AFFIX: Automatic Acceleration Framework for FPGA Implementation of OpenVX Vision Algorithms

Sajjad Taheri[1], Payman Behnam[2], Eli Bozorgzadeh[1], Alexander Veidenbaum[1], Alexandru Nicolau[1]

[1]Department of Computer Science, UC Irvine

[2]School of Computing, University of Utah

27th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays

Motivation

FPGAs for Computer Vision Acceleration

Computer Vision

- ✓ Important applications in automation, entertainment, healthcare, etc.
- × Complex algorithms and demanding workloads

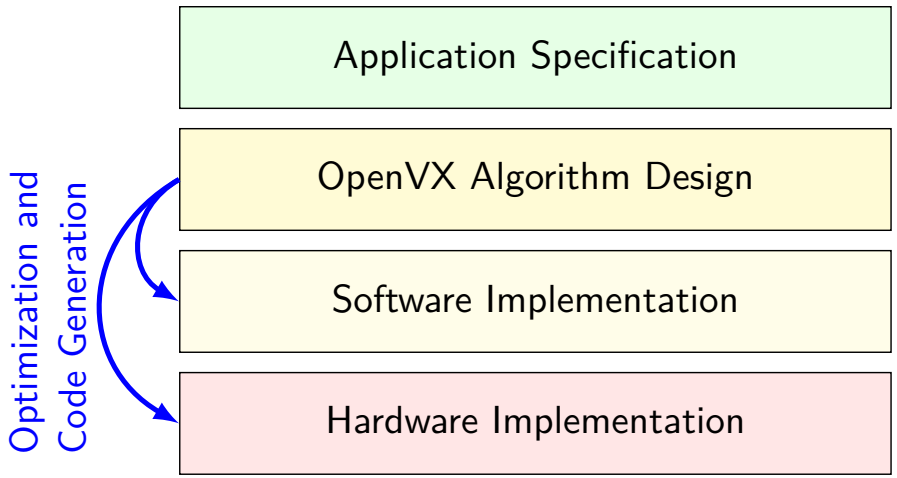
FPGAs offer

- ✓ Inherrent parallelism, high performance, low latency, energy efficiency.

Efficient FPGA acceleration requires hardware design expertise and considerable amount of engineering man-hours.

AFFIX Goal

- FPGA acceleration flow for high level graph-based computer vision algorithms
- "Conventional" computer vision algorihtms based on OpenVX specification.
  - Related work include domain specific languages to represent image processing pipeline: e.g., Halide (2016), PolyMage (2016)



Our Approach

Domain Knowledge

- Design accelerator architecture by considering image processing kernel behaviors
  - Domain Specific Representation and implementataion
- Apply algorithm-specific optimizations on the algorithm graphs

FPGA Design Methodology

- Use High Level Synthesis (OpenCL).
  - Portablity
  - Maintainability
- Apply Hardware specific optimizations

Outline

Overview of OpenVX

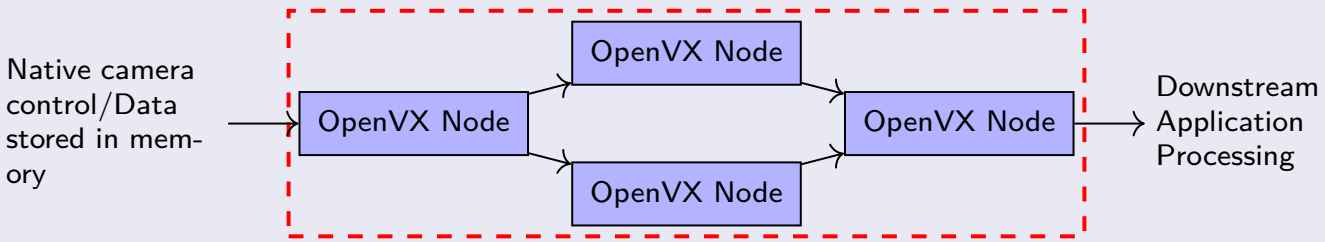
OpenVX

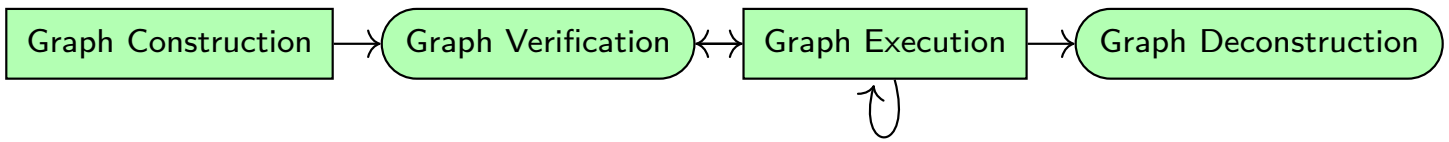
- Open, royalty-free standard for cross platform acceleration of computer vision applications
- Performance and power-optimized computer vision processing
- Graph-based execution model to enable task and data-independent execution

Overview of OpenVX

OpenVX defined objects

- Kernel: Abstract representation of a vision functions, predicates, and delay objects
- Node: An instance of a kernel
- Virtual Image: Represents an image
- Graph: A set of nodes connected in a directed acyclic fashion.





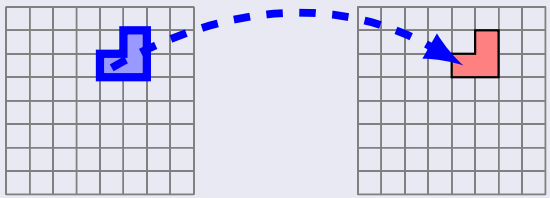
Graph Lifecycle

Customizable library of vision functions

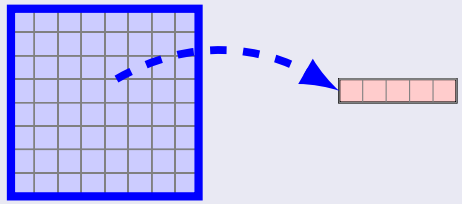
Vision Function Categorization

- Vision functions are categorized based on their data access patterns
- We have implemented streaming kernels for each category in OpenCL

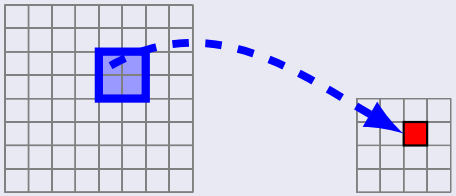
Pixel wise



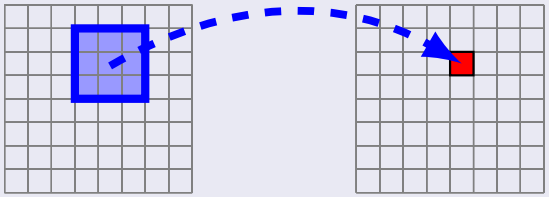
Statistics



Downsample



Stencil



Geometric

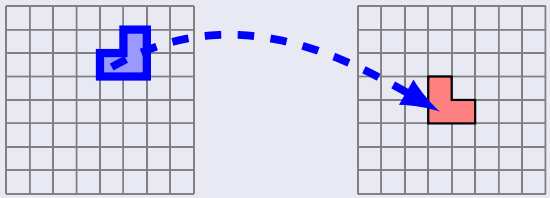
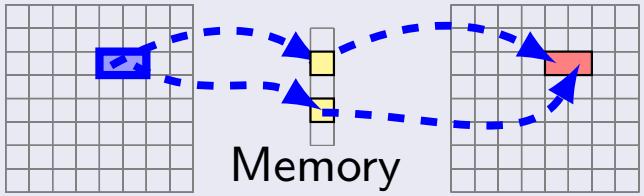


Table Lookup



Vision Function Categorization

Kernels from OpenVX Specification 1.2

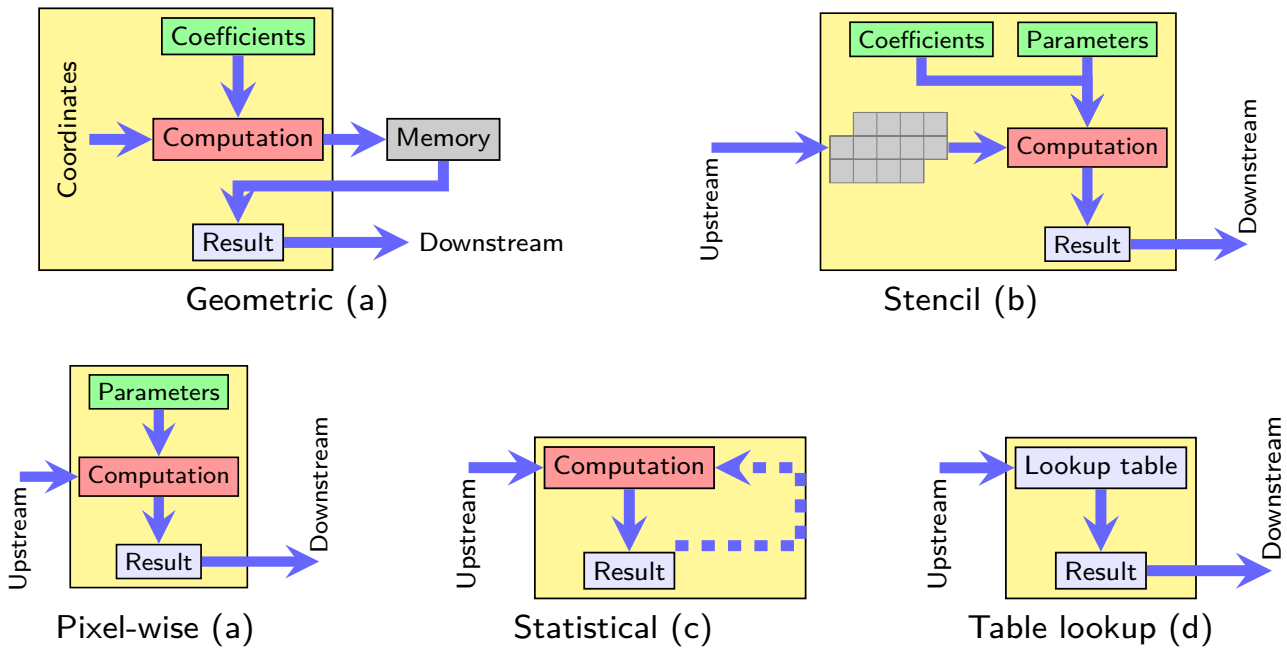
Category	Formal Definition	OpenVX Vision Function
Pixel-wise	$out(x,y) = f(in(x,y))$	Absolute difference, Accumulate, Accumulate squared, Accumulate weighted, Addition/subtraction, Bitwise operations, Channel combine, Channel extract, Color convert, Convert bit depth, Magnitude, Phase, Pixel-wise multiplication, Threshold, Min, Max
Fixed-rate Stencil	$out(x,y) = \sum_{i=-k}^{i=k} \sum_{j=-k}^{j=k} g(in(x+i, y+j))$	Box filter, Sobel, Non-maxima suppression, Custom convolution, Erode, Dilate, Gaussian blur, Nonlinear filter, Integral image, Median filter
Multi-rate Stencil	$out(x,y) = \sum_{i=-k}^{i=k} \sum_{j=-k}^{j=k} g(in(Nx+i, Ny+j))$	Down-sample, Scale image
Statistical	$out = \sum_{i=0}^{i=Width} \sum_{j=0}^{j=Height} g(in(i,j))$	Histogram, Mean, Standard deviation, Min,max location
Geometric	$out(x,y) = in(h(x,y), h'(x,y))$	Remap, Warp affine, warp perspective
Table lookup	$out(x,y) = table[in(x,y)]$	table lookup
Non-primitive	N/A	Equalize histogram, Fast corners, Harris corners, Gaussian image pyramid, Canny edges, LBP, HOG, HoughLinesP

Categorization of Supported OpenVX vision functions

Template-based FPGA Kernel Implementation

Templates based on vision function categorization.

- Easier testing and optimization (5 cases vs 50+ cases)
- Easier to extend OpenVX with user defined functions



General implementation of different kernel categories

Template-based FPGA Kernel Implementation

Kernels can be specialized with

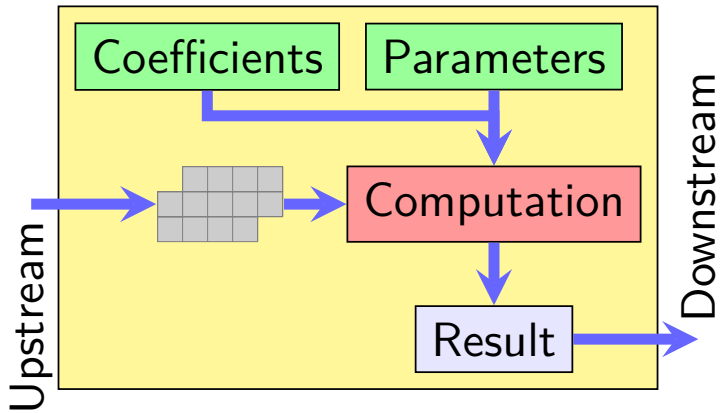
- Specific compute function (similar to function pointers)
- Input and output types (OpenCL standard types)
- SIMD size (1 to 32)
- Sliding Window size
- Local memory configuration (banking, etc.)
- Arithmetic precision (double, float and potentially fixed point)

Channels can be specialized with

- Channel type and width
- Channel depth

OpenCL Programming Interface

- A Domain Specific Language (DSL) on top of OpenCL.
- C-style macros are used to instantiate and specialize generic templates in OpenCL



STENCIL_KERNEL(*name*, *win_size*, *SIMD_size*, *kernel_size*, *in_type*, *out_type*, *func*, *params*, *in_ch*, *out_ch*)

- Vision functions such as Gaussian blur, erode, dilate, and box filter can be implemented with this template

OpenCL Programming Interface (Cont'd)

C-style macros are used to instantiate and specialize channels in OpenCL as well.

- Channels are dynamic FIFOs
- Kernels communicate through channels

Example

```
#define SIMD_SZ 8
CHANNEL(ch_con_col, uchar, SIMD_SZ)
CHANNEL(ch_thresh, uchar, SIMD_SZ)
THRESH(ch_conv_col, SIDM_SZ, thresh_val, ch_thresh)
```

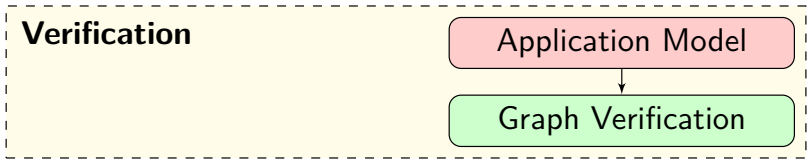
AFFIX framework

AFFIX Framework Flow

Automatic generation of accelerator systems from input algoritms.

1. Checks input algorihtm graph for correctness

Input graph is represented in a textual format



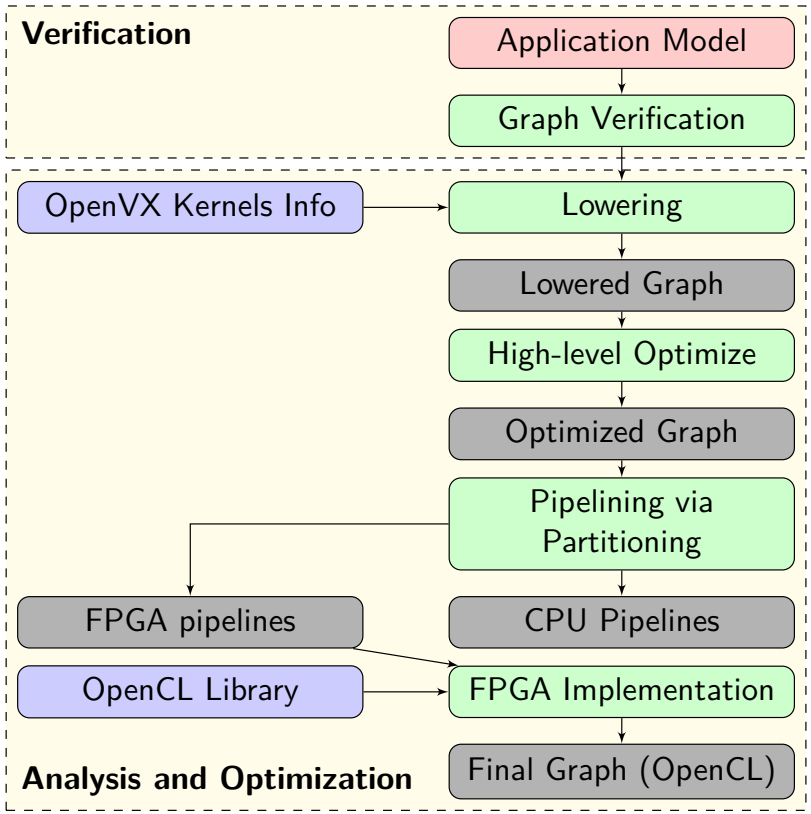
AFFIX Framework Flow

Automatic generation of accelerator systems from input algoritms.

1. Checks input algorihtm graph for correctness

Input graph is represented in a textual format

2. Analysises, paritions, and optimizes the algorithm graph



AFFIX Framework Flow

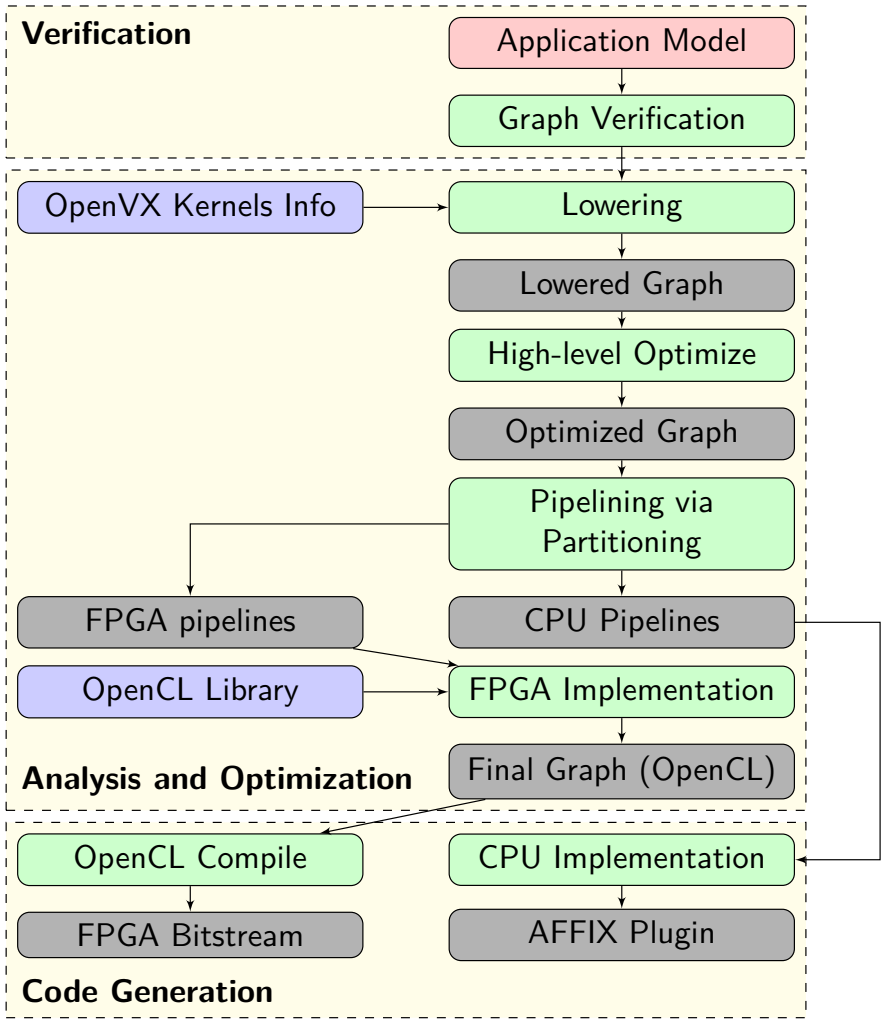
Automatic generation of accelerator systems from input algoritms.

1. Checks input algorihtm graph for correctness

Input graph is represented in a textual format

2. Analysises, paritions, and optimizes the algorithm graph

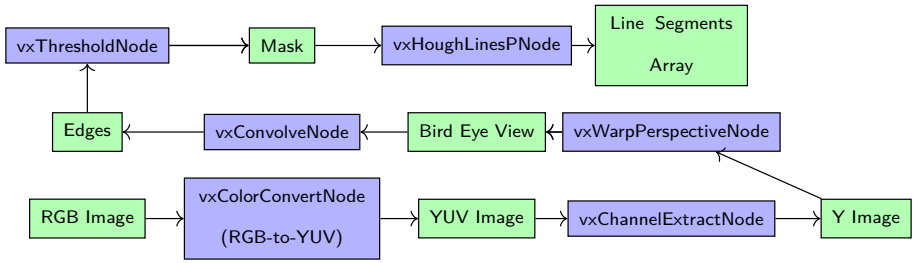
3. Generates code for both FPGA and CPU components



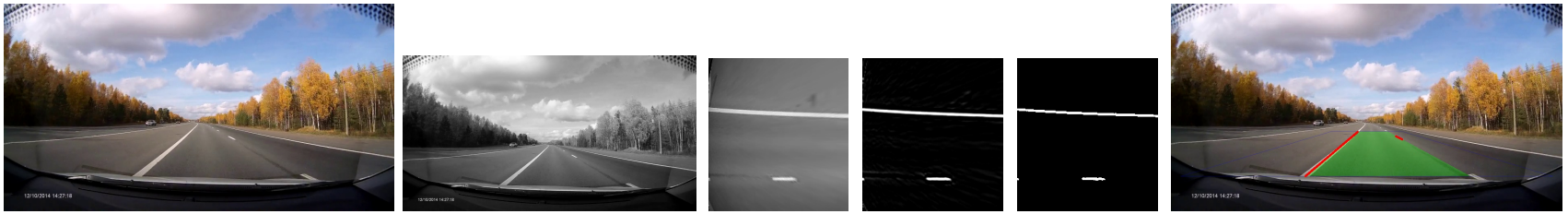
OpenVX Example: Lane Detection Algorithm

```
1  Inputs: Image in_img, Matrix transoform_mat, Matrix filter,
2          Integer thresh_val
3  Output: Array line_segments
4  begin
5    y_img ← convert_to_grayscale(in_img)
6    b_img ← warp_perspective(y_img, transoform_mat)
7    f_img ← filter(b_img, filter)
8    t_img ← threshold(f_img, thresh_val)
9    line_segments ← houghlinsp(t_img)
10   return line_segments
11 end
```

Algorithm



OpenVX Graph

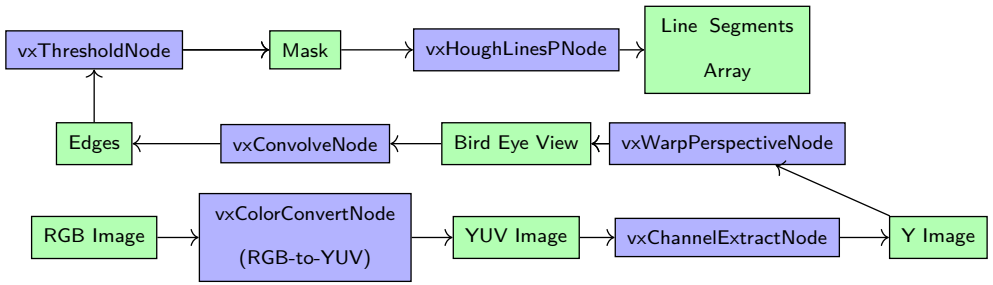


Input image | Grayscale image | *b_image* | *f_image* | *t_image* | Highlighted lanes

Lane Detection Algorithm Demonstration (Input video obtained from software.intel.com)

Simplify and optimize OpenVX graphs

- Decomposition of OpenVX vision functions into simpler primitives
- Removal of nodes that are not connected to an output node
- Separable and symmetric 2D filter implementation
- More steps can be incorporated...

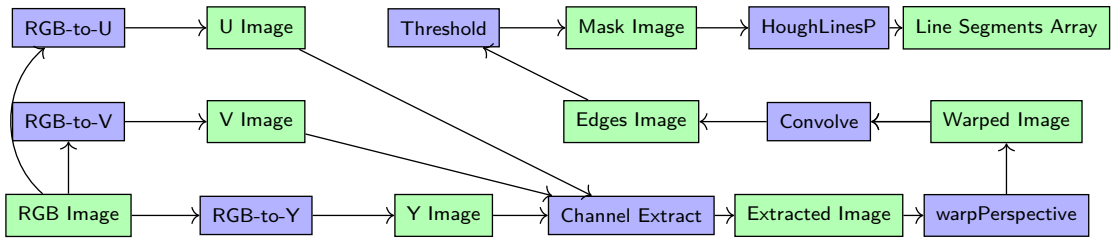


Step 1: Input Lange Detection Algorithm Graph.

High-Level Analysis and Optimization

Simplify and optimize OpenVX graphs

- Decomposition of OpenVX vision functions into simpler primitives
- Removal of nodes that are not connected to an output node
- Separable and symmetric 2D filter implementation
- More steps can be incorporated...

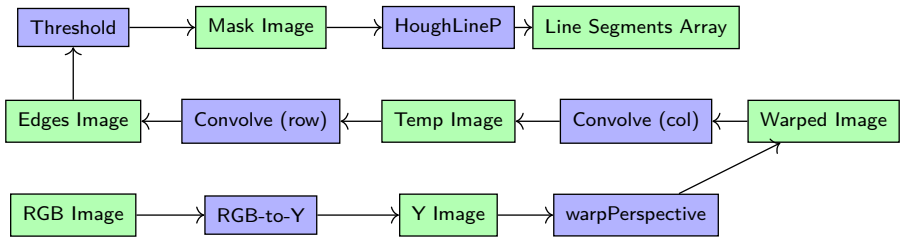


Step 2: Lowered Lane Detection Algorithm. RGB-to-YUV node is replaced with RGB2Y, RGB2U, and RGB2V nodes. Channel extract node drops U and V images.

High-Level Analysis and Optimization

Simplify and optimize OpenVX graphs

- Decomposition of OpenVX vision functions into simpler primitives
- Removal of nodes that are not connected to an output node
- Separable and symmetric 2D filter implementation
- More steps can be incorporated...

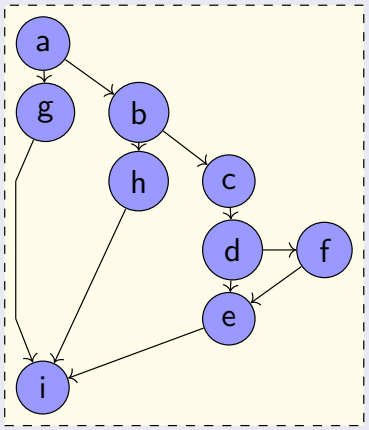


Step 3: Optimized Lane Detection Algorithm

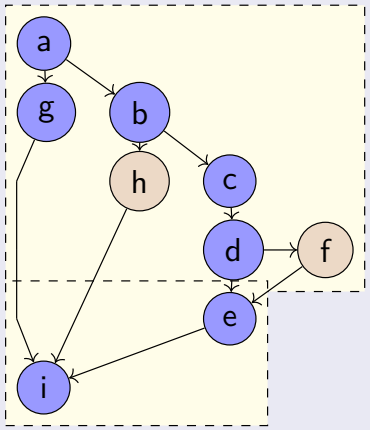
Graph Partitioning

Algorithm graph is partitioned based on vision functions data dependencies

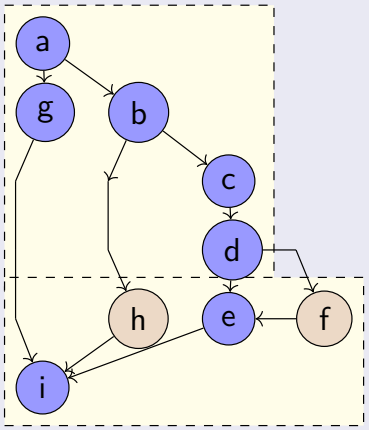
FPGA Graph Partitionin



Graph with only pixel-wise, stencil, and Lookup nodes can be fully pipelined.



Statistical Nodes (h,f) must be last nodes of any pipeline



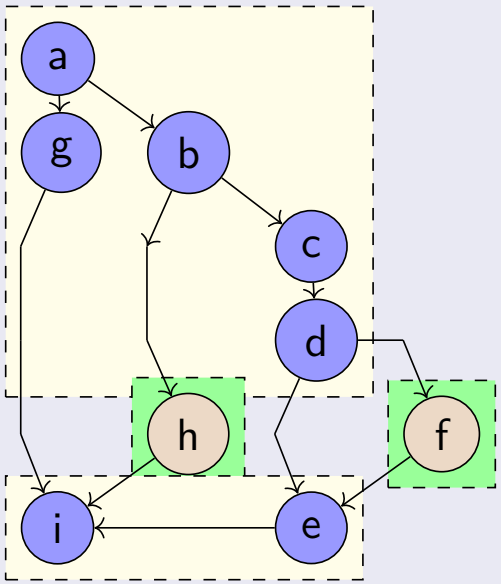
Statistical Nodes (h,f) must be last nodes of any pipeline

Graph Partitioning

Not all vision functions are accelerated on the FPGA

- Kernels with irregular data access
- Kernels with high resource usage or complex implementations

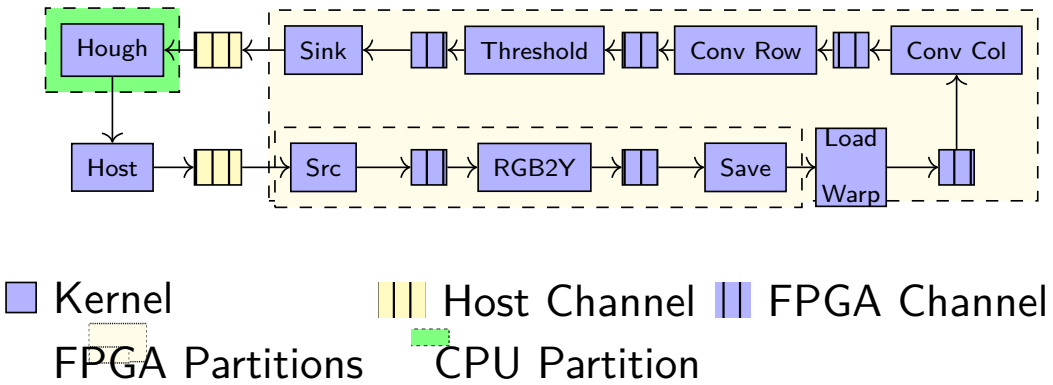
Heterogeneous Graph Partitioning



Graph partitioning in case of CPU nodes: predecessors and successors of CPU nodes(h,f) cannot be mapped to a same pipeline.

Graph Partitioning of Lane Detection Example

- FPGA partitions are implemented in OpenCL
- Partitions are executed in topological order.



Lane Detection Partitioning

FPGA Partitions are described in OpenCL

```
#define SIMD_SZ 8
#define WIN_SZ 240

// Partition 1
CHANNEL(ch_in, uint, SIMD_SZ)
CHANNEL(ch_y, uchar, SIMD_SZ)
SRC(ch_in)
RGBTOY(SIMD_SZ,ch_in,ch_y)
SAVE(ch_y)

// Partition 2
CHANNEL(ch_warped, uchar, SIMD_SZ)
CHANNEL(ch_conv_row, uchar, SIMD_SZ)
CHANNEL(ch_con_col, uchar, SIMD_SZ)
CHANNEL(ch_thresh, uchar, SIMD_SZ)
float[9] conv_col = {...};
float[3] conv_row = {...};
WARP_LOAD(ch_warped, SIMD_SZ)
CONV_ROW(ch_warped, ch_conv_row, 9, conv_row, ...)
CONV_COL(ch_conv1, ch_con_col, 3, conv_col, ...)
THRESH(ch_conv_col, SIDM_SZ, thresh_val, ch_thresh)
SINK(ch_thresh)
```

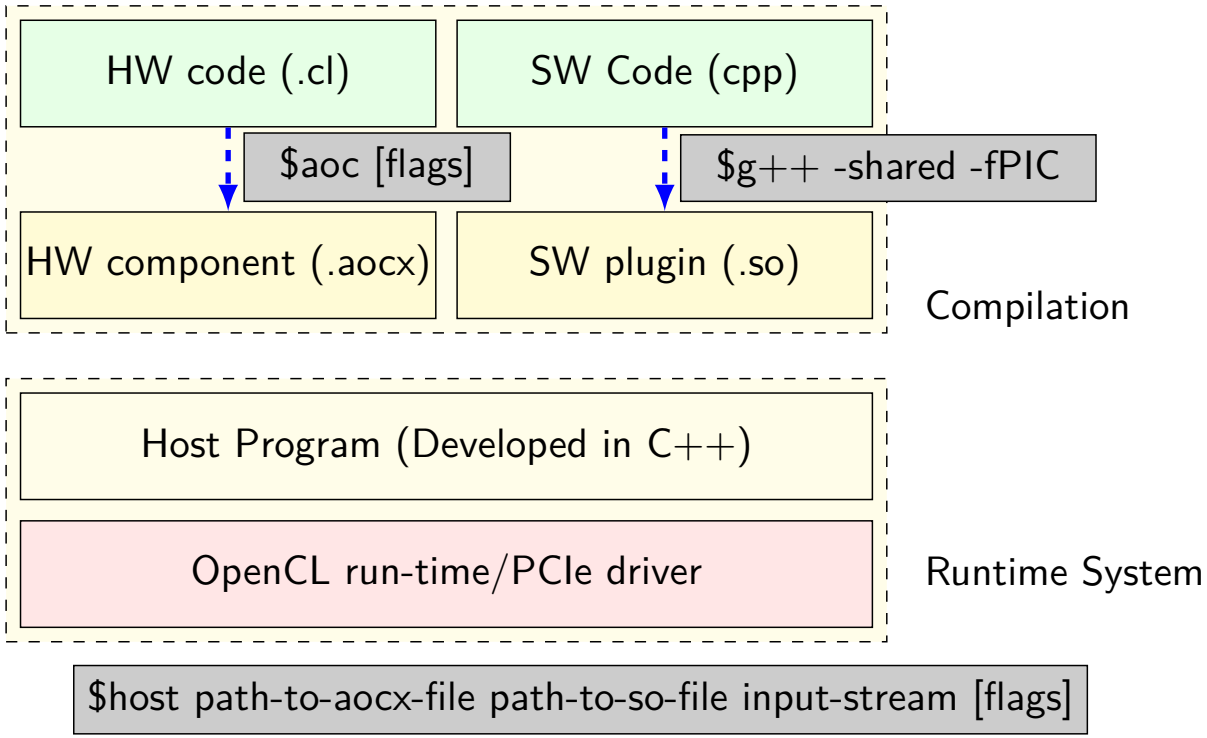
Simplifed OpenCL Implementation of Lane Detection

Software components are described in C++

They implement *Algorithm* class interface:

- Describe both hardware and software pipeline and their ordering
- Implement pre- and post- processing steps for each partition
- Guide OpenCL runtime (Memory allocation, etc)
- USes OpenCV to implement CPU vision functions

Overall System Components



Software Components

Evaluation

Evaluation: Workload and Platform Characterization

- Diverse set of vision algorithms developed
  - Application domain, number of vision functions, and combination of vision function types

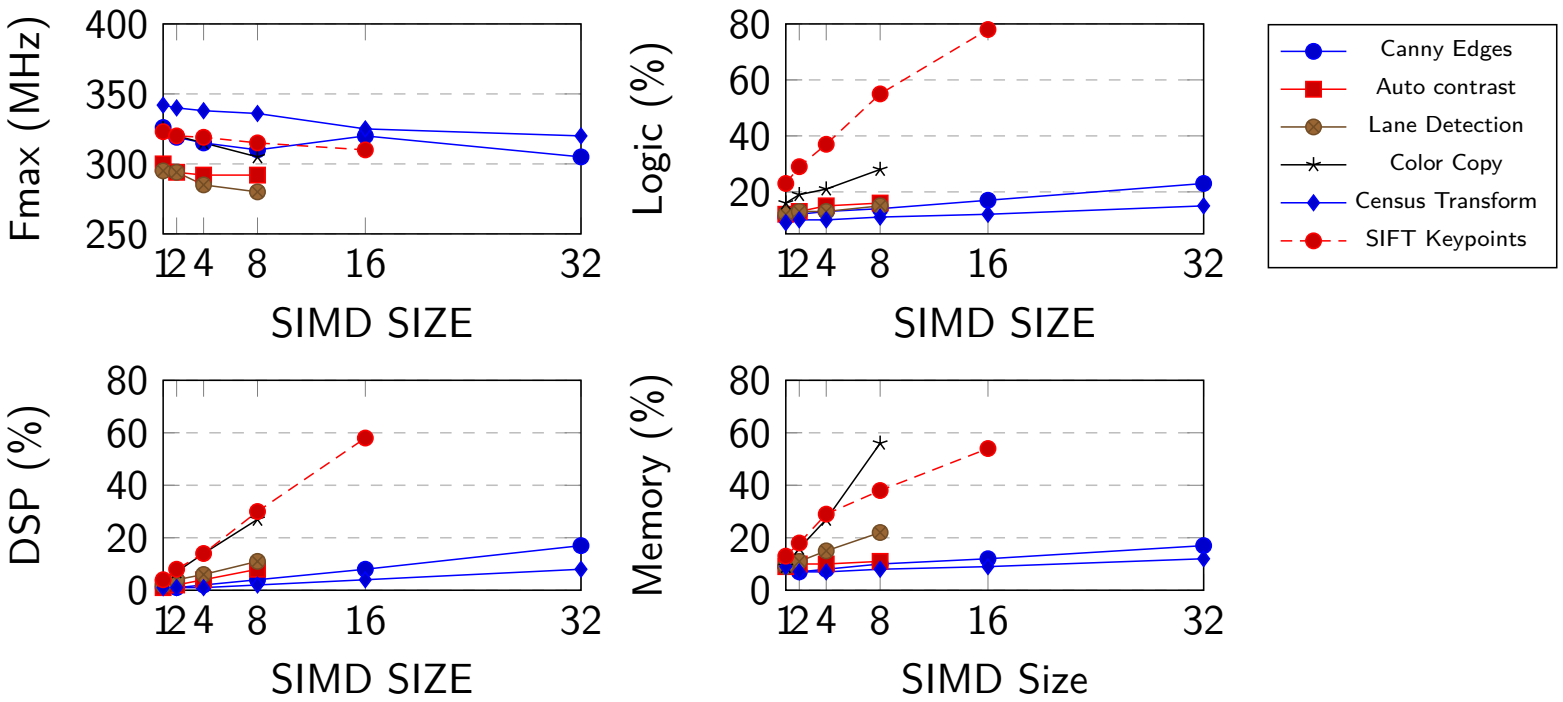
Benchmark	Domain	No VX Fns	No Extension Fns	No CPU Fns	No Geo Fns	No Stats Fns	No Graph Partitions
Canny Edges	Image Processing	1	0	1	0	0	2
Automatic Contrast	Image Processing	6	0	0	0	1	2
Lane Detection	Image Processing	6	0	0	1	0	3
Color Copy	Color Printing	42	4	1	0	0	3
Census Transform	Visual Descriptor	4	1	0	0	0	1
SIFT keypoints	Visual Descriptor	116	2	0	0	0	1

Workload Characterization

Test Platfrom Spec

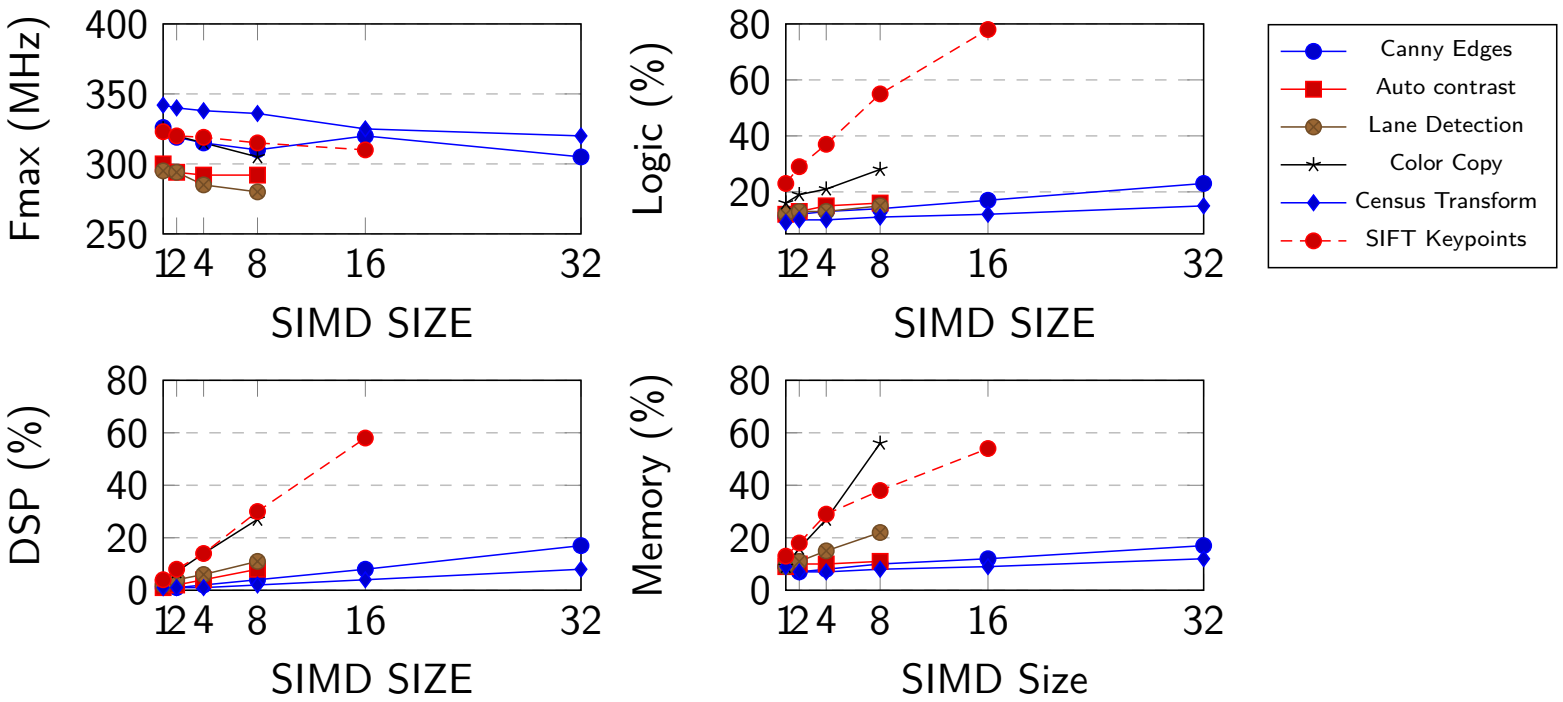
Intel Arria-10GX board connected to host with Intel Core i7 4770 processor via PCIe Gen3x8.

Evaluation: Results



Arria 10 Resource utilization and FMax

Evaluation: Results



Arria 10 Resource utilization and FMax

1. No significant drop on FMax by increasing SIMD size
2. Resources usage grows linearly by increasing SIMD size

Evaluation: Results

Benchmark	Input Size	CPU (AVX+8 Cores)	FPGA SIMD=1	FPGA SIMD=2	FPGA SIMD=4	FPGA SIMD=8	FPGA SIMD=16	FPGA SIMD=32
Canny Edges	3840x2160x1	15 ms	28 ms	15 ms	8 ms	5 ms	4 ms	4 ms
Automatic Contrast	3840x2160x4	21 ms	84 ms	45 ms	22 ms	13 ms	N/A	N/A
Lane Detection	3840x2160x4	46 ms	57 ms	27 ms	14 ms	12 ms	N/A	N/A
Color Copy	3840x2160x4	83 ms	45 ms	32 ms	23 ms	19 ms	N/A	N/A
Census Transform	3840x2160x4	12 ms	27 ms	14 ms	7 ms	4 ms	3 ms	3 ms
SIFT keypoints	3840x2160x1	223 ms	56 ms	27 ms	14 ms	10 ms	10 ms	N/A

Average total execution time of CPU only vs CPU(Intel Core i7)+Arria10 accelerated algorithms with different SIMD sizes

- *SIMD_SIZE* is limited by PCIe width (256 bits)
- FPGA performance increases linearly by improving SIMD until hitting PCIe max bandwidth (Gen3x8)

Conclusion and future direction

Conclusion

Contributions

1. Library of costumizable OpenVX vision functions implementated in OpenCL
   - Support for a wide variety of OpenVX vision elements (functions, data types, control structures)
   - Extendable with new user defined functions
2. Algorithm graph and low level hardware optimization
3. Heterogeneous Graph partitioning and implementation

- Salable and efficient hardware generation
- Ease of development

Future Direction

- Power and energy optimization
- More sophisticated CPU scheduling using multiple processor cores
- Neural Network Integration

Questions?

contact sajjadt@uci.edu

This work is supported by the Intel Corp.