Portable performance via the OpenVX™ computer vision library: Case studies

Frank Brill
Cadence Design Systems
May 2019
• Wide range of vision hardware architectures
• OpenVX provides a high-level Graph-based abstraction
  - Enables Graph-level optimizations!
  - Can be implemented on almost any hardware or processor!
• Portable, Efficient Vision Processing!
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
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 cascade detector/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-compiled vendor binary (e.g., pre-compiled NN as a kernel)
OpenVX Case Studies
Cadence OpenVX case study

Application: background subtraction for video security

Graph Speed-up

Higher memory access penalty → greater graph benefits
Cadence OpenVX case study

Application: feature tracking for video security

Graph Speed-up

Fewer, “bigger” nodes → reduced graph benefits
Axis Communications and OpenVX

Uses OpenVX API internally for accelerating algorithm on custom HW blocks

Compute heavy algorithm for reliable motion detection

**Before:**
Hand optimized custom assembler by algorithm developers

**After:**
Algorithm developers “draw” algorithms as graphs.
Driver developers implement the needed graph API

example “algorithm”  $\rightarrow$ (part of) real world algorithm
Axis Communications and Portable Performance

- OpenVX provided a well-defined API enabling parallelized development
- Auto-generates OpenVX C-code from graphical representation of algorithm using internal tool
- Negligible performance loss, significant portability gain
- Same algorithm implementation on custom HW and generic CPU via different OpenVX backends
OpenVX Graph for a Front Camera ADAS Use-case

Image PreProcess \(\rightarrow\) IMG \(\rightarrow\) Image Pyramid \(\rightarrow\) PYR \(\rightarrow\) Feature Plane Compute

\(\frac{1}{2}\) down scale

Lane Detect \(\rightarrow\) ARR \(\rightarrow\) Traffic Sign Detect \(\rightarrow\) ARR

Traffic Light Detect \(\rightarrow\) ARR

Sparse Optical Flow \(\rightarrow\) ARR \(\rightarrow\) Depth Estimation (SFM) \(\rightarrow\) ARR

Feature Plane Compute

Object Classify \(\rightarrow\) ARR

Object Draw

OpenVX Image Data Object
OpenVX Array Data Object
OpenVX Pyramid Data Object

© 2019 Khronos Group
TIOVX and TI Vision SDK Performance Comparison for Front Camera ADAS Use-case

• The table compares TI legacy Vision SDK framework with TI OpenVX framework for an ADAS front-camera application

• The low level algorithms and operating conditions like SoC used, CPU speed, cache settings, OS used are same in both cases

• Vision SDK framework is heavily optimized for TI SoC and therefore goal for TIOVX is to match or improve upon Vision SDK performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Vision SDK</th>
<th>TIO OpenVX</th>
</tr>
</thead>
<tbody>
<tr>
<td>System frame-rate</td>
<td>30 fps</td>
<td>30 fps</td>
</tr>
<tr>
<td>DSP1 Load</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>DSP2 Load</td>
<td>50%</td>
<td>47%</td>
</tr>
<tr>
<td>EVE1 Load</td>
<td>78%</td>
<td>78%</td>
</tr>
<tr>
<td>EVE2 Load</td>
<td>49%</td>
<td>49%</td>
</tr>
<tr>
<td>IPU1-0 Load</td>
<td>21%</td>
<td>23%</td>
</tr>
<tr>
<td>Capture -&gt; results latency</td>
<td>46 msec</td>
<td>46 msec</td>
</tr>
</tbody>
</table>

Results confirm OpenVX matches performance of highly-optimized Vision SDK, while adding flexibility, ease of use, and scalability.
AMD case study: skin tone detect

OpenVX
- Allows graph level processing optimizations
- Allows node fusion for better overall performance
- Allows auto graph level memory optimizations

OpenCV
- Independent function invocations (no graph)
- New OpenCV graph mode has limited functionality
- Not currently performance-portable
AMD case study: Inference with OpenVX

- Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
- Embed NN node(s) in OpenVX graph
  - Add nodes for pre & post processing
- Optimize across entire flow
  - Enables graph optimizations across OpenVX and NN nodes
- Run optimized full-flow inference on target hardware

Diagram:

Preprocessing:
- Resize
- Warp
- Filter
- Denoise
- Pixel-wise ops
- Pyramids
- Optical flow
- User defined ops
- Etc.

Model:

Post-processing:
- Resize
- Draw & label
- Filter
- Denoise
- Pixel-wise Ops
- User defined ops
- Etc.
AMD case study: Inference with OpenVX

Step 1: Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph

a) convert Pre-Trained model to AMD NNIR
b) apply optimization
c) convert AMD NNIR to OpenVX C Code
AMD case study

Inference with OpenVX

Step 2: Add pre & post processing nodes
Step 3: Run Optimized inference on target hardware

<table>
<thead>
<tr>
<th>Model</th>
<th>ms/frame</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>InceptionV4</td>
<td>16.10</td>
<td></td>
</tr>
<tr>
<td>Resnet50</td>
<td>5.52</td>
<td></td>
</tr>
<tr>
<td>VGG16</td>
<td>10.31</td>
<td></td>
</tr>
<tr>
<td>GoogleNet</td>
<td>3.70</td>
<td></td>
</tr>
<tr>
<td>Resnet101</td>
<td>9.70</td>
<td></td>
</tr>
<tr>
<td>Resnet152</td>
<td>14.04</td>
<td></td>
</tr>
<tr>
<td>VGG19</td>
<td>11.29</td>
<td></td>
</tr>
</tbody>
</table>

InceptionV4 — coffee mug
Resnet50 — coffee mug
VGG16 — coffee mug
GoogleNet — coffee mug
Resnet101 — coffee mug
Resnet152 — coffee mug
VGG19 — coffee mug

* FP32 Inference Sample
OpenVX delivers portable performance

- Application code **portable** across a broad range of hardware platforms
- **Performance** comparable to hand-optimized, non-portable code
  - Real, complex applications on real, complex hardware
  - Much lower development effort than hand-optimized
- **Integrate** neural-network and pre/post processing to **optimize** globally
OpenVX Roadmap and Resources
OpenVX Roadmap

• **OpenVX 1.3 expected to be released in June**
  • Enhanced neural-network support
    • NNEF import with conformance tests
  • Feature sets to enable compliance for diverse application spaces
    • Classical computer vision / image processing
    • Neural networks via OpenVX extension nodes or NNEF import
    • Binary (one bit) image processing
  • Merge safety-critical features into single main specification
• **Open-source implementation on Raspberry Pi in development**
OpenVX and NNEF resources

- OpenVX Overview: [https://www.khronos.org/openvx](https://www.khronos.org/openvx)

- OpenVX Specifications: current, previous, and extensions
  - [https://www.khronos.org/registry/OpenVX](https://www.khronos.org/registry/OpenVX)

- OpenVX implementations, tutorials, reference guides, etc.
  - [https://www.khronos.org/openvx/resources](https://www.khronos.org/openvx/resources)

- NNEF Specification: [https://www.khronos.org/registry/NNEF](https://www.khronos.org/registry/NNEF)

- Embedded Vision Summit Workshop
  “Hardware acceleration for Machine Learning and Computer Vision through Khronos open standard APIs”
  Thursday, May 23, 2019 from 9:00am-5:00pm