HeteroHalide: From Image Processing DSL to Efficient FPGA Acceleration

Jiajie Li\textsuperscript{1,2}, Yuze Chi\textsuperscript{2}, Jason Cong\textsuperscript{2}

Tsinghua University\textsuperscript{1}, University of California, Los Angeles\textsuperscript{2}

li-jj16@mails.tsinghua.edu.cn\textsuperscript{1},\{chiyuze,cong\}@cs.ucla.edu\textsuperscript{2}

*Work mainly done at UCLA during Jiajie's research internship in Summer 2019.*
Background

- **Halide**[SIGGRAPH’12]: a popular image processing DSL
  - Decoupled algorithm & schedule
    - Same algorithm, schedule everywhere (?)

Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, Jonathan Ragan-Kelley et al., SIGGRAPH’12
Motivation

◆ Existing effort synthesizing Halide to FPGA: Halide-HLS[TACO’17]

- **Vendor-specific**
  - When vendor tool behavior changes/switching vendor...
  - Portability 😞

- **Microarchitecture-specific**
  - When better microarchitectures are found...
  - Maintainability 😞
  - Performance 😞
HeteroHalide: Our Approach

- Leverage HeteroCL as an intermediate representation
  - Vendor-neutral
  - Microarchitecture-neutral
  - Semantics-preserving

- Portability
- Maintainability
- Performance

SODA: Stencil with Optimized Dataflow Architecture, Yuze Chi et al., ICCAD’18
PolySA: Polyhedral-Based Systolic Array Auto-Compilation, Jason Cong and Jie Wang, ICCAD’18
Algorithm Transformation

◆ C++-based Halide syntax →

Python-based HeteroCL syntax

Func blur_x("blur_x");
blur_x(x, y) = (input(x, y) + input(x + 1, y) + input(x + 2, y)) / 3;

Func blur_y("blur_y");
blur_y(x, y) = (blur_x(x, y) + blur_x(x, y + 1) + blur_x(x, y + 2)) / 3;

def top(input_hcl):
    with heterocl.Stage("blur_x"):
        with heterocl.for_(y_min, y_max) as y:
            with heterocl.for_(x_min, x_max) as x:
                tensor_blur_x[x, y] = (input_hcl[x, y] + input_hcl[x + 1, y] + input_hcl[x + 2, y]) / 3

    with heterocl.Stage("blur_y"):
        with heterocl.for_(y_min, y_max) as y:
            with heterocl.for_(x_min, x_max) as x:
                tensor_blur_y[x, y] = (tensor_blur_x[x, y] + tensor_blur_x[x, y + 1] + tensor_blur_x[x, y + 2]) / 3

    return tensor_blur_y
## Schedule Transformation

### Immediate transformation

\[
\text{blur}_x(x, y) = \frac{\text{input}(x, y) + \text{input}(x + 1, y) + \text{input}(x + 2, y)}{3}
\]

\[
\text{blur}_x\text{.unroll}(x, 4)
\]

for \(y\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 1\)]:
for \(x\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 4\)]:
\[
\begin{align*}
\text{blur}_x(y, x) &= \ldots \\
\text{blur}_x(y, x + 1) &= \ldots \\
\text{blur}_x(y, x + 2) &= \ldots \\
\text{blur}_x(y, x + 3) &= \ldots
\end{align*}
\]

### Lazy transformation

\[
\text{blur}_x(x, y) = \frac{\text{input}(x, y) + \text{input}(x + 1, y) + \text{input}(x + 2, y)}{3}
\]

\[
\text{blur}_x\text{.lazy_unroll}(x, 4)
\]

for \(y\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 1\)]:
for \(x\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 1\); \text{unrolled}; \text{factor} = 4]:
\[
\text{blur}_x(y, x) = \ldots
\]

### Halide IR

\[
\text{blur}_x\text{.unroll}(x, 4)
\]

for \(y\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 1\)]:
for \(x\) [\(\text{min} = \ldots; \text{extent} = \ldots; \text{stride} = 1\); unrolled; factor = 4]:
\[
\text{blur}_x(y, x) = \ldots
\]

### Merlin C

for (int \(y = \ldots; y < \ldots; y++\))
for (int \(x = \ldots; x < \ldots; x += 4\))
\[
\begin{align*}
\text{blur}_x[y][x] &= \ldots \\
\text{blur}_x[y][x+1] &= \ldots \\
\text{blur}_x[y][x+2] &= \ldots \\
\text{blur}_x[y][x+3] &= \ldots
\end{align*}
\]

#pragma ACCEL parallel factor = 4 flatten
for (int \(x = \ldots; x < \ldots; x++\))
\[
\text{blur}_x[y][x] = \ldots
\]
Evaluation: Productivity

◆ xfOpenCV
  ▪ An HLS library for image processing

◆ For new applications
  ▪ HeteroHalide is more compact

◆ For existing Halide programs
  ▪ HeteroHalide requires minimal changes

<table>
<thead>
<tr>
<th>Application</th>
<th>Lines of Code (algorithm + schedule)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HeteroHalide</td>
</tr>
<tr>
<td>Harris</td>
<td>26 + 14</td>
</tr>
<tr>
<td>Gaussian</td>
<td>8 + 3</td>
</tr>
<tr>
<td>Dilation</td>
<td>2 + 1</td>
</tr>
<tr>
<td>Erosion</td>
<td>2 + 1</td>
</tr>
<tr>
<td>Median Blur</td>
<td>2 + 1</td>
</tr>
<tr>
<td>Sobel</td>
<td>3 + 2</td>
</tr>
<tr>
<td>Geo. Mean</td>
<td>—</td>
</tr>
</tbody>
</table>

Xilinx xfOpenCV Library: https://github.com/Xilinx/xfopencv
FPGA: Zynq 7020

HeteroHalide scales better by leveraging state-of-the-art microarchitecture
Evaluation: Comparison w/ Original Halide on CPU

- Different platforms × different backends
- Energy efficient & performant on both platforms and all backends

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Data Size &amp; Type</th>
<th>VU9P (AWS F1)</th>
<th>Stratix 10 MX</th>
<th>Pattern (Backend)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Energy Eff.</td>
<td>Speedup</td>
<td>Energy Eff.</td>
</tr>
<tr>
<td>Harris</td>
<td>2448×3264, Uint8</td>
<td>29.11</td>
<td>10.31</td>
<td>12.36</td>
</tr>
<tr>
<td>Blur</td>
<td>648×482, UInt16</td>
<td>10.98</td>
<td>3.89</td>
<td>9.34</td>
</tr>
<tr>
<td>Linear Blur</td>
<td>768×1280×3, Float32</td>
<td>12.65</td>
<td>4.48</td>
<td>10.75</td>
</tr>
<tr>
<td>Stencil Chain</td>
<td>1536×2560, UInt16</td>
<td>4.29</td>
<td>1.52</td>
<td>3.64</td>
</tr>
<tr>
<td>Dilation</td>
<td>6480×4820, UInt16</td>
<td>4.69</td>
<td>1.66</td>
<td>1.99</td>
</tr>
<tr>
<td>Median Blur</td>
<td>6480×4820, UInt16</td>
<td>12.51</td>
<td>4.43</td>
<td>5.30</td>
</tr>
<tr>
<td>GEMM</td>
<td>$1024^3$, Int16</td>
<td>9.97</td>
<td>3.53</td>
<td>—</td>
</tr>
<tr>
<td>K-Means</td>
<td>320×32, k=15, Int32</td>
<td>29.00</td>
<td>10.27</td>
<td>—</td>
</tr>
<tr>
<td>Geo. Mean</td>
<td>—</td>
<td>11.44</td>
<td>4.05</td>
<td>6.02</td>
</tr>
</tbody>
</table>

CPU: dual Xeon 2680v4, 14nm, 2.4GHz, 240W; VU9P on AWS F1, 16nm, 250MHz, 85W; Stratix 10 MX, 14nm, 480MHz, 192W
Not to serve as a fair comparison between the two FPGAs
Conclusion

◆ HeteroHalide

- Enables end-to-end compilation from Halide to FPGA
  - Simplified flow from Halide to accelerators
  - Minimal modifications on existing Halide programs

- Extends the existing Halide schedules
  - Generate efficient code for the backend tools

- Produces efficient accelerators by leveraging HeteroCL
  - 4.82× average speedup over 28 CPU cores
  - 2-4× speedup over existing work
References

◆ Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, Jonathan Ragan-Kelley et al., SIGGRAPH’12

◆ Programming Heterogeneous Systems from an Image Processing DSL, Jing Pu et al., TACO’17

◆ SODA: Stencil with Optimized Dataflow Architecture, Yuze Chi et al., ICCAD’18

◆ PolySA: Polyhedral-Based Systolic Array Auto-Compilation, Jason Cong and Jie Wang, ICCAD’18

Thank you
See you in the poster session!

Acknowledgments

This work is supported by the Intel and NSF joint research programs for Computer Assisted Programming for Heterogeneous Architectures (CAPA), Tsinghua Academic Fund for Undergraduate Overseas Studies, and Beijing National Research Center for Information Science and Technology (BNRist). We thank Prof. Zhiru Zhang (Cornell) and his research group for their help on HeteroCL and Prof. Mark Horowitz (Stanford) and his research group for their help on Halide-HLS. We also thank Amazon for providing AWS F1 credits.