Utrecht University

PUBLIC DEFENSE THESIS

13 September 2023

Maintaining parallelism in reverse-mode automatic differentiation on functional parallel array languages

Simon van Hus
6147879

The Main Idea

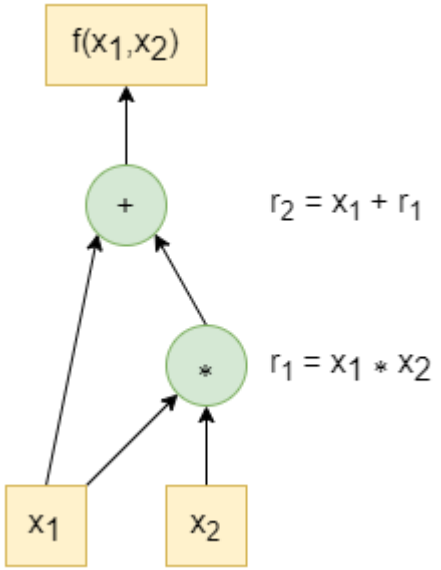
- Tracing
  - How does the program work?
- Automatic Differentiation
  - Finding the derivative of the program
- Parallelism
  - Find the derivative *quickly*

Utrecht University

Tracing

Tracing

- Single-Line program
- Function Elimination
- Generalized Type Elimination



Tracing

- Generalized Type Elimination
- Trace Consistency
- Trace Minimalism

$$\forall s \in S$$
$$\forall i \in I$$
$$\text{trace}(s, i) = (t \in T, o \in O)$$
$$(t \in S \cap T) \rightarrow (\text{eval}(s, i) = \text{eval}(t, i) = o)$$

$$\forall s \in S$$
$$\forall i \in I$$
$$\text{trace}(s, i) = (t \in T, o \in O)$$
$$\text{trace}(t, i) = (t, o)$$

Tracing

Function Tracing example

- Insert tracing into lambda expression
- Rediscover on application

Tracing Arrays

- Instantiation
- Map
- Fold

```
1  trace :: TEnvironment -> Expression -> (Value, Trace)
2  trace n (EApply e1 e2) =
3      -- First trace e1 and e2
4      let (v1, t1) = trace n e1
5          (v2, t2) = trace n e2
6      -- Check if v1 actually returns a function
7      in  case v1 of
8          -- Do the application, return the result and the combined trace
9          TFunc f -> let (vf, tf) = f v2
10                    in  (vf, tf ++ t2 ++ t1)
11         _       -> error ''Type mismatch in trace/EApply''
12
13 trace n (ELambda s e1) =
14     -- Define the function, insert value x as variable s into the environment that is currently
15     -- present, and trace the body
16     let f = TFunc (\x -> trace (insert s x) e1)
17     -- Return the function as abstracted function as a value, and no trace
18     in  (f, [])
```

Utrecht University

Automatic Differentiation

Automatic Differentiation

By hand
Finite Differencing
Automatic Differentiation

- Forward Mode
- Backwards/Reverse Mode

```
 x1 = 15
dx1 = 1
 x2 = 7
dx2 = 0
 r1 = x1 + x2
dr1 = dx1 + dx2
 y  = r1 × x2
dy  = r1 × dx2 + dr1 × x2
```

Listing 1: An example of forward mode AD by source transformation, with the AD statements in red

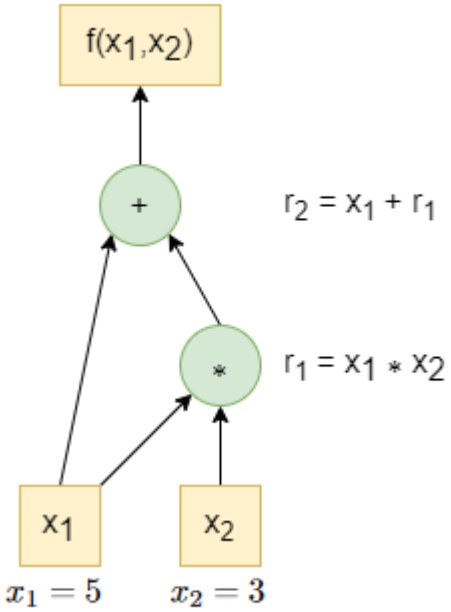
```
 x1 = 15
 x2 = 7
 r1 = x1 + x2
 y  = r1 × x2

dy  = 1
dr1 = dy × x2
dx2 = dy × r1 + dr1 × 1
dx1 = dr1 × 1
```

Listing 2: An example of reverse mode AD by source transformation, with the AD statements in red

Automatic Differentiation

Forward Pass

- Operations performed
- Reference counters
- Intermediate values



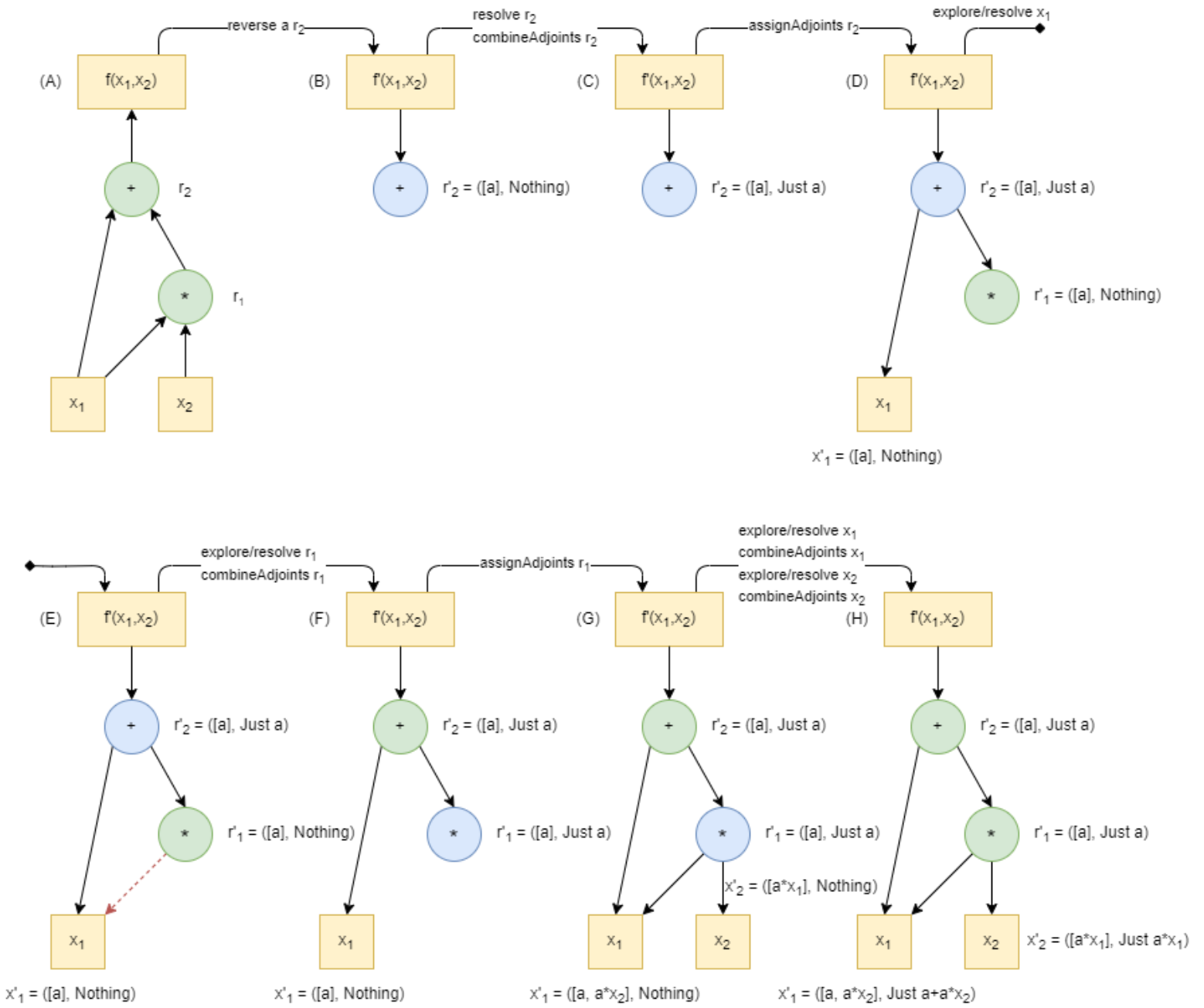

```
f :: Value -> Value -> Expression
f x1 x2 = ELet "x1" (ELift x1) (
    ELet "x2" (ELift x2) (
        EOp2 Add (ERef "x1") (
            EOp2 Mul (ERef "x1") (ERef "x2")
        )
    ))

trace_result :: (TValue, Trace)
trace_result = (TReal "r2" 20.0, [
    ("x1", TLift (TReal "x1" 5.0)),
    ("x2", TLift (TReal "x2" 3.0)),
    ("r1", TOp2 Mul "x1" "x2"),
    ("r2", TOp2 Add "x1" "r1")
])
```

Listing 21: DSL definition of f and its trace

Automatic Differentiation

Reverse Pass

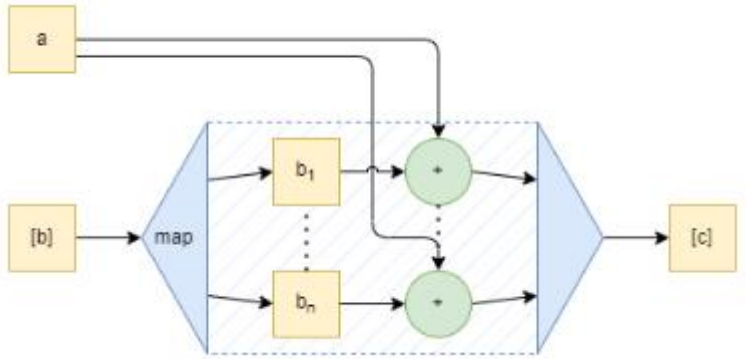
- Resolve node
- Combine adjoints
- Assign adjoints to ancestors
- Explore and resolve ancestors



Automatic Differentiation

Reverse Pass on arrays

- Map
- Fold
- Closures



Utrecht University

Parallelism

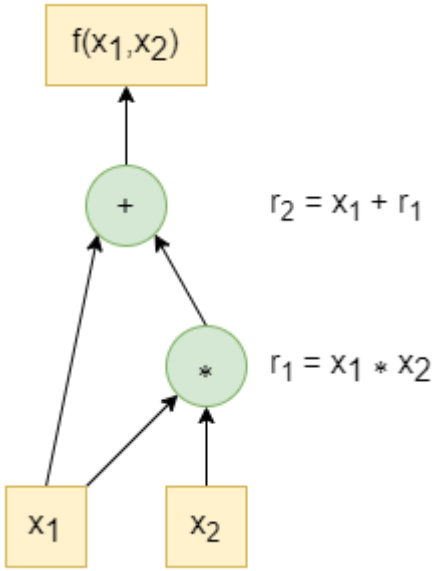
Parallelism

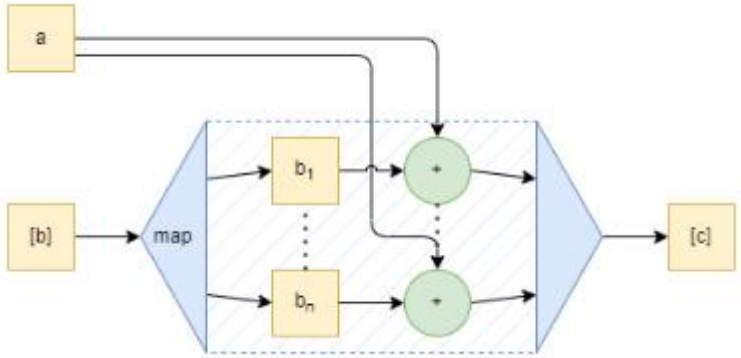
Data Parallelism
Task Parallelism
Distributed/Machine Parallelism

Parallelism

Task parallelism
- Concurrency
- Naming

Data parallelism
- Vectorization
- Segmentation

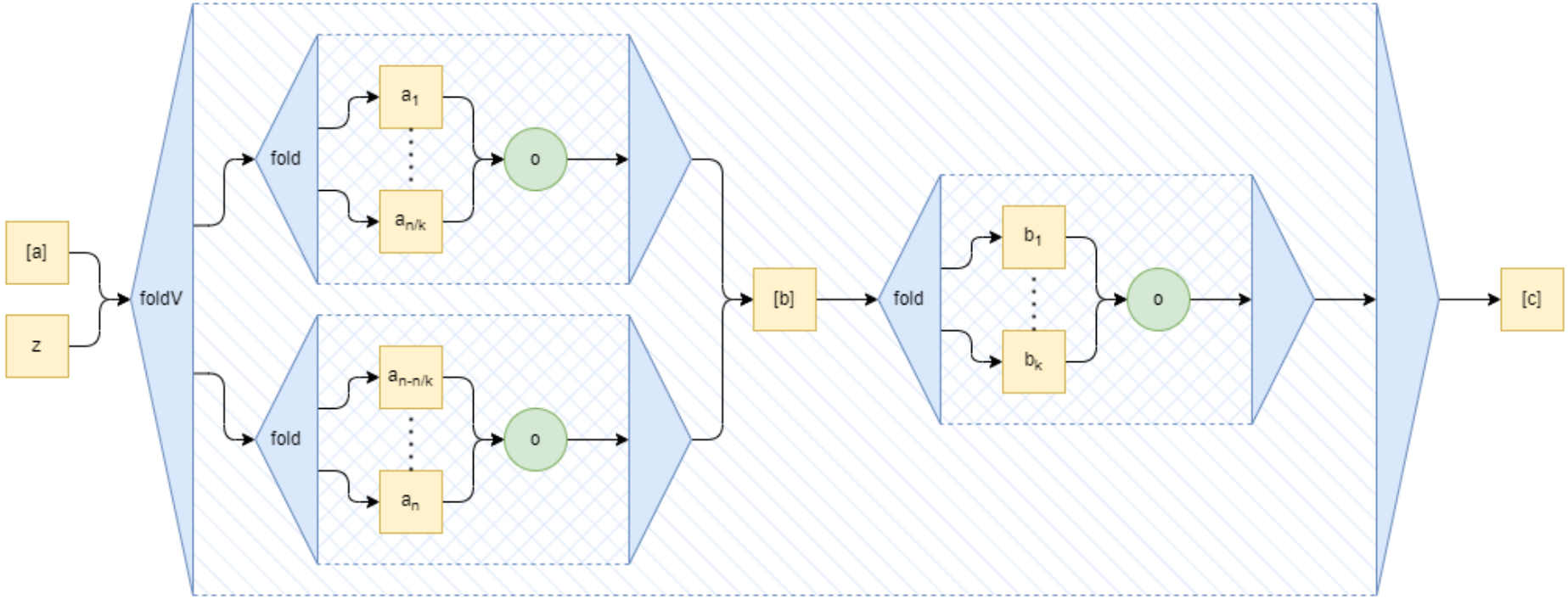




Parallelism

Task parallelism
- Concurrency
- Naming

Data parallelism
- Vectorization
- Segmentation



Conclusion

Conclusion

Tracing
- Formal Definition, Consistency, Minimalism

Automatic Differentiation
- Reference counting, intermediate values

Parallelism
- Task and data parallelism

That's it!

Simon van Hus
6147879

Utrecht University

Sharing science,
shaping tomorrow