HARDWARE ACCELERATION FOR MACHINE LEARNING AND COMPUTER VISION THROUGH KHRONOS OPEN STANDARD APIs

Thursday, May 23, 2019 from 9:00 am - 5:00 pm

Inference with OpenVX

2:30 – 5:00 PM
Rajy Rawther
Kiriti Nagesh Gowda
MIVisionX toolkit is a comprehensive set of computer vision and machine intelligence libraries, utilities and applications bundled into a single toolkit.

AMD OpenVX is delivered as Open Source with MIVisionX.

Primarily targeted at applications requiring a combination of machine learning inference and computer vision or image/video processing.

Includes a model compiler for converting and optimizing a pretrained model from existing formats such as Caffe, NNEF and ONNX to an OpenVX backend.

After compilation, MIVisionX generates an optimized library specific for a backend to run inferencing and vision pre- and post-processing modules.

It is beneficial to have lightweight and dedicated APIs optimized for AMD hardware for inference deployment as opposed to heavyweight frameworks.
All the software for MIVisionX is available under ROCm - AMD’s platform for GPU computing and machine learning.

In this tutorial we will show how to use MIVisionX toolkit to run some sample neural net applications doing image classification, object detection or segmentation.

The tutorial includes hands on training examples that allow participants to develop real world applications including computer vision, inferencing and visualization.

The tutorial demonstrates how the OpenVX framework with neural network extension is used to deploy some sample applications.
Inference with OpenVX

- Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
- Optimize NN model in OpenVX
  - Layer Fusion
  - Memory Optimization
  - Quantization
  - Tensor Fusion
  - Dynamic batch size update
- Add pre- & post-processing nodes
- Run Optimized inference on target hardware
Inference with OpenVX

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Exchange Formats</th>
<th>Compiler &amp; Optimizer</th>
<th>RunTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chainer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyTorch</td>
<td></td>
<td></td>
<td>Layer Fusion</td>
</tr>
<tr>
<td>Caffe2</td>
<td></td>
<td></td>
<td>Memory Optimization</td>
</tr>
<tr>
<td>mxnet</td>
<td>ONNX</td>
<td>* Quantization</td>
<td>* Tensor Fusion</td>
</tr>
<tr>
<td>TensorFLow</td>
<td></td>
<td></td>
<td>* Dynamic Batch Size Update</td>
</tr>
<tr>
<td>CNTK</td>
<td>NNEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddlePaddle</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Layer Fusion
* Memory Optimization
* Quantization
* Tensor Fusion
* Dynamic Batch Size Update
# Inference with OpenVX

## ResNet50 Overall Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images With Ground Truth</td>
<td>7975</td>
</tr>
<tr>
<td>Images Without Ground Truth</td>
<td>1905</td>
</tr>
<tr>
<td>Total Images</td>
<td>9880</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Top 5 Match</td>
<td>6307</td>
</tr>
<tr>
<td>Accuracy on Top 5</td>
<td>79.08 %</td>
</tr>
<tr>
<td>Average Pass Confidence for Top 5</td>
<td>56.18 %</td>
</tr>
<tr>
<td>Total Mismatch</td>
<td>1668</td>
</tr>
<tr>
<td>Mismatch Percentage</td>
<td>20.92 %</td>
</tr>
<tr>
<td>Average mismatch Confidence for Top 1</td>
<td>37.19 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Match</td>
<td>4307</td>
<td>54.01 %</td>
</tr>
<tr>
<td>2nd Match</td>
<td>999</td>
<td>12.53 %</td>
</tr>
<tr>
<td>3rd Match</td>
<td>528</td>
<td>6.62 %</td>
</tr>
<tr>
<td>4th Match</td>
<td>283</td>
<td>3.55 %</td>
</tr>
<tr>
<td>5th Match</td>
<td>190</td>
<td>2.38 %</td>
</tr>
</tbody>
</table>
Inference with OpenVX

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.

Frameworks: Caffe, Torch, Caffe2, MXnet
Exchange Formats: ONNX, NNEF
Compiler & Optimizer: Layer Fusion, Memory Optimization, Quantization
RunTime: Dynamic, Batch Size Update

OpenVX, OpenCV, ROCm

KHRONOS Group
Prerequisites

• Ubuntu 16.04/18.04 or CentOS 7.5/7.6
• ROCm supported hardware
  • AMD Radeon GPU or APU required
• ROCm 2.4 and above
• Build & Install MIVisionX
  • MIVisionX installs model compiler at /opt/rocm/mivisionx

Tutorial: We will be using MIVisionX Dockers
## AMD ML SOFTWARE Stack with ROCm

### Data Platform Tools
- Machine Learning Apps
- MIvisionX Apps

### Frameworks
- Caffe2
- PyTorch
- TensorFlow

### Exchange formats
- NNEF
- ONNX

### Middleware and Libraries
- MIOpen
- BLAS, FFT, RNG
- RCCL
- Eigen

### ROCm
- OpenMP
- HIP
- OpenCL™
- Python

### Devices
- GPU
- CPU
- APU
- Future Accelerators

### OPEN SOURCE FOUNDATION FOR MACHINE LEARNING
- Latest Machine Learning Frameworks
- Optimized Math & Communication Libraries
- Docker and Kubernetes support
- Up-Streamed for Linux Kernel Distributions
Tutorial Example 1: Image classification

- Using a pretrained ONNX model
  
  **Step 1**: Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
  - Convert Pre-Trained model to AMD NNIR

```python
# Caffe
To convert a pre-trained caffemodel into AMD NNIR model:

```python
% python caffe_to_nnir.py <net.caffemodel> <nnirOutputFolder> --input-dims <n,c,h,w> [--verbose <0|1>]
```

```python
# ONNX
To convert an ONNX model into AMD NNIR model:

```python
% python onnx_to_nnir.py <model.onnx> <nnirModelFolder> [OPTIONS]

OPTIONS:
  - --input_dims n,c,h,w
```

```python
# NNEF
To convert a NNEF model into AMD NNIR model:

```python
% python nnef_to_nnir.py <nnefInputFolder> <nnirOutputFolder>
```
Tutorial Example 1: Image classification

- Using a pretrained ONNX model

  - **Step 1**: Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
    - Convert Pre-Trained model to AMD NNIR

```
input: data; offset; loss;
conv1_w; conv1_b;
conv2_w; conv2_b;
conv3_w; conv3_b;
conv4_w; conv4_b;
conv5_w; conv5_b;
conv6_w; conv6_b;
conv7_w; conv7_b;
conv8_w; conv8_b;
conv9_w; conv9_b;
conv10_w; conv10_b;
conv11_w; conv11_b;
conv12_w; conv12_b;
conv13_w; conv13_b;
conv14_w; conv14_b;
conv15_w; conv15_b;
conv16_w; conv16_b;
conv17_w; conv17_b;
conv18_w; conv18_b;
conv19_w; conv19_b;
conv20_w; conv20_b;
conv21_w; conv21_b;
conv22_w; conv22_b;
conv23_w; conv23_b;
conv24_w; conv24_b;
conv25_w; conv25_b;
conv26_w; conv26_b;
conv27_w; conv27_b;
conv28_w; conv28_b;
conv29_w; conv29_b;
conv30_w; conv30_b;
conv31_w; conv31_b;
conv32_w; conv32_b;
conv33_w; conv33_b;
conv34_w; conv34_b;
conv35_w; conv35_b;
conv36_w; conv36_b;
conv37_w; conv37_b;
conv38_w; conv38_b;
conv39_w; conv39_b;
conv40_w; conv40_b;
conv41_w; conv41_b;
conv42_w; conv42_b;
conv43_w; conv43_b;
conv44_w; conv44_b;
conv45_w; conv45_b;
conv46_w; conv46_b;
conv47_w; conv47_b;
conv48_w; conv48_b;
conv49_w; conv49_b;
conv50_w; conv50_b;
conv51_w; conv51_b;
conv52_w; conv52_b;
conv53_w; conv53_b;
conv54_w; conv54_b;
conv55_w; conv55_b;
conv56_w; conv56_b;
conv57_w; conv57_b;
conv58_w; conv58_b;
conv59_w; conv59_b;
conv60_w; conv60_b;
conv61_w; conv61_b;
conv62_w; conv62_b;
conv63_w; conv63_b;
conv64_w; conv64_b;
conv65_w; conv65_b;
conv66_w; conv66_b;
conv67_w; conv67_b;
conv68_w; conv68_b;
conv69_w; conv69_b;
conv70_w; conv70_b;
conv71_w; conv71_b;
conv72_w; conv72_b;
conv73_w; conv73_b;
conv74_w; conv74_b;
conv75_w; conv75_b;
conv76_w; conv76_b;
conv77_w; conv77_b;
conv78_w; conv78_b;
conv79_w; conv79_b;
conv80_w; conv80_b;
conv81_w; conv81_b;
conv82_w; conv82_b;
conv83_w; conv83_b;
conv84_w; conv84_b;
conv85_w; conv85_b;
conv86_w; conv86_b;
conv87_w; conv87_b;
conv88_w; conv88_b;
conv89_w; conv89_b;
conv90_w; conv90_b;
conv91_w; conv91_b;
conv92_w; conv92_b;
conv93_w; conv93_b;
conv94_w; conv94_b;
conv95_w; conv95_b;
conv96_w; conv96_b;
conv97_w; conv97_b;
conv98_w; conv98_b;
conv99_w; conv99_b;
```
Tutorial Example 1: Image classification

• Using a pretrained ONNX model

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

To update batch size in AMD NNIR model:

```python
% python nnir_update.py --batch-size <N> <nnirModelFolder> <nnirModelFolderN>
```

To fuse operations in AMD NNIR model (like batch normalization into convolution):

```python
% python nnir_update.py --fuse-ops <1> <nnirModelFolderN> <nnirModelFolderFused>
```

To quantize the model to float 16:

```python
% python nnir_update.py --convert-fp16 <1> <nnirModelFolderN> <nnirModelFolderFused>
```
Tutorial Example 1: Image classification

- Using a pretrained ONNX model

  - **Step 1**: Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
    - Convert AMD NNIR to OpenVX C Code

To convert an AMD NNIR model into OpenVX C code:

```
% python nnir_to_openvx.py --help
```

Usage: python nnir_to_openvx.py [OPTIONS] <nnirInputFolder> <outputFolder>

OPTIONS:
- --argmax UINT8  -- argmax at the end with 8-bit output
- --argmax UINT16 -- argmax at the end with 16-bit output
- --argmax <fileNamePrefix>rgb.txt  -- argmax at the end with RGB color mapping using LUT
- --argmax <fileNamePrefix>rgba.txt  -- argmax at the end with RGBA color mapping using LUT
- --help            -- show this help message

LUT File Format (RGB): 8-bit R G B values one per each label in text format
R0 G0 B0
R1 G1 B1
...

LUT File Format (RGBA): 8-bit R G B A values one per each label in text format
R0 G0 B0 A0
R1 G1 B1 A1
Tutorial Example 1: Image classification

• Using a pretrained ONNX model

- Step 1: Convert Pre-trained models in Caffe/NNEF/ONNX to OpenVX graph
  - Convert AMD NNIR to OpenVX C Code
Tutorial Example 1: Image classification

- Using a pretrained ONNX model
  - Step 2: Add pre & post processing nodes

**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.
Tutorial Example 1: Image classification

- Using pretrained ONNX model

  ➢ Step 3: Run Optimized inference on target hardware
Tutorial Example 2: Object Detection

- Using a Pre-Trained Caffe model
Tutorial Example 3: Image Classification

• Using a Pre-Trained NNEF model

<table>
<thead>
<tr>
<th>Model</th>
<th>ms/frame</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGGNet-16</td>
<td>13.57</td>
<td></td>
</tr>
</tbody>
</table>

Confidence
MIVisionX Inference Deployment API

- Enables a pre-trained CNN model to be deployed on AMD’s ROCm stack with MIVisionX
- Architected to support different backends including OpenCL-ROCm, OpenCL-HIP, WinML etc. All options use OpenVX for executing the inference engine.
- Two stage process
  - mv_compile for compiling the model (Caffe, ONNX, NNEF) for the specific backends with the option to run Model Optimizer for fuse operations, quantization etc.
  - mv_compile generates deployment library (libmvdeploy.so), binary weights for the model, and some .cpp and .h files required to run inference engine.
  - Also it generates a post-processing module for running argmax or bounding box object detection.
  - Users can use the deployment package to develop real-world applications for both inference and vision.
Pre-trained and Optimized NN Model (Caffe, ONNX, NNEF) → Convert to NNIR → Optimize NNIR Model → NNIR2clib compiler → Deployment package: library, weights & sample

(Application) Use deployment package and add pre/post processing modules to build real-world application

Fuse kernels, memory optimization, quantize etc.
Compile model for specific backend using `mv_compile`.

- `mv_compile` options:
  - `--model <model_name>`: name of the trained model with full path [required]
  - `--install_folder <install_folder>`: the location for compiled model [required]
  - `--input_dims <n,c,h,w>`: dimension of input for the model given in format NCHW [required]
  - `--backend <backend>`: is the name of the backend for compilation [optional: default: OpenVX_Rocm_OpenCL]
  - `--fuse_cba <0/1>`: enable or disable Convolution_bias_activation fuse mode (0/1) [optional: default: 0]
  - `--quant_mode <fp32/fp16>`: if enabled the model and weights would be converted [optional: default: fp32]

Deploy the compiled model with optional pre and/or postprocessing functions. `mvdeploy_api.h` has all the API function definitions for deploying the model.
# MIVisionX Deployment API

<table>
<thead>
<tr>
<th>API Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mvInitializeDeployment</code> (const char* <code>install_folder</code>)</td>
<td>Initializes the deployment engine with the URL of the <code>install_folder</code></td>
</tr>
<tr>
<td><code>QueryInference</code> (int *num_inputs, int *num_outputs,..)</td>
<td>Returns the model config such as input and output dimensions</td>
</tr>
<tr>
<td><code>SetPreProcessCallback</code> (mv_add_preprocess_callback_f preproc_f, ...)</td>
<td>Callback function to add preprocessing nodes to the same OpenVX graph as the inference</td>
</tr>
<tr>
<td><code>mvCreateInferenceSession</code> (mivid_session *inf_session, const char *install_folder)</td>
<td>Creates an inference deployment session and returns a handle</td>
</tr>
<tr>
<td><code>mvSetInputDataFromMemory/File</code> (mivid_session inf_session, int input_num, void *input_data, ..)</td>
<td>sets input for the inference or copies it to input tensor</td>
</tr>
<tr>
<td><code>mvRunInference</code> (mivid_session inf_session, ...)</td>
<td>Executes the inference for a single frame for #iterations</td>
</tr>
<tr>
<td><code>mvGetOutput</code> (mivid_session inf_session, void *out_mem,..)</td>
<td>Copies the output to user specified memory</td>
</tr>
<tr>
<td><code>mvShutdown</code> (mivid_session inf_session)</td>
<td>Releases inference session and free all resources</td>
</tr>
</tbody>
</table>
Tutorial Example 5: Object detection from video using OpenVX video decoder preprocessing node.

- This example shows how to add pre-processing nodes, such as decoding with the amd_video_decoder node along with inferencing for a realistic object detection application.

- Shows the use of MIVisionX utility mv_compile, which is used to compile an inference deployment package from a pre-trained model for a specific backend. The compiled package consists of a library, header files, application specific source files and the binary weights of the model.

- Also it shows how to add a pre-processing node to the inference graph using the MIVisionX deployment API.

- It demonstrates how to build an application which does objection detection using YoloV2 from one video stream or multiple video streams. This example shows the advantage of using OpenVX which avoids extra data copies thereby achieving a significance performance increase.
Tutorial Example 5: Flow chart

- **yolov2 pretrained model**
  - mv_compile utility
  - mvInitializeDeployment()
  - mvCreateInference()
    - Creates vxGraph, vxContext and calls vxVerifyGraph()
  - mvRunInference(), called in a loop for all frames in the video
  - mvShutdown(), Releases vxGraph, all vx memory resources

- Lib_mvdeploy.so, weights.bin, mv_deployapi.cpp, sample app
  - preprocess_addnodes_cb()
    - Adds vx_amdMediaDecoder node, vx_convertImageToTensor() node
  - VxProcessGraph() with preprocessing nodes and CNN nodes

- **mv_compile utility**
  - mvobjdetect example

- Initializes deployment from the library; loads weights for the constant tensors
Tutorial Example 5: use of mv_compile.exe

Step 1: mv_compile.exe --model <model> --install_folder <install_folder> --input_dims n,c,h,w --fuse-cba <0/1>
- Generates library libmv_deploy.so, mvdeploy.h, weights.bin, mvdeploy_api.cpp, mvtestdeploy.cpp in the install_folder
- Generates mv_extras library for doing the most common postprocessing functions like argmax and bounding box detections

Step 2: Compile and run mvobjdetect example using the precompiled deployment package
- Copy additional source files and CMakeLists.txt from mvobjdetect folder to the install_folder
- Build the mvobjdetect example
- Run using mvobjdetect <input_file.mp4> --install_folder . --frames <num> --bb cls, th1, th2 --v
- Shows the usage of OpenCV for visualizing the results
Example shows decoding 4 video streams simultaneously using amd_media_decoder OpenVX node and running the inference on 4 streams and visualizing the results using OpenCV.
Resources

- Tutorial part 1: https://github.com/kiritigowda/MIVisionX-Inference-Tutorial
- Tutorial part 2: https://github.com/rrawther/MIVisionX-OpenVX-Tutorial
DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change. "Polaris", "Vega", "Radeon Vega", "Navi", "Zen" and "Naples" are codenames for AMD architectures, and are not product names.

While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale.

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale. GD-18

©2019 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Ryzen, Threadripper, EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.