Optimizing Processing Pipelines with a Rewrite-Based Domain-Extensible Compiler

Thomas Kœhler   Michel Steuwer

University of Glasgow   The University of Edinburgh

Huawei CSI Paris Seminar — December 2021

Domain-Agnostic Compilers

Some compilers are domain-agnostic:

+ generic program abstractions and optimizations
+ compile programs from any domain (turing complete)
- no automation of domain-specific optimizations
- manual optimization takes months and risks introducing bugs

Domain-Specific Compilers

Some compilers are domain-specific:

+ convenient programming
+ high-performance

Halide

tvm

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);
```

Domain-Specific Compilers

Halide

tvm

Some compilers are domain-specific:

- fixed set of abstractions and optimizations
- lack of flexibility and extensibility

Halide Development Roadmap #5055

Open abadams opened this issue on Jun 19 · 44 comments

- How do we make Halide easier to use for researchers wanting to cannibalize it, extend it, or compare to it?
- How do we make Halide more useful on current and upcoming hardware?
- How do we make Halide more useful for new types of application?

https://github.com/halide/Halide/issues/5055

Domain-Extensible Compilers

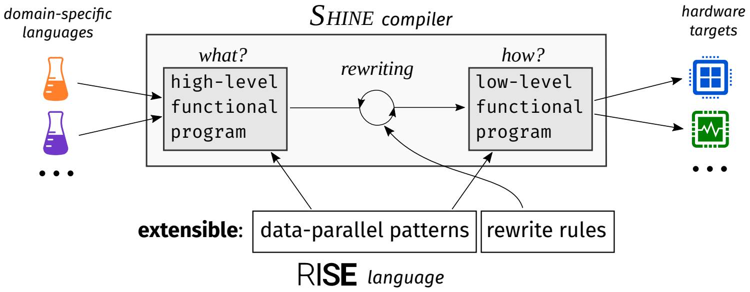
Compilers should be domain-extensible:

- + extensible set of abstractions and optimizations

Domain-Extensible Compilers

Compilers should be domain-extensible:

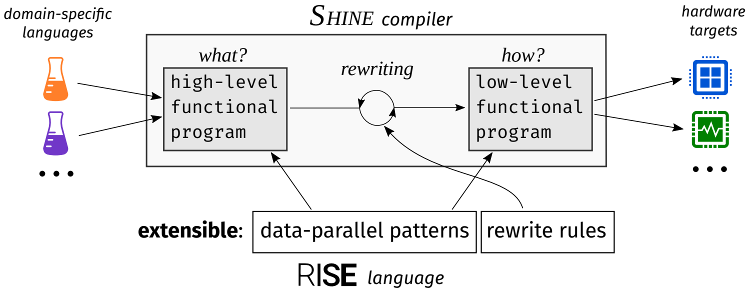
+ extensible set of abstractions and optimizations



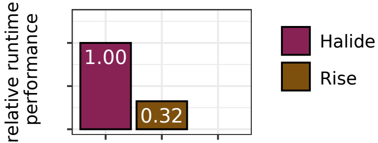
Domain-Extensible Compilers

Compilers should be domain-extensible:

+ extensible set of abstractions and optimizations
- competitive with domain-specific compilers?



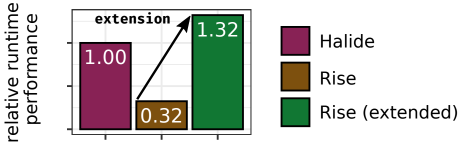
Is RISE Competitive with Domain-Specific Compilers?



important image processing pipeline optimizations are missing

[Kœhler and Steuwer, CGO 2021, *Towards a Domain-Extensible Compiler: Optimizing an Image Processing Pipeline on Mobile CPUs*]

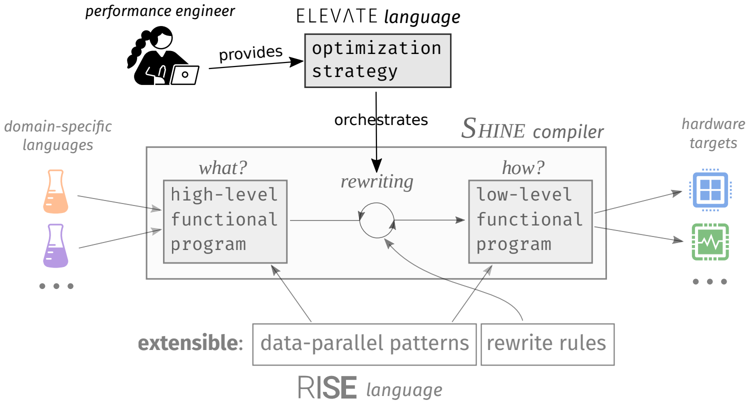
Is RISE Competitive with Domain-Specific Compilers?



6 well-known image processing pipeline optimizations can be encoded as compositions of RISE rewrite rules

[Kœhler and Steuwer, CGO 2021, *Towards a Domain-Extensible Compiler: Optimizing an Image Processing Pipeline on Mobile CPUs*]

Orchestrating Compositions of Rewrite Rules



[Hagedorn et al, ICFP 2020, *Achieving high-performance the functional way: a functional pearl on expressing high-performance optimizations as rewrite strategies*]

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

```
zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

Rewriting

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

```
zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

Rewriting

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

```
zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

Rewriting

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

```
zip(a, b) ▷ reduceSeq(fun acc, x. acc + fst(x) × snd(x), 0)
```

Low-Level
RISE Program

Compilation Example

dot product

High-level RISE program

```
def dot(a, b) = zip(a, b) ▷ map(×) ▷ reduce(+, 0)
```

ELEVATE optimization strategy

```
strategy lowerDot = applyOnce(reduceMapFusion)

rule reduceMapFusion = map(f) ▷ reduce(g, init)
  ↦ reduceSeq(fun (acc, x). g(acc, f(x)), init)
```

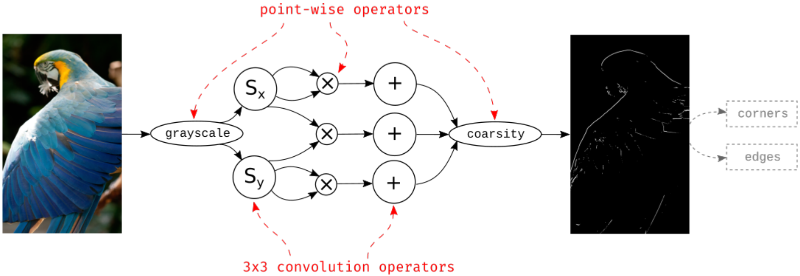
```
zip(a, b) ▷ reduceSeq(fun acc, x. acc + fst(x) × snd(x), 0)
```

Low-Level RISE Program

```
void dotSeqC(float* out, int n, float* a, float* b) {
  float acc;
  acc = 0.0f;
  for (int i = 0; i < n; i++) {
    acc = acc + (a[i] * b[i]);
  }
  out[0] = acc;
}
```

Low-Level C Code

Harris Case Study



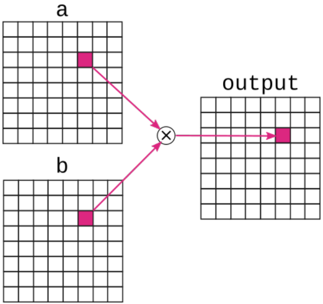
The Harris corner (and edge) detector is a well established image processing pipeline

How de we represent these operators in RISE?

Harris Case Study

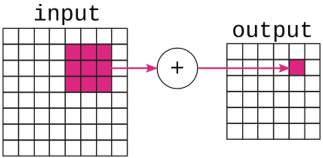
High-level point-wise operator

```
def ×2D(a, b: [n][m]f32): [n][m]f32 =
  zip2d(a, b) ▷ map2d(×)
```



High-level convolution operator

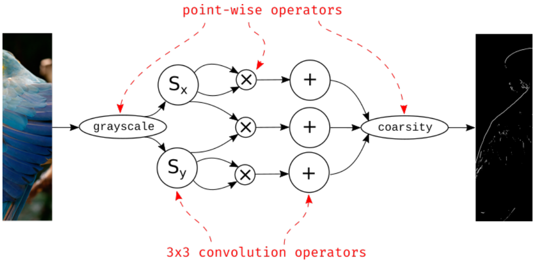
```
def +3×3: [n+2][m+2]f32 → [n][m]f32 =
  slide2d(3, 1, 3, 1) ▷ map2d(fun w. reduce(+, 0 join(w)))
```



Harris Case Study

High-level Harris operator

```
def harris(RGB: [3][n+4][m+4]f32): [n][m]f32 =
  def I = grayscale(RGB)
  def Ix  = Sx(I)
  def Iy  = Sy(I)
  def Ixx = ×2D(Ix, Ix)
  def Ixy = ×2D(Ix, Iy)
  def Iyy = ×2D(Iy, Iy)
  def Sxx = +3×3(Ixx)
  def Sxy = +3×3(Ixy)
  def Syy = +3×3(Iyy)
  coarsity(Sxx, Sxy, Syy, 0.04)
```



Reference Optimizations

CPU schedule for Harris

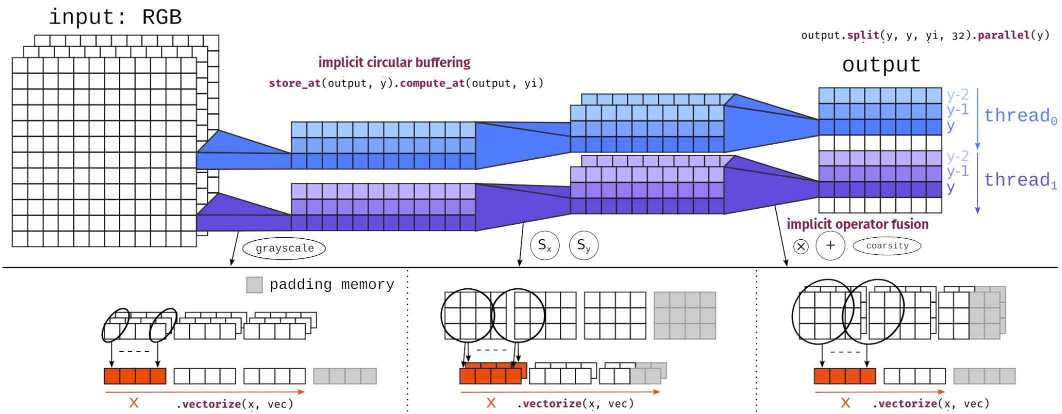
from the Halide GitHub repository

```
const int vec = natural_vector_size<float>();
output.split(y, y, yi, 32).parallel(y)
  .vectorize(x, vec);
gray.store_at(output, y).compute_at(output, yi)
  .vectorize(x, vec);
Ix.store_at(output, y).compute_at(output, yi)
  .vectorize(x, vec);
Iy.store_at(output, y).compute_at(output, yi)
  .vectorize(x, vec);
Ix.compute_with(Iy, x);
```

Simplified internal representation of lowered code

```
let t1226 = ((output.extent.1 + 31)/32)
parallel (output.s0.y.y, 0, t1226) {
 allocate gray[float32 * (output.extent.0 + 4) * 8]
 allocate Iy[float32 * t1247 * 4]
 allocate Ix[float32 * t1247 * 4]
 for (output.s0.y.yi, 0, 32) {
  for (gray.s0.y, gray.s0.y.min_2, gray.s0.y.loop_extent) {
   for (gray.s0.x.x, 0, t1265) {
    gray[ramp(((gray.s0.x.x*4) + t1268), 1, 4)] = [...] }}
  for (Iy.s0.fused.y, Iy.s0.y.min_2, t1269) {
   for (Iy.s0.x.fused.x, 0, t1251) {
    Iy[ramp(((Iy.s0.x.fused.x*4) + t1275), 1, 4)] = [...]
    Ix[ramp(((Iy.s0.x.fused.x*4) + t1275), 1, 4)] = [...] }}
  for (output.s0.x.x, 0, t1250) {
   output[ramp(((output.s0.x.x*4) + t1281), 1, 4)] = [...] }}
 free gray
 free Iy
 free Ix }
```

Reference Optimizations



Reference Optimizations

In RISE and ELEVATE

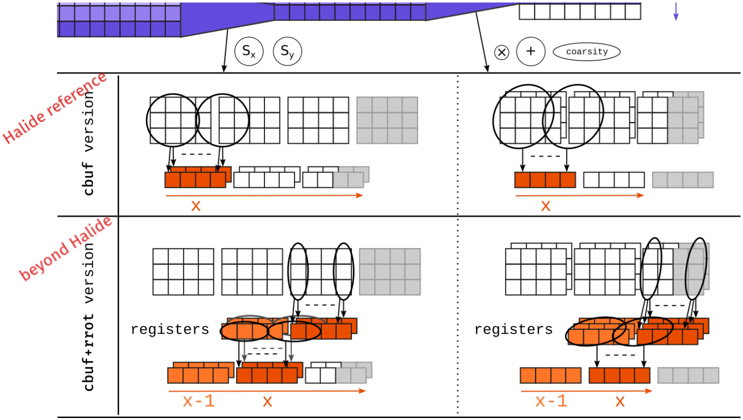
ELEVATE optimization strategy

```
strategy cbufVersion =
  fuseOperators;
  splitPipeline(32); parallel;
  vectorizeReductions(vec);
  harrisIxWithIy;
  circularBufferStages;
  sequentialLines;
  usePrivateMemory; unrollReductions
```

Harris after applying `cbufVersion`

```
slide(32+4, 32) ▷ mapGlobal(
  circularBuffer(global, 3, grayLine) ▷
  circularBuffer(global, 3, sobelLine) ▷
  mapSeq(coarsityLine)
) ▷ join
```

Optimizations beyond Halide



Optimizations beyond Halide

In RISE and ELEVATE

ELEVATE optimization strategy

```
strategy cbuf+rrotVersion =
    fuseOperators;
    splitPipeline(32); parallel;
    separateConvolutions;
    vectorizeReductions(vec);
    harrisIxWithIy;
    circularBufferStages;
    rotateValuesAndConsumeLines;
    usePrivateMemory; unrollReductions
```

Typical 2D Convolution

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    dot(join(weights2d), join(nbh2d)))
```

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Optimizations beyond Halide

In RISE **and** ELEVATE

ELEVATE optimization strategy

```
strategy cbuf+rrotVersion =
    fuseOperators;
    splitPipeline(32); parallel;
    separateConvolutions;
    vectorizeReductions(vec);
    harrisIxWithIy;
    circularBufferStages;
    rotateValuesAndConsumeLines;
    usePrivateMemory; unrollReductions
```

Typical 2D Convolution

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    dot(join(weights2d), join(nbh2d)))
```

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    nbh2d ▷ transpose ▷ map(dot(wV)) ▷ dot(wH))
```

Optimizations beyond Halide

In RISE **and** ELEVATE

ELEVATE optimization strategy

```
strategy cbuf+rrotVersion =
    fuseOperators;
    splitPipeline(32); parallel;
    separateConvolutions;
    vectorizeReductions(vec);
    harrisIxWithIy;
    circularBufferStages;
    rotateValuesAndConsumeLines;
    usePrivateMemory; unrollReductions
```

Typical 2D Convolution

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    dot(join(weights2d), join(nbh2d)))
```

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    nbh2d ▷ transpose ▷ map(dot(wV)) ▷ dot(wH))
```

```
nbhV ▷ transpose  ▷ map(dot(wV))
     ▷ slide(3,1) ▷ map(dot(wH))
```

Optimizations beyond Halide

In RISE and ELEVATE

ELEVATE optimization strategy

```
strategy cbuf+rrotVersion =
    fuseOperators;
    splitPipeline(32); parallel;
    separateConvolutions;
    vectorizeReductions(vec);
    harrisIxWithIy;
    circularBufferStages;
    rotateValuesAndConsumeLines;
    usePrivateMemory; unrollReductions
```

Typical 2D Convolution

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    dot(join(weights2d), join(nbh2d)))
```

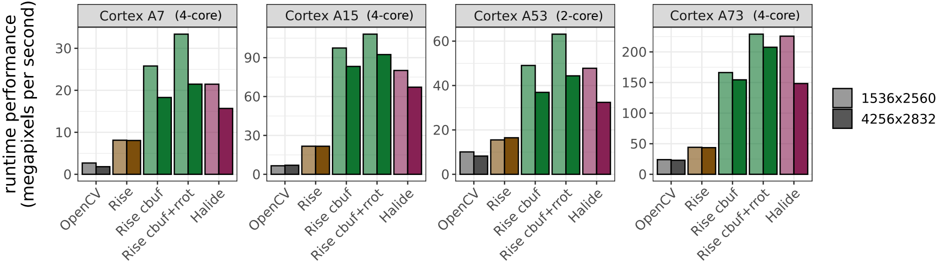
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

```
nbhV ▷ map(slide(3,1)) ▷ transpose ▷ map(fun nbh2d.
    nbh2d ▷ transpose ▷ map(dot(wV)) ▷ dot(wH))
```

```
nbhV ▷ transpose  ▷ map(dot(wV))
     ▷ slide(3,1) ▷ map(dot(wH))
```

```
nbhV ▷ transpose ▷ map(dot(wV))
     ▷ rotateValues(private, 3) ▷ mapSeq(dot(wH))
```

Experimental Evaluation



- All compilers outperform the OpenCV library: RISE by up to 16×
- RISE improved by up to 4.5×
- RISE `cbuf` is roughly on par with Halide
- RISE `cbuf+rrot` is faster than Halide by up to 40%

Harris Case Study on ARM CPUs

Summary

- We reproduced an optimized Halide schedule by defining compositional ELEVATE optimization strategies; by extending and re-using RI**SE** patterns.
- We reached higher performance through additional optimizations that cannot be expressed in a Halide schedule, showing the benefit of compiler extensibility.

Harris Case Study on ARM CPUs

Summary

- We reproduced an optimized Halide schedule by defining compositional ELEVATE optimization strategies; by extending and re-using RI**SE** patterns.
- We reached higher performance through additional optimizations that cannot be expressed in a Halide schedule, showing the benefit of compiler extensibility.

But, ELEVATE optimization strategies are difficult to write!

Optimization Strategies are Difficult to Write

ELEVATE optimization strategy

```
strategy cbufVersion =
  fuseOperators;
  splitPipeline(32); parallel;
  vectorizeReductions(vec);
  harrisIxWithIy;
  circularBufferStages;
  sequentialLines;
  usePrivateMemory; unrollReductions
```

Behind the scenes:

- 400 lines of ELEVATE strategies
- hard to write: 1 month of work
- hard to read, hard to reuse

Optimization Strategies are Difficult to Write

ELEVATE optimization strategy

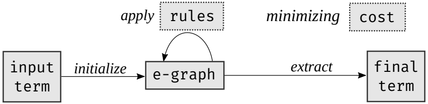
```
strategy cbufVersion =
  fuseOperators;
  splitPipeline(32); parallel;
  vectorizeReductions(vec);
  harrisIxWithIy;
  circularBufferStages;
  sequentialLines;
  usePrivateMemory; unrollReductions
```

Behind the scenes:

- 400 lines of ELEVATE strategies
- hard to write: 1 month of work
- hard to read, hard to reuse

Can we automatically apply rewrite rules instead of writing strategies?

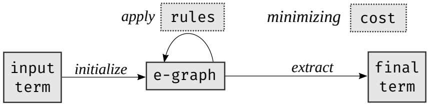
Exploring Many Ways to Apply Rewrite Rules



Equality Saturation
[Tate et al. 2009 "Equality saturation: a new approach to optimization"]
[Willsey et al. 2021 "egg: fast and extensible equality saturation"]

- An e-graph efficiently represents a large set of equivalent programs.
- All possible rewrite rules are applied in a purely additive way, growing the e-graph.
- After growing the e-graph, the best program found is extracted.

Exploring Many Ways to Apply Rewrite Rules



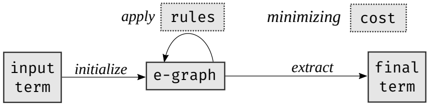
Equality Saturation

[Tate et al. 2009 “Equality saturation: a new approach to optimization”]

[Willsey et al. 2021 “egg: fast and extensible equality saturation”]

+ No need to decide which rewrite to apply next.
+ Decide which program you want in the end.
- Does not scale to our Harris case study

Exploring Many Ways to Apply Rewrite Rules



Equality Saturation

[Tate et al. 2009 "Equality saturation: a new approach to optimization"]

[Willsey et al. 2021 "egg: fast and extensible equality saturation"]

Can we make a trade-off between precise control and full automation?

Declaring Rewrite Goals using Sketches

Harris after applying `cbufVersion`

```
slide(32+4, 32) ▷ mapGlobal(
  circularBuffer(global, 3, grayLine) ▷
  circularBuffer(global, 3, sobelLine) ▷
  mapSeq(coarsityLine)
) ▷ join
```

- When designing optimizations, it is useful to think about the desired shape of the optimized program.

Declaring Rewrite Goals using Sketches

Harris after applying `cbufVersion`

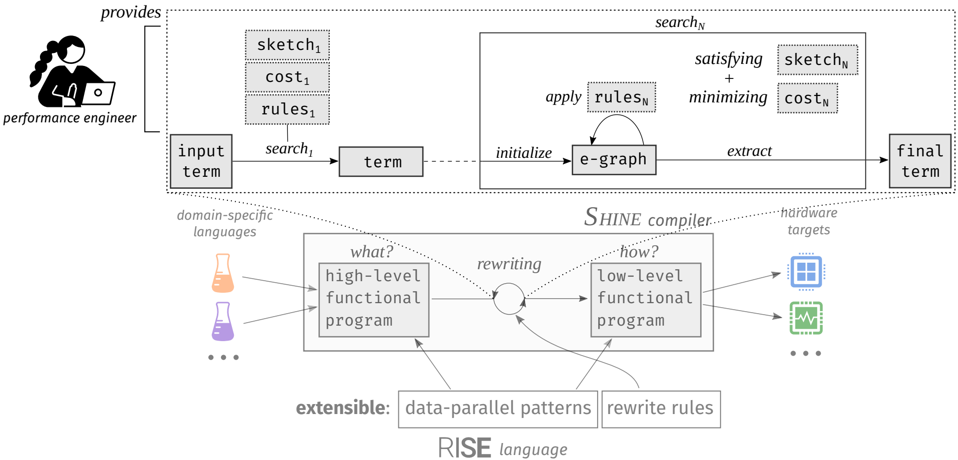
```
slide(32+4, 32) ▷ mapGlobal(
  circularBuffer(global, 3, grayLine) ▷
  circularBuffer(global, 3, sobelLine) ▷
  mapSeq(coarsityLine)
) ▷ join
```

Harris sketch corresponding to `cbufVersion`

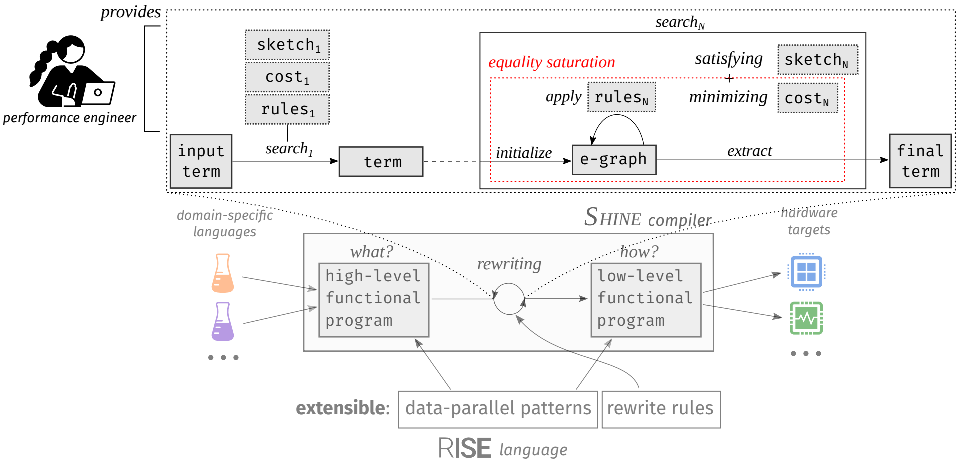
```
contains(? ▷ slide(32+4, 32) ▷ mapGlobal(
  contains(? ▷
  circularBuffer(global, 3, containsGrayLine) ▷
  circularBuffer(global, 3, containsSobelLine) ▷
  mapSeq(containsCoarsityLine))
))
```

- When designing optimizations, it is useful to think about the desired shape of the optimized program.
- *Sketches* are program patterns that capture this intuition while leaving details unspecified using program holes (`?`) and other constructs (`contains`).
- Sketches can be used to guide searches such as equality saturation.

Sketch-Guided Equality Saturation



Sketch-Guided Equality Saturation



Matrix Multiplication Case Study

Optimization Time and Memory Consumption

Single-Sketch[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+	???

Multi-Sketch[2]:

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

[1] Intel Xeon E5-2640 v2

[2] AMD Ryzen 5 PRO 2500U

Conclusion

- We encode 6 well-known image processing pipeline optimizations as compositions of rewrite rules.
- We propose sketch-guided equality saturation to offer novel trade-offs between precise control and full automation of optimizations.

Conclusion

- We encode 6 well-known image processing pipeline optimizations as compositions of rewrite rules.
- We propose sketch-guided equality saturation to offer novel trade-offs between precise control and full automation of optimizations.

---

thomas.koehler@thok.eu
thok.eu

Thanks!
We are open to collaboration!

rise-lang.org
elevate-lang.org

Deciding How to Apply Rewrite Rules

Fully automated search?

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

This is fully **automated**!

The search takes forever ..

Why is the performance so bad in this case?

I wish I could have control over the optimizations!

Manually written recipe?

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

I have full **control**!

This was more difficult to write than I expected!

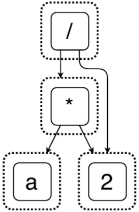
How can I generalize this recipe to other cases?

Guided search!

I can combine **control and automation**!

E-Graphs

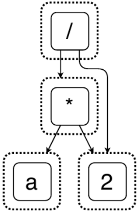
$$(a * 2)/2 \longrightarrow^* a$$

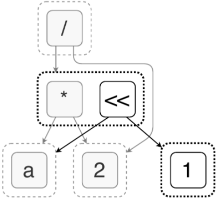


$(a * 2)/2$

E-Graphs

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



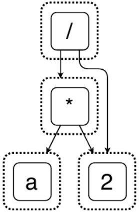


$(a * 2)/2$

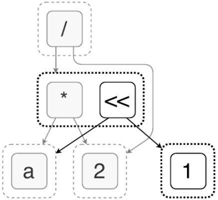
$x * 2 \longrightarrow x \ll 1$

E-Graphs

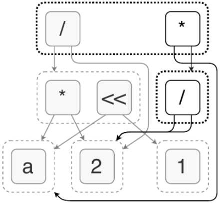
$$(a * 2)/2 \longrightarrow^{*} a$$



$(a * 2)/2$



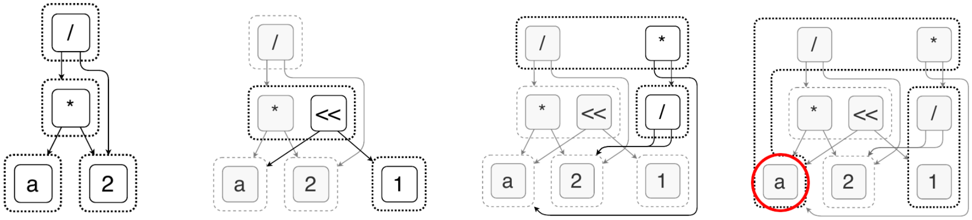
$x * 2 \longrightarrow x \ll 1$



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

E-Graphs

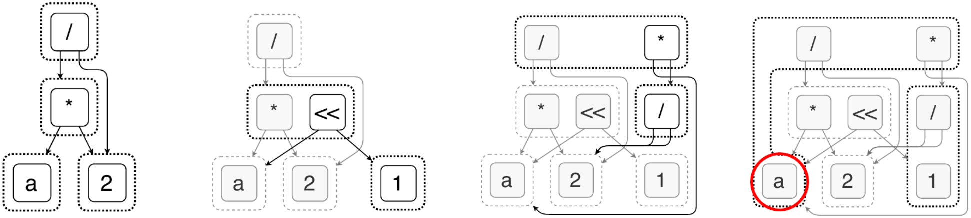
$$(a * 2)/2 \longrightarrow^{*} a$$



$(a * 2)/2$ $\quad$ $x * 2 \longrightarrow x \ll 1$ $\quad$ $(x * y)/z \longrightarrow x * (y/z)$ $\quad$ $x/x \longrightarrow 1$, $x * 1 \longrightarrow x$

E-Graphs

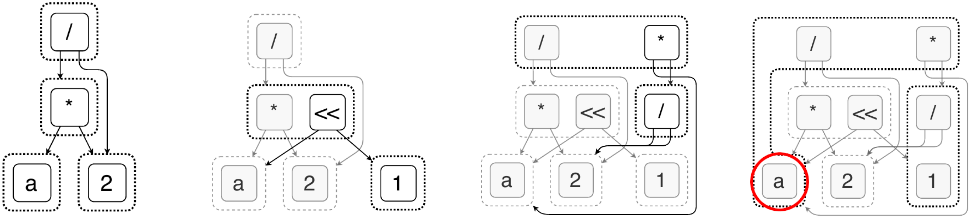
$$(a * 2)/2 \longrightarrow^* a$$



Congruence invariant: $a = b \implies f(a) = f(b)$

E-Graphs

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



How does it work for functional programs?

Equality Saturation for RISE

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

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

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

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

How can we implement substitution, predicates and name bindings?

- State-of-the-art is very inefficient, trivial optimizations are our of reach.
- We made substitution order of magnitudes more efficient using a practical approximation.
- We made name bindings order of magnitudes more efficient using DeBruijn indices.