The OpenVX Computer Vision and Neural Network Inference Standard for Portable, Efficient Code

Radhakrishna Giduthuri | Editor, OpenVX Khronos Group
radha.giduthuri@amd.com | @RadhaGiduthuri
Khronos Open Standards

- Vision sensor(s)
- Tracking and Positioning
- Geometric scene reconstruction
- Semantic scene understanding (Neural Networks)
- Download 3D augmentation object and scene data
- Generate Low Latency 3D Content and Augmentations
- VR/AR Application
- Interact with sensor, haptic and display devices

Copyright © 2018 Khronos Group
The OpenVX Standard

Wide range of vision and neural network hardware architectures

OpenVX enables high-level Graph-level optimizations!
Can be implemented on almost any hardware or processor!

Portable, Efficient Vision Processing!

Power Efficiency

Computation Flexibility

Vision Engines
Middleware
Applications

CPU
GPU
DSP
Hardware

Software Portability

Vision Processing Graph

Vision Node
CNN Nodes
Vision Node

Copyright © 2018 Khronos Group
OpenVX Evolution

Conformant Implementations

- AMD
- Imagination
- NVIDIA
- Texas Instruments
- VeriSilicon
- cadence
- Intel
- Synopsys
- Socionext

Current Progress

OpenVX 1.2
- Spec released May 2017
- Adopters Program November 2017

Functionality
- Conditional node execution
- Feature detection
- Classification operators

Extensions
- Neural Network Acceleration
- Pipeline, stream, batch processing
- OpenCL interop
- Import vendor kernel, ...

Safety Critical
- OpenVX 1.1 SC for safety-certifiable systems

OpenVX 1.3 or 2.0?
- Roadmap under Discussion
- Enhance NN support
- Profiles/subsets
- NN-only profile
- Merge safety-critical
- More feature detectors
- Stereo vision support
- SLAM support
- More flow control

AMD OpenVX Tools
- Open source, highly optimized for x86 CPU and OpenCL for GPU
- “Graph Optimizer” & Tools
- Modules: 360° video stitching, Neural Network Inference

OpenVX Key Features

• Faster development of efficient and portable vision and neural network applications
  - Graph-level abstraction using C APIs
  - Opaque memory model
  - Developers are protected from hardware complexities
  - No platform-specific performance optimizations needed

• Performance portability to diverse hardware
  - Hardware agility for different use case requirements
  - Application software investment is protected as hardware evolves
OpenVX Graph Level Abstraction

- OpenVX developers express a graph of image or tensor operations (‘Nodes’)
  - Using a C API
- Nodes can be executed on any hardware or processor coded in any language
  - Implementers can optimize under the high-level graph abstraction
- Graphs are the key to run-time power and performance optimizations
  - e.g. Node fusion, pipelining, tiled graph processing for cache efficiency etc.
- CNN nodes can be mixed with traditional vision nodes
OpenVX Efficiency through Graphs

**Graph Scheduling**
- Split the graph execution across the whole system: CPU / GPU / dedicated HW
- Faster execution or lower power consumption

**Memory Management**
- Reuse pre-allocated memory for multiple intermediate data
- Less allocation overhead, more memory for other applications

**Kernel Fusion**
- Replace a sub-graph with a single faster node
- Better memory locality, less kernel launch overhead

**Data Tiling**
- Execute a sub-graph at tile granularity instead of image granularity
- Better use of data cache and local memory
Simple Edge Detector in OpenVX

```c
vx_image input = vxCreateImage(1280, 720, U8);
vx_image output = vxCreateImage(1280, 720, U8);

vx_graph g = vxCreateGraph();

vx_image horiz = vxCreateVirtualImage(g);
vx_image vert = vxCreateVirtualImage(g);
vx_image mag = vxCreateVirtualImage(g);

vxSobel3x3Node(g, input, horiz, vert);
vxMagnitudeNode(g, horiz, vert, mag);
vxThresholdNode(g, mag, THRESH, output);

status = vxVerifyGraph(g);
status = vxProcessGraph(g);
```

1. Declare Input and Output Images
2. Declare Intermediate Images
3. Construct the Graph topology
4. Compile the Graph
5. Execute the Graph
OpenVX Functions

[Computer Vision: 63 functions]
- Image Color Conversion
- Pixel-wise Image Operations
- Image Scaling & Filters
- Pyramids: Gaussian, Laplacian
- Histogram & Statistical
- Tensor Operations
- Classification, Feature Detection, and Tracking
- Control Flow Operations

[Neural Network Extension: 8 functions]
- Convolution
- Deconvolution
- Fully Connected
- Max Pooling & Average Pooling
- Activations
- LRN
- Softmax
- ROI Pooling
OpenVX Extensions

• **Neural Network**: run inference as part of a graph
  - Layers are represented as OpenVX nodes

• **Classification**: detect and recognize objects in an image based on a set of features
  - Import a classifier model trained offline
  - Classify objects based on a set of input features

• **Pipelining**: increase hardware utilization and throughput
  - Provide a way of pipelining, streaming, and batch processing
  - Multiple initiations of a graph with different inputs and outputs

• **OpenCL Interop**: interop between OpenVX and OpenCL application & user-kernels

• **Import/Export**: provide a way of exporting and importing pre-verified graphs & objects

• **Import Kernel**: import pre-compiler vendor binary (e.g., pre-compiled NN as a kernel)
Neural Network Workflow

Datasets

Neural Network Architecture

Neural Net Training Frameworks

Desktop and Cloud Hardware

Trained Models

Vision and Neural Net Inferencing Runtime

Vision/IAI Applications

Diverse Inferencing Acceleration Hardware

CPU  GPU  CPU  GPU  FPGA  DSP  Custom Hardware
OpenVX - Solving Fragmentation

Before OpenVX & NNEF - NN Training and Inferencing Fragmentation

- NN Authoring Framework 1
- NN Authoring Framework 2
- NN Authoring Framework 3

Every Tool Needs an Exporter to Every Accelerator

Inference Engine 1
Inference Engine 2
Inference Engine 3

With OpenVX & NNEF - NN Training and Inferencing Interoperability

- NN Authoring Framework 1
- NN Authoring Framework 2
- NN Authoring Framework 3

Optimization and processing tools

Inference Engine 1
Inference Engine 2
Inference Engine 3
NNEF 1.0 Provisional

- Released as Provisional to get industry feedback before finalization
  - Welcoming comments and feedback on Khronos GitHub
- Initial focus on passing trained frameworks to embedded inference engines
  - Authoring interchange, importing NNEF into tools, is also an emerging use case
- Support deployable range of network topologies
  - Rapid evolution to encompass new network types as they emerge from research

NNEF Working Group Participants
OpenVX Benefits

• Faster development of efficient and portable vision and NN applications
  - Developers are protected from hardware complexities
  - No platform-specific performance optimizations needed
  - OpenVX with NNEF solves fragmentation problem

• Graph description enables significant automatic optimizations
  - Kernel fusion, pipelining, and tiling

• Performance portability to diverse hardware
  - Hardware agility for different use case requirements
  - Application software investment is protected as hardware evolves

• Roadmap driven by community participation
  - Join us and bring your ideas
OpenVX Resources

- OpenVX Resources
  - OpenVX Overview
    - https://www.khronos.org/openvx
  - OpenVX Specifications: current, previous, and extensions
    - https://www.khronos.org/registry/OpenVX
  - OpenVX Resources: implementations, tutorials, reference guides, etc.
    - https://www.khronos.org/openvx/resources

- NNEF Resources
  - NNEF Specification
    - https://www.khronos.org/registry/NNEF

- Embedded Vision Summit Workshop
  - “Khronos Standards for Neural Networks and Embedded Vision”
  - Thursday, May 24, 2018 from 8:00am-5:30pm
    - https://www.khronos.org/events/2018-embedded-vision-summit