Fusing filters with Integer Linear Programming

Amos Robinson (that’s me!)
Gabriele Keller
Ben Lippmeier
I don’t want to write *this*

DO 10 I = 1, SIZE(XS)
    SUM1 = SUM1 + XS(I)
    IF (XS(I) .GT. 0) THEN
        SUM2 = SUM2 + XS(I)
    END IF
10  CONTINUE

DO 20 I = 1, SIZE(XS)
    NOR1(I) = XS(I) / SUM1
    NOR2(I) = XS(I) / SUM2
20  CONTINUE
I’d rather write this

```
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
```
But I also want speed

- Naive compilation: one loop for each combinator
- We need fusion!
Vertical fusion

\[
\begin{align*}
\text{sum1} &= \text{fold} \ (+) \ 0 \ \ xs \quad \text{-- loop 1} \\
\text{nor1} &= \text{map} \ (\div \ \text{sum1}) \ xs \quad \text{-- loop 2} \\
\text{ys} &= \text{filter} \ (> \ 0) \ \ xs \quad \text{-- loop 3} \\
\text{sum2} &= \text{fold} \ (+) \ 0 \ \ ys \quad \text{-- loop 4} \\
\text{nor2} &= \text{map} \ (\div \ \text{sum2}) \ xs \quad \text{-- loop 5}
\end{align*}
\]
Vertical fusion

\[
\begin{align*}
\text{sum1} &= \text{fold} \ (+) \ 0 \ \text{xs} \quad \text{-- loop 1} \\
\text{nor1} &= \text{map} \ (\div \ \text{sum1}) \ \text{xs} \quad \text{-- loop 2} \\
\text{sum2} &= \text{filterFold} \\
\quad (> \ 0) \ (+) \ 0 \ \text{xs} \quad \text{-- loop 3} \\
\text{nor2} &= \text{map} \ (\div \ \text{sum2}) \ \text{xs} \quad \text{-- loop 4}
\end{align*}
\]
Horizontal fusion

\[
\text{sum1} = \text{fold} \ (\ + \ ) \ 0 \ \ xs \quad \text{-- loop 1}
\]

\[
\text{nor1} = \text{map} \ (\ / \ \text{sum1}) \ xs \quad \text{-- loop 2}
\]

\[
\text{sum2} = \text{filterFold} \\
\quad (> \ 0) \ (\ + \ ) \ 0 \ \ xs \quad \text{-- loop 3}
\]

\[
\text{nor2} = \text{map} \ (\ / \ \text{sum2}) \ xs \quad \text{-- loop 4}
\]
Horizontal fusion

\[
\text{sum1} = \text{fold} \ (+) \ 0 \ \text{xs} \quad \text{-- loop 1}
\]

\[
(nor1, \ \text{sum2})
\]

\[
= \text{mapFilterFold} \ (\div \ \text{sum1})
\]

\[
(> 0) \ (+) \ 0 \ \text{xs} \quad \text{-- loop 2}
\]

\[
\text{nor2} = \text{map} \ (\div \ \text{sum2}) \ \text{xs} \quad \text{-- loop 3}
\]
sum1 = fold (+) 0 xs -- loop 1
(nor1, sum2) = mapFilterFold (/ sum1)
   (> 0) (+) 0 xs -- loop 2
nor2 = map (/ sum2) xs -- loop 3
Multiple choices

- What if we applied the fusion rules in a different order?
- There are far too many to try all of them, but…
Order matters

\[
\begin{align*}
\text{sum1} &= \text{fold} \ (+) \ 0 \ \text{xs} \quad -- \ \text{loop 1} \\
\text{nor1} &= \text{map} \ (\div \ \text{sum1}) \ \text{xs} \quad -- \ \text{loop 2} \\
\text{ys} &= \text{filter} \ (> \ 0) \ \text{xs} \quad -- \ \text{loop 3} \\
\text{sum2} &= \text{fold} \ (+) \ 0 \ \text{ys} \quad -- \ \text{loop 4} \\
\text{nor2} &= \text{map} \ (\div \ \text{sum2}) \ \text{xs} \quad -- \ \text{loop 5}
\end{align*}
\]
Order matters

\[
\begin{align*}
\text{sum1} &= \text{fold} \ (+) \ 0 \ \ \text{xs} \quad \text{-- loop 1} \\
\text{nor1} &= \text{map} \ (\div \ \text{sum1}) \ \text{xs} \quad \text{-- loop 2} \\
\text{sum2} &= \text{filterFold} \ \ (> \ 0) \ (+) \ 0 \ \ \text{xs} \quad \text{-- loop 3} \\
\text{nor2} &= \text{map} \ (\div \ \text{sum2}) \ \text{xs} \quad \text{-- loop 5}
\end{align*}
\]
Order matters

\[(\text{sum}1, \text{sum}2) = \text{foldFilterFold} \]
\[
\quad (+) 0
\]
\[
\quad (> 0) (+) 0 \quad \text{xs} \quad -- \text{loop 1}
\]
\[
\text{nor}1 = \text{map} \quad (/ \ \text{sum}1) \ \text{xs} \quad -- \text{loop 2}
\]
\[
\text{nor}2 = \text{map} \quad (/ \ \text{sum}2) \ \text{xs} \quad -- \text{loop 3}
\]
Order matters

\[(\text{sum1}, \text{sum2}) = \text{foldFilterFold} \]

\[\begin{align*}
(+ & 0 \\
(> & 0) \quad (+) \quad 0 \quad x & \text{xs} \quad -- \quad \text{loop 1} \\
nor1 & = \text{map} \quad (/ \quad \text{sum1}) \quad x & \text{xs} \quad -- \quad \text{loop 2} \\
nor2 & = \text{map} \quad (/ \quad \text{sum2}) \quad x & \text{xs} \quad -- \quad \text{loop 3}
\end{align*}\]
Order matters

\[(\text{sum1}, \text{sum2}) = \text{foldFilterFold}\]

\[
(+) 0
\]

\[
(> 0) (+) 0 \quad \text{xs} \quad -- \text{loop 1}
\]

\[(\text{nor1}, \text{nor2}) = \text{mapMap}\]

\[
(/ \text{sum1}) (/ \text{sum2}) \quad \text{xs} \quad -- \text{loop 2}
\]
Order matters

(sum1, sum2) = foldFilterFold

(+) 0

(> 0) (+) 0 xs -- loop 1

(nor1, nor2) = mapMap

(/ sum1) (/ sum2) xs -- loop 2
Which order?

- Finding the *best* order is the hard part.
- That’s why we use…
Integer Linear Programming!

Minimise $y - x$ \hspace{1cm} Objective

Subject to $0 \leq x \leq 2$ \hspace{1cm} Constraints

$0 \leq y \leq 2$

$x + 2y \geq 3$

Where $x : \mathbb{Z}$ \hspace{1cm} Variables

$y : \mathbb{Z}$
Integer Linear Programming!

Minimise \( y - x \)  

Subject to \( 0 \leq x \leq 2 \)  
\( 0 \leq y \leq 2 \)  
\( x + 2y \geq 3 \)  

Where \( x : \mathbb{Z} = 2 \)  
\( y : \mathbb{Z} = 1 \)
Create a graph
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys   = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
ys   = filter (> 0) xs

nor1 = map (/ sum1) xs

ys   = filter (> 0) xs

sum1 = fold (+) 0 xs

nor2 = map (/ sum2) xs

nor1 = map (/ sum1) xs

sum1 = fold (+) 0 xs

sum2 = fold (+) 0 ys

nor2 = map (/ sum2) xs
sum1 = fold (+) 0 xs
nor1 = map (/ sum1) xs
ys = filter (> 0) xs
sum2 = fold (+) 0 ys
nor2 = map (/ sum2) xs
Different size loops

```plaintext
xs
-sum1
 |xs|   |ys|  sum2
 |xs|   |ys|
|xs|   |xs|
nor1 nor2
```
Different size loops

Diagram:
- xs
  - sum1
    - |xs|
    - nor1
      - |xs|
  - sum2
    - |ys|
    - nor2
      - |xs|
- ys
  - |xs|

Connections: Green arrows indicate interconnections between nodes.
Different size loops

Diagram:
- Nodes: xs, sum1, ys, sum2, nor1, nor2
- Edges:
  - xs to sum1
  - xs to nor1
  - ys to sum2
  - ys to nor2
  - sum1 to sum2
  - nor1 to nor2
  - Dotted lines indicate different size loops.
Filter constraint

Minimise ... 

Subject to ...

\[ f(sum1, ys) \leq f(sum1, sum2) \]
\[ f(sum2, ys) \leq f(sum1, sum2) \]

\[ f(a,b) = 0 \text{ iff } a \text{ and } b \text{ are fused together} \]
Objective function
Objective function

Minimise $100f(sum1, ys) + 1f(sum1, sum2)$

$\quad + 100f(sum1, nor2) + 100f(ys, sum2)$

$\quad + 100f(ys, nor1) + 1f(sum2, nor1)$

$\quad + 100f(nor1, nor2)$
Cyclic clusterings cannot be executed
Non-fusible edge

(xs)

(sum1
  nor1)

(ys
  sum2
  nor2)
Non-fusible edge

\[ o(\text{sum1}) < o(\text{nor1}) \]
Fusible edge
if \( f(ys, \text{sum2}) = 0 \)

then \( o(ys) = o(\text{sum2}) \)

else \( o(ys) < o(\text{sum2}) \)
Fusible edge

\[ 1f(ys, sum2) \leq o(sum2) - o(ys) \leq 100f(ys, sum2) \]
Fusible edge - fused

\[1f(ys, sum2) \leq o(sum2) - o(ys) \leq 100f(ys, sum2)\]

\[0 \leq o(sum2) - o(ys) \leq 0\]

\[o(sum2) = o(ys)\]
Fusible edge - unfused

1 \( f(ys, \text{sum2}) \leq o(\text{sum2}) - o(ys) \leq 100f(ys, \text{sum2}) \)

1 \( \leq o(\text{sum2}) - o(ys) \leq 100 \)

\( o(\text{sum2}) > o(ys) \)
No edge

Diagram:
- Nodes: `xs`, `sum1`, `ys`, `sum2`, `nor1`, `nor2`
- Edges:
  - `xs` to `sum1`
  - `xs` to `ys`
  - `sum1` to `nor1`
  - `ys` to `sum2`
  - `sum2` to `nor2`
No edge

if \( f(sum1, ys) = 0 \)

then \( o(sum1) = o(ys) \)
No edge

-100f(sum1, ys) ≤ o(ys) - o(sum1) ≤ 100f(sum1, ys)
No edge - fused

\[-100f(\text{sum1}, \text{ys}) \leq o(\text{ys}) - o(\text{sum1}) \leq 100f(\text{sum1}, \text{ys})\]

\[0 \leq o(\text{ys}) - o(\text{sum1}) \leq 0\]

\[o(\text{ys}) = o(\text{sum1})\]
No edge - unfused

\[-100f(sum1, ys) \leq o(ys) - o(sum1) \leq 100f(sum1, ys)\]

\[-100 \leq o(ys) - o(sum1) \leq 100\]
All together

Minimise  
\[ 100f(\text{sum1}, \text{ys}) + 1f(\text{sum1}, \text{sum2}) + 100f(\text{sum1}, \text{nor2}) + 100f(\text{ys}, \text{sum2}) + 100f(\text{ys}, \text{nor1}) + 100f(\text{nor1}, \text{nor2}) \]

Subject to
\[ f(\text{sum1}, \text{ys}) \leq f(\text{sum1}, \text{sum2}) \]
\[ f(\text{sum2}, \text{ys}) \leq f(\text{sum1}, \text{sum2}) \]
\[ -100f(\text{sum1}, \text{ys}) \leq o(\text{ys}) - o(\text{sum1}) \leq 100f(\text{sum1}, \text{ys}) \]
\[ -100f(\text{sum1}, \text{sum2}) \leq o(\text{sum2}) - o(\text{sum1}) \leq 100f(\text{sum1}, \text{sum2}) \]
\[ 1f(\text{ys}, \text{sum2}) \leq o(\text{sum2}) - o(\text{ys}) \leq 100f(\text{ys}, \text{sum2}) \]
\[ -100f(\text{nor1}, \text{nor2}) \leq o(\text{nor2}) - o(\text{nor1}) \leq 100f(\text{nor1}, \text{nor2}) \]
\[ o(\text{sum1}) < o(\text{nor1}) \]
\[ o(\text{sum2}) < o(\text{nor2}) \]
Result clustering

\[ f(sum1, ys) = 0 \]
\[ f(ys, sum2) = 0 \]
\[ f(sum1, sum2) = 0 \]
\[ f(sum1, nor2) = 1 \]
\[ f(ys, nor1) = 1 \]
\[ f(sum2, nor1) = 1 \]
\[ f(nor1, nor2) = 0 \]
In conclusion

• Integer linear programming isn’t as scary as it sounds!

• We can fuse small (<10 combinator) programs in adequate time

• But we still need to look into large programs

• And we need to support more combinators
Timing: small programs

- Quickhull, Normalize2, Closest points, Quad tree and other test cases
- GLPK and CPLEX both took < 100ms.
Timing: large program

- Randomly generated with 24 combinators
- GLPK (open source) took > 20min
- COIN/CBC (open source) took 90s
- CPLEX (commercial) took < 1s!
References

• Megiddo 1997: Optimal weighted loop fusion for parallel programs

• Darte 1999: On the complexity of loop fusion

• Lippmeier 2013: Data flow fusion with series expressions in Haskell
Differences from Megiddo

• With *combinators* instead of loops, we have more semantic information about the program.

• Which lets us recognise size-changing operations like filters, and fuse together.
Future work

• Currently only a few combinators: map, map2, filter, fold, gather (bpermute), cross product

• Need to support: length, reverse, append, segmented fold, segmented map, segmented…