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 ≈ “mathematics in typewriter font”)
- ...

NIRICT - GPU
Executable Mathematics
Mathematical spec

⇒ Simulation in Haskell

Mathematics

Platforms

Data dependencies

Time/Space

Mathematics

Platforms

Single core

FPGA

Multi core

GPU

···
### Higher order functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Diagram</th>
<th>Equation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map</code></td>
<td><img src="image" alt="Diagram for map" /></td>
<td>$f \ x \Rightarrow z$</td>
<td>$zs = \text{map} \ f \ xs$</td>
</tr>
<tr>
<td><code>zipWith</code></td>
<td><img src="image" alt="Diagram for zipWith" /></td>
<td>$x \ \ast \ y \Rightarrow z$</td>
<td>$zs = \text{zipWith} \ (\ast) \ xs \ ys$</td>
</tr>
<tr>
<td><code>foldl</code></td>
<td><img src="image" alt="Diagram for foldl" /></td>
<td>$a \ \ast \ x \Rightarrow a'$</td>
<td>$w = \text{foldl} \ (\ast) \ a \ xs$</td>
</tr>
<tr>
<td><code>scanl</code></td>
<td><img src="image" alt="Diagram for scanl" /></td>
<td>$a \ \ast \ x \Rightarrow a'$</td>
<td>$zs = \text{scanl} \ (\ast) \ a \ xs$</td>
</tr>
<tr>
<td><code>mapAccumL</code></td>
<td><img src="image" alt="Diagram for mapAccumL" /></td>
<td>$f \ a \ x \Rightarrow (a', z)$</td>
<td>$(w, zs) = \text{mapAccumL} \ f \ a \ xs$</td>
</tr>
</tbody>
</table>
Dot product

\[ \vec{x} \cdot \vec{y} = \sum_{i=0}^{n-1} x_i y_i = x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1} \]

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

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

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

\[ \text{where} \]

\[ ws = \text{zipWith (*) xs ys} \]
Generated Hardware + Imperative Code

\[
\text{foldl } (+) \ 0 \ (\text{zipWith } (*) \ \text{xs} \ \text{ys})
\]

for (i0...) {
    \text{v0}[i0] = \text{xs}[i0] \ast \text{ys}[i0];
};
Z = 0;
for (i1...) {
    Z = Z + \text{v0}[i1];
};
Transformations

\[
\begin{align*}
\text{foldl } f \ a \ (\text{zipWith } g \ xs \ ys) \\
\Rightarrow \\
\text{foldl } (f \triangleleft g) \ (\text{zip } xs \ ys)
\end{align*}
\]

\[
f \triangleleft g = \lambda a \ z \to f \ a \ (g \ z)
\]
foldl \(((+) \odot (\ast))\) (zip xs ys)

\[ Z = 0; \]
\[ \text{for } (i0 \ldots) \{ \]
\[ \quad Z = Z + x \ast y; \]
\[ \} ; \]
Some other transformations

\[
foldl (+) 0 \, xs
\]

\[
foldl (foldl (+)) 0 \, xss
\]

\[
foldl (+) 0 \,(map \,(foldl (+) 0) \, xss)
\]

\[
foldl (+) 0 \,(foldl (zipWith (+) 0) \, xss)
\]
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
- ...

NIRICT - GPU Executable Mathematics
Thanks