$$(a * 2)/2 \longrightarrow^* a$$

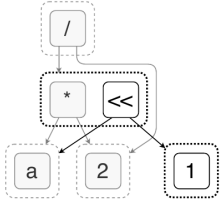


$(a * 2)/2$

E-Graph Example

$$(a * 2)/2 \longrightarrow^* a$$



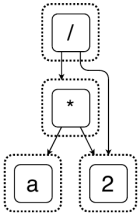


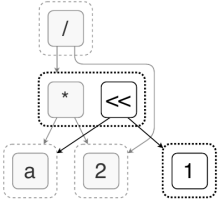
$(a * 2)/2$

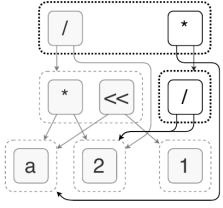
$x * 2 \longrightarrow x \ll 1$

E-Graph Example

$$(a * 2)/2 \longrightarrow^* a$$







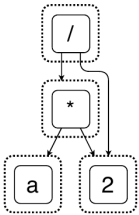
$(a * 2)/2$

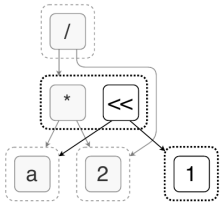
$x * 2 \longrightarrow x \ll 1$

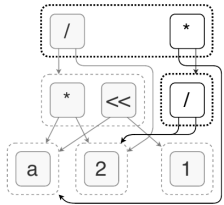
$(x * y)/z \longrightarrow x * (y/z)$

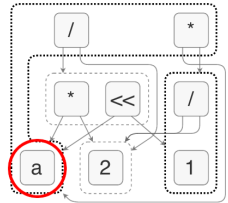
E-Graph Example

$$(a * 2)/2 \longrightarrow^* a$$









$(a * 2)/2$

$x * 2 \longrightarrow x \ll 1$

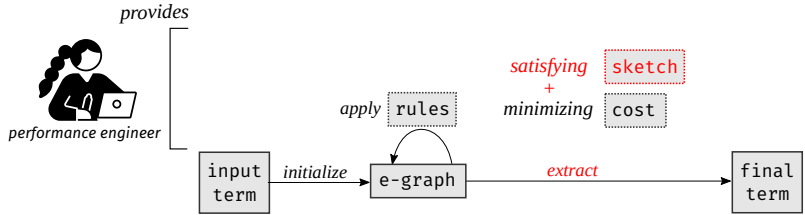
$(x * y)/z \longrightarrow x * (y/z)$

$x/x \longrightarrow 1$

$x * 1 \longrightarrow x$

cost = term size

Equality Saturation with Sketches



Questions:

1. How does it work for functional programs like RI**SE**?
   - no efficient support for name bindings, rewritten languages are usually first order
2. Does it scale to complex optimizations of realistic programs?
   - the search could be too costly

Equality Saturation for Functional Programs

Consider 2 standard λ-calculus rules + 2 rules that introduce names on the right:

$$(\lambda x.\ b)\,e \longrightarrow b[e/x] \qquad (\beta\text{-reduction})$$

$$\lambda x.\ f\,x \longrightarrow f \qquad \text{if } x \text{ not free in } f \qquad (\eta\text{-reduction})$$

$$map\ f\ (map\ g\ arg) \longrightarrow map\ (\lambda x.\ f\ (g\ x))\ arg \qquad (\text{map-fusion})$$

$$map\ (\lambda x.\ f\,g\,x) \longrightarrow \lambda y.\ map\ f\ (map\ (\lambda x.\ g\,x)\ y) \qquad \text{if } x \text{ not free in } f \qquad (\text{map-fission})$$

How can we implement substitution, predicates and name bindings?

Equality Saturation for Functional Programs

- We made substitution much more efficient, at the cost of ignoring possibilities.
  only substitute using one representative term by equivalence class

- State-of-the-art predicates are efficient but ignore possibilities.
  predicate has to hold ∀ terms in equivalence class
- We made name bindings much more efficient using DeBruijn indices.
  the goal is to avoid duplicating alpha-equivalent terms

Equality Saturation for Functional Programs

- We made substitution much more efficient, at the cost of ignoring possibilities.

  discovering a rewrite goal requiring less than 10 rewrite rule applications:

method	time	RAM	e-nodes	e-classes	found
before	40s	4GB	13M	3M	no
after	<1ms	MBs	364	277	yes

- We made name bindings much more efficient using DeBruijn indices.

  discovering a rewrite goal requiring less than 50 rewrite rule applications:

method	time	RAM	e-nodes	e-classes	found
before	2mn	4GB	2.9M	1.4M	no
after	100ms	MBs	3K	1K	yes

Matrix Multiplication Case Study

Optimization Time and Memory Consumption

- ELEVATE strategies take ~1s to execute, reproducing 7 optimizations from TVM.
- Equality Saturation with Sketches[1]:

version	sketches	found	time	RAM	rules
baseline	1	yes	0.5s	20 MB	2
blocking	1	yes	1h+	35GB	5M
+ 5 others	1	**no**	~1h+	60GB+	???

[1] Intel Xeon E5-2640 v2

Matrix Multiplication Case Study

Optimization Time and Memory Consumption

- ELEVATE strategies take ~1s to execute, reproducing 7 optimizations from TVM.
- Equality Saturation with Sketches[1]:

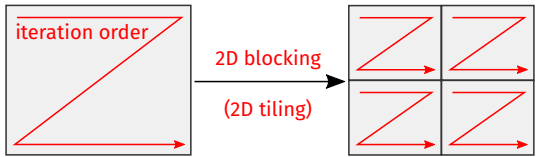
version	sketches	found	time	RAM	rules
baseline	1	yes	0.5s	20 MB	2
blocking	1	yes	1h+	35GB	5M
+ 5 others	1	**no**	~1h+	60GB+	???

What else can we do to improve scaling?

[1] Intel Xeon E5-2640 v2

Matrix Multiplication Case Study

Sketches



baseline version:

```
containsMap(m,          | for m:
 containsMap(n,         |  for n:
  containsReduceSeq(k,  |   for k:
   containsAddMul)))    |    .. * .. * ..
```

blocking version (3D):

```
containsMap(m / 32,           | for m / 32:
 containsMap(n / 32,          |  for n / 32:
  containsReduceSeq(k / 4,    |   for k / 4:
   containsReduceSeq(4,       |    for 4:
    containsMap(32,           |     for 32:
     containsMap(32,          |      for 32:
      containsAddMul))))))    |       .. * .. * ..
```

Matrix Multiplication Case Study

Intermediate Sketches

baseline version:

```
containsMap(m,          | for m:
 containsMap(n,         |  for n:
  containsReduceSeq(k,  |   for k:
   containsAddMul)))    |     .. + .. * ..
```

Intermediate sketch:

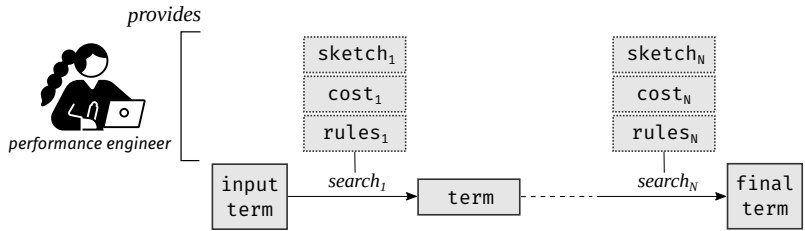
how to split the loops?

```
containsMap(m / 32,              | for m / 32:
 containsMap(32,                 |  for 32:
  containsMap(n / 32,            |   for n / 32:
   containsMap(32,               |    for 32:
    containsReduceSeq(k / 4,     |     for k / 4:
     containsReduceSeq(4,        |      for 4:
      containsAddMul))))))       |       .. + .. * ..
```

blocking version (3D):

```
containsMap(m / 32,              | for m / 32:
 containsMap(n / 32,             |  for n / 32:
  containsReduceSeq(k / 4,       |   for k / 4:
   containsReduceSeq(4,          |    for 4:
    containsMap(32,              |     for 32:
     containsMap(32,             |      for 32:
      containsAddMul))))))       |       .. + .. * ..
```

Sketch-Guided Equality Saturation



- decompose a complex search into a series of simpler searches
- additional sketches specify intermediate goals

Matrix Multiplication Case Study

Intermediate Sketches

version	**sketches**
blocking	split + reorder_1
vectorization	split + reorder_1 + lower_1
loop-perm	split + reorder_2 + lower_2
array-packing	split + reorder_2 + store + lower_3
cache-blocks	split + reorder_2 + store + lower_4
parallel	split + reorder_2 + store + lower_5

Decomposition of each version into logical steps. A sketch is defined for each logical step.

Matrix Multiplication Case Study

Optimization Time and Memory Consumption

- ELEVATE strategies take ~1s to execute, reproducing 7 optimizations from TVM.
- Equality Saturation with Sketches[2]:

version	sketches	found	time	RAM	rules
baseline	1	yes	0.5s	20 MB	2
blocking	1	yes	1h+	35GB	5M
+ 5 others	1	**no**	~1h+	60GB+	???

- Sketch-Guided Equality Saturation[3]:

version	sketches	found	time	RAM	rules
baseline	1	yes	0.5s	20 MB	2
blocking	2	yes	7s	0.3 GB	11K
+ 5 others	3-4	yes	<7s	<0.5 GB	<11K

[2] Intel Xeon E5-2640 v2

[3] AMD Ryzen 5 PRO 2500U

Conclusion

We propose:

- *sketches* for program optimization, alternative to *optimization strategies*
- practical techniques to support efficient equality saturation for lambda calculi
- *sketch-guided equality saturation*, a novel, semi-automatic optimization technique

Conclusion

We propose:

- *sketches* for program optimization, alternative to *optimization strategies*
- practical techniques to support efficient equality saturation for lambda calculi
- *sketch-guided equality saturation*, a novel, semi-automatic optimization technique

---

thomas.koehler@thok.eu
thok.eu

Thanks!
We are open to collaboration!

rise-lang.org
elevate-lang.org

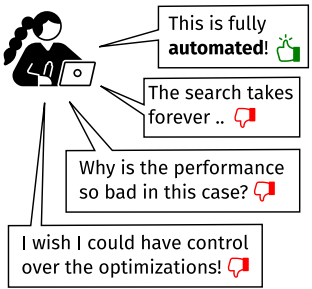
---

paper: https://arxiv.org/abs/2111.13040

Deciding How to Apply Rewrite Rules

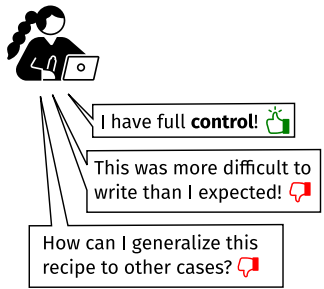
Fully automated search?

e.g. heuristic search, equality saturation, ...



Manually written recipe?

e.g. Halide/TVM schedules, Elevate strategies, ...



Guided search!