Darkroom:
Compiling High-Level Image Processing Code into Hardware Pipelines (Proc. SIGGRAPH’14)

James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley*, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, Pat Hanrahan
Stanford University, *MIT CSAIL

Presented by: Claire Chen, Curran Sinha, Mark Zhao
1. Motivation
What is Darkroom?

▷ Domain Specific Language (DSL)
▷ Language and compiler for image processing
▷ Translates code into line-buffered pipelines
Why do we need DSLs?

▷ Specialized languages (vs. general purpose languages)
  ○ Eg. DSL for data organization: Unix shell scripts
▷ Easier to express domain specific constructs
▷ Data dependencies are more explicit - aids compiler in generating more efficient outputs
DSLs for Image Processing

▷ Cameras are everywhere
  ○ Photography and computer vision
▷ A camera image signal processor pipeline requires many operations
▷ Achieve speed and energy efficiency with specialized hardware that exploits memory access patterns
▷ Line buffers!

Raw input

<table>
<thead>
<tr>
<th>Black Level Correction</th>
<th>Dead Pixel Correction</th>
<th>Crosstalk Correction</th>
<th>White Balance</th>
<th>Demosaic</th>
<th>Color Correction Matrix</th>
<th>Output input</th>
</tr>
</thead>
</table>
Line-buffered pipelines

▷ Single stage: function of pixel position and surrounding stencil

▷ Line buffers between stages of pipeline
DSLs for Image Processing

But...

▷ Tedious to implement in HDL
▷ Synthesis is slow
▷ DSLs, such as Darkroom, ease programming for developers
2. Programming Model
Programming Model

▷ Declarative programming - express computation logic without specifying control flow

▷ Image functions declared with: \texttt{im}(x,y)

▷ Brighten image \( I \): multiply all pixels by 1.1
\[
\text{brighter} = \text{im}(x,y) \times I(x,y) \times 1.1 \quad \text{end}
\]

▷ Convolve image \( I \): access neighboring pixels
\[
C = \text{im}(x,y) \times (I(x-1,y)+I(x,y)+I(x+1,y))/3 \quad \text{end}
\]
3. Darkroom in Practice
1-D Convolution

\[ \text{Out} = \text{im}(x) \cdot k_0 \cdot \text{In}(x) + k_1 \cdot \text{In}(x-1) + k_2 \cdot \text{In}(x-2) \]
Output: Line-buffered pipeline that consumes one pixel, outputs one pixel at every time step.

- Line buffers to store intermediate values
- Combinational nodes for arithmetic
Convolution Line-buffered Pipeline (t=0)

\[ \text{Out}(0) = k_0 \times \text{In}(0) \]
Convolution Line-buffered Pipeline (t=1)

\[ \text{Out}(1) = k_0 \times \text{In}(1) + k_1 \times \text{In}(0) \]
Convolution Line-buffered Pipeline 
(t=2)

Out(2) = k_0 \cdot \text{In}(2) + k_1 \cdot \text{In}(1) + k_2 \cdot \text{In}(0)
Convolution Line-buffered Pipeline (t=x)

\[ \text{Out}(x) = k_0 \times \text{In}(x) + k_1 \times \text{In}(x-1) + k_2 \times \text{In}(x-2) \]

2-D Stencils can be reduced into 1-D by concatenating each row of the input image.
4. Ensuring Causality
What about non-causal stencils? (RAW Hazards)

- Example: 1-D Richardson-Lucy deconvolution
  - \( \text{Rel} = \text{im}(x) \frac{\text{Obs}(x)}{k_0 \cdot \text{Lat}(x-1) + k_1 \cdot \text{Lat}(x) + k_2 \cdot \text{Lat}(x+1)} \) end
  - \( \text{LatN} = \text{im}(x) \frac{\text{Lat}(x) \cdot (k_2 \cdot \text{Rel}(x-1) + k_1 \cdot \text{Rel}(x) + k_0 \cdot \text{Rel}(x+1))}{\text{Rel}(x)} \) end
Solution: Shift Operator

- Shift functions $s$ time steps later $f_s(x) = f(x-s)$
  - $\text{Rel}_1 = \frac{\text{im}(x) \text{Obs}(x)}{k_0 \cdot \text{Lat}(x-2) + k_1 \cdot \text{Lat}(x-1) + k_2 \cdot \text{Lat}(x)}$
  - $\text{LatN}_2 = \frac{\text{im}(x) \cdot \text{Lat}(x-2) \cdot (k_2 \cdot \text{Rel}_1(x-2) + k_1 \cdot \text{Rel}_1(x-1) + k_0 \cdot \text{Rel}_1(x))}$
5. Optimizing Line-Buffers
Optimizing Pipeline

- Shifted pipelines are not always optimal

- Goal: Ensure causality while minimizing line buffer size
Solution: ILP to minimize line buffers

- Let $F$ be the set of image functions (e.g. Out, Rel$_1$)

- Define a use triple $(c,p,d)$ for each value required to calculate $c(x)$
  - $c$ is the consumer function, $p$ is the producer function, and $d$ is the time offset between production and consumption.
  - For example, $\text{Out}(x) = \text{im}(x) \text{In}(x-1) + \text{In}(x) + \text{In}(x+1)$
    - $F = \{\text{Out}, \text{In}\}$
    - $U = \{(\text{Out}, \text{In}, -1), (\text{Out}, \text{In}, 0), (\text{Out}, \text{In}, 1)\}$

- Goal: For each function $f \in F$, solve for the optimal shift $s_f$ to minimize line buffers

- For each use $\in U$, calculate the number of buffers needed as
  - $n_{(c,p,d)} = s_c - s_p - d$

- Constraint: Ensure Causality
  - $n_{(c,p,d)} \geq 0$

- Objective Function: minimize the number of line buffers necessary
  - Define $S$ as the total size of line buffers needed by all producers.
  - Minimize $S$

$$S = \sum_{p \in F} \left( \max_{(c,p,d) \in U} n_{(c,p,d)} \right) * b_p$$
6. Implementation
FPGA/ASIC

▷ Optimized pipeline converted to SystemVerilog
  ○ Line buffers -> Block RAM
  ○ Image function -> Combinational Logic

▷ Block RAM issues
  ○ Discrete sizes
  ○ Limited bandwidth

▷ Darkroom only supports straight pipelines
  ○ Conversion needed for certain pipelines
Converting DAG to Linearized Pipeline

1. Group IR nodes by distance from the input:

   1. Obs
   2. Lat
   3. Rel
   4. LatN

2. Add passthrough nodes whenever an edge cross a stage:

   1. Obs
   2. Lat
   3. Rel
   4. Lat_pass
   5. LatN

3. Merge values in each stage, producing larger pixel widths:

   1. Obs, Lat
   2. Rel, Lat_pass
   3. LatN
CPU Implementation

▷ Parallel approach
▷ Divide output image and assign sections to different cores
▷ One loop -> one clock cycle

```plaintext
for each line y
  for each pixel x in line of S1
    compute S1(x, y)
  for each pixel x in line of S2
    // loading S1 from line buffer
    compute S2(x, y)
  rotate line buffers
```
Results

▷ Typical image processing algorithms

▷ FPGA: ~130 megapixels/second
▷ ASIC: ~1000 megapixels/second
▷ HD video: ~124 megapixels/second
Results

▷ Buffering inefficiencies
▷ Block RAM sizes
▷ Reasonable resource utilization on midrange FPGA
Results

- **CPU speedup compared to reference C implementation:** 6.7x
  - Multithreading and vectorization
- **Similar speedup compared to Halide (image processing language):**
  - 8 hours vs. 1 second to find optimal schedule

<table>
<thead>
<tr>
<th>Run</th>
<th>Runtime (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C++/GCC</td>
<td>Darkroom</td>
</tr>
<tr>
<td>ISP</td>
<td>2.2</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>Halide autotuned</td>
<td>Darkroom</td>
</tr>
<tr>
<td>DEBLUR (float)</td>
<td>0.37</td>
<td>0.35</td>
</tr>
</tbody>
</table>
Thanks!

Any questions?
7. Quiz
Which pipeline corresponds to the following Darkroom function

\[ \text{Out} = \text{im}(x) \cdot k_0 \cdot \text{In}(x) + k_1 \cdot \text{In}(x-1) + k_2 \cdot \text{In}(x-2) \]

A

B

C