# Hardware Generation from Halide

Jeff Setter and Mark Horowitz

## Introduction

Motivation

- Need to design many image-processing accelerators
- Desire to increase hardware design efficiency for design space exploration

Halide

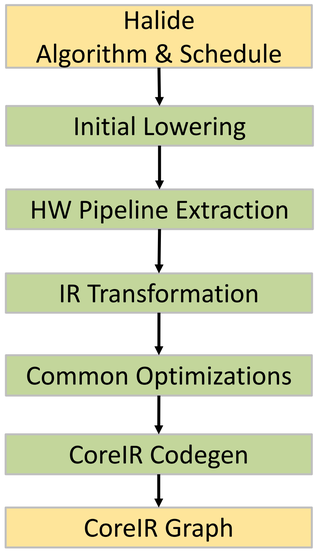
- Image-processing DSL
- Splits *algorithm* from *schedule*
- Explore using different backends: CPU, GPU, FPGA, ASIC, CGRA

CoreIR

- New language to represent hardware
- Uses compiler passes to optimize designs

## Lowering Steps

Steps to transform Halide DSL to CoreIR hardware.




lower: replace RDom with for loops and create loop nests

extract: interpret Halide code and gather loop bounds (window, stride, domain)

transform: use loop bounds to modify code representation and insert linebuffers into dataflow

optimize: unroll loops and eliminate unneeded code (i.e. `a-b+b = a`)

codegen: create instance of each hardware element using CoreIR generators and connect instances into a graph

## Halide to CoreIR Mappings

A table of the Halide operators and associated CoreIR.

| | Halide representation | CoreIR instances |
|---|---|---|
| **Input / Output** | InputParam, Param | def.input, const (set during configuration) |
| | *const* | const |
| **Algorithm functions** | * / + - | mul, ashr, add, sub |
| | != == < <= > >= | neq, eq, {u,s}lt, {u,s}le, {u,s}gt, {u,s}ge |
| | && \|\| ! | and, or, not |
| | & \| ~ ^ >> << | and, or, not, xor, {a,l}shr, shl |
| | select max min | mux, {u,s}max, {u,s}min |
| | absd, * + | absd, mad |
| **Control flow** | for, if | counter, *enable wire* |
| | *var load linebuffer stencil*, *var load array* | *input* => muxn, *const* => muxn |
| **Schedule primitives** | accelerate | Create circuit between input and output |
| | linebuffer = comp+store_at | Create linebuffer (memories and registers) |
| | compute_root | Define order of computation |
| | RDom | Define stencil input size for linebuffer |
| | update | Get update handle for unrolling |
| | unroll | Duplicate algorithm operators by amount. Can be used to remove counters / var load. |
| | tile | Define linebuffer width |
| | reorder | Define how data is read from image |

## Harris Corner Example

A Halide algorithm/schedule to hardware with linebuffers.

```
// algorithm
// sobel filter
padded16(x,y) = padded(x,y);
grad_x(x, y) =  - padded16(x-1,y-1) +   padded16(x+1,y-1)
                -2*padded16(x-1,y)   + 2*padded16(x+1,y)
                 - padded16(x-1,y+1) +   padded16(x+1,y+1));
grad_y(x, y) =   padded16(x-1,y+1) - padded16(x-1,y-1)
               + 2*padded16(x,y+1) - 2*padded16(x,y-1)
               +   padded16(x+1,y+1) - padded16(x+1,y-1));
// product of gradients
grad_xx(x, y) = grad_x(x,y) * grad_x(x,y);
grad_yy(x, y) = grad_y(x,y) * grad_y(x,y);
grad_xy(x, y) = grad_x(x,y) * grad_y(x,y));
// box filter (i.e. windowed sum)
grad_gx(x, y)  += grad_xx(x+box.x, y+box.y);
grad_gy(x, y)  += grad_yy(x+box.x, y+box.y);
grad_gxy(x, y) += grad_xy(x+box.x, y+box.y);
// calculate corner measure (Cim)
Expr lgx  = grad_gx(x, y)  >> 7;
Expr lgy  = grad_gy(x, y)  >> 7;
Expr lgxy = grad_gxy(x, y) >> 7;
Expr det = lgx*lgy - lgxy*lgxy;
Expr trace = lgx + lgy;
cim(x, y) = det - (trace*trace >> 4);
// Perform non-maximal suppression (nms)
Expr is_max = cim(x,y) > cim(x-1,y-1) && cim(x,y) > cim(x,y-1)
           && cim(x,y) > cim(x+1,y-1) && cim(x,y) > cim(x-1,y)
           && cim(x,y) > cim(x+1,y)   && cim(x,y) > cim(x-1,y+1)
           && cim(x,y) > cim(x,y+1)   && cim(x,y) > cim(x+1,y+1);
hw_output(x, y) = select( is_max && (cim(x, y) >= threshold),
                          255, 0);

// schedule
grad_{x,y,xx,yy,xy,gx,gy,gxy}.linebuffer().unroll(x);
grad_{gx,gy,gxy}.update(0).unroll(x).unroll(box.x).unroll(box.y);
```

Input

Sobel filter

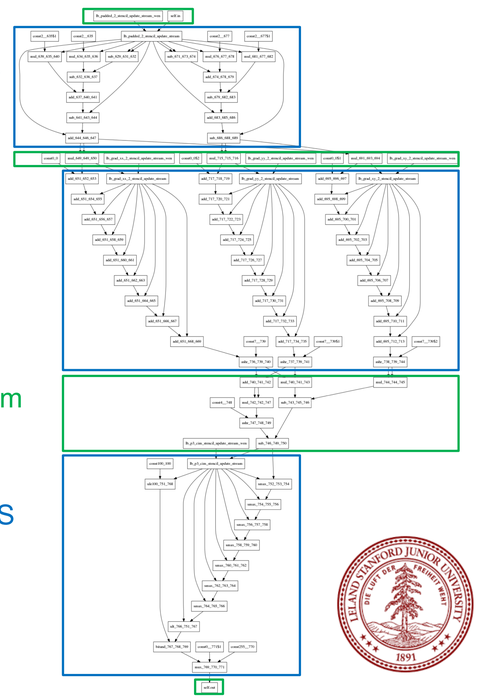
Product of gradients

Box filter

Calculate Cim

Perform NMS

Output



## Application Suite

Testing

- Input image(s) are read by CPU to create reference output image
- Each generated CoreIR design is also tested using the CoreIR interpreter to find bugs early

Current Plan

- Update compiler to top-of-tree Halide and express scheduling more consistently
- Merge generation of double-buffered pipelines for matrix multiplication with current linebuffer generation

| Application | Description | Halide Written | Compiles to CoreIR | CoreIR Tested |
|---|---|---|---|---|
| pointwise | multiply by 2 | ✓ | ✓ | ✓ |
| conv_bw | convolution with 3x3 kernel | ✓ | ✓ | ✓ |
| cascade | two back-to-back conv | ✓ | ✓ | ✓ |
| harris | gradient corner detection | ✓ | ✓ | ✓ |
| fast_corner | FAST corner detection | ✓ | ✓ | ✓ |
| unsharp | mask to sharpen image | ✓ | ✓ | |
| bilateral_filter | blurs while preserving edges | ✓ | | |
| optical_flow | find motion of objects | ✓ | ✓ | |
| stereo | depth map from two views | ✓ | ✓ | |
| camera_pipe | full camera pipeline | ✓ | ✓ | |
| demosaic | demosaic input to rgb | ✓ | ✓ | ✓ |
| demosaic_harris | demosaic then harris | ✓ | ✓ | |
| demosaic_flow | demosaic then optical flow | ✓ | ✓ | |

ISTC Agile