

Executable Mathematics

Jan Kuper

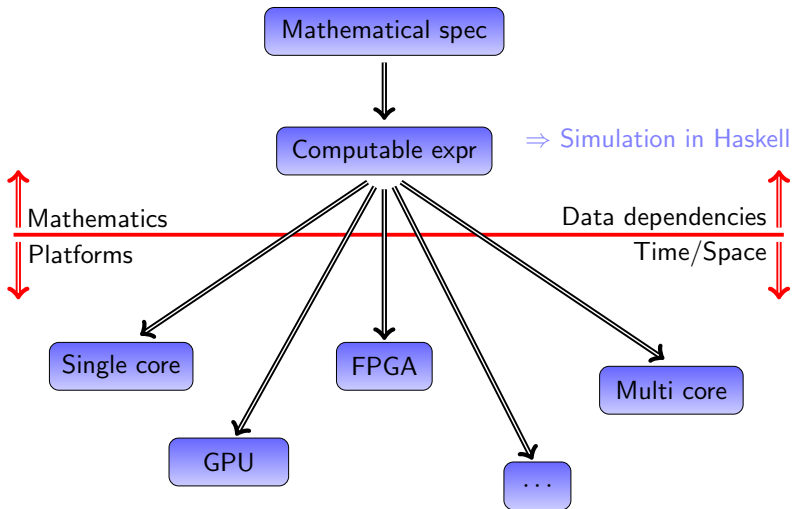
Embedded Systems Group
University of Twente

GPU-NIRICT
Utrecht

December 3, 2015

Key aspects

- ▶ Programming model, one programming environment for various platforms
- ▶ Hardware/Software codesign
- ▶ Stay within mathematical realm as long as possible (data dependencies)
- ▶ Not starting from imperative reference implementation
- ▶ Specification, simulation, implementation in one language (Haskell \approx “mathematics in typewriter font”)
- ▶ ...



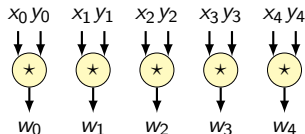
Higher order functions

<i>map</i>	<p>Diagram illustrating the <i>map</i> function: five input elements x_0, x_1, x_2, x_3, x_4 are each processed by a function f to produce output elements z_0, z_1, z_2, z_3, z_4.</p>	$f\ x \Rightarrow z$	$zs = \text{map } f\ xs$
<i>zipWith</i>	<p>Diagram illustrating the <i>zipWith</i> function: pairs of input elements $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ are each processed by a function $*$ to produce output elements w_0, w_1, w_2, w_3, w_4.</p>	$x\ * \ y \Rightarrow z$	$zs = \text{zipWith } (*)\ xs\ ys$
<i>foldl</i>	<p>Diagram illustrating the <i>foldl</i> function: an initial value a is processed sequentially with input elements x_0, x_1, x_2, x_3, x_4 using a function $*$ to produce a final result z.</p>	$a\ * \ x \Rightarrow a'$	$w = \text{foldl } (*)\ a\ xs$
<i>scanl</i>	<p>Diagram illustrating the <i>scanl</i> function: an initial value a is processed sequentially with input elements x_0, x_1, x_2, x_3, x_4 using a function $*$ to produce a list of intermediate results $z_0, z_1, z_2, z_3, z_4, z_5$.</p>	$a\ * \ x \Rightarrow a'$ $z = a$	$zs = \text{scanl } (*)\ a\ xs$
<i>mapAccumL</i>	<p>Diagram illustrating the <i>mapAccumL</i> function: an initial value a is processed sequentially with input elements x_0, x_1, x_2, x_3, x_4 using a function f to produce a final result w and a list of intermediate results z_0, z_1, z_2, z_3, z_4.</p>	$f\ a\ x \Rightarrow (a', z)$	$(w, zs) = \text{mapAccumL } f\ a\ xs$

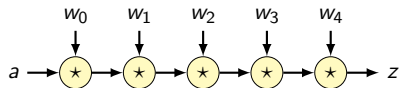
Dot product

$$\vec{x} \bullet \vec{y} = \sum_{i=0}^{n-1} x_i y_i = x_0 y_0 + x_1 y_1 + \dots + x_{n-1} y_{n-1}$$

$ws = \text{zipWith } (*) \text{ } xs \text{ } ys$



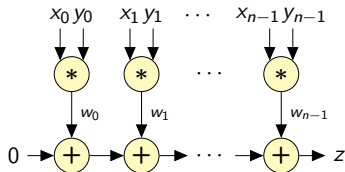
$z = \text{foldl } (+) \text{ } 0 \text{ } ws$



$z = \text{foldl } (+) \text{ } 0 \text{ } ws$

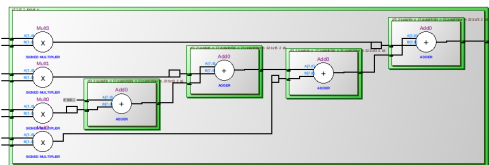
where

$ws = \text{zipWith } (*) \text{ } xs \text{ } ys$



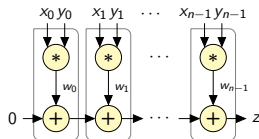
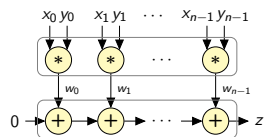
Generated Hardware + Imperative Code

foldl (+) 0 (zipWith () xs ys)*



```
for (i0...) {  
  v0[i0] = xs[i0] * ys[i0];  
};  
Z = 0;  
for (i1...) {  
  Z = Z + v0[i1];  
};
```

Transformations



$\text{foldl } f \ a \ (\text{zipWith } g \ xs \ ys)$

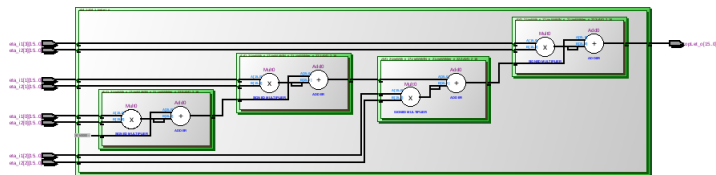
\Rightarrow

$\text{foldl } (f \triangleleft g) \ (zip \ xs \ ys)$

$f \triangleleft g = \lambda a \ z \rightarrow f \ a \ (g \ z)$

Generated Hardware + Imperative Code

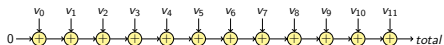
foldl ((+) ◁ (*)) (zip xs ys)



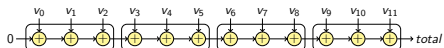
```
Z = 0;  
for (i0...) {  
  Z = Z + x * y;  
};
```


Some other transformations

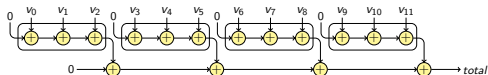
$\text{foldl } (+) 0 \text{ xs}$



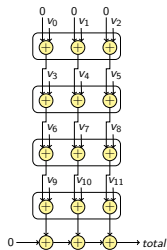
$\text{foldl } (\text{foldl } (+)) 0 \text{ xss}$



$\text{foldl } (+) 0 (\text{map } (\text{foldl } (+) 0) \text{ xss})$



$\text{foldl } (+) 0 (\text{foldl } (\text{zipWith } (+) 0s) \text{ xss})$



Future work

- ▶ Extend code generation towards OpenCL, and also towards specific architectures (Xentium, WaveCore, ...)
- ▶ Develop more transformations
- ▶ Generation of data-dependency graphs, combine with mapping and dataflow analysis for performance
- ▶ Case studies: HPC, image processing, signal processing, adaptive cruise control, particle filtering, solving differential equations
- ▶ ...

Thanks