Robust Program Specialization using Equality Saturation

Jonathan Van der Cruysse
Christophe Dubach

McGill University

November 15, 2022

Overview

Goal: Recognize idioms in a functional array language

Goal: Recognize idioms in a functional array language

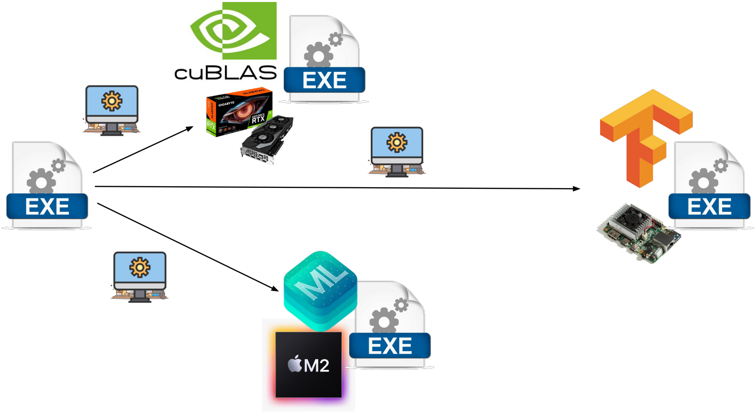
List of ingredients:

- Equality saturation
- The `build` and `ifold` operators

Let's accelerate!

Many hardware/software accelerators to speed up **specific patterns** in programs

Support all the platforms!



The idiom:

$$\alpha \cdot A \cdot B + \beta \cdot C \rightarrow \texttt{gemm(}\alpha\texttt{, } A\texttt{, } B\texttt{, } \beta\texttt{, } C\texttt{)}$$

An exact match: $1 \cdot X \cdot Y + 0 \cdot Z \rightarrow \texttt{gemm(1, } X\texttt{, } Y\texttt{, 0, } Z\texttt{)}$

The idiom:

$$\alpha \cdot A \cdot B + \beta \cdot C \rightarrow \texttt{gemm}(\alpha\texttt{, } A\texttt{, } B\texttt{, } \beta\texttt{, } C\texttt{)}$$

An exact match: $1 \cdot X \cdot Y + 0 \cdot Z \rightarrow \texttt{gemm(1, } X\texttt{, } Y\texttt{, 0, } Z\texttt{)}$
No match:

- $X \cdot Y$
- $2 \cdot X \cdot Y$
- $2 \cdot X \cdot Y + Z$
- $X \cdot Y + 2 \cdot Z$
- $Y + 2 \cdot Z$
- ...

A flexible pattern:

$$[\alpha\cdot]\, A \cdot B\, [+\, [\beta\cdot]\, C]$$

Limitations:

- Enumerate all variants
- Encode in idiom description language
- How to rewrite?

What if a machine could discover that

$$X \cdot Y = 1 \cdot X \cdot Y + 0 \cdot \mathbf{0} = \texttt{gemm(1, } X\texttt{, } Y\texttt{, 0, } \mathbf{0}\texttt{)}?$$

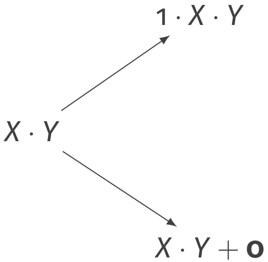
What if a machine could discover that

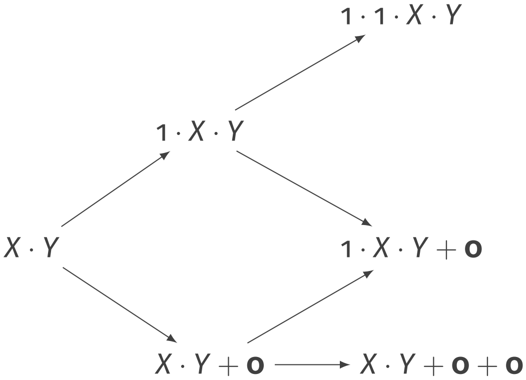
$$X \cdot Y = 1 \cdot X \cdot Y + 0 \cdot \mathbf{0} = \texttt{gemm(}1\texttt{, } X\texttt{, } Y\texttt{, } 0\texttt{, } \mathbf{0}\texttt{)}?$$

Principle:

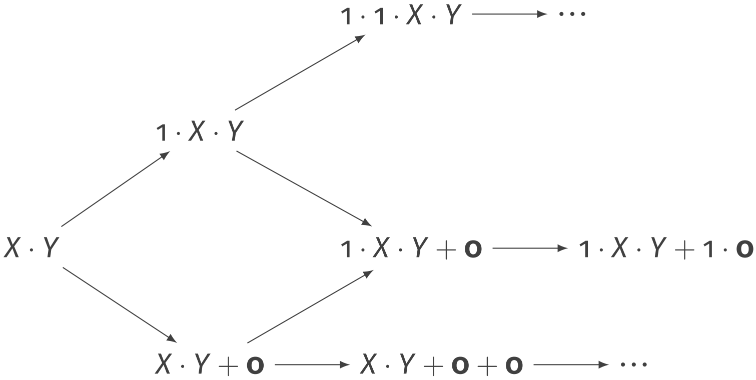
- Specify standard idiom, *e.g.*,
  $\alpha \cdot A \cdot B + \beta \cdot C \to \texttt{gemm(}\alpha\texttt{, } A\texttt{, } B\texttt{, } \beta\texttt{, } C\texttt{)}$
- Use language semantics to make patterns match
  $\Rightarrow A \to 1 \cdot A$ and $A \to A + \mathbf{0}$

Brute force



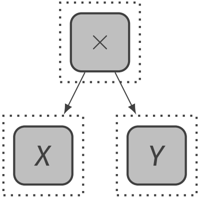

1 · 1 · X · Y
1 · X · Y
X · Y
1 · X · Y + 0
X · Y + 0
X · Y + 0 + 0

Brute force

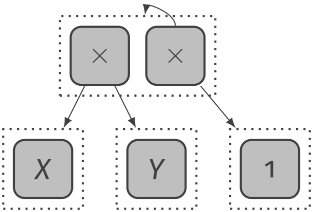

$1 \cdot 1 \cdot X \cdot Y$
$\cdots$
$1 \cdot X \cdot Y$
$X \cdot Y$
$1 \cdot X \cdot Y + \mathbf{0}$
$1 \cdot X \cdot Y + 1 \cdot \mathbf{0}$
$X \cdot Y + \mathbf{0}$
$X \cdot Y + \mathbf{0} + \mathbf{0}$
$\cdots$

Elegant alternative: Equality Saturation

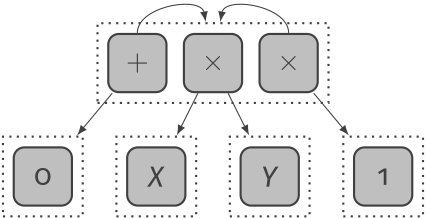
e-Graph for $X \cdot Y$



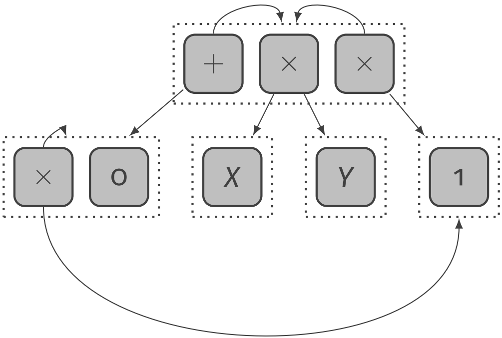
Apply $A \rightarrow 1 \cdot A$

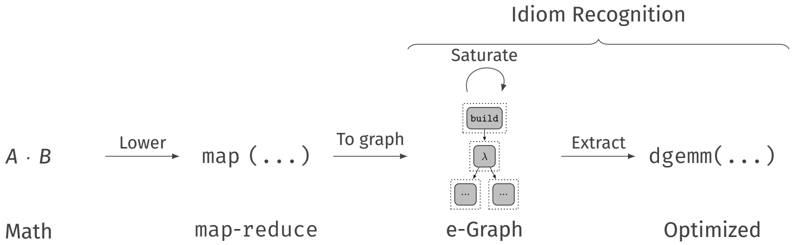


Apply $A \to A + \mathbf{0}$



Apply $A \to 1 \cdot A$




Idiom Recognition
Saturate
A · B
Lower
map (...)
To graph
build
λ
...
...
Extract
dgemm(...)
Math
map-reduce
e-Graph
Optimized

Putting the pieces together (naively)

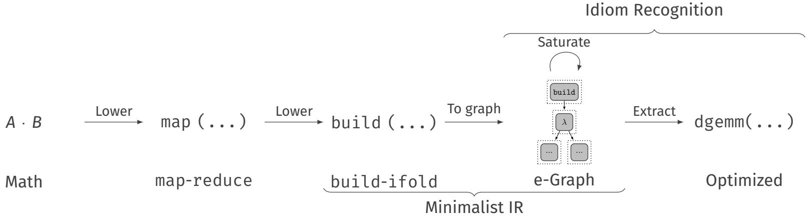
Need two components to recognize idioms:

1. Description of language semantics
2. Description of idioms

LANGUAGE SEMANTICS RULES (map-reduce)

$$\texttt{map f (map g xs)} = \texttt{map (g} \circ \texttt{f) xs}$$
$$\texttt{reduce f z (map g xs)} = \texttt{reduce (}\lambda \text{x. } \lambda \text{acc. } \texttt{g (f x) acc) z xs}$$
$$\texttt{zip (map f) (map g)} = \texttt{map (}\lambda \text{t. } \texttt{tuple (f (fst t)) (g (snd t)))}$$
$$\texttt{map f (join xs)} = \texttt{join (map (map f) xs)}$$
$$\texttt{split N (map f xs)} = \texttt{map (map f) (split N xs)}$$
$$\texttt{slide N (map f xs)} = \texttt{map (map f) (slide N xs)}$$

```
join (split N xs) = xs
zip (join xs ys) (join zs ws) = join (zip xs zs) (zip ys ws)
...
```


Idiom Recognition
Saturate
build
λ
...
...
A · B
Lower
map (...)
Lower
build (...)
To graph
Extract
dgemm(...)
Math
map-reduce
build-ifold
e-Graph
Optimized
Minimalist IR

build N f = | f 0 | f 1 | … | f (N - 1) |

$(\boxed{a_0 \mid a_1 \mid \ldots \mid a_{N-1}})[i] = a_i$

```
tuple a b = (a, b)
fst (a, b) = a
snd (a, b) = b
```

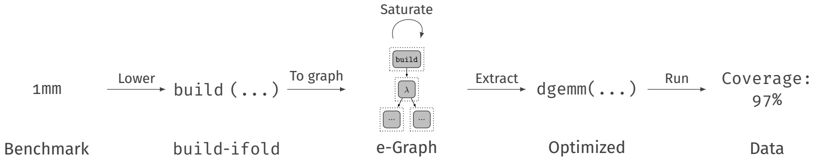
```
ifold 0 init f = init
ifold (N + 1) init f = f N (ifold N init f)
```

Simplification	**Expansion**
`(build N f)[i]` $\rightarrow$ `f i`	`f i` $\rightarrow$ `(build N f)[i]`
`fst (tuple a b)` $\rightarrow$ `a`	`a` $\rightarrow$ `fst (tuple a b)`
`snd (tuple a b)` $\rightarrow$ `b`	`b` $\rightarrow$ `snd (tuple a b)`
`(`λ`x. e) y` $\rightarrow$ `([y/x]e)`	`e` $\rightarrow$ `(`λ`x. e) y`

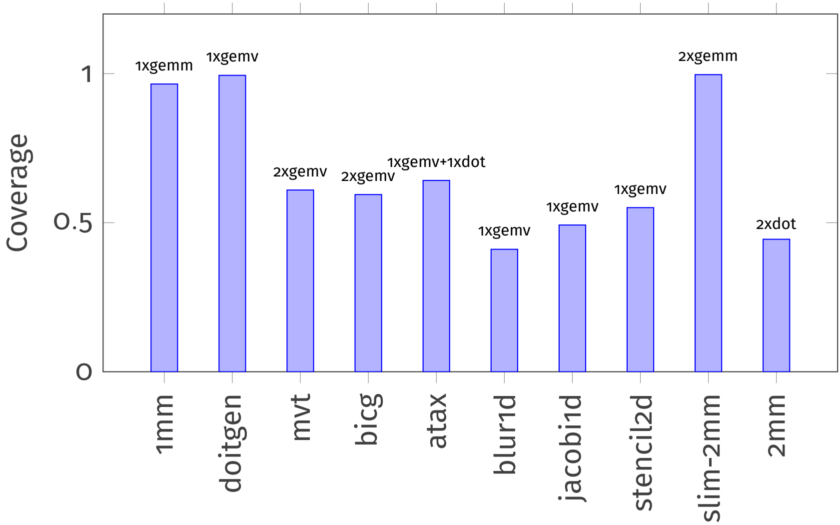
Idiom rewrite rule: gemm

```
build N (λi.
  build K (λj.
    ifold M 0 (λacc. λk.
      alpha * A[i][k] * B[k][j] + acc)
    + beta * C[i][j]))
→ gemm(alpha, A, B, beta, C)
```

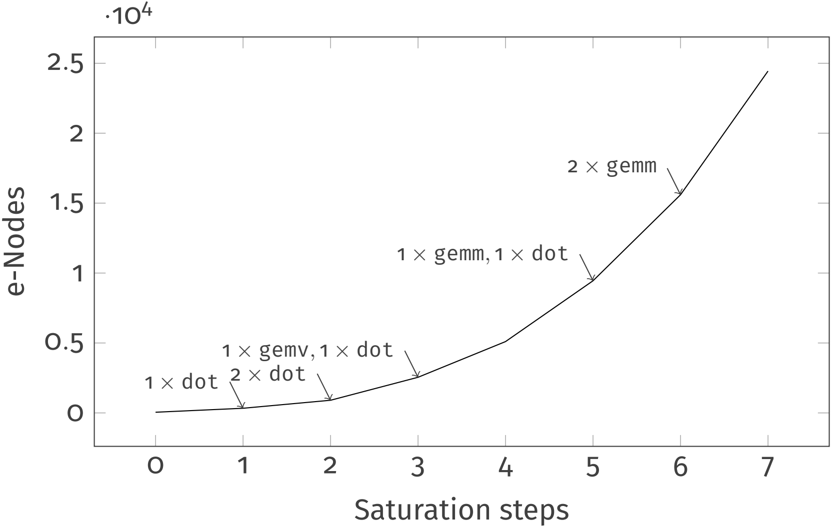
Evaluation



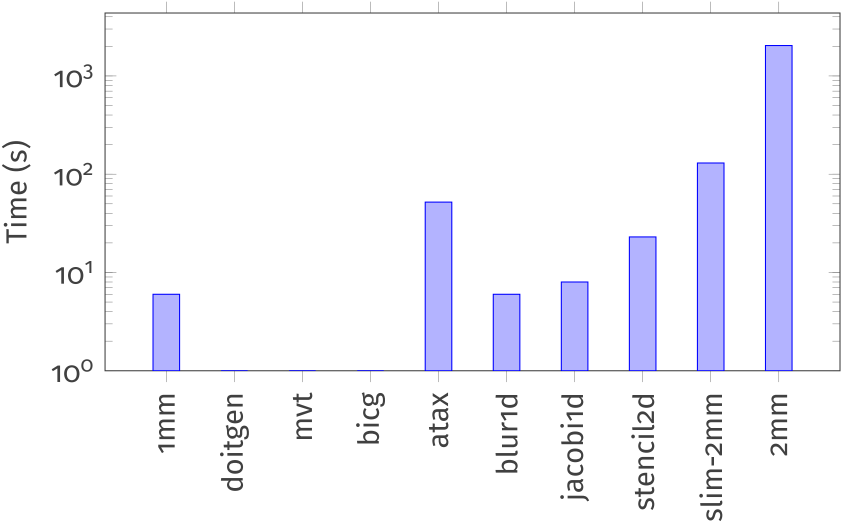
Saturate
1mm
Lower
build (...)
To graph
build
λ
...
...
Extract
dgemm(...)
Run
Coverage:
97%
Benchmark
build-ifold
e-Graph
Optimized
Data


Coverage
1
0.5
0
1xgemm
1xgemv
2xgemv
2xgemv
1xgemv+1xdot
1xgemv
1xgemv
1xgemv
2xgemm
2xdot
1mm
doitgen
mvt
bicg
atax
blur1d
jacobi1d
stencil2d
slim-2mm
2mm

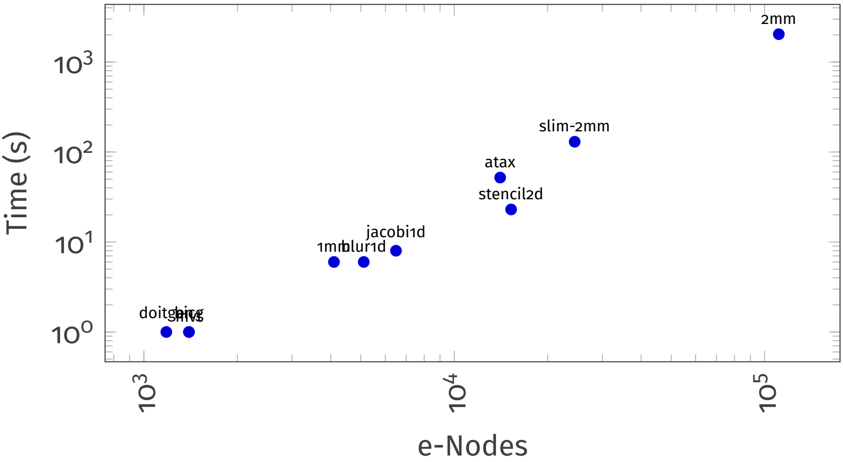
Solutions over time



Saturation speed



Saturation speed



References I

Philip Ginsbach, Toomas Remmelg, Michel Steuwer, Bruno Bodin, Christophe Dubach, and Michael F. P. O'Boyle.
Automatic matching of legacy code to heterogeneous apis: An idiomatic approach.
In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, page 139–153, New York, NY, USA, 2018. Association for Computing Machinery.

Zhitao Lin and Christophe Dubach.
From functional to imperative: Combining destination-passing style and views.
In *Proceedings of the 8th ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming*, ARRAY 2022, page 25–36, New York, NY, USA, 2022. Association for Computing Machinery.

Amir Shaikhha, Andrew Fitzgibbon, Simon Peyton Jones, and Dimitrios Vytiniotis.
Destination-passing style for efficient memory management.
In *Proceedings of the 6th ACM SIGPLAN International Workshop on Functional High-Performance Computing*, FHPC 2017, page 12–23, New York, NY, USA, 2017. Association for Computing Machinery.

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner.
Equality saturation: A new approach to optimization.
In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, page 264–276, New York, NY, USA, 2009. Association for Computing Machinery.

De Bruijn indices

Named	**De Bruijn**
λx. x	λ p_0
λy. y	λ p_0
λx. λy. x	λ λ p_1
λx. λy. y	λ λ p_0
λx. λy. λz. x	λ λ λ p_2

Language semantics rules: `build-ifold`

`(build N f)[i]` $\rightarrow$ `f i`	`f i` $\rightarrow$ `(build N f)[i]`
`fst (tuple a b)` $\rightarrow$ `a`	`a` $\rightarrow$ `fst (tuple a b)`
`snd (tuple a b)` $\rightarrow$ `b`	`b` $\rightarrow$ `snd (tuple a b)`
$(\lambda$ `e) y` $\rightarrow ([y/p_0]\, e){\downarrow}$	`e` $\rightarrow (\lambda\ e{\uparrow})\ y$

Idiom rewrite rule: gemm

```
build N (λ
  build K (λ
    ifold M 0 (λ λ
      alpha * A[p₃][p₁] * B[p₁][p₂] + p₀)
    + beta * C[p₁][p₀]))
→ gemm(alpha↓⁴, A↓⁴, B↓⁴, beta↓², C↓²)
```

Arithmetic rules

```
x + 0 → x    x → 1 * x
x * 1 → x    x → x * 1
1 * x → x    x * y → y * x
x → x + 0
```

Idioms

```
ifold N 0 (λ λ A[p₁] * B[p₁] + p₀)
→ dot(A↓², B↓²)
```

```
build N (λ
  ifold M 0 (λ λ
    alpha * A[p₂][p₁] * B[p₁] + p₀)
  + beta * C[p₀])
→ gemv(alpha↓³, A↓³, B↓³, beta↓, C↓)
```

```
build N (λ
  build K (λ
    ifold M 0 (λ λ
      alpha * A[p₃][p₁] * B[p₁][p₂] + p₀)
    + beta * C[p₁][p₀]))
→ gemm(alpha↓⁴, A↓⁴, B↓⁴, beta↓², C↓²)
```

Benchmark Results

Benchmark	Suite	Steps	Time (s)	e-Nodes
1mm	custom	5	6	4096
doitgen	polybench	3	1	1181
mvt	polybench	3	1	1397
bicg	polybench	3	1	1397
atax	polybench	7	52	14058
blur1d	polybench	3	6	5110
jacobi1d	polybench	3	8	6491
stencil2d	custom	3	23	15251
slim-2mm	custom	7	130	24439
2mm	polybench	9	2038	111123

Benchmark Results

Benchmark	`dot`	`gemv`	`gemm`	Coverage
1mm	0	0	1	97%
doitgen	0	1	0	99%
mvt	0	2	0	61%
bicg	0	2	0	59%
atax	1	1	0	64%
blur1d	0	1	0	41%
jacobi1d	0	1	0	49%
stencil2d	0	1	0	55%
slim-2mm	0	0	2	100%
2mm	2	0	0	44%