UCLA COMPUTER SCIENCE DEPARTMENT

Center for Domain Specific Computing

HeteroHalide: From Image Processing DSL to Efficient FPGA Acceleration

Jiajie Li[1,2], Yuze Chi[2], Jason Cong[2]

Tsinghua University[1], University of California, Los Angeles[2]

li-jj16@mails.tsinghua.edu.cn[1],{chiyuze,cong}@cs.ucla.edu[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 (?)

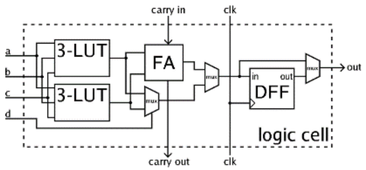
```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

CPU

x64/ARM/PPC/…

GPU

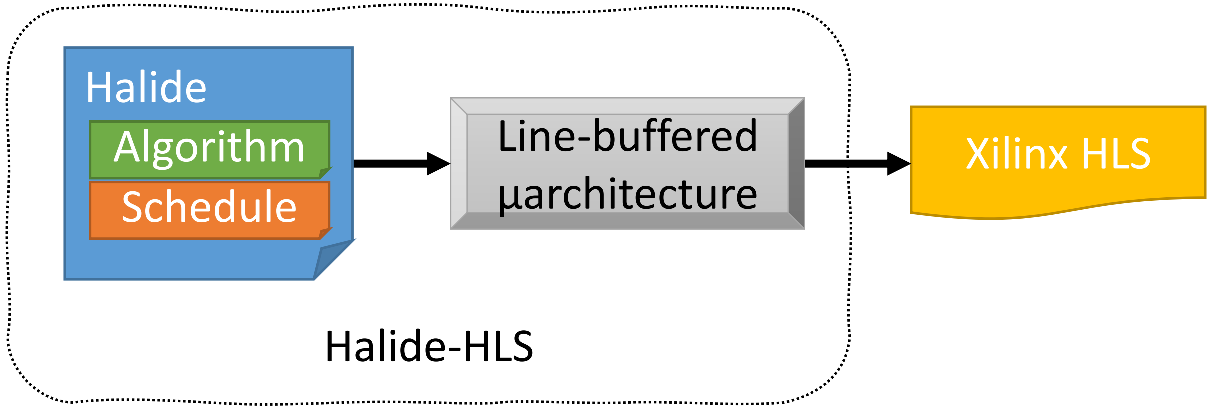
CUDA/OpenCL/…



FPGA?

Motivation

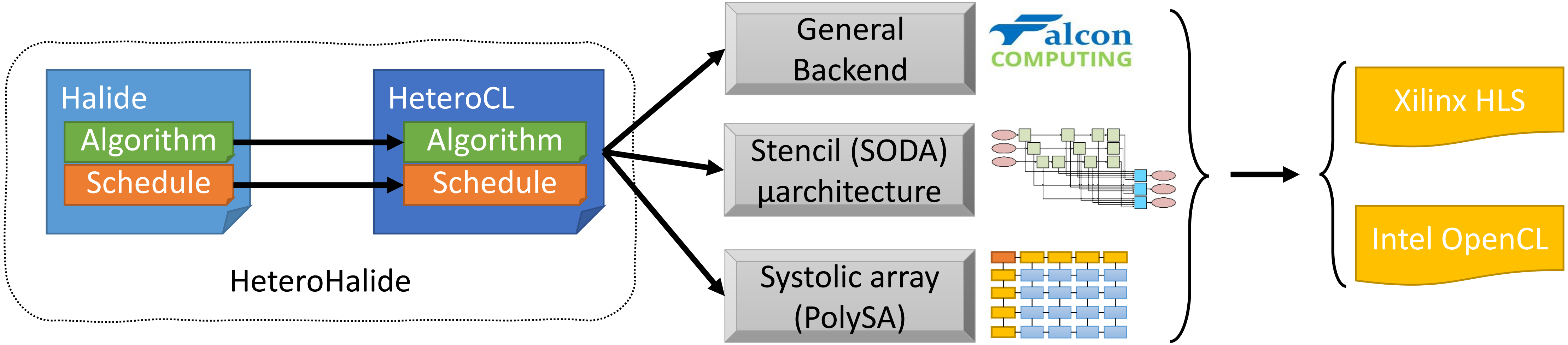
- Existing effort synthesizing Halide to FPGA: Halide-HLS[TACO'17]
  - Vendor-specific
    - When vendor tool behavior changes/switching vendor…
    - Portability
  - Microarchitecture-specific
    - When better microarchitectures are found…
    - Maintainability
    - Performance



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

HeteroHalide: Our Approach

- Leverage HeteroCL as an intermediate representation
  - Vendor-neutral — *Portability*
  - Microarchitecture-neutral — *Maintainability*
  - Semantics-preserving — *Performance*



HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing, Yi-Hsiang Lai et al., FPGA'19
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

Lazy transformation

Halide

```
blur_x(x, y) = (input(x, y) + input (x + 1, y)
               + input(x + 2, y)) / 3

blur_x.unroll(x, 4)
```

```
blur_x(x, y) = (input(x, y) + input (x + 1, y) +
               input(x + 2, y)) / 3

blur_x.lazy_unroll(x, 4)
```

Halide IR

```
for y [min = ...; extent = ...; stride = 1]:
  for x [min = ...; extent = ...; stride = 4]:
    blur_x(y, x)     = ...
    blur_x(y, x + 1) = ...
    blur_x(y, x + 2) = ...
    blur_x(y, x + 3) = ...
```

```
for y [min = ...; extent = ...; stride = 1]:
  for x [min = ...; extent = ...; stride = 1;
        unrolled; factor = 4]:
    blur_x(y, x) = ...
```

Merlin C

```
for (int y = ...; y < ...; y++)
  for (int x = ...; x < ...; x += 4)
    blur_x[y][x] = ...
    blur_x[y][x+1] = ...
    blur_x[y][x+2] = ...
    blur_x[y][x+3] = ...
```

```
for (int y = ...; y < ...; y++)
#pragma ACCEL parallel factor = 4 flatten
  for (int x = ...; x < ...; x++)
    blur_x[y][x] = ...
```

Evaluation: Productivity

- xfOpenCV
  - An HLS library for image processing
- For new applications
  - HeteroHalide is more compact
- For existing Halide programs
  - HeteroHalide requires minimal changes

Application	Lines of Code (algorithm + schedule)	
	HeteroHalide	xfOpenCV
Harris	26 + 14	117 (2.9×)
Gaussian	8 + 3	104 (9.5×)
Dilation	2 + 1	80 (26.7×)
Erosion	2 + 1	79 (26.3×)
Median Blur	2 + 1	81 (27.0×)
Sobel	3 + 2	208 (41.6×)
Geo. Mean	—	(16.7×)

Xilinx xfOpenCV Library: https://github.com/Xilinx/xfopencv

Evaluation: Comparison with Prior Work

Application	Data Size & Type	Throughput (pixel/cycle)		Speedup
		Halide-HLS	HeteroHalide	
Harris	640×640, uint8	2	4	2
Gaussian	640×640, uint8	2	8	4
Unsharp	640×640×3, uint8	1	4	4
Geo. Mean	—	—	—	3.2

- 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

Benchmark	Data Size & Type	VU9P (AWS F1)		Stratix 10 MX		Pattern (Backend)
		Energy Eff.	Speedup	Energy Eff.	Speedup	
Harris	2448×3264, Uint8	29.11	10.31	12.36	9.89	Stencil (SODA)
Blur	648×482, UInt16	10.98	3.89	9.34	7.47	Stencil (SODA)
Linear Blur	768×1280×3, Float32	12.65	4.48	10.75	8.60	Stencil (SODA)
Stencil Chain	1536×2560, UInt16	4.29	1.52	3.64	2.91	Stencil (SODA)
Dilation	6480×4820, UInt16	4.69	1.66	1.99	1.59	Stencil (SODA)
Median Blur	6480×4820, UInt16	12.51	4.43	5.30	4.24	Stencil (SODA)
GEMM	1024^3, Int16	9.97	3.53	—	—	Systolic Array (PolySA)
K-Means	320×32, k=15, Int32	29.00	10.27	—	—	General (Merlin Compiler)
Geo. Mean	—	11.44	4.05	6.02	4.82	—

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
- HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing, Yi-Hsiang Lai et al., FPGA'19

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.