Building a Neural Network Inference Application

Radhakrishna Giduthuri
Advanced Micro Devices
radha.giduthuri@ieee.org
Agenda

- Workflow from pre-trained neural network model to OpenVX application
- Sample Neural Network Inference Application
- AMD open-source project: bit.ly/openvx-amd
Neural Network End-to-End Workflow

- **Neural Network Training Frameworks**
  - Caffe
  - Caffe2
  - mxnet
  - TensorFlow
  - CNTK
  - PaddlePaddle
  - PYTORCH

- **Third Party Tools**
  - OpenVX

- **Trained Network Model**

- **Vision and Neural Network Inferencing Runtime**

- **Vision/Al Applications**

- **Desktop and Cloud Hardware**
  - cuDNN
  - MIOpen
  - MKL-DNN

- **Embedded/Mobile/Desktop/Cloud Vision/Inferencing Hardware**
  - GPU
  - DSP
  - CPU
  - Custom
  - FPGA
Neural Network Import

Use open-source tools to import a “Neural Network Inference Computational Graph” into OpenVX Graph (used during the application development stage to produce OpenVX binary for deployment)
OpenVX 1.2 and Neural Net Extension

• Convolution Neural Network topologies can be represented as OpenVX graphs
  - Layers are represented as OpenVX nodes
  - Layers connected by multi-dimensional tensors objects
  - Layer types include convolution, activation, pooling, fully-connected, soft-max
  - CNN nodes can be mixed with traditional vision nodes

• Import/Export Extension
  - Efficient handling of network Weights/Biases or complete networks

• OpenVX will be able to import NNEF files into OpenVX Neural Nets
OpenVX Neural Network Extension

- Tensor types of INT16, INT7.8, INT8, and UINT8 are supported
  - Other types may be supported by a vendor
- Conformance tests will be up to some “tolerance” in precision
  - To allow for optimizations, e.g., weight compression
- Eight neural network “layer” nodes:

<table>
<thead>
<tr>
<th>vxActivationLayer</th>
<th>vxConvolutionLayer</th>
<th>vxDeconvolutionLayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxFullyConnectedLayer</td>
<td>vxNormalizationLayer</td>
<td>vxPoolingLayer</td>
</tr>
<tr>
<td>vxSoftmaxLayer</td>
<td>vxROIPoolingLayer</td>
<td>...</td>
</tr>
</tbody>
</table>
## CIFAR-10 Example: NNEF Container

<table>
<thead>
<tr>
<th>Filename</th>
<th>Type</th>
<th>Description</th>
<th>Size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph.nnef</td>
<td>Text file</td>
<td>Neural network structure</td>
<td>1,195</td>
</tr>
<tr>
<td>conv1/weights.dat</td>
<td>Tensor file</td>
<td>Weights of conv1</td>
<td>9,632</td>
</tr>
<tr>
<td>conv1/bias.dat</td>
<td>Tensor file</td>
<td>Biases of conv1</td>
<td>148</td>
</tr>
<tr>
<td>conv2/weights.dat</td>
<td>Tensor file</td>
<td>Weights of conv2</td>
<td>102,432</td>
</tr>
<tr>
<td>conv2/bias.dat</td>
<td>Tensor file</td>
<td>Biases of conv2</td>
<td>148</td>
</tr>
<tr>
<td>conv3/weights.dat</td>
<td>Tensor file</td>
<td>Weights of conv3</td>
<td>204,832</td>
</tr>
<tr>
<td>conv3/bias.dat</td>
<td>Tensor file</td>
<td>Biases of conv3</td>
<td>276</td>
</tr>
<tr>
<td>ip1/weights.dat</td>
<td>Tensor file</td>
<td>Weights of ip1</td>
<td>262,176</td>
</tr>
<tr>
<td>ip1/bias.dat</td>
<td>Tensor file</td>
<td>Biases of ip1</td>
<td>276</td>
</tr>
<tr>
<td>ip2/weights.dat</td>
<td>Tensor file</td>
<td>Weights of ip2</td>
<td>2,592</td>
</tr>
<tr>
<td>ip2/bias.dat</td>
<td>Tensor file</td>
<td>Biases of ip2</td>
<td>60</td>
</tr>
</tbody>
</table>

*Recommended: IEEE 1003.1-2008 tar archive with optional compression*
Useful open-source projects

- [https://github.com/KhronosGroup/NNEF-Tools](https://github.com/KhronosGroup/NNEF-Tools)
  NNEF-Tools: repository contains tools to generate and consume NNEF documents
  - parser/cpp: sample NNEF parser and validator
  - converter/caffe: Caffe to NNEF converter
  - converter/tensorflow: TensorFlow to NNEF converter

- [https://github.com/rgiduthuri/vidnn](https://github.com/rgiduthuri/vidnn)
  Some useful tools
  - translators/nnef2openvx: generate OpenVX graph from an NNEF container
Translating NNEF to OpenVX

<table>
<thead>
<tr>
<th>File in NNEF Container</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph.nnef</td>
<td>Text file</td>
</tr>
<tr>
<td>conv1/weights.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>conv1/bias.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>conv2/weights.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>conv2/bias.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>conv3/weights.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>conv3/bias.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>ip1/weights.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>ip1/bias.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>ip2/weights.dat</td>
<td>Tensor file</td>
</tr>
<tr>
<td>ip2/bias.dat</td>
<td>Tensor file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OpenVX Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annmodule.h annmodule.cpp</td>
<td>Function <strong>annCreateGraph()</strong></td>
</tr>
<tr>
<td>weights.bin</td>
<td>Binary file needed by <strong>annCreateGraph()</strong></td>
</tr>
<tr>
<td>anntest.cpp</td>
<td>Sample inference test program</td>
</tr>
<tr>
<td>CMakeLists.txt</td>
<td>Sample build script</td>
</tr>
</tbody>
</table>

*See [https://github.com/rgiduthuri/vidnn](https://github.com/rgiduthuri/vidnn) for **nnef2openvx** translator.*
CIFAR-10 Example: NNEF Structure

Input: 32x32 RGB image

Output: 10 classes
airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

version 1.0
graph cifar10( data ) -> ( prob ) {
  data = external(shape = [0,3,32,32]);
  conv1w = variable(shape = [32,3,5,5], label = ‘conv1/weights’);
  conv1b = variable(shape = [1,32], label = ‘conv1/bias’);
  conv1 = conv(data, conv1w, conv1b, padding = [(2,2),(2,2)]);
  pool1 = max_pool(conv1, size=[1,1,3,3], stride=[1,1,2,2], no_border);
  relu1 = relu(pool1);
  conv2w = variable(shape = [32,32,5,5], label = ‘conv2/weights’);
  conv2b = variable(shape = [1,32], label = ‘conv2/bias’);
  conv2 = conv(relu1, conv2w, conv2b, padding = [(2,2),(2,2)]);
  relu2 = relu(conv2);
  pool2 = avg_pool(relu2, size=[1,1,3,3], stride=[1,1,2,2], no_border);
  conv3w = variable(shape = [64,32,5,5], label = ‘conv3/weights’);
  conv3b = variable(shape = [1,64], label = ‘conv3/bias’);
  conv3 = conv(pool2, conv3w, conv3b, padding = [(2,2),(2,2)]);
  relu3 = relu(conv3);
  pool3 = avg_pool(relu3, size=[1,1,3,3], stride=[1,1,2,2], no_border);
  ip1w = variable(shape = [64,64,4,4], label = ‘ip1/weights’);
  ip1b = variable(shape = [1,64], label = ‘ip1/bias’);
  ip1 = conv(pool3, ip1w, ip1b, padding = [(0,0),(0,0)]);
  ip2w = variable(shape = [10,64,1,1], label = ‘ip2/weights’);
  ip2b = variable(shape = [1,10], label = ‘ip2/bias’);
  ip2 = conv(ip1, ip2w, ip2b, padding = [(0,0),(0,0)]);
  prob = softmax(ip2);
}
CIFAR-10 Example: OpenVX Code

```c
vx_status annCreateGraph(vx_graph graph, vx_tensor data, vx_tensor prob,
    const char * binaryFilename)
{
    vx_context context = vxGetContext((vx_reference)graph);
    vxLoadKernels(context, "vx_nn");

    vx_size conv1w_dims[4] = { 5, 5, 3, 32 };  
    vx_tensor conv1w = vxCreateTensor(context, 4, conv1w_dims,
        vx_size conv1b_dims[2] = { 32, 1 };  
    vx_tensor conv1b = vxCreateTensor(context, 2, conv1b_dims,
        vx_size conv1_dims[4] = { 32, 32, 32, 1 };  
    vx_tensor conv1 = vxCreateVirtualTensor(graph, 4, conv1_dims,
        { vx_nn_convolution_params_t par = { 0 };  
            par.padding_x = 2;  
            par.padding_y = 2;  
            par.dilation_x = 0;  
            par.dilation_y = 0;  
            par.overflow_policy = VX_CONVERT_POLICY_SATURATE;
            par.rounding_policy = VX_ROUND_POLICY_TO NeAREST_EVEN;
            par.down_scale_size_rounding = VX_NN_DS_SIZE_ROUNDING_FLOOR;
            vx_node node = vxConvolutionLayer(graph, data, conv1w, conv1b, &par, sizeof(par), conv1);
        }  
    vxReleaseNode(&node);
}
```

Application Integration

- Create context
- annCreateGraph
- Graph execution
- Release all objects
AMD OpenVX open-source project with NN

bit.ly/openvx-amd
All OpenVX projects on GitHub

LOOM: 360° live video stitching

Neural network acceleration for inference

Inference Application Development Workflow

Pre-trained CAFFE model

inference_generator

annmodule library

Application

openvx and vx_nn libraries

MIOpenGEMM, MIOpen, and OpenCL libraries

dog

Workstation

Select/Upload pre-trained NN Models
Browse Images
Submit for Inference
Inference Result Viewer (GUI)

Sample Inference Application

Server Node

annInferenceServer

Network Server (tcp/ip)

JPEG decode & scaling on CPUs

Inference Generator

Pre-compiled Models

Uploaded Models (Temporary Storage)
Hands-on Session
Benefits and Resources

• Faster development of efficient and portable vision applications
  - Developers are protected from hardware and training framework complexities
  - No platform-specific performance optimizations needed

• Graph description enables significant automatic optimizations
  - Scheduling, memory management, kernel fusion, and tiling

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

• Resources
  - Overview & Resources
    - https://www.khronos.org/nnef
    - https://www.khronos.org/openvx
  - Specifications: current, previous, and extensions
    - https://www.khronos.org/registry/NNEF
    - https://www.khronos.org/registry/OpenVX
Any Questions?

• Khronos is working on a comprehensive set of solutions for vision and inferencing
  - Layered ecosystem that include multiple developer and deployment options
• Please let us know if we can help you participate in any Khronos activities!
• A quick hands-on OpenVX Tutorial
  - https://github.com/rgiduthuri/openvx_tutorial
• Please contact us with any comments or questions!
  - Radhakrishna Giduthuri | radha.giduthuri@ieee.org

• These slides and further information on all these standards at Khronos
  - www.khronos.org