The OpenVX™ Kernel Import Extension

The Khronos® OpenVX Working Group, Editor: Radhakrishna Giduthuri, Intel

Version 1.3 (provisional), Thu, 08 Aug 2019 20:27:50 +0000

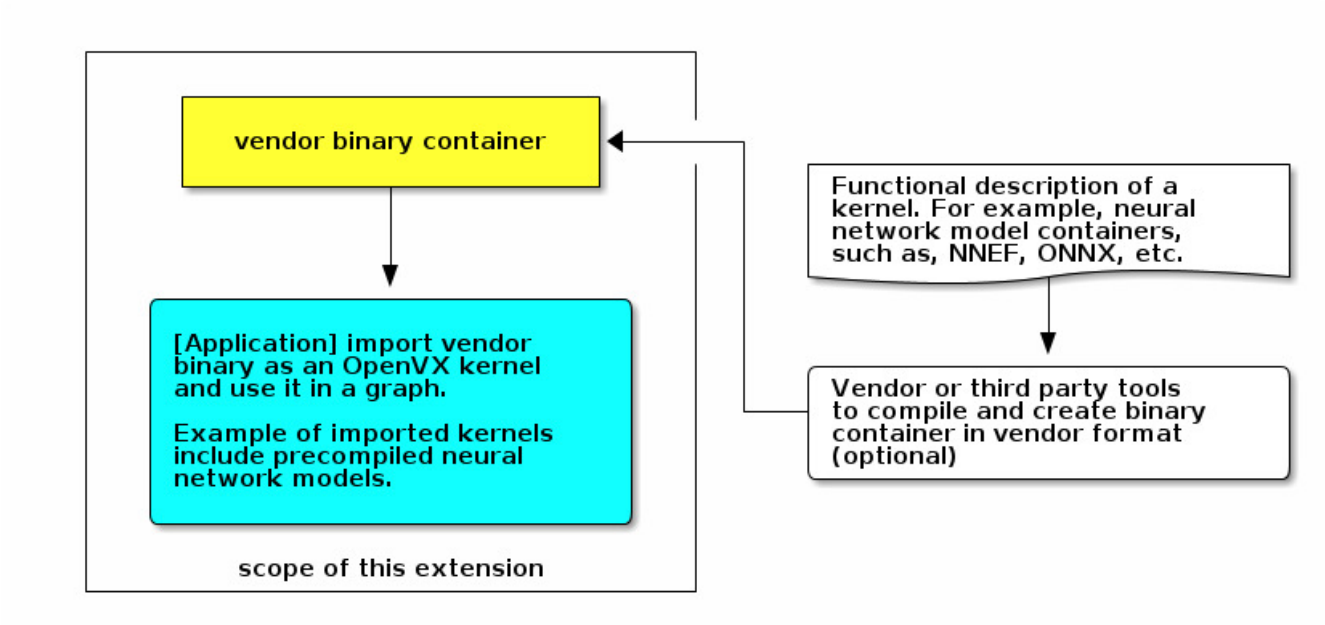
Table of Contents

Chapter 1. Kernel Import Extension to OpenVX 1.3

1.1. Purpose

This document details an extension to OpenVX 1.3, and references some APIs and symbols that may be found in that API, at https://www.khronos.org/registry/OpenVX/specs/1.3/OpenVX_Specification_1_3.html.

Provide a way of importing an OpenVX kernel from a vendor binary specified by URL.



The name of this extension is **vx_khr_import_kernel**.

1.2. Acknowledgements

This specification would not be possible without the contributions from this partial list of the following individuals from the Khronos Working Group and the companies that they represented at the time:

- Radhakrishna Giduthuri - AMD
- Niclas Danielsson, Axis Communications AB
- Frank Brill, Cadence
- Thierry Lepley, Cadence
- Adam Herr, Intel
- Jesse Villarreal, Texas Instruments

1.3. Example: AlexNet graph

In order to use a neural network in OpenVX graph, one may to use the process outlined below:

- Import a pre-trained neural network kernel into the context from a vendor binary specified by URL. Use the `vxImportKernelFromURL` API to import the neural network kernel.
- Create an OpenVX graph that will use the imported neural network kernel.
- Create tensor objects for all neural network parameters (i.e., both input and output)
- Instantiate a neural network node into the graph using the `vxCreateGenericNode` and `vxSetParameterByIndex` APIs.
- Use the `vxVerifyGraph` API to verify and optimize the graph.
- Run the OpenVX graph in a loop

```
#include <VX/vx_khr_import_kernel.h>

void AlexNet( )
{
    vx_uint32 num_params, i;
    vx_tensor tensors[MAX_TENSORS] = { NULL };

    // create OpenVX context
    vx_context context = vxCreateContext();

    // import neural network kernel
    const char * type = "vx_xyz_folder"; // XYZ's kernel binary container
    const char * url = "/assets/alexnet/"; // folder with AlexNet binary
    vx_kernel nn_kernel = vxImportKernelFromURL(context, type, url);

    // create OpenVX graph
    vx_graph graph = vxCreateGraph(context);

    // add neural network instance as a node in the OpenVX graph
    vx_node node = vxCreateGenericNode(graph, nn_kernel);

    // query number of parameters in imported kernel
    vxQueryKernel(nn_kernel, VX_KERNEL_PARAMETERS, num_params, sizeof(vx_uint32));

    // query parameters of kernel to create tensor objects and add to node
    for(i=0; i<num_params; i++)
    {
        vx_type_e type;
        vx_parameter prm = vxGetKernelParameterByIndex(nn_kernel, i);
        vxQueryParameter(prm, VX_PARAMETER_TYPE, &type, sizeof(vx_type_e));

        if(VX_TYPE_TENSOR == type)
        {
            vx_meta_format meta;
            vx_size num_dims;
            vx_size sizes[MAX_SIZES];
            vx_enum tensor_type;
            vx_int8 fixed_point_precision;

            vxQueryParameter(prm, VX_PARAMETER_META_FORMAT, &meta,
                             sizeof(vx_meta_format));

            // Query data needed to create tensor
            vxQueryMetaFormatAttribute(meta, VX_TENSOR_NUMBER_OF_DIMS,
```

```
                                    &num_dims, sizeof(vx_size));
            vxQueryMetaFormatAttribute(meta, VX_TENSOR_DIMS,
                                    &sizes, sizeof(sizes));
            vxQueryMetaFormatAttribute(meta, VX_TENSOR_DATA_TYPE,
                                    &tensor_type, sizeof(vx_enum));
            vxQueryMetaFormatAttribute(meta, VX_TENSOR_FIXED_POINT_PRECISION,
                                    &fixed_point_precision, sizeof(vx_int8));

            tensors[i] = vxCreateTensor(context, num_dims, sizes, tensor_type,
                                     fixed_point_precision);
        }
        vxSetParameterByIndex(node, i, tensors[i]);
    }

    vxReleaseNode(&node);

    // verify graph
    vxVerifyGraph(graph);

    // process graph with one batch at a time
    while( userGetNextJobInput(tensors[0]) == VX_SUCCESS )
    {
        // execute the graph to run AlexNet
        vxProcessGraph(graph);

        // consume the output from AlexNet
        userConsumeOutput(tensors[i-1]);
    }

    vxReleaseGraph(&graph);
    for(i=0; i<num_params; i++)
    {
        vxReleaseTensor(&tensors[i]);
    }
    vxReleaseContext(&context);
}
```

Chapter 2. Module Documentation

Functions

- vxImportKernelFromURL

2.1. Functions

2.1.1. vxImportKernelFromURL

Import a kernel from binary specified by URL.

```
vx_kernel vxImportKernelFromURL(
    vx_context                          context,
    const vx_char*                      type,
    const vx_char*                      url);
```

Parameters

- [in] *context* - OpenVX context
- [in] *type* - Vendor-specific identifier that indicates to the implementation how to interpret the **url**. For example, if an implementation can interpret the **url** as a *file*, a *folder* a *symbolic label*, or a *pointer*, then a vendor may choose to use "vx_<vendor>_file", "vx_<vendor>_folder", "vx_<vendor>_label", and "vx_<vendor>_pointer", respectively for this field. Container types starting with "vx_khr_" are reserved. Refer to vendor documentation for list of container types supported.
- [in] *url* - URL to binary container.

Returns: A vx_kernel reference. Any possible errors preventing a successful import should be checked using vxGetStatus.

Note

An implementation may provide several different error codes to give useful diagnostic information in the event of failure to create the context.

Note

The name of kernel parameters can be queried using the vxQueryReference API with vx_parameter as **ref** and VX_REFERENCE_NAME as **attribute**.

Index