Introduction to OpenVX

Niclas Danielsson, Mikael Pendse
Axis Communications AB
Motivation

- Code portability
- Fast efficient execution
- Neural Network support (recent addition)
API Portability

- Every vendor supporting OpenVX does their own implementation of the standard
- Implementations tailored to be optimally efficient for each platform
- API designed to be hardware independent
- Hardware independent -> Code portable

Same code runs everywhere
Without (m)any changes
The Three Legs of OpenVX

- **Definition**
  - API Specification

- **Guideline**
  - Sample Implementation

- **Validation**
  - Conformance Tests
OpenVX Graphs

- Directed Acyclic Graphs, DAG
- Nodes = computational kernels
- No more spaghetti algorithms
Enabling Efficiency

- Graph Scheduling
- Memory Management
- Kernel Merge
- Data Tiling
Example Graph

OpenVX Graph

OpenVX Nodes

Camera Input

RGB Frame

Color Conversion

YUV Frame

Channel Extract

Gray Frame

Image Pyramid

Pyr_t

Optical Flow Track

Harris Corners

Array of Keypoints

Pts_{t-1}

Tracking Output
Vision Functions

Element-wise Functions
  Add, Subtract, Multiply, AbsDiff,
  And, Or, Xor, Not, Min, Max,
  Magnitude, Phase,
  Threshold, TableLookup, ColorDepth,
  ChannelExtract, ChannelCombine,
  ColorConvert, Copy, AccumulateImage,
  Tensor Add/Subtract/Multiply/LUT/…

Filtering Functions
  Box3x3, Convolve, Dilate3x3, Erode3x3,
  Gaussian3x3, Median3x3, Sobel3x3,
  GaussianPyramid, NonLinearFilter,
  LaplacianPyramid/Reconstruct,
  NonMaxSupression, Bilateral, LBP, HOG, …

Reduction Functions
  Histogram, MeanStdDev, MinMaxLoc

Geometric Functions
  Remap, ScaleImage, WarpAffine,
  WarpPerspective, HalfScaleGaussian

Control Flow
  Scalar Operations, Select

Complex Functions
  CannyEdgeDetector, EqualizeHist,
  FastCorners, HarrisCorners, IntegralImage,
  OpticalFlowPyrLK, HoughLinesP, MatrixMult,
  …
User Kernels

You name it!

- Get custom algorithm into OpenVX runtime
  - Vendor extensions
  - Portable OpenCL acceleration
A Single Node

```c
vx_node VX_API_CALL vxThresholdNode ( vx_graph graph,
                                      vx_image input,
                                      vx_threshold thresh,
                                      vx_image output )
```

Parameters
- [in] graph
- [in] input
- [in] thresh
- [out] output
OpenVX Graph

```c
vx_context context = vxCreateContext();
vx_image input   = vxCreateImage( context, 640, 480, VX_DF_IMAGE_U8 );
vx_image output  = vxCreateImage( context, 640, 480, VX_DF_IMAGE_U8 );

vx_graph graph   = vxCreateGraph( context );
vx_image intermediate = vxCreateVirtualImage( graph, 640, 480, VX_DF_IMAGE_U8 );
vx_node F1        = vxF1Node( graph, input, intermediate );
vx_node F2        = vxF2Node( graph, intermediate, output );
vxVerifyGraph( graph );
while(...) {
    // … write to input image …
    vxProcessGraph( graph );
    // … read from output image …
}
```
OpenVX Components

Context

Data Objects
- Image, Tensor, Pyramid, Array, LUT, Remap, Scalar, Threshold, Distribution, Matrix, Convolution, Delay, ObjectArray

Kernels
- Built-in vision functions, Vendor extensions, User-defined

Graphs

Nodes
- Kernel instances, parameters, attributes

Virtual Data Objects
- Intermediate data without host access, enables several optimizations

Extensions
- NN, Import/Export, ...

Miscellaneous
- Directives, Hints, Logging, Performance Measurements
Opaque Data Object Access

- **Data Memory Ownership**
  - Unless explicitly granted, the OpenVX framework owns the memory
  - The framework has the flexibility to move memory anywhere in the system
Creating Images

vxCreateImage + map + copy + unmap

vxCreateImageFromHandle
The Role of Extensions (to the core spec)

- Compliance flexibility
- A way to test ideas
- A process to add to the core spec
  - ...and sometimes NOT!
  - OpenCL Interop Extension
    - Makes no sense if the platform does not support OpenCL
Extending OpenVX for Deep Learning

• Standards usually evolve slowly...
• Deep Learning is in some sense the opposite to this!

Is it meaningful to standardize Deep Learning at all?
”Flexifying” Standardization

- Core CNN Extension - the tested and stable parts
  - Convolutions
  - Fully connected
  - Activation layers
  - etc.

- Core OpenVX can also be used to construct CNN layers

- Vendor extensions - the changing parts, ”current hype”
  - Can be (semi) quickly changed
  - Doable, but requires vendor to implement the HW support
  - Established extensions can migrate to core spec over time.
Application Developer-level Flexibility

- User nodes - flexibility on the application level
  - True day-to-day flexibility
  - Executed on CPU.
  - Most user nodes are not computationally heavy.
  - But what if your new layer really NEEDS HW acceleration?
    - You can not wait for the standard or vendor extension to catch up?
    - You have a proprietary layer? (a ”secret sauce”)

OpenCL Interoperability Extension
**Alternative - Direct Import**

- OpenVX Kernel Import Extension allows import of “opaque blob” object
  - Only inputs and outputs (and parameters) are set externally
- No need to bother about core vs vendor extensions.
  - Might be simpler for Vendor.
  - Possibility to have a profile with limited OpenVX support (only runtime support)
- Future work
  - Extending import functionality to support user nodes (CPU or OpenCL)

---

**Diagram:**

1. **Training framework**
   - TensorFlow
   - PyTorch
   - MxNet
   - etc...
2. **Exchange format**
3. **CNN “blob”**
CNN + Vision Example
CNN + Vision Example

Input Frame

pre-processing

BG model
BG image

CNN classifier
Object classes

in-paint

Output frame
privacy masked

© Copyright Khronos™ Group 2018 - Page 21
Thank You, Happy Hacking!