Parallel evaluation of general vector-arithmetic trees

András Leitereg - Eötvös Loránd University

Dániel Berényi - Wigner Research Centre for Physics

Overview

- Motivation
- Category theory
- Implementation
- Conclusion

The problem

Scientists:
field knowledge and large scale computational needs

Programmers:
experience in parallel programming and optimization

The solution

Scientist

Expression
to evaluate

Library
for parallel
optimization

Objective

- Focusing on linear algebra applications
- Only CPU threads
- Choose execution strategy (sequential or parallel) based on cost estimation

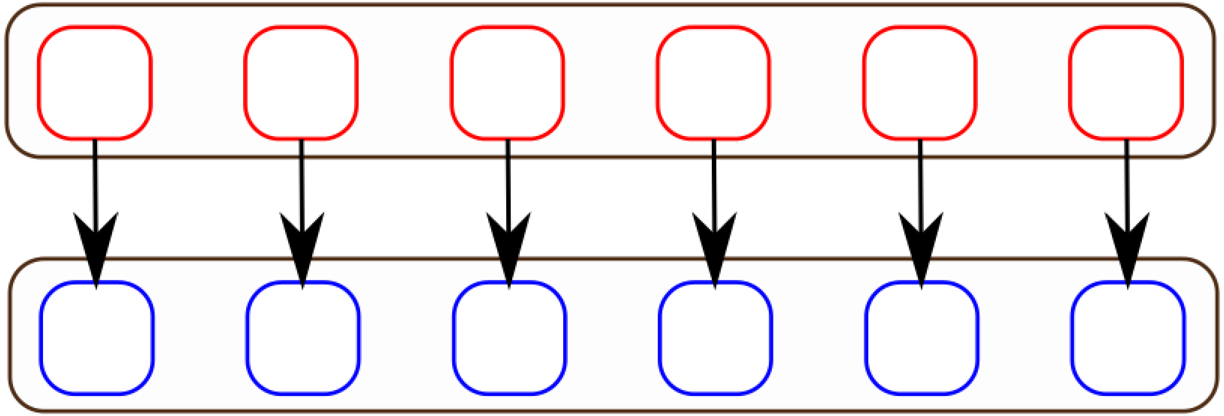
Workflow

1. Expression construction
2. Type checking
3. Cost estimation
4. Code generation
5. Building and loading evaluator function

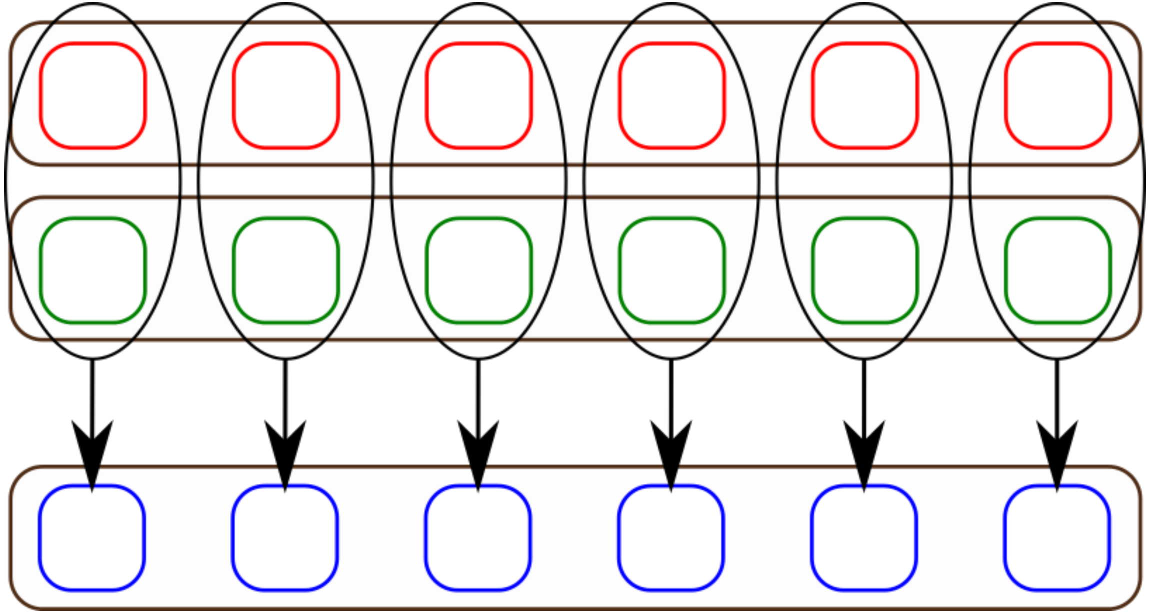
Functional approach

- Natural representation of calculations
- Easy to optimize
  - Theoretically proven equivalences
- Less room for user errors
- Easy to check correctness
- Describes the intention much better than a low level representation

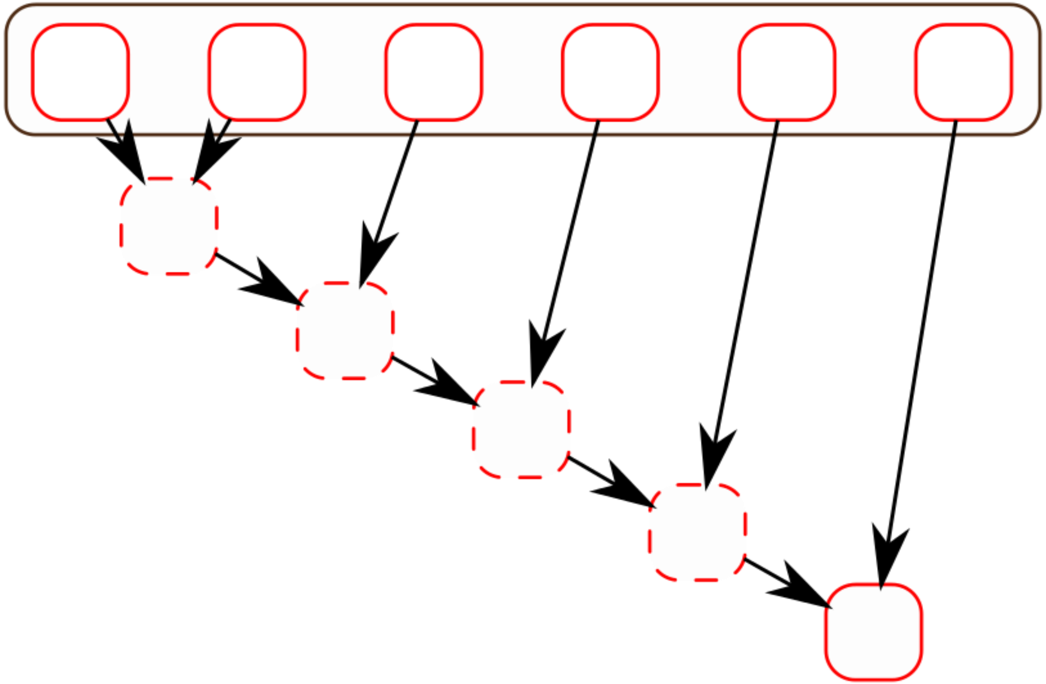
Map



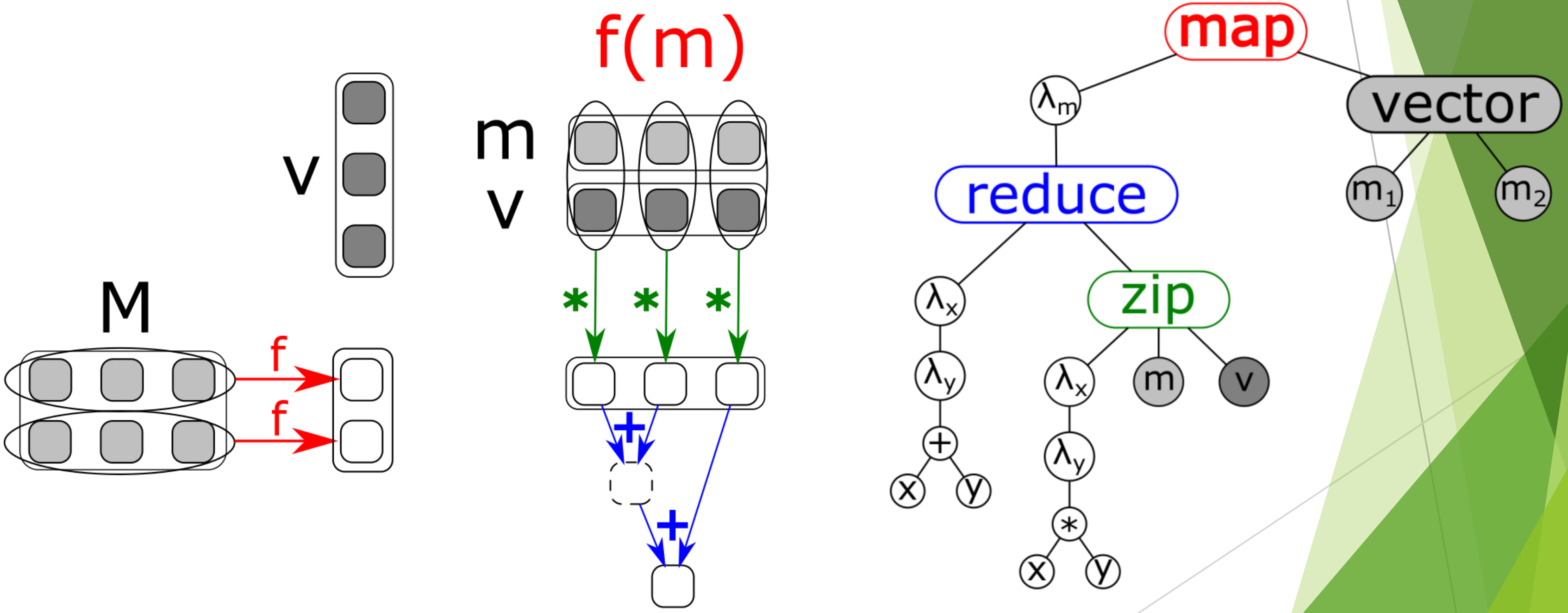
Zip



Reduce



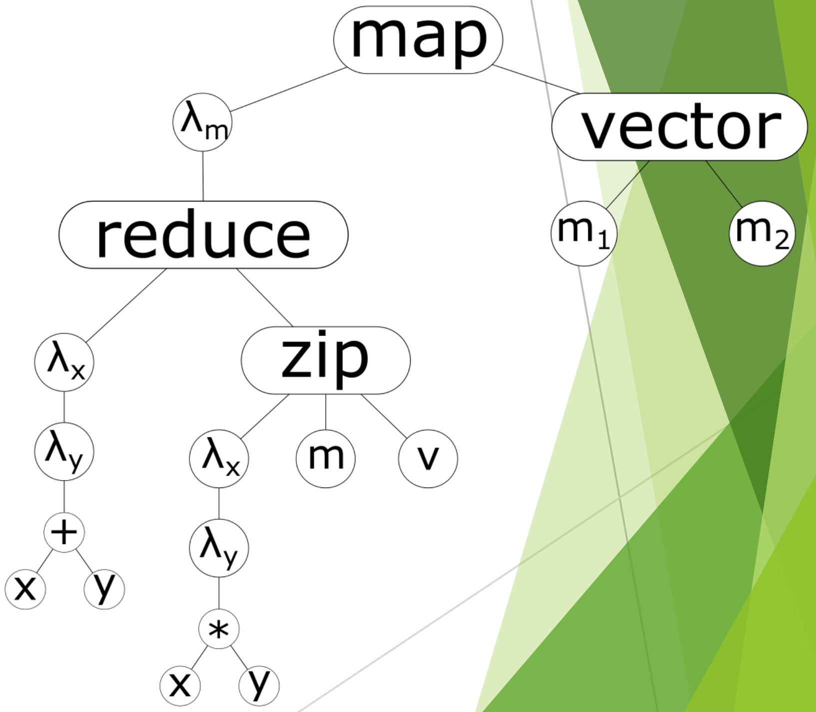
Matrix-vector multiplication



Processing passes on arbitrary trees

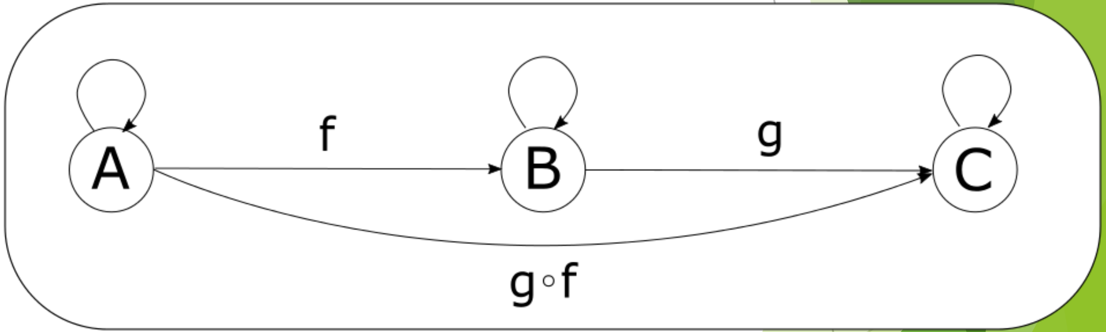
- Type checking
- Cost estimation
- Code generation

How to define these abstractly?

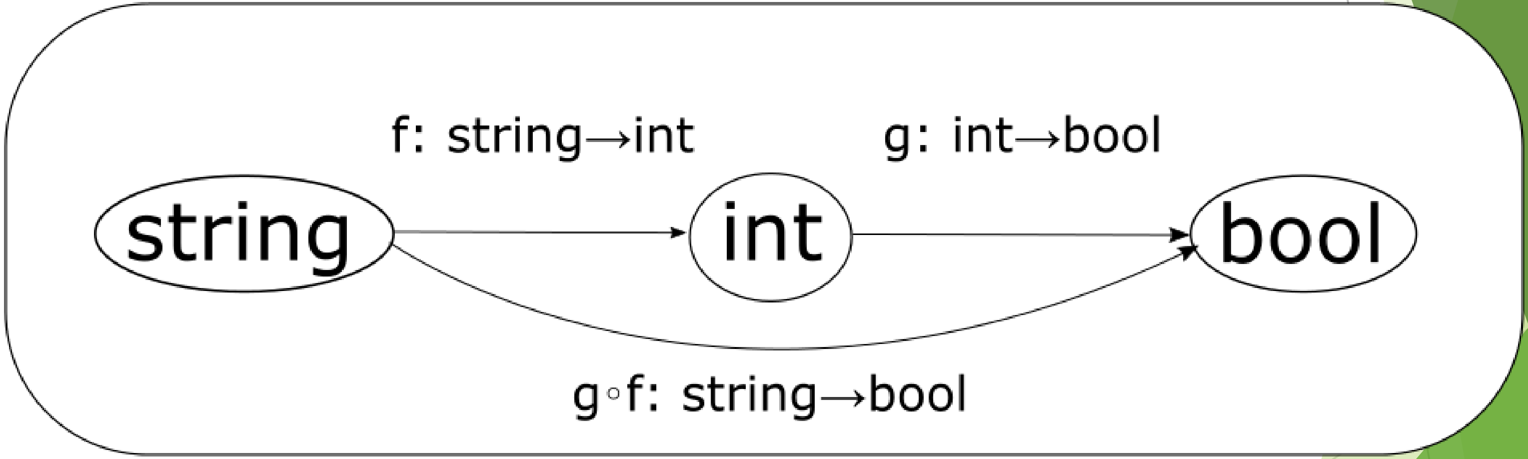


Category theory

- Formalizes relations between algebraic structures
- Objects and morphisms (arrows)
- Identity morphism for every object
- Morphisms can be composed associatively



Example with types



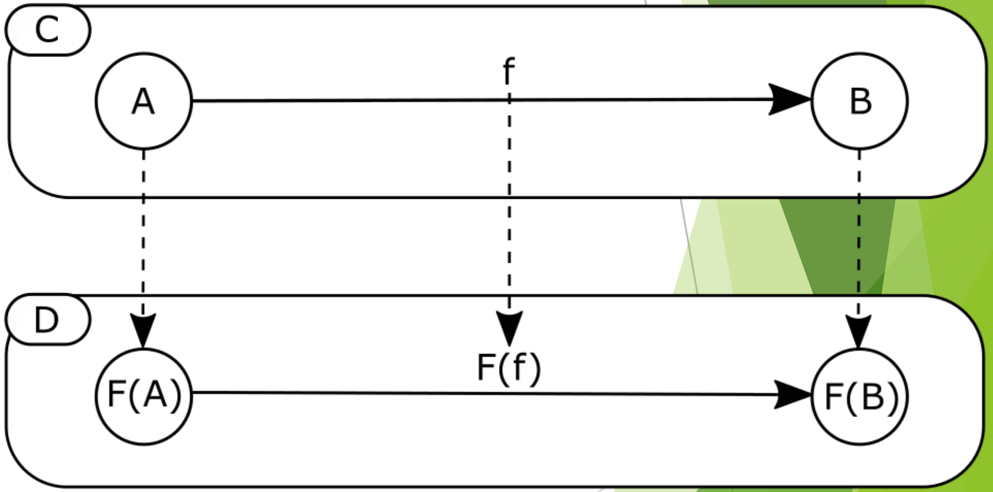
Functor

- Mapping between categories
- Mapping of morphisms (fmap) preserves
- Identity:

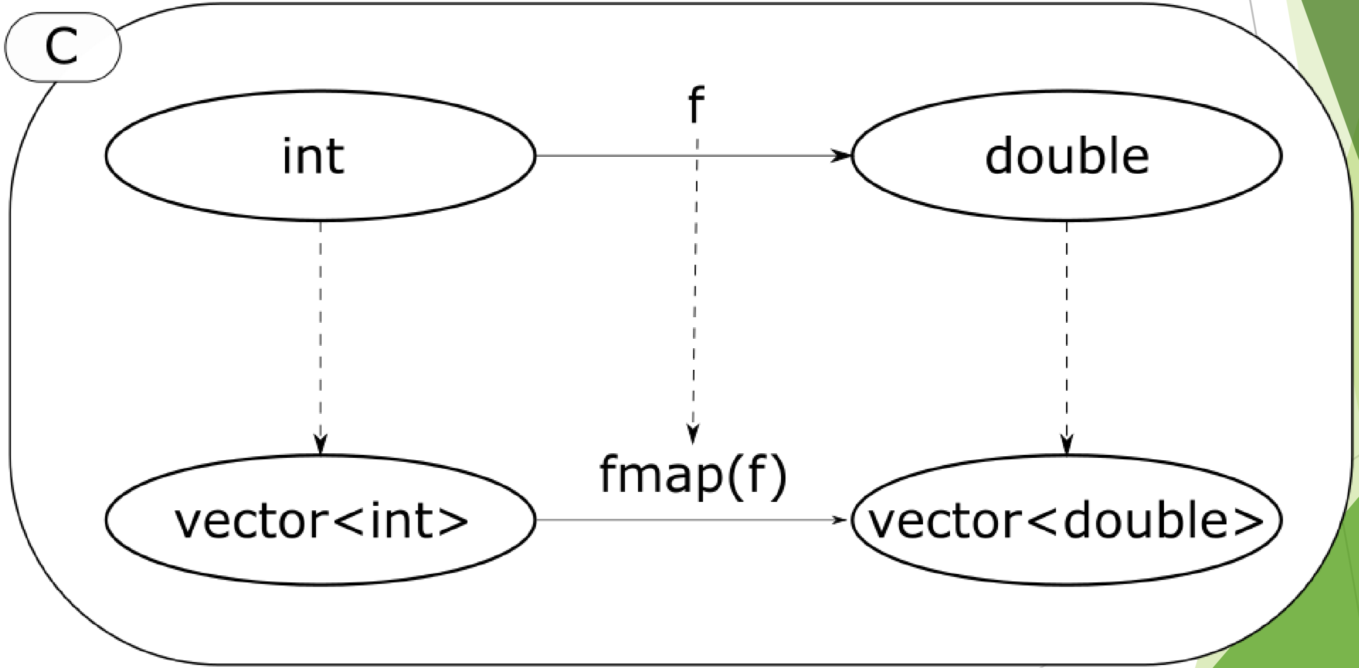
$$F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$$

- Composition:

$$F(g \circ f) = F(g) \circ F(f)$$



Functor example

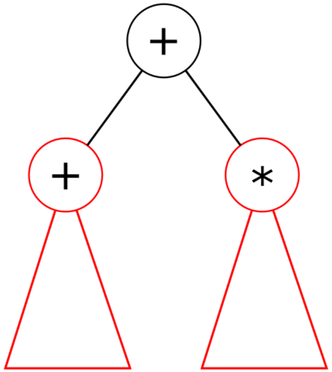


Expression trees

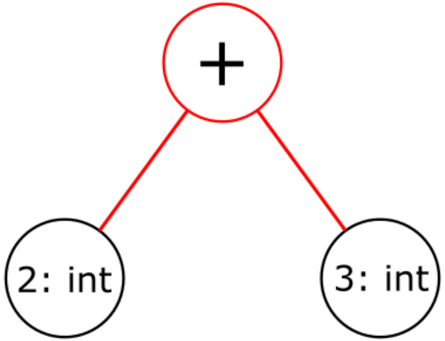
- We represent the operations in an expression tree with types,
- so we can use results from category theory to work with the abstract trees.
- We have Scalar, Variable, +, ×, Function, Application, Map, Reduce and Zip nodes.

Recursive tree processing

An arbitrary expression tree can be evaluated by recursively:



applying the evaluator function to each child of the current node through an fmap, and



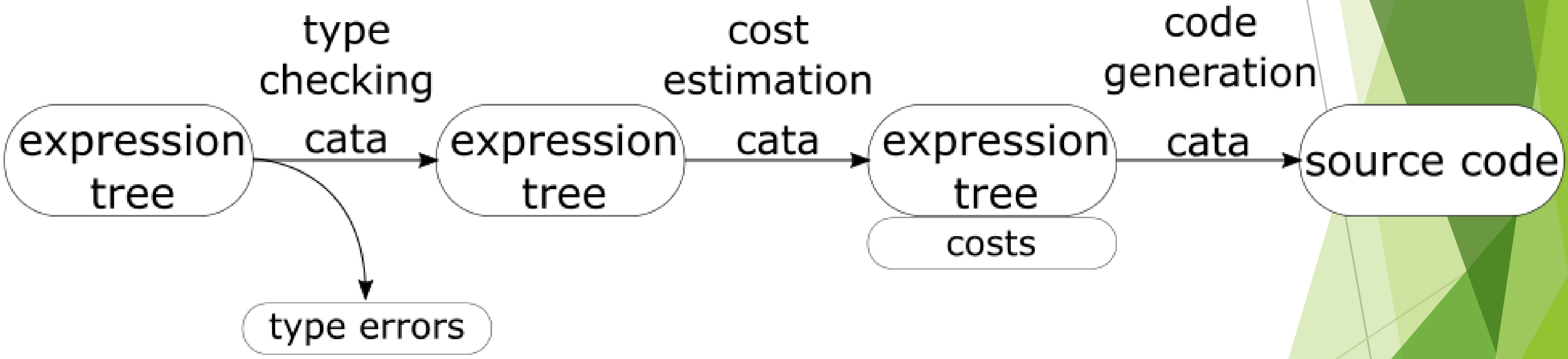
evaluating the current node using specific logic of the represented operation.

The result of the evaluation must be the same type for any tree.

Catamorphism

- A general concept from category theory.
- Formalizes the previous process for arbitrary tree representations and transformations.
- Algebra: the collection of “operation specific logics”.
- Carrier type: the type of the transformation result.
- $cata(alg) = alg \circ fmap\,(cata\ alg) \circ unfix$

Catamorphic transformations



Implementation

- The library is implemented in standard C++.
- We built upon Eric Niebler’s sample F-algebra implementation
- The type of the expression tree nodes is expressed with `boost::variant`, and the algebras are `boost::static_visitor`-s.

Performance testing

Matrix-vector multiplication (16 rows, 10^7 cols)

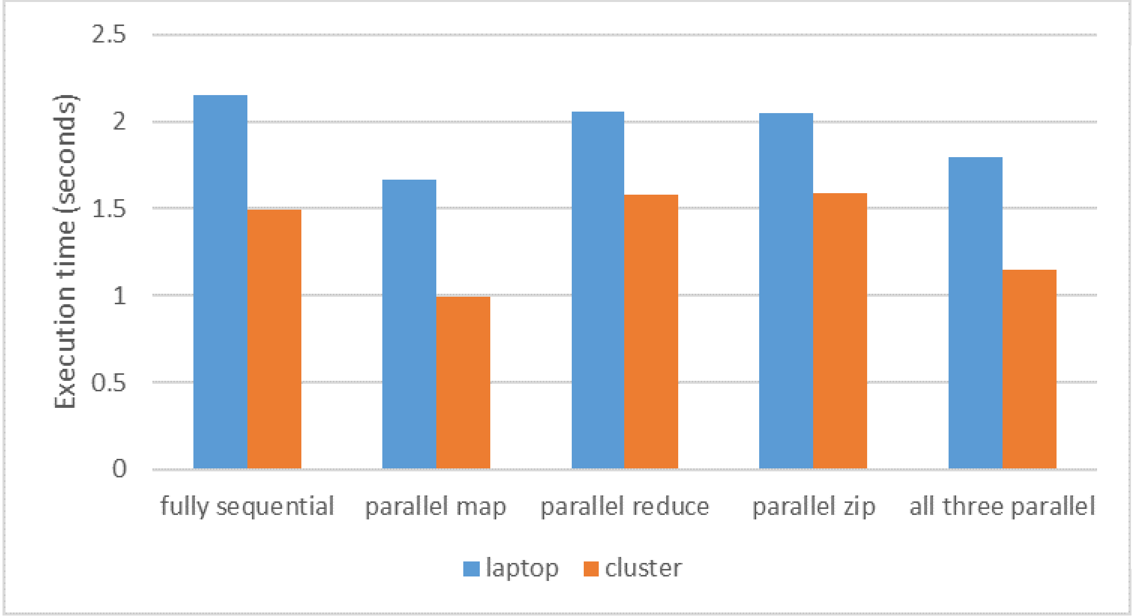
5 different execution strategies:

- Sequential
- Parallelizing only one of the map, reduce and zip
- Parallelizing all three

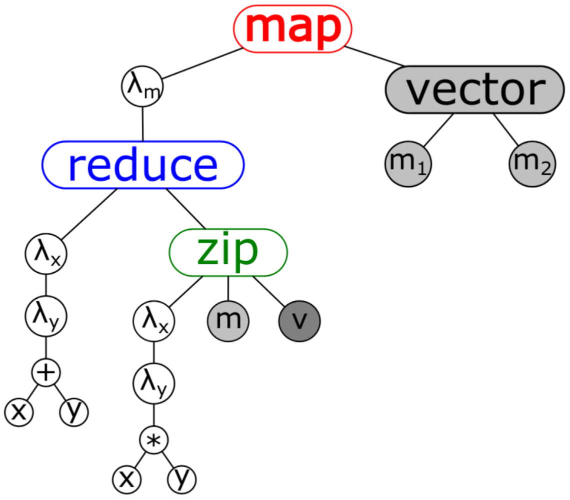
Two platforms:

- Laptop with 4 threads and 8GB memory
- HPC cluster with 8 threads and 32GB memory

Results



Execution time (seconds)
2.5
2
1.5
1
0.5
0
fully sequential
parallel map
parallel reduce
parallel zip
all three parallel
laptop
cluster



map
λ_m
vector
reduce
m_1
m_2
λ_x
zip
λ_y
λ_x
m
v
+
λ_y
x
y
*
x
y

Conclusion

- The implemented library successfully demonstrates that the functional approach is feasible
- Selective parallelization can be faster, than blindly parallelizing everything
- Easy to select parallelization level

Further plans

- Deduce memory allocations automatically from the expression tree
- Use category theory to optimize e.g.
  - Fmap:
    $$fmap(g) \circ fmap(f) = fmap(g \circ f)$$
  - Catafusion:
    $$cata(alg2) \circ cata(alg1) = cata(alg2 \circ alg1)$$
  - …
- Move to additional platforms (GPUs first)

Links

- You can follow the project on github.
- More detailed explanation here and here.