Introduction to OpenVX

www.khronos.org/registry/vx
(for specification and header files)

Radhakrishna Giduthuri | AMD
OpenVX Graphs

- OpenVX developers express a graph of data operations (‘Nodes’)
  - Nodes can be on any hardware or processor coded in any language
- Graphs can execute almost autonomously
  - Possible to Minimize host interaction during frame-rate graph execution
- Graphs are the key to run-time optimization opportunities...
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 Merge**
- 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 Corner Detector Example

vx_context

Input Image from CAMERA

vx_image

Convert to Grayscale

keypoint array (x, y, …)

vx_array

Harris Detector

vx_node

vx_image (virtual)

vx_graph
An Example of Convolution Neural Network

AlexNet Architecture

OpenVX Graph
Convolution Neural Network Nodes

- Convolution Layer

```c
vx_nn_convolution_params_t params = { ... };
vxConvolutionLayer(graph, x1, w1, b1, &params, sizeof(params), z1);
```

- Activation Layer

```c
vxActivationLayer(graph, z1, VX_CONVOLUTIONAL_NETWORK_ACTIVATION_RELU, 0, 0, y1);
```

- Pooling Layer

```c
vxPoolingLayer(graph, y1, VX_CONVOLUTIONAL_NETWORK_POOLING_MAX, 3, 3, 2, 2,
                VX_CONVOLUTIONAL_NETWORK_DS_SIZE_ROUNDING_FLOOR, x2);
```
OpenVX Application Deployment Model

Development Application

1. Create and verify OpenVX graph
2. Export all the objects that need access during deployment
3. Release all objects

Deployment Application

1. Create context
2. Import objects from binary
3. Graph execution
4. Release all objects
Context

- Context
  - OpenVX world: need to be created first
  - All objects belong to a context

```c
vx_context context = vxCreateContext();
```

See “VX/vx_api.h” for framework API function definitions
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, ICD, …

Miscellaneous
- Directives, Hints, Logging, Performance Measurements
Data objects

- The application gets only references to objects, not the objects
  - References should be released by the application when not needed
  - Ref-counted object is destroyed by OpenVX when not referenced any more

```c
vx_image img = vxCreateImage( context, 640, 480, VX_DF_IMAGE_RGB );
// Use the image
vxReleaseImage( &img );
```

- Object-Oriented Behavior
  - Strongly typed (good for safety-critical applications)
  - OpenVX are really pointers to structs
    - Any object may be down-cast to a `vx_reference`, e.g., for passing to `vxGetStatus()`

- Opaque
  - Access to content explicit and temporary (map/unmap or copy)
    - No permanent pointer to internal data
  - Needed to handle complex memory hierarchies
    - DSP local memory
    - GPU dedicated memory
Error Management

- **Methods return a status**
  - `vx_status` returned: `VX_SUCCESS` when no error

  ```c
  if( vxProcessGraph( graph ) != VX_SUCCESS) { /* Error */ }
  ```

- **Explicit status check**
  - Object creation: use `vxGetStatus` to check the object

  ```c
  vx_context context = vxCreateContext();
  if( vxGetStatus( (vx_reference)context ) != VX_SUCCESS ) { /* Error */ }
  ```

- **More info from the log callback**

  ```c
  void logCallback( vx_context c, vx_reference r, vx_status s,
                   const vx_char string[] )
  { /* Do something */ }
  ...
  vxRegisterLogCallback( context, logCallback, vx_false_e );
  ...
  vxAddLogEntry( reference, VX_INVALID_VALUE, "specified value is out of range" );
  ```

See “VX/vx_types.h” for type definitions and error codes
Data Object Creation

```c
vx_image img = vxCreateImage( ctx, 640, 400, VX_DF_IMAGE_UYVY ); // 13 standard formats

vx_pyramid pyr = vxCreatePyramid( ctx, levels, VX_SCALE_PYRAMID_HALF, 640, 400, VX_DF_IMAGE_U8 );

vx_array arr = vxCreateArray( ctx, VX_TYPE_KEYPOINT, capacity ); // array of vx_keypoint_t[]

vx_lut lut = vxCreateLUT( ctx, VX_TYPE_UINT8, 256 ); // 8-bit look-up table

vx_remap remap = vxCreateRemap( ctx, src_width, src_height, dst_width, dst_height );

vx_matrix mat = vxCreateMatrix( ctx, VX_TYPE_FLOAT32, columns, rows );

vx_distribution dist = vxCreateDistribution( ctx, num_bins, offset, range );

vx_float32 scalar_initial_value = 1.25f;
vx_scalar scalar = vxCreateScalar( ctx, VX_TYPE_FLOAT32, &scalar_initial_value );

vx_delay delay = vxCreateDelay( ctx, (vx_reference)ex, num_slots );

vx_tensor tensor = vxCreateTensor( ctx, 4, dims, VX_TYPE_INT16, 8 );
```
Virtual Data Object Creation

```c
vx_<object> obj = vxCreateVirtual<Object>(graph, ...);
vx_status status = vxGetStatus((vx_reference)obj);
if (status == VX_SUCCESS) {
    // object is good
}
```
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 1.2 Built-in Vision Functions

Kernels

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

Reduction Functions
- Histogram, MeanStdDev, MinMaxLoc

Control Flow
- Scalar Operations, Select

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

Geometric Functions
- Remap, ScaleImage, WarpAffine,
- WarpPerspective, HalfScaleGaussian

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

See “VX/vx_nodes.h” for functions to create kernel instances (nodes) in a graph
Simple Corner Detector Example

vx_context

Input Image from CAMERA

vx_image

Convert to Grayscale

vx_image (virtual)

keypoint array (x, y, ...)

vx_array

Harris Detector

vx_node

vx_graph
Simple Corner Detector Example

- **vx_context**
- **vx_array**
- **keypoint array** (x, y, ...)
- **Harris Detector**
- **vx_node**
- **additional parameters**
- **vx_image (U8) virtual**
- **Channel Extract**
- **vx_node**
- **VX_CHANNEL_Y**
- **vx_image (IYUV) virtual**
- **vx_node**
- **Color Convert**
- **vx_image** (RGB)
- **vx_node**
- **vx_graph**
Vision Functions in a Graph

- **RGB -> YUV**
  
  ```c
  vxColorConvertNode( graph, input_rgb_image, harris_yuv_image );
  ```

  ![Diagram of RGB to YUV conversion]

- **YUV -> Y**
  
  ```c
  vxChannelExtractNode( graph, harris_yuv_image, VX_CHANNEL_Y, harris_gray_image );
  ```

  ![Diagram of YUV to Y conversion]

- **Harris corner**
  - `strength_thresh`: 0.0005f
  - `min_distance`: 5.0f
  - `sensitivity`: 0.04f
  - `gradient_size`: 3
  - `block_size`: 3

  ```c
  vxHarrisCornersNode( graph, harris_gray_image, strength_thresh, min_distance, sensitivity, gradient_size, block_size, keypoint_array_output, NULL );
  ```

  ![Diagram of Harris corner detection]
Image Access: Write to OpenVX image

• Copy using application-controlled address and memory layout
  - vxCopyImagePatch: copy (Read or Write)

```c
vx_imagepatch_addressing_t addr = { /* fill stride_x & stride_y */ };  
vx_rectangle_t rect = { 0u, 0u, width, height };  
vxCopyImagePatch( img, &rect, plane, &addr, my_array,  
                  VX_WRITE_ONLY, VX_MEMORY_TYPE_HOST, VX_NOGAP_X );
```
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

The data memory must NOT be accessed directly by the application without explicit ownership grant.
Execute a Graph in a Loop to Process Input

- Before executing a graph
  - `vxVerifyGraph()` should return VX_SUCCESS (outside the loop)

- Inside the loop -- process each image from input video sequence
  - write pixels from input video into input RGB image
  - Execute Graphs using `vxProcessGraph()`
  - Read the output keypoint data

- After the processing loop
  - Query VX_GRAPH_ATTRIBUTE_PERFORMANCE for performance measurements
  - Release all objects - context should be the last one
Summary

• OpenVX is a low-level programming framework to enable software developers to
  - efficiently access computer vision hardware acceleration
  - with both functional and performance portability

• OpenVX contains:
  - a library of predefined and customizable vision functions
  - a graph-based execution model with task and data-independent execution
  - a set of memory objects that abstract the physical memory

• OpenVX is defined as a C API
  - object-oriented design
  - synchronous and asynchronous execution model
  - well-defined extension model
Opaque Data Object Access

- Data Memory Ownership of Image created using Externally Allocated Memory
  - Unless explicitly granted, the OpenVX framework owns the memory

Application

\[ \text{vxCreateImageFromHandle}(...) \]

\[ \text{vxSwapImageHandle}(...) \text{ or vxMapImagePatch}(...) \]

\[ \text{vxReleaseImage}(...) \text{ [ref\_count=0]} \]

OpenVX Framework

\[ \text{vxSwapImageHandle}(...) \text{ or vxUnmapImagePatch}(...) \]

The original pointer must NOT be accessed directly by the application after image object creation.
Image Access (2/4) : View / edit OpenVX image

• Access limited in time
  - vxMapImagePatch: get access (Read, Write, Read & Write)
  - vxUnmapImagePatch: release the access

```c
vx_map_id  map_id;
void * ptr;
vx_imagepatch_addressing_t  addr;
vx_rectangle_t  rect = { 0u, 0u, width, height };
vxMapImagePatch( img, &rect, plane, &map_id, &addr, &ptr,
  VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, VX_NOGAP_X );

// Access data in ptr
vxUnmapImagePatch( img, map_id );
```
typedef struct _vx_rectangle_t {
    vx_uint32 start_x;/*!< \brief The Start X coordinate. */
    vx_uint32 start_y;/*!< \brief The Start Y coordinate. */
    vx_uint32 end_x;/*!< \brief The End X coordinate. */
    vx_uint32 end_y;/*!< \brief The End Y coordinate. */
} vx_rectangle_t;
typedef struct _vx_imagepatch_addressing_t {
    vx_uint32 dim_x;
    vx_uint32 dim_y;
    vx_int32 stride_x;
    vx_int32 stride_y;
    vx_uint32 scale_x;
    vx_uint32 scale_y;
    vx_uint32 step_x;
    vx_uint32 step_y;
} vx_imagepatch_addressing_t;

Num of (logical) pixels in a row
Num of (logical) pixels in a column
Num of bytes between the beginning of 2 successive pixels
Num of bytes between the beginning of 2 successive lines

Sub-sampling:
1 physical pixel every ‘step’ logical pixel
scale = VX_SCALE_UNITY / step
Array Access (1/2): Write to array

- Copy using application controlled address and memory layout
  - `vxCopyArrayRange`: copy (Read or Write)

```c
vxQueryArray( arr, VX_ARRAY_ATTRIBUTE_NUMITEMS, &num_items, sizeof(num_items) );
vxCopyArrayRange( arr, 0, num_items, sizeof(my_array[0]), &my_array[0], VX_READ_ONLY, VX_MEMORY_TYPE_HOST );
```
Array Access (2/2) : View / edit array

- Access limited in time
  - vxMapArrayRange: get access (Read, Write, Read & Write)
  - vxUnmapArrayRange: release the access

```c
vx_map_id  map_id;
void * ptr;
vxQueryArray( arr, VX_ARRAY_ATTRIBUTE_NUMITEMS, &num_items, sizeof(num_items) );
vxMapArrayRange( arr, 0, num_items, &map_id, &stride, &ptr,
                 VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, 0 );

// Access data in ptr
vxUnmapArrayRange( arr, map_id );
```
OpenVX Object Attributes

• Specific information about OpenVX objects can be queried using `vxQueryType(...)` API

```c
vxQueryImage(image, VX_IMAGE_ATTRIBUTE_WIDTH, &width, sizeof(width)); // meta-data
vxQueryImage(image, VX_IMAGE_ATTRIBUTE_HEIGHT, &height, sizeof(height));
vxQueryImage(image, VX_IMAGE_ATTRIBUTE_FORMAT, &format, sizeof(format));
vxQueryImage(image, VX_IMAGE_ATTRIBUTE_PLANES, &planes, sizeof(planes)); // derived
```

```c
vxQueryArray(array, VX_ARRAY_ATTRIBUTE_ITEMTYPE, &type, sizeof(type));
vxQueryArray(array, VX_ARRAY_ATTRIBUTE_CAPACITY, &capacity, sizeof(capacity));
vxQueryArray(array, VX_ARRAY_ATTRIBUTE_ITEMSIZE, &size, sizeof(size));
vxQueryArray(array, VX_ARRAY_ATTRIBUTE_NUMITEMS, &num_items, sizeof(num_items));
```

```c
vxQueryGraph(graph, VX_GRAPH_ATTRIBUTE_PERFORMANCE, &perf, sizeof(perf)); // changes
```

• Specific information about OpenVX objects can be set

```c
vxSetConvolutionAttribute(conv, VX_CONVOLUTION_ATTRIBUTE_SCALE, &scale, sizeof(scale));
vxSetNodeAttribute(node, VX_NODE_ATTRIBUTE_BORDER_MODE, &mode, sizeof(mode));
```