Sketch-Guided Program Optimisation

Thomas Kœhler   Phil Trinder   Michel Steuwer

University of Glasgow   THE UNIVERSITY of EDINBURGH

Saarland University — April 2022

Program Optimisation

- program optimisation is critical in performance-demanding domains

  e.g. image processing, physics simulation, machine learning
- typically leads to order of magnitudes performance improvements
- hand optimisation takes months and risks introducing bugs in low level languages

  e.g. C, OpenCL, CUDA

Example: Matrix Multiplication

```
for (int im = 0; im < m; im++) {
  for (int in = 0; in < n; in++) {
    float acc = 0.0f;
    for (int ik = 0; ik < k; ik++) {
      acc += a[ik + (k * im)] * b[in + (n * ik)];
    }

    output[in + (n * im)] = acc;
  }
}
```

Optimised program on the right:

- \+ 110× faster runtime

  Intel i5-4670K CPU

- \- 6× more lines of code where things can go wrong

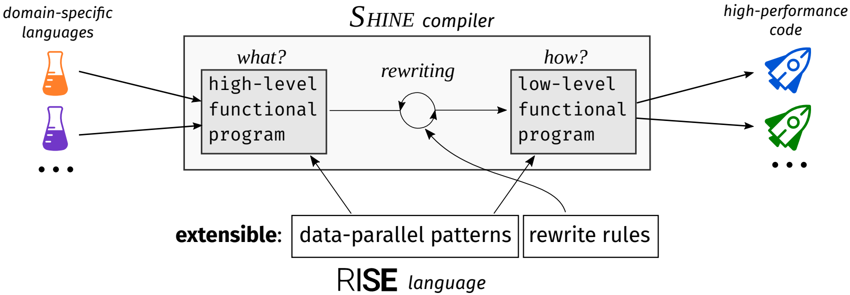
  threads, SIMD, index computations

- \- hardware specific (not portable)

```
float aT[n * k];
#pragma omp parallel for
for (int in = 0; in < (n / 32); in = 1 + in) {
  for (int ik = 0; ik < k; ik = 1 + ik) {
    #pragma omp simd
    for (int jn = 0; jn < 32; jn = 1 + jn) {
      aT[(ik + ((32 * in) * k)) + (jn * k)] = a[(jn + (32 * in)) + (ik * n)];
    }
  }
}
#pragma omp parallel for
for (int im = 0; im < (m / 32); im = 1 + im) {
  for (int in = 0; in < (n / 32); in = 1 + in) {
    float tmp1[1024];
    for (int jm = 0; jm < 32; jm = 1 + jm) {
      for (int jn = 0; jn < 32; jn = 1 + jn) {
        tmp1[jn + (32 * jm)] = 0.0f;
      }
    }
    for (int ik = 0; ik < (k / 4); ik = 1 + ik) {
      for (int jm = 0; jm < 32; jm = 1 + jm) {
        float tmp2[32];
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp2[jn] = tmp1[jn + (32 * jm)];
        }
        #pragma omp simd
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp2[jn] += (a[((4 * ik) + ((32 * im) * k)) + (jm * k)] * aT[((4 * ik) + ((32 * in) * k)) + (jn * k)]);
        }
        #pragma omp simd
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp2[jn] += (a[((1 + (4 * ik)) + ((32 * im) * k)) + (jm * k)] *
            aT[((1 + (4 * ik)) + ((32 * in) * k)) + (jn * k)]);
        }
        #pragma omp simd
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp2[jn] += (a[((2 + (4 * ik)) + ((32 * im) * k)) + (jm * k)] *
            aT[((2 + (4 * ik)) + ((32 * in) * k)) + (jn * k)]);
        }
        #pragma omp simd
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp2[jn] += (a[((3 + (4 * ik)) + ((32 * im) * k)) + (jm * k)] *
            aT[((3 + (4 * ik)) + ((32 * in) * k)) + (jn * k)]);
        }
        for (int jn = 0; jn < 32; jn = 1 + jn) {
          tmp1[jn + (32 * jm)] = tmp2[jn];
        }
      }
    }
    for (int jm = 0; jm < 32; jm = 1 + jm) {
      for (int jn = 0; jn < 32; jn = 1 + jn) {
        output[((jn + ((32 * im) * n)) + (32 * in)) + (jm * n)] = tmp1[jn + (32 * jm)];
      }
    }
  }
}
```

How can we automate the optimisation process?

Optimisation via Term Rewriting



- convenient, hardware agnostic programming
- high-performance code generation
- extensible set of abstractions and optimisations

The RISE language

- anonymous functions: `λx. e`
- function application: `f e`
- identifiers
- literals
- data-parallel patterns over multi-dimensional arrays:
  - **`map`** `f a` $= [f(a_1), \dots, f(a_n)]$
  - **`reduce`** `+ i a` $= i + a_1 + \cdots + a_n$
  - **`split`**, **`join`**, **`transpose`**, **`zip`**, **`unzip`** reshape arrays in various ways
  - …

Example: Matrix Multiplication Blocking

```
map ① (λaRow.
  map ② (λbCol.
    dot ③ aRow bCol)
    (transpose b)) a

def dot a b = reduce + 0
  (map (λy. (fst y) × (snd y))
    (zip a b))
```

↦*

```
join (map (map join) (map transpose
  map ⓐ (map ⓑ λx2.
   reduceSeq ⓒ (λx3. λx4.
     reduceSeq ⓓ (λx5. λx6.
       map ⓔ (map ⓕ (λx7. (fst x7) +
              (fst (snd x7)) ×
              (snd (snd x7)))
         (map (λx7. zip (fst x7) (snd x7))
           (zip x5 x6)))
     (transpose (map transpose
       (snd (unzip (map unzip map (λx5.
         zip (fst x5) (snd x5))
         (zip x3 x4)))))))
       (generate (λx3. generate (λx4. 0)))
       transpose (map transpose x2))
   (map (map (map (map (split 4))))
     (map transpose
       (map (map (λx2. map (map (zip x2)
         (split 32 (transpose b)))))
           split 32 a))))))
```

```
for m ①:
 for n ②:
  for k ③:
   ..
```

↦*

```
for m / 32 ⓐ:
 for n / 32 ⓑ:
  for k / 4 ⓒ:
   for 4 ⓓ:
    for 32 ⓔ:
     for 32 ⓕ:
      ..
```

How do we decide which rewrite rules to apply?

Rewriting Strategies

- programmers describe optimisations as compositions of rewrite rules
- MM blocking:

```
def blocking = ( baseline ‘;‘
  tile(32,32)      ‘@‘ outermost(mapNest(2))   ‘;;‘
  fissionReduceMap ‘@‘ outermost(appliedReduce)‘;;‘
  split(4)         ‘@‘ innermost(appliedReduce)‘;;‘
  reorder(List(1,2,5,6,3,4)))
```

+ empowers programmers to manually control the rewrite process
+ `tile`, `split`, `reorder` are not built-in but programmer-defined

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

- programmers describe optimisations as compositions of rewrite rules
- MM blocking:

```
def blocking = ( baseline ‘;‘
  tile(32,32)       ‘@‘ outermost(mapNest(2))   ‘;;‘
  fissionReduceMap ‘@‘ outermost(appliedReduce)‘;;‘
  split(4)          ‘@‘ innermost(appliedReduce)‘;;‘
  reorder(List(1,2,5,6,3,4)))
```

- requires programmers to order all rewrite steps deterministically
- strategies are often program-specific and complex to implement
- transformed program is hidden state that needs to be reasoned about

Sketch-Guided Program Optimisation

Observation:

- the *shape* of the optimised program is often used to explain optimisations:

```
for m:
 for n:
  for k:
   ..
```

$\mapsto^*$

```
for m / 32:
 for n / 32:
  for k / 4:
   for 4:
    for 32:
     for 32:
      ..
```

Sketch-Guided Program Optimisation

Observation:

- the *shape* of the optimised program is often used to explain optimisations:

```
for m:
 for n:
  for k:
   ..
```

$\mapsto^*$

```
for m / 32:
 for n / 32:
  for k / 4:
   for 4:
    for 32:
     for 32:
      ..
```

Key Insight:

- explanatory shapes can be formalized as sketches and used to guide a search
- replaces step-by-step rewriting strategies with declarative goals

Sketch-Guided Program Optimisation

Sketches

- *sketches* are program patterns that leave details unspecified
- MM blocking:

baseline sketch:

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

blocking sketch:

```
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 Program Optimisation

Search

Challenge:

- automatically find a program that
    1. satisfies the sketch
    2. is equivalent to the unoptimised program
- by exploring many different ways to apply semantic-preserving rewrite rules

Sketch-Guided Program Optimisation

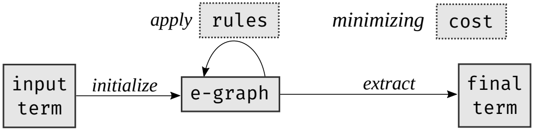
Search

Challenge:

- automatically find a program that
  1. satisfies the sketch
  2. is equivalent to the unoptimised program
- by exploring many different ways to apply semantic-preserving rewrite rules

To do this efficiently, we look at *Equality Saturation*

Equality Saturation



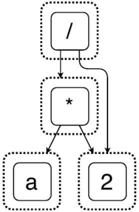
- An e-graph efficiently represents a large set of equivalent programs.
- The e-graph is grown by applying all possible rewrite rules in a purely additive way.
- After growing the e-graph, the best program found is extracted.

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

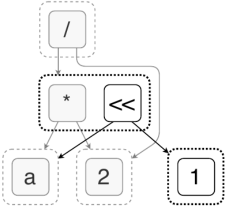
Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. “egg: fast and extensible equality saturation”. In: POPL (2021)

Equality Saturation

E-Graph Example

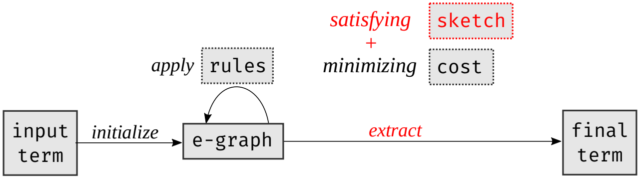


$(a * 2)/2$



$x * 2 \longrightarrow x \ll 1$

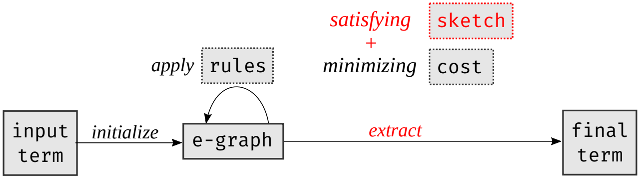
Equality Saturation



Questions:

1. How do we implement a sketch-satisfying extraction procedure?
2. How does it work for functional R**ISE** programs?
   - no efficient support for name bindings, rewritten languages are usually first order
3. Does it scale to complex optimisations?
   - as the e-graph grows, iterations become slower and require more memory

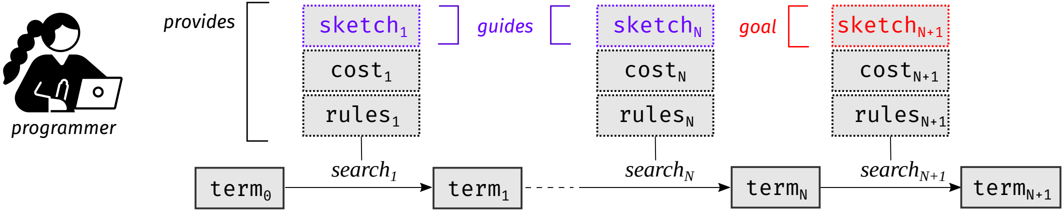
Equality Saturation



Focus in this talk:

3. Does it scale to complex optimisations?
    - as the e-graph grows, iterations become slower and require more memory

Sketch-Guided Equality Saturation



- factors a complex search into a sequence of smaller searches
- additional sketch guides specify intermediate goals
- each search should be sufficiently simple for equality saturation

Sketch-Guided Equality Saturation

MM Blocking

baseline sketch:

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

sketch guide:

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 sketch:

```
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))))))   |       .. + .. * ..
```

Evaluation

Search Runtime and Memory Consumption

- 7 matrix multiplication optimisation goals
- Equality Saturation without Sketch Guides[1]:

goal	found?	runtime	RAM
baseline	✓	0.5s	0.02 GB
blocking	✓	>1h	35 GB
+ 5 others	✗	>1h	>60 GB

- Sketch-Guided Equality Saturation[2]:

goal	sketch guides	found?	runtime	RAM
baseline	0	✓	0.5s	0.02 GB
blocking	1	✓	7s	0.3 GB
+ 5 others	2-3	✓	≤7s	≤0.5 GB

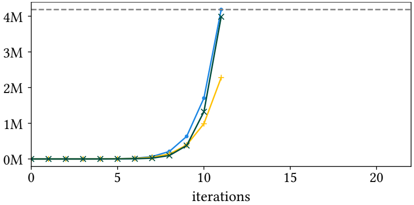
---

[1] Intel Xeon E5-2640 v2

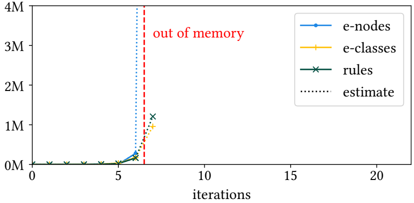
[2] AMD Ryzen 5 PRO 2500U

Evaluation

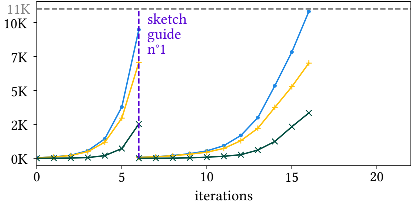
E-Graph Evolution



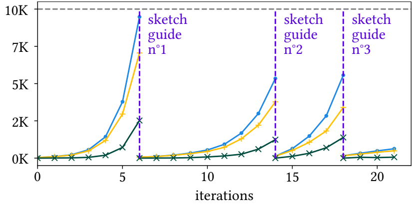
(a) unguided, *blocking*, found: ✓



(b) unguided, *parallel*, found: ✗



(c) sketch-guided, *blocking*, found: ✓



(d) sketch-guided, *parallel*, found: ✓

Evaluation

Sketch Guides

goal	sketch guides	sketch goal	sketch sizes	program size
blocking	*split*	*reorder*$_1$	7	90
vectorization	*split* + *reorder*$_1$	*lower*$_1$	7	124
loop-perm	*split* + *reorder*$_2$	*lower*$_2$	7	104
array-packing	*split* + *reorder*$_2$ + *store*	*lower*$_3$	7-12	121
cache-blocks	*split* + *reorder*$_2$ + *store*	*lower*$_4$	7-12	121
parallel	*split* + *reorder*$_2$ + *store*	*lower*$_5$	7-12	121

- each sketch corresponds to a logical transformation step
- sketches elide around 90% of the program
- intricate details such as array reshaping patterns are not specified (e.g. `split`, `join`, `transpose`)

Conclusion

We propose:

- *sketches* to guide optimisation, alternative to step-by-step *rewriting strategies*
- *sketch-guided equality saturation*, a novel, semi-automatic optimisation technique

Future work:

- design effective sketch guides for more diverse applications
- synthesize sketch guides from a sketch goal
- use in an interactive optimisation assistant

Conclusion

We propose:

- *sketches* to guide optimisation, alternative to step-by-step *rewriting strategies*
- *sketch-guided equality saturation*, a novel, semi-automatic optimisation technique

Future work:

- design effective sketch guides for more diverse applications
- synthesize sketch guides from a sketch goal
- use in an interactive optimisation assistant

✉ 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, ...

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**! 👍 👍

Sketch Definition

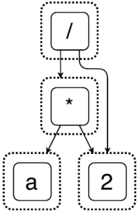
$$S \;::=\; ? \;\mid\; F(S, .., S) \;\mid\; contains(S)$$

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

```
def containsMap(n: NatSketch, f: Sketch): Sketch =
  contains(app(map :: ?t → n.?dt → ?y, f))
def containsReduceSeq(n: NatSketch, f: Sketch): Sketch =
  contains(app(reduceSeq :: ?t → ?t → n.?dt → ?t, f))
def containsAddMul: Sketch =
  contains(app(app(+, ?), contains(×)))
```

E-Graph Example

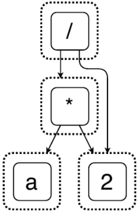
$$(a * 2)/2 \longrightarrow^{*} a$$



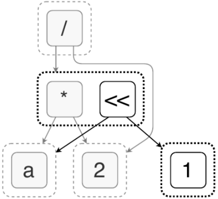
$(a * 2)/2$

E-Graph Example

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



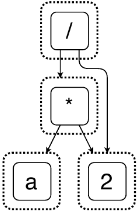
$(a * 2)/2$



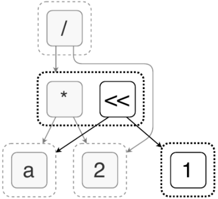
$x * 2 \longrightarrow x \ll 1$

E-Graph Example

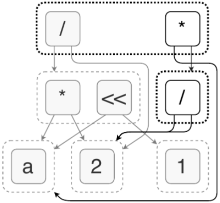
$$(a * 2)/2 \longrightarrow^{*} a$$



$(a * 2)/2$



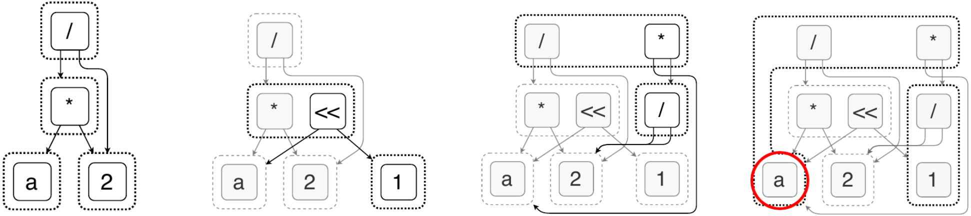
$x * 2 \longrightarrow x \ll 1$



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

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