Practical Verification for Edge AI use and Effort for Functional Improvement

June 20st, 2018
Yasumitsu Takahashi
NTT DATA MSE Corporation
Who am I?

NTT DATA MSE Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Yasumitsu Takahashi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Deputy Manager</td>
</tr>
</tbody>
</table>
| Carrier            | - Technical leader of R&D projects in NTT DATA MSE  
|                    | - Linux-based embedded devices  
|                    | - Application, middleware, kernel development |

© 2018 NTT DATA MSE Corporation
Agenda

• Introduction to our Activities

• Effort for performance improvement
  ✓ System Overview
  ✓ How to integrate app utilizing OpenCL into AGL
  ✓ Performance Verification & Consideration

• Possibility verification of continuous improvement of inference accuracy for edge AI
  ✓ System Overview
  ✓ Demonstration & Verification Result
Introduction to our Activities

- **Our team have started AI related activities since October 2017**
  - There are AI related news almost everyday
  - First Step: Let’s use an AI engine on edge devices
    ✓ Implemented a demo system “Handwritten Digit Recognition App”
  - Try creating own neural network architecture
  - Performance improvement
    ✓ More complicated neural network architecture and graphic processing cause performance issues
  - Consider how to improve inference accuracy on a practical scene and update calibration such as personal preference

- **In this presentation**
  - Performance verification
    ✓ How to integrate app utilizing OpenCL(GPU)
    ✓ Verify the performance effect of OpenCL
  - Verification to improve inference accuracy for edge AI continuously
Effort for performance improvement
System Overview (1/2)

Perform the followings to the pre-developed demo system (Handwritten Digit Recognition)

- Investigate how to apply OpenCL (GPU) and implement it
- Verify the effect of OpenCL

Preparation

Deep learning on a PC

Demo System

Deep learning on a PC

GUI: Handwritten Digit Recognition App

App utilizes OpenCL (GPU) via OpenCV

Performance:

- CPU: Cortex-A57, Quad core, 1.500 GHz
- GPU: PowerVR Rogue GX6650, 192 core, 600MHz
- AGL: EE 5.0.0
- NNabla: v0.9.6 (https://github.com/sony/nnabla)
- OpenCV: v3.2
- OpenCL: v1.2

© 2018 NTT DATA MSE Corporation
What is OpenCL?

OpenCL (Open Computing Language) is a cross-platform framework for parallel computing by using computing resources mixed by multicore CPU/GPU/DPS (Heterogeneous environment) in OpenCL C-language.

OpenCL Compilation Overview

1. Read kernel code
2. Compile kernel
3. Execute kernel
How to integrate app utilizing OpenCL into AGL (1/3)

Prepare OpenCL runtime libraries(proprietaries) supporting R-Car H3 provided by Renesas

Prepare OpenCL runtime libraries

(1) Execute 1. to 10. by following the steps in R-Car/Boards/Yocto-Gen3(https://elinux.org/R-Car/Boards/Yocto-Gen3) (Details omitted)

(2) Execute the steps up to “Building Yocto images” by following the ones in R-Car/Boards/Yocto-Gen3(https://elinux.org/R-Car/Boards/Yocto-Gen3/OpenCL) (Details omitted)

(3) Install OpenCL runtime to AGL target filesystem(SDCARD)

```
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libOpenCL.so $SDCARD/usr/lib/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libPVROCL.so $SDCARD/usr/lib/
$ sudo cp -a ./build/work/h3ulcb-poky-linux-cl-gles-user-module/1.0-r0/image/usr/local/bin/ocl_unit_test $SDCARD/usr/bin/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/liboclcompiler.so $SDCARD/usr/lib/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libglslcompiler.so $SDCARD/usr/lib/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libdbm.so $SDCARD/usr/lib/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libdlc_REL.so $SDCARD/usr/lib/
$ sudo cp -a ./build/tmp/sysroots/h3ulcb/usr/lib/libufwriter.so $SDCARD/usr/lib/
```

About AGL Getting Started. See also:
http://docs.automotivelinux.org/docs/getting_started/en/dev/reference/source-code.html

ocl_unit_test can check the build of OpenCL(GPU) runtime environment. For example, if any libraries like CL compiler (liboclcompiler.so) are short, the test will result in Fail...
How to integrate app utilizing OpenCL into AGL (2/3)

Need to enable OpenCL option of OpenCV and rebuild

**Build OpenCV with OpenCL enabled**

1. **Clean-up opencv & configure**
   - In case of pre-built, clean once and execute up to config.
   - $ bitbake opencv -c clean
   - $ bitbake opencv -c configure

2. **Enable OPENCL option**
   - Enable OPENCL option in Cmake environment variable definition file (otherwise, the error occurs in bitbake)
   - $ vi tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/build/CMakeVars.txt
   - WITH_OPENCL=OFF → WITH_OPENCL=ON
   - $ vi tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/build/CMakeCache.txt
   - WITH_OPENCL:BOOL=ON → WITH_OPENCL:BOOL=ON

3. **Modify cvconfig.h**
   - Define HAVE_OPENCL in config header file (probably, either of the followings is ok)
   - $ vi tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/build/cvconfig.h
   - #define HAVE_OPENCL
   - $ vi tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/build/opencv2/cvconfig.h
   - #define HAVE_OPENCL

4. **Rebuild opencv & install**
   - $ bitbake opencv -c compile
   - $ bitbake opencv -c install
   - $ sudo cp -a build-h3-cl/tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/image/* $SDCARD/
How to integrate app utilizing OpenCL into AGL (3/3)

Modify App code in order to use OpenCL(GPU) from OpenCV

Walk through modified demo application code

```cpp
// Video(camera) capture
.....
// Pre-processing image((1) - (5)) for AI engine
// (1) Cut out video image into a rectangle [100x100pix]
cv::Rect rect(GET_VIEW_SIZE_LEFT, GET_VIEW_SIZE_TOP, GET_VIEW_SIZE_WIDTH, GET_VIEW_SIZE_HEIGHT);
cv::Mat rectImg(frame, rect);

// (2) Convert gray scale
//cv::Mat grayImg;
//cv::cvtColor(rectImg, grayImg, CV_RGB2GRAY);
//cv::UMat u_rectImg, u_grayImg;
//rectImg.copyTo(u_rectImg);
//cv::cvtColor(u_rectImg, u_grayImg, CV_RGB2GRAY);

// (3) Convert binary image
//cv::Mat binImg;
//cv::threshold(grayImg, binImg, 127, 255, cv::THRESH_BINARY_INV);
//cv::UMat u_binImg;
//cv::threshold(u_grayImg, u_binImg, 127, 255, cv::THRESH_BINARY_INV);

// (4) Resize binary image[100x100pix -> 28x28pix]
//cv::Mat resizeImg;
//cv::resize(binImg, resizeImg, cv::Size(), PGM_WIDTH/grayImg.cols, PGM_HEIGHT/grayImg.rows);
//cv::UMat u_resizeImg;
//cv::resize(u_binImg, u_resizeImg, cv::Size(), PGM_WIDTH/u_grayImg.cols, PGM_HEIGHT/u_grayImg.rows);

// (5) Add pgm header
.....
// AI engine inference with MNIST classification model
.....
```

Just replace Mat class (matrix data and data property) with UMat class!
When OpenCL(GPGPU) is available in Target environment, switch to GPU operation, otherwise, to CPU operation.
Alternatively, it is able to switch to CPU/GPU by “cv::ocl::setUseOpenCL(false/true)”. 
**Performance Verification (1/5)**

### Measured demo application

<table>
<thead>
<tr>
<th>GPU/CPU</th>
<th>Processing time (a)</th>
<th>CPU load (b)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>11 ms</td>
<td>57.8 %</td>
<td>a : UI visual check for a set of processing time in AI demo</td>
</tr>
</tbody>
</table>
| CPU     | 9 ms                | 57.2 %       | b : Average on "CPUs utilized" measured 3 times by "perf stat -p PID[AI demo] -- sleep 10"

Faster process for CPU use. No difference for CPU load.

### Breakdown of measured (Function utilized OpenCL)

<table>
<thead>
<tr>
<th>OpenCL(GPU) supporting process</th>
<th>GPU/CPU</th>
<th>Processing time</th>
<th>CPU load</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grayscale conversion[100x100pix]</td>
<td>GPU</td>
<td>1.325 ms</td>
<td>N/A</td>
<td>8 times slower for GPU use</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>0.168 ms</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Binary conversion[100x100pix]</td>
<td>GPU</td>
<td>0.440 ms</td>
<td>N/A</td>
<td>16 times slower for GPU use</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>0.028 ms</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Resize [100x100pix -&gt; 28x28pix]</td>
<td>GPU</td>
<td>0.364 ms</td>
<td>N/A</td>
<td>9 times slower for GPU use</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>0.042 ms</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

OpenCL(GPU) effect cannot be recognized for image process used in pre-developed demo. Approx 10 times slower for each supporting process.

Assumed “GPU effect cannot be worked due to the small image size (operating volume)” in pre-developed demo case. Re-verify each OpenCL supporting process by enlarging the image size (VGA/Full-HD).
Performance Verification (2/5)

Verify by measuring the processing time [ms] & CPU load [%] of 100 times target process calls. Average value on 3 times turns out result.

Example test code: cvtColor(BGR2GRAY)

```c
int main(int argc, char *argv[]) {
    [snip]
    if( false == parser.get<bool>("openCL") ) { // Not use OpenCL[use CPU] if command line param is false.
        cv::ocl::setUseOpenCL(false);
        clMode = "CPU";
    } 
    cv::Mat frame = cv::imread(parser.get<string>("input"), cv::IMREAD_COLOR); // input image data
    const int countN = parser.get<int>("countN");
    double f = 1000.0f / cv::getTickFrequency();

    // for UMat cout<"Convert gray scale."<<endl;
    int64 start = cv::getTickCount();
    cv::UMat u_grayImg, u_frame;
    frame.copyTo(u_frame);
    for(int i =0; i < countN; i++) { 
        cv::cvtColor(u_frame, u_grayImg, CV_RGB2GRAY); // Verification target 
    }
    int64 end = cv::getTickCount();
    cout << "[
    UMat""<<clMode"" <<enate) " " "f " "[ms]" " << end;
    return 0;
}
```

Build & Install to target filesystem

$ source /opt/poky-agl/5.0.0/environment-setup-aarch64-agl-linux
$ $CXX TestOpenCL_cvtColor.cpp -I$SDKTARGETSYSROOT/usr/include/opencv2 -L$SDKTARGETSYSROOT/usr/lib -lopencv_core -lopencv_imgcodecs -lopencv_imgproc -o TestOpenCL_cvtColor
$ sudo cp TestOpenCL_cvtColor $SDCARD/usr/bin/

Perform test code and measure performance

root@h3ulcb:~# perf stat TestOpenCL_cvtColor --cl=true -i=/home/data/testdata-vga.png -o=/home/data/testdata-vga-out-gpu.png
Performance Verification (3/5)

Convert to gray scale image: cvtColor(BGR2GRAY)

GPU supporting effect works for both processing time & CPU load after VGA size!
Shortened processing time by “58ms -> 36ms” and lowered CPU load by “178% -> 55%” in VGA.
GPU effects work more in Full-HD because of shortening processing time by “272ms -> 62ms” and lowering CPU load by “274% -> 28%”.

Snap shot) Case CPU / Full-HD
Performance counter stats for 'TestOpenCL_cvtColor --cl=true ..'

<table>
<thead>
<tr>
<th></th>
<th>task-clock (msec)</th>
<th>context-switches</th>
<th>cpu-migrations</th>
<th>page-faults</th>
<th>cycles</th>
<th>instructions</th>
<th>branch-misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1090.676095</td>
<td>2.733 CPUs utilized</td>
<td>0.001 M/sec</td>
<td>0.020 K/sec</td>
<td>1.496 GHz</td>
<td>2.20 insn per cycle</td>
<td>0.00% of all branches</td>
<td></td>
</tr>
</tbody>
</table>

Snap shot) Case CPU / Full-HD
Performance counter stats for 'TestOpenCL_cvtColor --cl=false ..'

<table>
<thead>
<tr>
<th></th>
<th>task-clock (msec)</th>
<th>context-switches</th>
<th>cpu-migrations</th>
<th>page-faults</th>
<th>cycles</th>
<th>instructions</th>
<th>branch-misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>211.759924</td>
<td>0.280 CPUs utilized</td>
<td>0.001 M/sec</td>
<td>0.000 K/sec</td>
<td>0.041 M/sec</td>
<td>1.496 GHz</td>
<td>0.97 insn per cycle</td>
<td>0.00% of all branches</td>
</tr>
</tbody>
</table>

Note) CPU load[%] (same for page 2 and after)
1 thread=100%
400 threads at most because of 4 threads for R-Car H3
Sampled the value in perf command “CPUs utilized” for CPU load
Convert to binary image: threshold (THRESH_BINARY_INV)

**Longer processing time but lower CPU load by “128%->85%” in VGA.**
**Shorter processing time by “62ms->35ms” and lower CPU load by “173%->106%” in Full-HD and GPU effects work for both!**

**Snap shot) Case CPU / Full-HD**
Performance counter stats for TestOpenCL_threshold --cl=false

- 315,582,294 task-clock (msec)  # 1.795 CPUs utilized
- 1130 context-switches  # 0.004 M/sec
- 10 cpu-migrations  # 0.032 K/sec
- 2742 page-faults  # 0.009 M/sec
- 46,989,407 cycles  # 1.489 GHz
- 35,348,7896 instructions  # 0.75 insn per cycle
- <not supported> branches  # 0.00% of all branches

**Snap shot) Case GPU / Full-HD**
Performance counter stats for TestOpenCL_threshold --cl=true

- 174,865,064 task-clock (msec)  # 0.701 CPUs utilized
- 304 context-switches  # 0.002 M/sec
- 1 cpu-migrations  # 0.006 K/sec
- 3771 page-faults  # 0.022 M/sec
- 26,155,8114 cycles  # 1.496 GHz
- 28,881,0600 instructions  # 1.10 insn per cycle
- <not supported> branches  # 0.00% of all branches
Performance Verification (5/5)

Resize image (Bi-linear interpolation) : resize(0.5)

Longer processing time but lower CPU load by “99%->86%” in VGA. Shorter processing time by “74ms->35ms” and lower CPU load by “170%->78%” in Full-HD and GPU effects work for both!

Snap shot) Case GPU／Full-HD
Performance counter stats for 'TestOpenCL_resize --cl=false "

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>task-clock (msec)</td>
<td>304.999219</td>
<td>1.701 CPUs utilized</td>
</tr>
<tr>
<td>context-switches</td>
<td>1300</td>
<td>0.004 M/sec</td>
</tr>
<tr>
<td>cpu-migrations</td>
<td>17</td>
<td>0.056 K/sec</td>
</tr>
<tr>
<td>page-faults</td>
<td>2353</td>
<td>0.008 M/sec</td>
</tr>
<tr>
<td>cycles</td>
<td>454090850</td>
<td>1.489 GHz</td>
</tr>
<tr>
<td>instructions</td>
<td>315853627</td>
<td>0.70 insn per cycle</td>
</tr>
<tr>
<td>branch-misses</td>
<td>1495224</td>
<td>0.00% of all branches</td>
</tr>
</tbody>
</table>

Snap shot) Case CPU／Full-HD
Performance counter stats for 'TestOpenCL_resize --cl=true "

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>task-clock (msec)</td>
<td>159.925113</td>
<td>0.753 CPUs utilized</td>
</tr>
<tr>
<td>context-switches</td>
<td>305</td>
<td>0.002 M/sec</td>
</tr>
<tr>
<td>cpu-migrations</td>
<td>1</td>
<td>0.006 K/sec</td>
</tr>
<tr>
<td>page-faults</td>
<td>3735</td>
<td>0.023 M/sec</td>
</tr>
<tr>
<td>cycles</td>
<td>239104934</td>
<td>1.495 GHz</td>
</tr>
<tr>
<td>instructions</td>
<td>254117677</td>
<td>1.06 insn per cycle</td>
</tr>
<tr>
<td>branch-misses</td>
<td>1716419</td>
<td>0.00% of all branches</td>
</tr>
</tbody>
</table>
Consideration

- **Verified OpenCL(GPU) supporting effects in R-Car H3 / AGL**
  - Effect to shorten the processing time and lower CPU load **according to process contents (adaptable algorithm)**
  - The **larger the image size (calculation amount)** is, the more the effect works (Because the image size and operating volume are small due to 100x100[pix]/8bit for image processing in pre-developed demo, it is suggested that the copy overhead to device array and the increase in CPU stall have a stronger control.)

- **Next Step**
  - Considered that it is important to determine the cases and process contents applied by OpenCL (adaptable algorithm and image size)
  - Perform the profiling and tuning (e.g. what causes CPU[ARM] stall? how is it improved?) in order to utilize CPU/GPU features at most

For example, GPU profiling

For example, GPU profiling. See also: https://community.imgtec.com/developers/powervr/tools/
Possibility verification of continuous improvement of inference accuracy for edge AI
Mechanism to enable continuous improvement of inference accuracy for edge AI

- Training dataset upload (unrecognized / low recognition rate) from edge device
- Additional training on PC via the cloud
- Pre-trained models delivery to edge device and updates
Demonstration & Verification Result
Found the possibility of **continuous improvement of inference accuracy** for edge AI

- Improved the recognition result
- For us, available to build new trained models with a little effort and time because of only additional data training (if the pre-trained models are in place)
  - First training (60000 samples): 90 sec.
  - Additional training (e.g. 13 samples): 4 sec.
- Maintained a generalization accuracy (in-house verification: about 99 %)
  - Low possibility of incorrect recognition of correctly pre-recognized data.
  - Also, high possibility of correct recognition of unknown input data.
- Enable to improve the inference results only by updating the pre-trained model (just one .npp file) on edge device
  - Expect to perform "Recognition accuracy improvement" in the background
Next Step

- What types of **practical use cases** are available?

**Future use-cases**

- Use various information of embedded devices
- Detect driver's drowsiness
- Improvement of autonomous driving technology
Questions?
Thank you very much!!

Smart Life Community®