GDAŃSK UNIVERSITY OF TECHNOLOGY

intel

International Summer School on Deep Learning

July 2nd-6th 2018, Gdansk, Poland

Deep Learning
Inference with Movidius™ Neural Compute Stick

Jacek Czaja, Krzysztof Biniaś

Intel Corporation

Introduction to Intel Movidius C API (V2)

Jacek Czaja Krzysztof Binias

Intel Corporation

June 28, 2018

Table of contents

Intel Movidius Neural Compute Stick

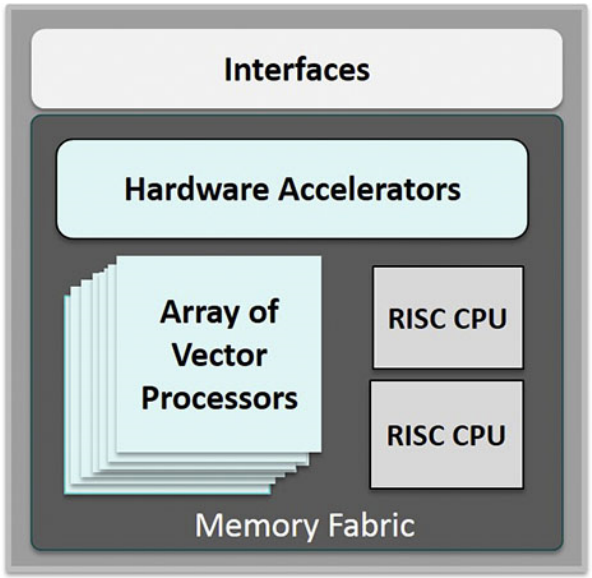
NCS is a USB-thumb-drive-sized deep learning machine that you can use to learn AI programming at the edge

- Based on Myriad 2 processor (28 nm)
- 80-150 GFLOPS performance
- Consumes only 1W of power
- Connectivity: USB 3.0 Type-A
- Operating temperature: 0 - 40 C

Intel Movidius Neural Compute Stick

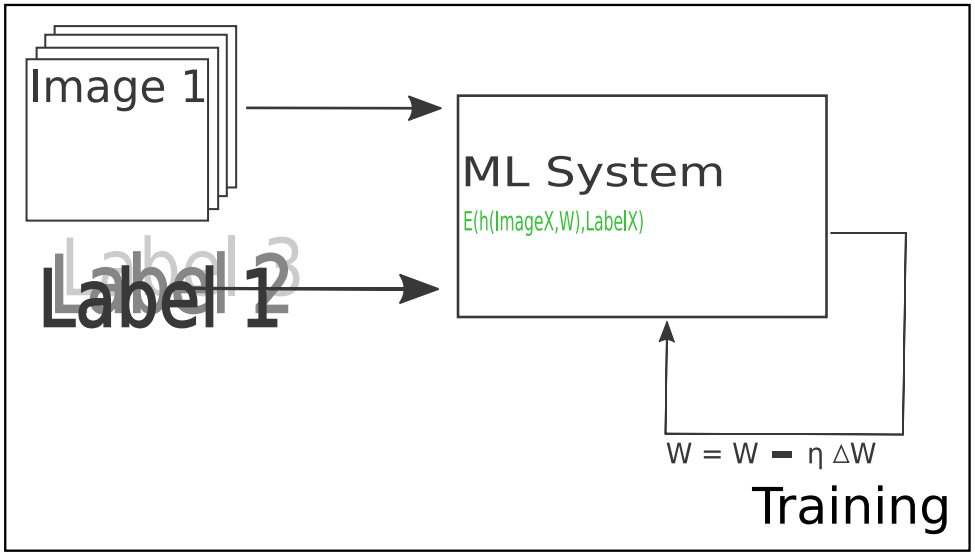
The Intel Movidius Myriad 2 VPU

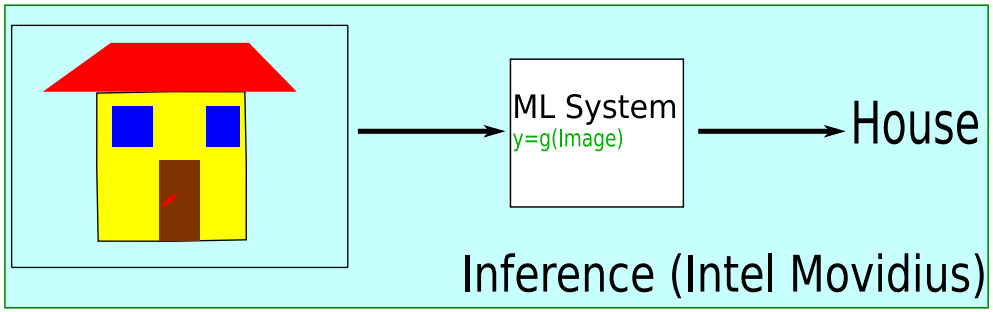
- An ultra-low power design
- Featuring 12 VLIW programmable SHAVE cores, dedicated vision accelerators and 2 CPUS
- 12 programmable SHAVE cores
- A small-area footprint
- Support for 16/32-bit floating point and 8/16/32-bit integer operations



Myriad 2 Vision Processor Unit (VPU)

Stages of Deep learning





Prerequisities

- Raspbian is officially supported
- Ubuntu 16.04 Linux is officially supported
  - native installation
  - Virtual machine
  - docker
- Other Linux flavours are not supported
  - Python3 is needed

Configuring and Building project that uses C API - Commandline

Example Commandline :

```
g++ main.cpp -I<dir with mvnc.h> -lmvnc -L<dir with libmvnc.so> -o hello-movidius
```

Example Commandline from Fedora Linux :

```
g++ main.cpp -I/usr/local/include -lmvnc -L/usr/local/lib -o hello-movidius
```

Configuring and Building of project using C API – cmake

```
cmake_minimum_required(VERSION 2.8)
project(task1)

# ---[ Find Intel Movidius header
find_path(NCS_INCLUDE_DIR NAMES "mvnc.h"
          HINTS "/usr/local"
          PATHS "/usr/local"
          PATH_SUFFIXES "include" )

# ---[ Find  Intel Movidius library
find_library(NCS_LIBRARY
          NAMES mvnc
          PATHS /usr/local/lib)
if(NCS_LIBRARY)
  message(STATUS "Found Movidius NCS (include: ${NCS_INCLUDE_DIR}, lib: ${NCS_LIBRARY}")
  include_directories(${NCS_INCLUDE_DIR})
else()
  message(FATAL " Intel NCS not located properly")
endif()

add_executable(task2 main.cpp )
target_link_libraries(task1 ${NCS_LIBRARY} )
```

Configuring and Building project that uses C API - task

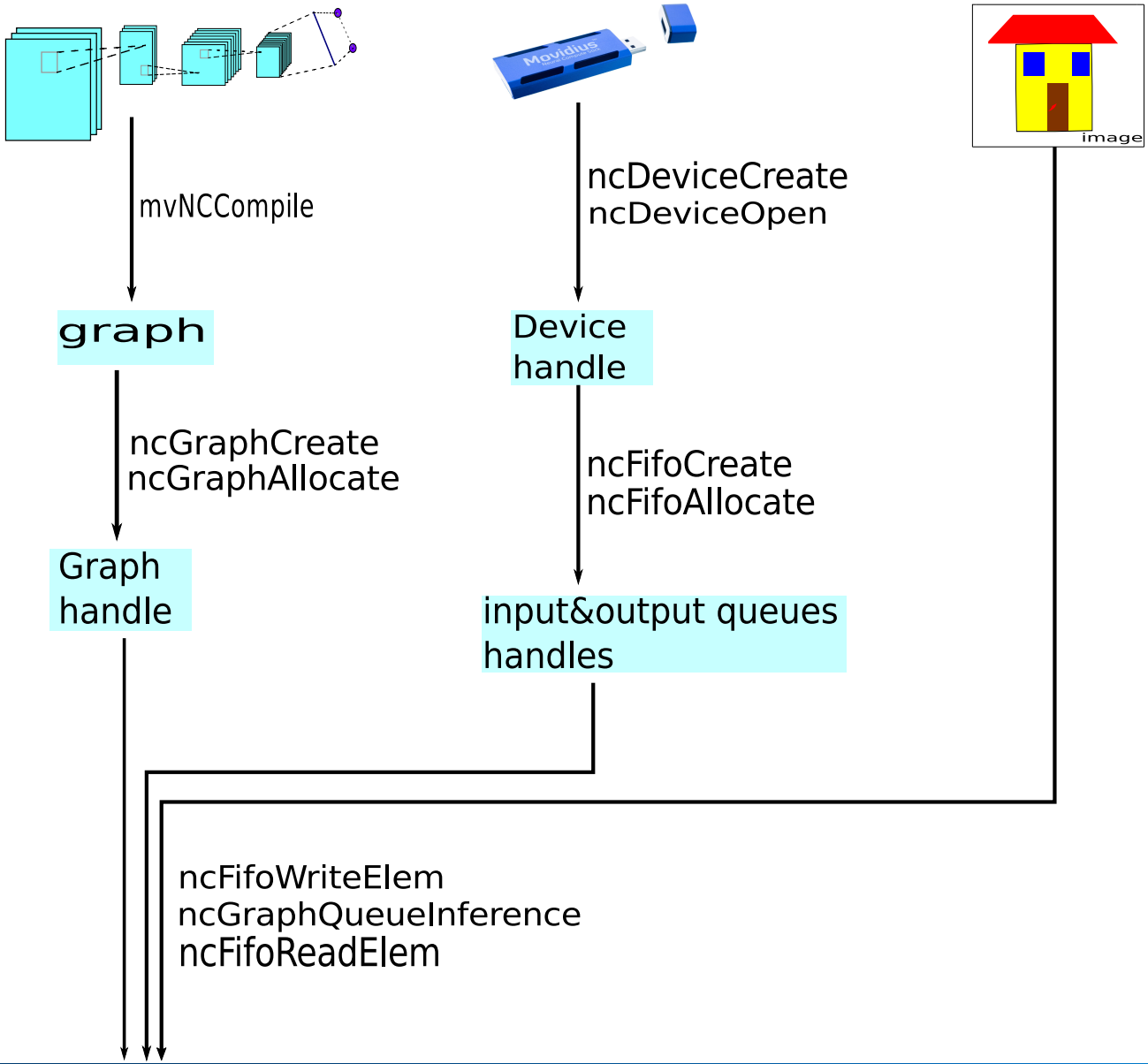
Example cmake commandline :

```
mkdir build; cd build; cmake ../
```

Task1:

1. Use cmake to build code in directory: task1
2. Run created binary without NCS inside
3. Run created binary with NCS

Working with Intel Movidius NCS



Locating NCS devices in a system

```
index = 0;  // Index of device to query for
ncStatus_t ret = ncDeviceCreate(index,&deviceHandle);
if (ret == NC_OK) {
  std::cout << "Found NCS named: " << index++ << std::endl;
}

// If not devices present the exit
if (index == 0) {
  throw std::string("Error: No Intel Movidius identified in a system!\n");
}
```

- can be called in multiple times (until an error is returned) to create device handles for multiple NCS devices.

Opening communication with NCS devices

```
ret = ncDeviceOpen(deviceHandle);
if(ret != NC_OK) {
   // If we cannot open communication with device then clean up resources
   destroy();
   throw std::string("Error: Could not open NCS device: ") + std::to_string(index-1);
}
```

- ncDeviceOpen can be called for each initialized NCS

Creating and Training model

- caffe or tensorflow are supported
- classification and detection is supported
- No batch processing mode for inference.
- not all corner cases of caffe functionality is supported

Conversion of model for NCS

Example Commandline for converting GoogleNet model prepared with Caffe:

```
mvNCCompile -w bvlc_googlenet.caffemodel deploy.prototxt -s 12 -o myGoogleNet
```

Actions performed by mvNCCompile:

- convert model's data layout from ZYX to YXZ
- convert data format : floating point 16
- merge(fuse) layers eg. Relu+BatchNorm

Supported DNN frameworks:

- Caffe
- Tensorflow

Conversion of model for NCS – task

Examples of conversions :

```
mvNCCompile -w bvlc_googlenet.caffemodel deploy.prototxt -s 1 -o myGoogleNet-shave1
mvNCCompile -w bvlc_googlenet.caffemodel deploy.prototxt -s 12 -o myGoogleNet-shave12
```

Task2:

1. find and enter task2 directory
2. convert trained googlenet model to graph suited for one shave
3. convert trained googlenet model to graph suited for 12 shaves

Loading NCS graph

```
// Creation of graph resources
unsigned int graphSize = 0;
loadGraphFromFile(graphFile, graphFileName, &graphSize);
ncStatus_t ret = ncGraphCreate(graphFileName.c_str(),&graphHandlePtr);
if (ret != NC_OK) {
    throw std::string("Error: Graph Creation failed!");
}
// Allocate graph on NCS
ret = ncGraphAllocate(deviceHandle,graphHandlePtr,graphFile.get(),graphSize);
if (ret != NC_OK) {
    destroy();
    throw std::string("Error: Graph Allocation failed!");
}
unsigned int optionSize = sizeof(inputDescriptor);
ret = ncGraphGetOption(graphHandlePtr,
        NC_RO_GRAPH_INPUT_TENSOR_DESCRIPTORS,
        &inputDescriptor,
        &optionSize);
if (ret != NC_OK) {
  destroy();
  throw std::string("Error: Unable to create input FIFO!");
}
```

- Many graphs can be loaded to single NCS
- Each graph can be send to only one NCS.

Loading NCS graph

```
void loadGraphFromFile(std::unique_ptr<char[]>& graphFile, const std::string& graphFileNa
{
  std::ifstream ifs;
  ifs.open(graphFileName, std::ifstream::binary);
  if (ifs.good() == false) {
    throw std::string("Error: Unable to open graph file: ") + graphFileName;
  }

  // Get size of file
  ifs.seekg(0, ifs.end);
  *graphSize = ifs.tellg();
  ifs.seekg(0, ifs.beg);

  graphFile.reset(new char[*graphSize]);
  ifs.read(graphFile.get(),*graphSize);
  ifs.close();
}
```

Creating Queues (FIFO)

```
  // Create input FIFO
ncStatus_t ret = ncFifoCreate("input1",fifotype, &FIFO_);
  if (ret != NC_OK) {
    throw std::string("Error: Unable to create FIFO!");
  }

....
  ret = ncFifoAllocate(input.FIFO_, deviceHandle, &inputDescriptor, 2);
  if (ret != NC_OK) {
    destroy();
    throw std::string("Error: Unable to allocate input  FIFO! on NCS");
  }
```

- It is nessesery to create at least one input and at least one output queues

Starting inference

```
ret = ncFifoWriteElem(input.FIFO_,tensor.get(),&inputLength, nullptr);
if (ret != NC_OK) {
  throw std::string("Error: Loading Tensor into input queue failed!");
}

ret = ncGraphQueueInference(graphHandlePtr,&(input.FIFO_), 1, &(output.FIFO_), 1);
if (ret != NC_OK) {
  throw std::string("Error:  Queing inference failed!");
}
```

- For each written element into input FIFO we need to call ncGraphQueueInference

Preparing tensor – 1

```
void prepareTensor(std::unique_ptr<unsigned char[]>& input, std::string& imageName,unsig
{
  // load an image using OpenCV
  cv::Mat imagefp32 = cv::imread(imageName, -1);
  if (imagefp32.empty())
    throw std::string("Error reading image: ") + imageName;

  // Convert to expected format
  cv::Mat samplefp32;
  if (imagefp32.channels() == 4 && net_data_channels == 3)
    cv::cvtColor(imagefp32, samplefp32, cv::COLOR_BGRA2BGR);
  else if (imagefp32.channels() == 1 && net_data_channels == 3)
    cv::cvtColor(imagefp32, samplefp32, cv::COLOR_GRAY2BGR);
  else
    samplefp32 = imagefp32;

  // Resize input image to expected geometry
  cv::Size input_geometry(net_data_width, net_data_height);

  cv::Mat samplefp32_resized;
  if (samplefp32.size() != input_geometry)
    cv::resize(samplefp32, samplefp32_resized, input_geometry);
  else
    samplefp32_resized = samplefp32;

  ....
```

Preparing tensor – 2

```
...
  // Convert to float32
  cv::Mat samplefp32_float;
  samplefp32_resized.convertTo(samplefp32_float, CV_32FC3);

  // Mean subtract
  cv::Mat input;
  cv::Mat mean = cv::Mat(input_geometry, CV_32FC3, net_mean);
  cv::subtract(samplefp32_float, mean, input);

  *inputLength = sizeof(short)*net_data_width*net_data_height*net_data_channels;
}
```

- By default data is accepted in float (32 bit format type), but it is possible to deliver input in 16 bit floating point type

Getting inference results

```
// Get size of outputFIFO element
optionSize = sizeof(unsigned int);

ret = ncFifoGetOption(output.FIFO_,
    NC_RO_FIFO_ELEMENT_DATA_SIZE,
    &outputFIFOsize,
    &optionSize);
if (ret != NC_OK) {
  throw std::string("Error: Getting
    output FIFO element size failed!");
}

// Prepare buffer for reading output
result.reset(new
    unsigned char[outputFIFOsize]);

ret = ncFifoReadElem(output.FIFO_,
    result.get(),
    &outputFIFOsize,
    nullptr);
if (ret != NC_OK) {
  throw std::string("Error: Reading
element failed !");
}
```

Example Results:

```
...
0
0
0.00014782
0.000174284
0
0
0.00274658
0.00568008
0
0.00163364
0.234131
0.495605
0.0139542
0.00143623
0.162109
0
0.0288391
0
0
...
```

- ncFifoReadElem blocks till results are available

Performance evaluation

```
unsigned int optionSize = sizeof(unsigned int);
unsigned int profilingSize = 0;

ncStatus_t ret = ncGraphGetOption(
    graphHandlePtr,
    NC_RO_GRAPH_TIME_TAKEN_ARRAY_SIZE,
    &profilingSize,
    &optionSize);

std::unique_ptr<float> profilingData(new
    float[profilingSize/sizeof(float)]);

ret = ncGraphGetOption(graphHandlePtr,
    NC_RO_GRAPH_TIME_TAKEN,
    profilingData.get(),
    &profilingSize);

std::cout << "Performance profiling:"
    << std::endl;
float totalTime = 0.0f;
int num_measuers = profilingSize/sizeof(float);
for(unsigned int i=0; i<num_measuers; ++i) {
  std::cout << " "
    << profilingData.get()[i]
    << " ms"<<std::endl;
  totalTime += profilingData.get()[i];
}
std::cout << "Total compute time: "
    << std::to_string(totalTime) << " ms"<< std::endl;
```

```
Performance profiling:
        0.005325 ms
        5.715929 ms
        1.142653 ms
        0.552343 ms
        1.450738 ms
        14.622865 ms
        1.481223 ms
        0.826488 ms
        0.895807 ms
        1.151683 ms
        5.986005 ms
        0.492310 ms
        1.383597 ms
        0.158708 ms
        ......
        0.734407 ms
        0.215032 ms
        0.781197 ms
        0.202845 ms
Total time: 116.748215 ms
```

Finishing work with NCS

```
// Deallocating graph
ncStatus_t ret = ncGraphDestroy(&graphHandlePtr);
if (ret != NC_OK)
  std::cout << "Error: Graph destroying failed!" << std::endl;

// Releasing queue
ncStatus_t ret = ncFifoDestroy(&FIFO_);
if (ret != NC_OK)
  std::cout << "Error: FIFO destroying failed!" << std::endl;

// Closing communication with NCS
ncStatus_t ret = ncDeviceClose(deviceHandle);
if (ret != NC_OK) {
  std::cerr << "Error: Closing of device: "
    <<  std::to_string(index-1) <<"failed!" << std::endl;
}

// Releasing resources for NCS handling
ncStatus_t ret = ncDeviceDestroy(&deviceHandle);
if (ret != NC_OK) {
  std::cerr << "Error: Freeing resources of device: "<<  std::to_string(index-1)
    <<"failed!" << std::endl;
}
```

- Lack of closing device may result in device not been available for some time

Performance evaluation - Task

Task3:

1. find and enter task3 directory
2. build main.cpp to use graph compiled for one shave

```
mkdir build; cd build; cmake ../; make
```

3. execute program task3 using cat.jpg and note top-1 classification result

```
cd ../; ./build/test-ncs-v2 cat.jpg -graph myGoogleNet-shave1
```

4. execute program task3 using cat.jpg with **profiling** and note performance results

```
./build/test-ncs-v2 cat.jpg -graph myGoogleNet-shave1 -profile
```

5. execute program task3 using cat.jpg with **profiling** but with graph compiled for 12 shaves. Note performance results

```
./build/test-ncs-v2 cat.jpg -graph myGoogleNet-shave12 -profile
```

References I

[1] Caffe* Optimized for Intel Architecture: Applying Modern Code Techniques. Improving the computational performance of a deep learning framework. Vadim Karpusenko, Ph.D. Andres Rodriguez, Ph.D. Jacek Czaja, Mariusz Moczala

[2] TensorFlow* Optimizations on Modern Intel Architecture. Elmoustapha Ould-Ahmed-Vall et el.