RISE in MLIR

A Functional Pattern-based Dialect

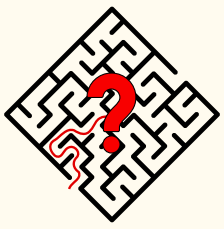
1

Machine Learning Systems are Stuck in a Rut!

(two original TensorFlow authors agree) [HotOS2019]

Problems:

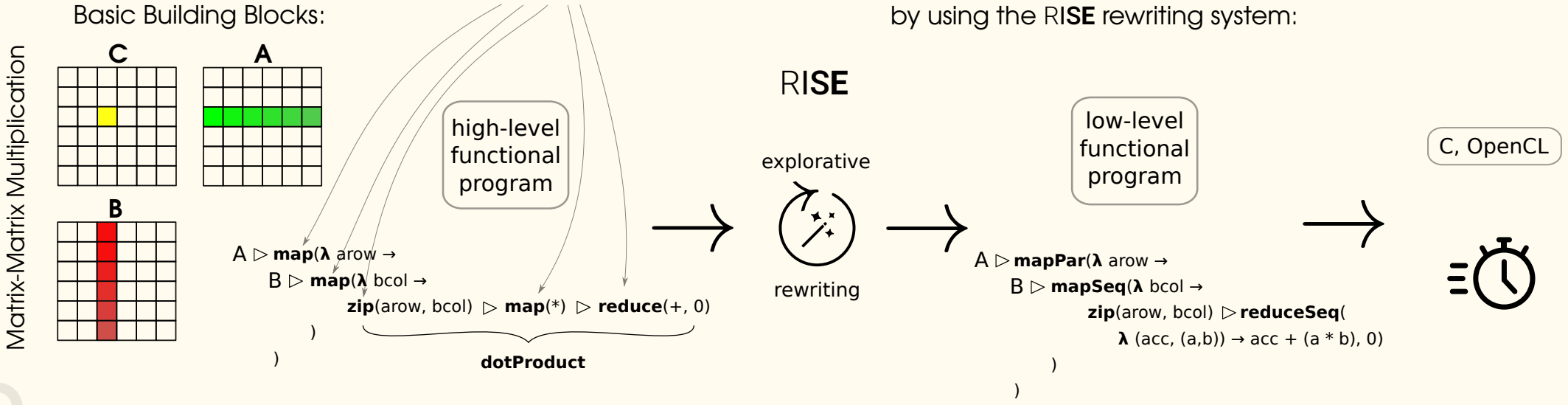
- Inflexibility of ML framework APIs halts innovation
- Lack of extensibility of ML systems due to highly optimized opaque operators
- Prototyping new ideas often requires new operators in the IR



2

A Modular Pattern-based Approach

- Expressing Computations in Terms of Patterns as Basic Building Blocks:
- High performance on heterogenous hardware by using the RISE rewriting system:



3

RISE as MLIR Dialect

Functional Lambda Calculus:

Function abstraction:

lambda (mlir::Op)
region : mlir::region funType : FunType
type: () → funType

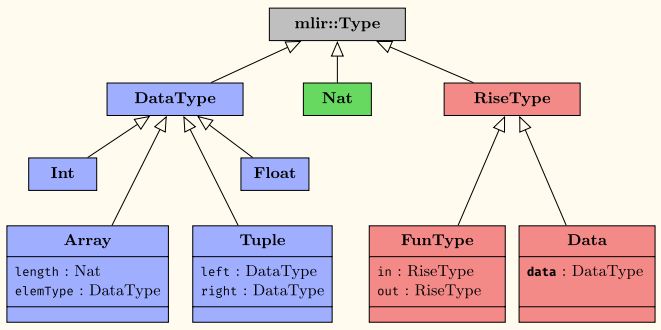
Function application:

apply (mlir::Op)
fun : mlir::Value arg0 : mlir::Value ... argN : mlir::Value
type: (arg0.type, . . . , argN.type) → fun.type.out

Identity example ($y \triangleright \lambda\, x \to x$):

```
%id  = rise.lambda (%x) : !rise.fun<data<int> -> data<int>> {
    rise.return %x : !rise.data<int>
  }
%res = rise.apply %id %y
```

Types:



High-Level Patterns:

zip (mlir::Op)	map (mlir::Op)	reduce (mlir::Op)
n : Nat dt1 : DataType dt2 : DataType	n : Nat dt1 : DataType dt2 : DataType	n : Nat dt : DataType
type: () → ([dt1]$_n$ ↦ ([dt2]$_n$ ↦ [(dt1, dt2)]$_n$))	type: () → ((dt1 ↦ dt2) ↦ ([dt1]$_n$ ↦ [dt2]$_n$))	type: () → ((dt ↦ (dt ↦ dt)) ↦ (dt ↦ ([dt]$_n$ ↦ dt)))

Matrix-Matrix Multiplication in MLIR RISE:

```
1  func @mm(%A: !rise.data<array<4, array<4, int>>>,
2           %B: !rise.data<array<4, array<4, int>>>) ->
3                (!rise.data<array<4, array<4, int>>>) {                  map(fun(arow =>
4    %f1 = rise.lambda (%arow) :
5                !rise.fun<data<array<4, int>> -> data<array<4, int>>> {  map(fun(bcol =>
6    %f2 = rise.lambda (%bcol) :
7                !rise.fun<data<array<4, int>> -> data<array<4, int>>> {
8     %zip = rise.zip #rise.nat<4> #rise.int #rise.int
9     %zipped = rise.apply %zip, %arow, %bcol                             zip(arow, bcol)
10
11    %f = rise.lambda (%t) : !rise.fun<data<tuple<int, int>> -> data<int>> {
12     %fst = rise.fst #rise.int #rise.int                                 ∨
13     %snd = rise.snd #rise.int #rise.int
14     %t1  = rise.apply %fst, %t
15     %t2  = rise.apply %snd, %t                                          map(*)
16     %mul = rise.mult #rise.int
17     %res = rise.apply %mul, %t1, %t2
18     rise.return %res
19    }
20    %map = rise.map #rise.nat<4> #rise.tuple<int, int> #rise.int         ∨
21    %mapped = rise.apply %map, %f, %zipped
22
23    %add = rise.add #rise.int
24    %init = rise.literal #rise.lit<int<0>>
25    %reduce = rise.reduce #rise.nat<4> #rise.int #rise.int               reduce(+, 0)
26    %res = rise.apply %reduce, %add, %init, %mapped
27    rise.return
28   }
29
30   %map = rise.map #rise.nat<4> #rise.array<4, int>  #rise.array<4, int>
31   %res = rise.apply %map, %f2, %B
32   rise.return                          4, array<4, int>>>
33  }                                                                       )
34
35  %map = rise.map #rise.nat<4> #rise.array<4,!rise.int> #rise.array<4,!rise.int>
36  %res = rise.apply %map, %f1, %A
37  return %res
38 }                                                                        )
```


Martin Lücke
Michel Steuwer
Aaron Smith

Univ. of Edinburgh
Univ. of Glasgow
Univ. of Edinburgh, Microsoft

RISE-lang.org/mlir

THE UNIVERSITY of EDINBURGH informatics