OpenCL Interoperability with OpenVX

**Introduction**

**OpenVX**
- Open computer vision framework
- Graph-based API => system-level optimization.
- Includes many kernels but not all.
- Supports C based user kernels.

**The problem**
- No accelerated user kernels.
- Two solutions proposed:
  1. OpenVX inserts OpenCL user kernels in the graph.
  2. The framework may use the same command queue for built-in kernels using OpenCL and Interop kernels.

**Interop Kernels – OpenCL Context**
- OpenVX inserts OpenCL user kernels in the graph.
- The framework may use the same command queue for built-in kernels using OpenCL and Interop kernels.
- An OpenCL (context, device) pair is associated with an OpenVX target.

**First: OpenCL-C kernels**
- OpenCL programs are passed to the framework
- Framework compiles kernels
- Users specify inputs and outputs and work item sizes.
- Map is needed between OpenVX objects and OpenCL kernel arguments.
- No OpenCL runtime required.
- Users specify inputs and outputs and work item sizes.

**Second: OpenCL Interop kernels**
- A general solution, no OpenCL kernel restrictions.
- Many OpenCL kernels in one OpenVX kernel.
- User determine how to enqueue them.

**Interop Kernel Example**
- Users can request the OpenCL context, create multiple OpenCL kernels and add them to an OpenVX user-kernel.

```c
// Get OpenCL context associated with an OpenVX target
cl_context clContext = vxGetOpenCLContext(vxContext, target);
// OpenCL standard code for creating kernels
cl_program clProgram = clCreateProgramWithSource(clContext, "k1", ...
cl_kernel clKernel0 = clCreateKernel(clProgram, "k0", ...);
cl_kernel clKernel1 = clCreateKernel(clProgram, "k1", ...);
...

// Create OpenCL interop kernel
vx_kernel vxKernel = vxAddOpenCLInteropKernel(targets, ...
  userFunc, userVal, userInit, userDeinit);
vxAddOpenCLKernelToDeviceKernel(vxKernel, 0, clKernel0);
vxAddOpenCLKernelToDeviceKernel(vxKernel, 1, clKernel1);
vxKernel vxKernel = vxFinalizeKernel(vxKernel);
// Create a graph
vx_node node = vxCreateGenericNode(graph, kernel);
vxSetParameterByIndex(node, 0, inputImage);
vxSetParameterByIndex(node, 1, outputImage);
vxProcessGraph(graph);
```

- OpenCL buffers can be obtained from OpenVX objects.
- **vx_image** can be interpreted as a buffer or image2d.

```c
vx_status userFunc(vx_node node, vx_reference* parameters, 
  cl_kernel* kernels, cl_command_queue queue) 
{ 
  vx_image vxImg = (vx_image) parameters[0]; 
  vx_array vxArr = (vx_array) parameters[1]; 
  cl_mem clBuf = vxGetOpenCLBufferFromImage(node, vxImg); 
  cl_mem clImage = vxGetOpenCLImage2DFromImage(node, vxImg);
  cl_mem clArray = vxGetOpenCLBufferFromArray(node, vxArr);
  // This can be moved to the initializer
  clSetKernelArg(buffer.kernelSize[8], 0, sizeof(cl_mem), &clImage);
  clEnqueueNDRangeKernel(queue, clKernel0, ...);
}
```

By Ben Ashbaugh and Ariel Bernal, Intel® Corporation

---

1. Design is not limited to Intel architectures.
2. Intel currently supports this device kernel extension.
3. Currently in POC stage, OpenVX1.1 sample.