Michel Steuwer

University of Edinburgh

FHNPC 2021

It’s all about Choices!

- I couldn’t agree more with Sven-Bodo about this statement

- I (of course) couldn’t disagree more with Sven-Bodo about all the choices made in SaC 😉

Shouldn’t the consequence be not to *make* choices but to *offer* them?

RISE + ELEVATE : Expose Optimization Choices

[ICFP 2020] [CGO 2021]

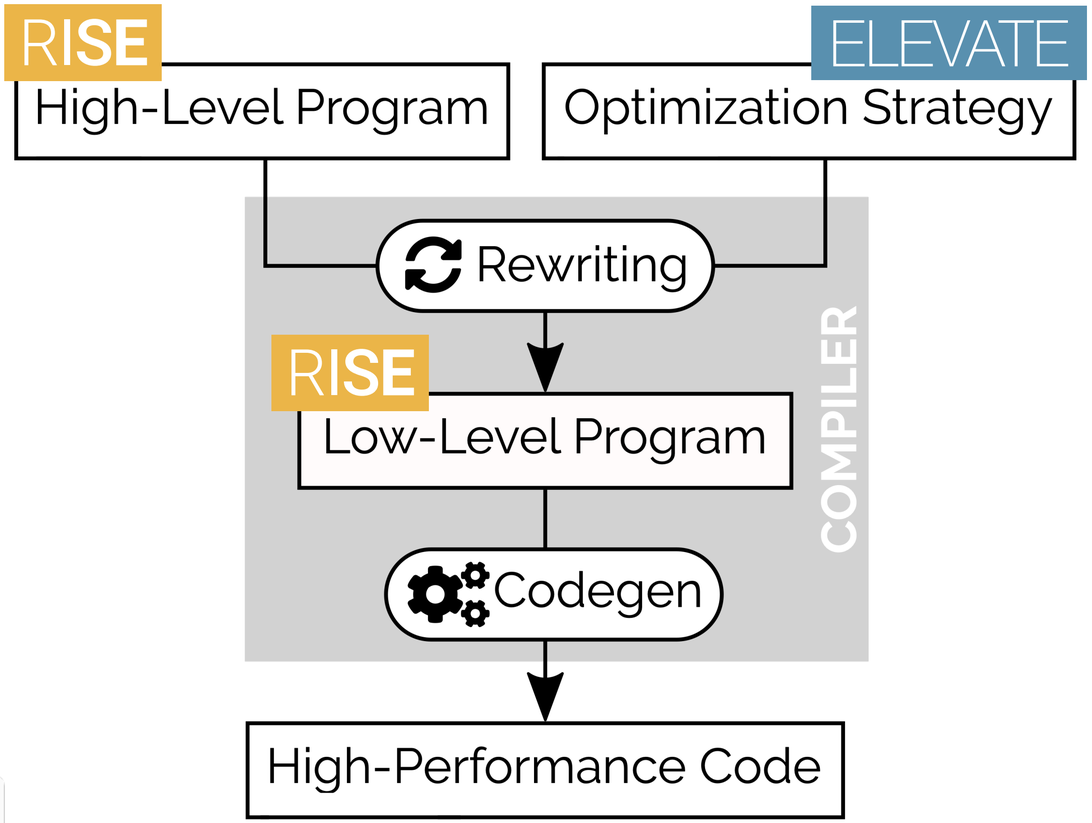
RISE A Pattern-Based Intermediate Languages

Successor to LIFT

```
def highLevelProgram =
  depFun((n: Nat, m: Nat, o: Nat) ⇒
    fun(A: n.o.f32 ⇒ fun(B: m.o.f32 ⇒
      A ▷ map(fun(rowOfA ⇒
        B ▷ map(fun(rowOfB ⇒
          zip(rowOfA)(rowOfB) ▷
            map(fun(x ⇒ fst(x) * snd(x))) ▷
              reduce(add)(0.0f) )) )) ))
```

ELEVATE A Programming Language for describing Optimization Strategies

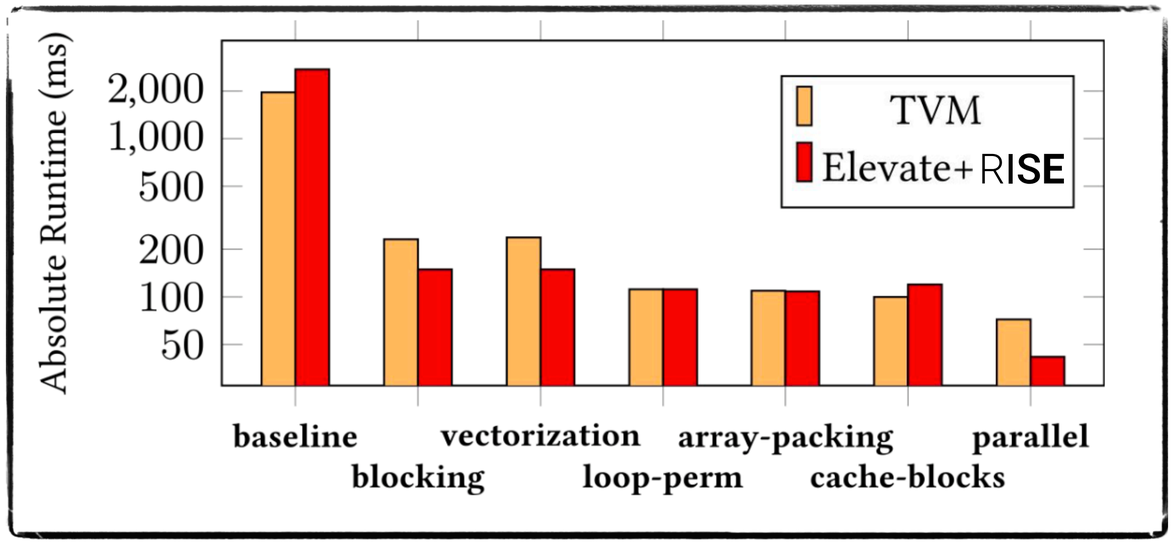
```
def optimizationStrategy =
  (`map ⟼ mapPar`        `@` outermost(isMap))   `;`
  (`map ⟼ mapSeq`        `@` outermost(isMap))   `;`
  (`reduce ⟼ reduceSeq` `@` everywhere)
```

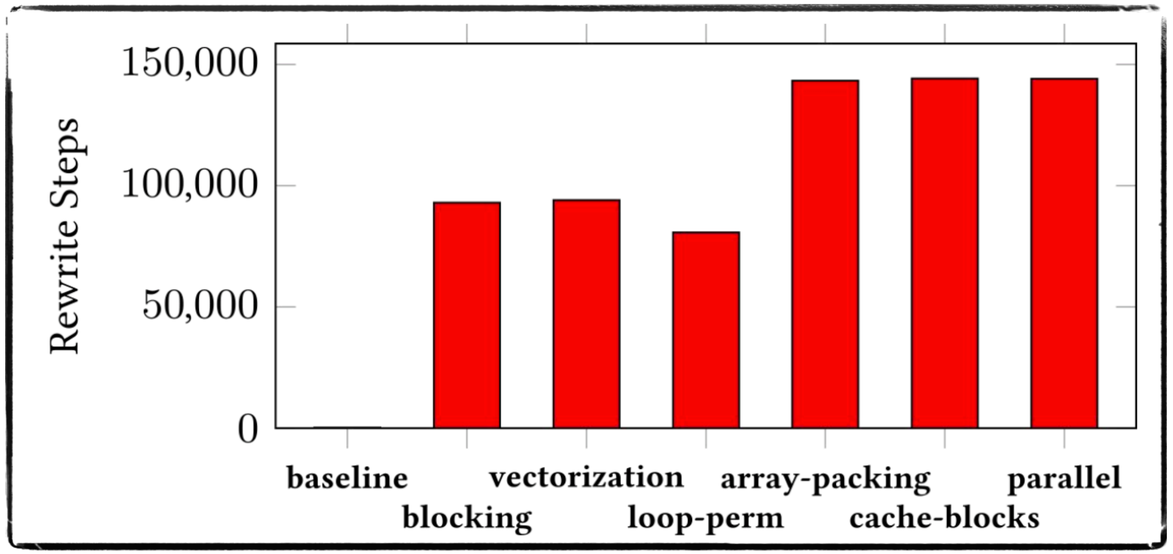


https://rise-lang.org

https://elevate-lang.org

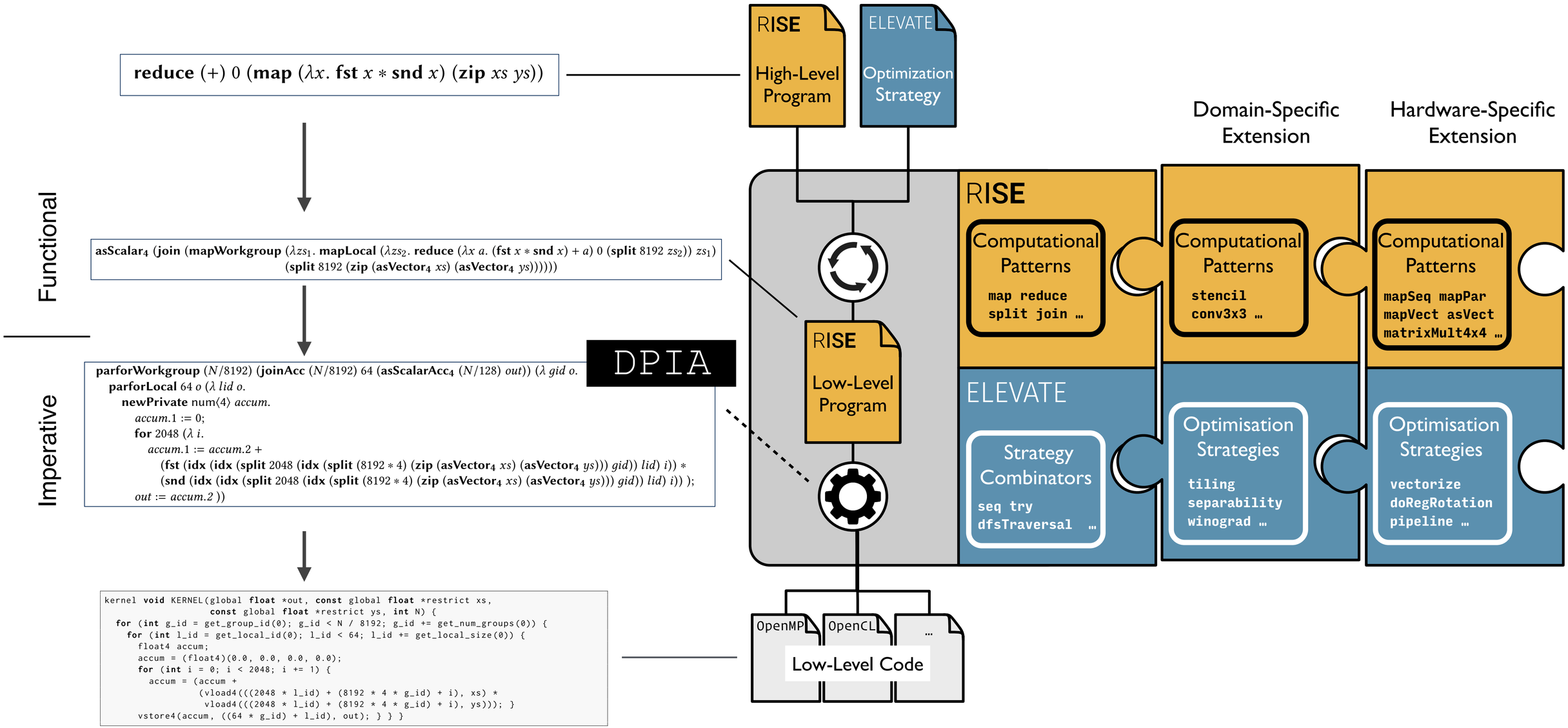
RISE + ELEVATE : Results for Matrix Multiplication





ELEVATE allows to implement state-of-the art scheduling APIs from first principle

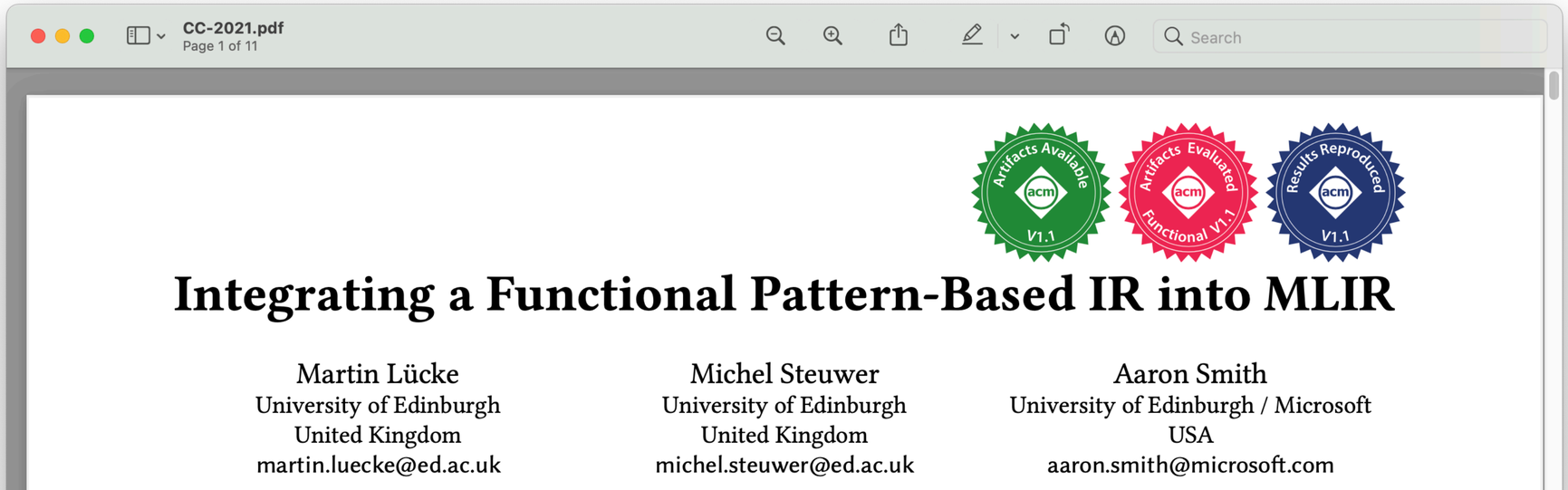
Compilation via multiple intermediate languages



MLIR: Offer integration choices

- Focus on compiler intermediate languages rather than user facing languages
  - Avoid flame war over functional programming
  - Type systems (e.g. dependent types) can be complex to carry rich information
  - Easy(er) to build fully integrated systems

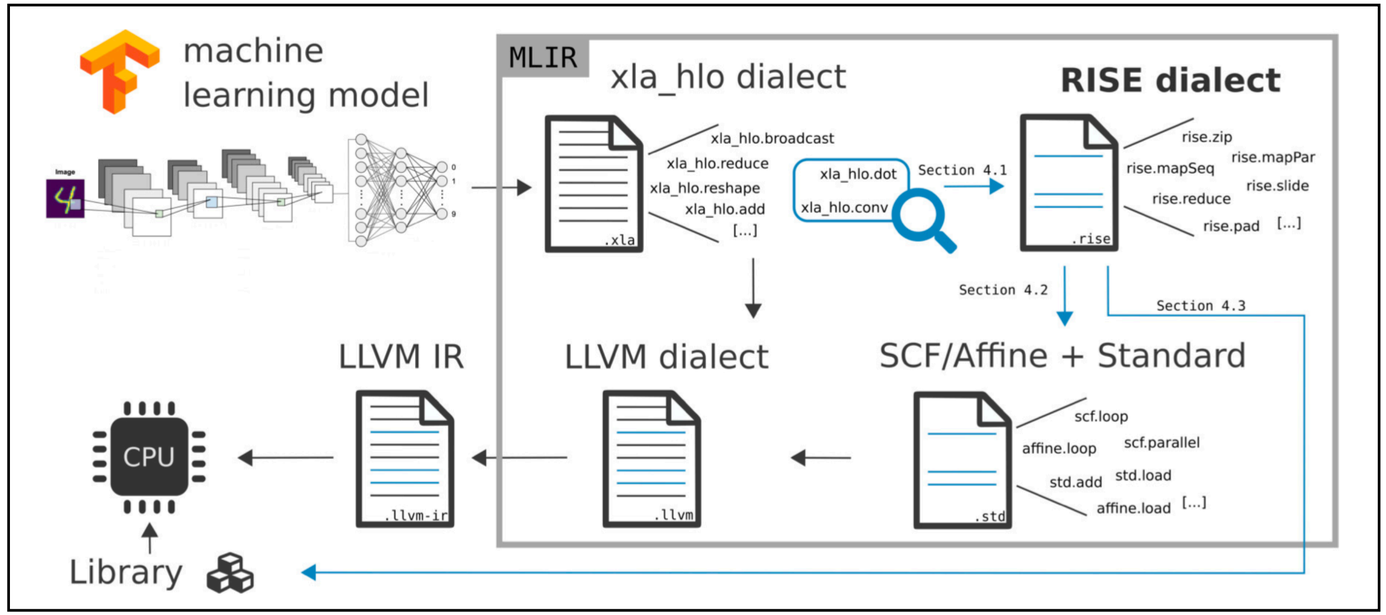
[CC 2021]



MLIR: Offer integration choices

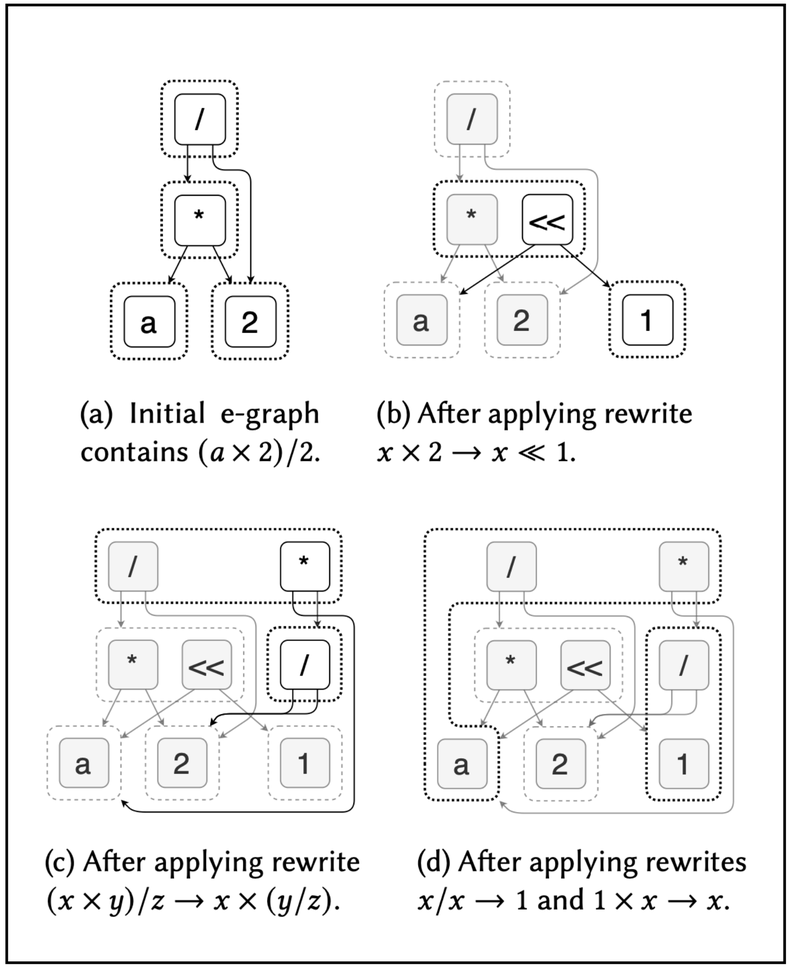
```
A |> map(fun(arow, B |> transpose |> map(fun(bcol,
     zip(arow,bcol) |> reduce(fun((ab,acc), (ab1×ab2)+acc),0)))))
```

```
func @mm_fused(%outArg, %inA, %inB) {
 %A = in %inA
 %B = in %inB
 %t = rise.transpose #rise.nat<2048>
                     #rise.nat<2048> #rise.scalar<f32>
 %B_t = rise.apply %t, %B
%m1fun = lambda (%arow) -> array<2048, scalar<f32>> {
 %m2fun = lambda (%bcol) -> scalar<f32> {
  %zipFun = zip #nat<2048> #scalar<f32> #scalar<f32>
  %zippedArrays = rise.apply %zipFun, %arow, %bcol
  %reduceLambda = lambda(%tuple, %acc)->scalar<f32> {
   %fstFun = rise.fst #scalar<f32> #scalar<f32>
   %sndFun = rise.snd #scalar<f32> #scalar<f32>
   %first  = rise.apply %fstFun, %tuple
   %second = rise.apply %sndFun, %tuple
   %result = rise.embed(%first, %second, %acc) {
    %product = mulf %first, %second :f32
    %result = addf %product, %acc : f32
    return %result : f32
   }
   return %result : scalar<f32>
  }
  %init = rise.literal #lit<0.0>
  %reduceFun = reduceSeq #nat<2048> #tuple
  %result = rise.apply %reduceFun, %reduceLambda,
                       %init, %zippedArrays
  return %result : scalar<f32>
 }
 %m2 = mapSeq #nat<2048> #array<2048, scalar<f32>>
                         #scalar<f32>
 %result = rise.apply %m2, %m2fun, %B_t
 return %result : array<2048, array<2048, scalar<f32>>>
}
%m1 = mapSeq  #nat<2048> #array<2048, scalar<f32>>
                         #array<2048, scalar<f32>>
%result = rise.apply %m1, %m1fun, %A
out %outArg <- %result
return
}
```



How to make choices?

- Fully manual via E LEVATE
- Fully automated via:
  - Stochastic methods **[ICFP 2015]**
  - Equality Saturation & E-graphs:
    Search *"Optimizing Functional Programs with Equality Saturation" on YouTube*
  - Reinforcement Learning & other machine learning methods

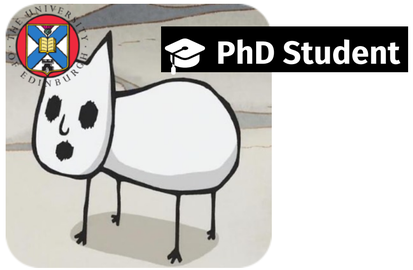


- Big open question: *How can we mix both modes conveniently?*

Team

Martin Lücke

Federico Pizzuti



Xueying Qin

Johannes Lenfers

Thomas Kœhler

Bastian Köpcke

Rongxiao Fu

Compilers

Programming Languages