OpenCL Implementation and Performance Verification on R-Car H3/AGL

May 25th, 2018

NTT DATA MSE Corporation
Yasumitsu Takahashi
Agenda

• Introduction to our Activities

• System Overview

• How to integrate app utilizing OpenCL into AGL

• Performance Verification
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”
  • Next Step: Try creating own neural network architecture
  • Performance improvement
    ✓ More complicated neural network architecture and graphic processing cause performance issues

In this presentation
  • Performance verification
    ✓ How to integrate app utilizing OpenCL(GPU)
    ✓ Verify the performance effect of OpenCL
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

Application
- AI engine
- NNabla
- OpenCV
- OpenCL

Trained Model

R-Car H3:
- 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

GUI: Handwritten Digit Recognition App

App utilizes OpenCL(GPU) via OpenCV
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

Host Program: OpenCV

OpenCL Runtime Libraries

OpenCL Device
GPU: PowerVR Rogue
GX6650

Look easy to learn on C-language basis. Look possible to secure the portability of source code by not depending to the specific HW PF.
How to integrate app utilizing OpenCL into AGL (1/3)

Prepare OpenCL runtime libraries(proprietary) 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/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/liboclcompiler.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/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/
Modify App code in order to use OpenCL(GPU) from OpenCV

Walk through modified demo application code

// Video(camera) capture
.....
// Pre-processing image[(1) - (5)] for AI engine
// (1) Cut out video image into a rectangle [100x100px]
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[ms] - 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>
<tr>
<td>CPU</td>
<td>9 ms</td>
<td>57.2 %</td>
<td>b : -- Average on &quot;CPUs utilized“ measured 3 times by &quot;perf stat -p PID[AI demo] sleep 10”</td>
</tr>
</tbody>
</table>

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[ms]</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<<"] " << (end - start) * f << "] ms" << endl;
    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 -lopenvc_core -lopenvc_imgcodecs -lopenvc_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=false`:

- task-clock (msec): 1090.676095, # of CPUs utilized: 2.733
- context-switches: 1106, # of M/sec: 0.001
- cpu-migrations: 22, # of K/sec: 0.020
- page-faults: 2703, # of M/sec: 0.002
- cycles: 1631814102, # of GHz: 1.496
- instructions: 3582132759, # per cycle: 2.20
- branch-misses: 1844910, # of all branches: 0.00%

Snap shot) Case GPU/Full-HD
Performance counter stats for `TestOpenCL_cvtColor --cl=true`:

- task-clock (msec): 211.759924, # of CPUs utilized: 0.280
- context-switches: 305, # of M/sec: 0.001
- cpu-migrations: 0, # of K/sec: 0.000
- page-faults: 8609, # of M/sec: 0.041
- cycles: 31688448, # of GHz: 1.496
- instructions: 306382990, # per cycle: 0.97
- branch-misses: 1984822, # of all branches: 0.00%

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)

**Processing Time [ms]**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FHD [1920×1080]</th>
<th>VGA [640×480]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>60</td>
<td>30</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Short</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

**CPU Load [%]**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FHD [1920×1080]</th>
<th>VGA [640×480]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Longer</td>
<td>150</td>
<td>100</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>Short</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**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 ...'

<table>
<thead>
<tr>
<th>Task-clock (msec)</th>
<th>Context-switches (msec)</th>
<th>CPU-migrations (msec)</th>
<th>Page-faults (cycles)</th>
<th>Instructions (instructions)</th>
<th>Branches (branch-misses)</th>
<th>Branch-misses (branch-misses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>315.582294</td>
<td>1130</td>
<td>2742</td>
<td>469894072</td>
<td>353487896</td>
<td>1320894</td>
<td>&lt;not supported&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Snap shot) Case GPU/Full-HD**

Performance counter stats for 'TestOpenCL_threshold --cl=true ...'

<table>
<thead>
<tr>
<th>Task-clock (msec)</th>
<th>Context-switches (msec)</th>
<th>CPU-migrations (msec)</th>
<th>Page-faults (cycles)</th>
<th>Instructions (instructions)</th>
<th>Branches (branch-misses)</th>
<th>Branch-misses (branch-misses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>174.865064</td>
<td>304</td>
<td>1</td>
<td>261558114</td>
<td>288810600</td>
<td>1558031</td>
<td>&lt;not supported&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2018 NTT DATA MSE Corporation
Performance Verification (5/5)

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

**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 CPU / Full-HD
Performance counter stats for ‘TestOpenCL_resize --cl=false ...’

- 304.999219 task-clock (msec)  # 1.701 CPUs utilized
- 1300 context-switches  # 0.004 M/sec
- 17 cpu-migrations  # 0.056 K/sec
- 2353 page-faults  # 0.008 M/sec
- 454090850 cycles  # 1.489 GHz
- 315853627 instructions  # 0.70 insn per cycle
- <not supported> branches  # <not supported> branch-misses  # 0.00% of all branches

Snap shot) Case GPU / Full-HD
Performance counter stats for ‘TestOpenCL_resize --cl=true ...’

- 159.925113 task-clock (msec)  # 0.753 CPUs utilized
- 305 context-switches  # 0.002 M/sec
- 1 cpu-migrations  # 0.006 K/sec
- 3735 page-faults  # 0.023 M/sec
- 239104934 cycles  # 1.495 GHz
- 254117677 instructions  # 1.06 insn per cycle
- <not supported> branches  # <not supported> branch-misses  # 0.00% of all branches

© 2018 NTT DATA MSE Corporation
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 (process volume) 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.)

Considered that it is important to determine the cases and process contents applied by OpenCL (adaptable algorithm and image size) and 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.
Thank you very much!!

Smart Life Community®
Performance Verification (Perf + Flame Graph)

Demo application Perf and Flame Graph
The image below is zoom in the core processing of demo app(Graphic pre-process + Inference). Refer to the attached file(*.svg) for details.

火焰图(aidemo_cpu.svg)
火焰图(aidemo_gpu.svg)
Performance Verification (1 time target call)

Verify by measuring the processing time[ms] & CPU load[%] of 1 time target process call. Average value on 3 times turns out result.

Convert to gray scale image: cvtColor(BGR2GRAY)

- Longer processing time but lower CPU load by “102.3%->97.4%” in VGA.
- Shorter processing time by “11.09ms->10.70ms” and lower CPU load by “104.6%->94.2%” in Full-HD and GPU effects work for both!
Performance Verification (1 time target call)

Convert to binary image : threshold(THRESH_BINARY_INV)

Longer processing time but lower CPU load by "100.9%->98.9%" in VGA.
Shorter processing time by "6.97ms->6.19ms" and lower CPU load by "102.6%->99.0%" in Full-HD and GPU effects work for both!
Performance Verification (1 time target call)

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

- **Fast**
  - Processing Time [ms]
  - CPU
  - GPU

- **Light**
  - CPU
  - GPU

Longer processing time but lower CPU load by “101.7%->99.0%” in Full-HD.
Performance Verification (Other processing of OpenCV)

Outline detection : Sobel x 1 time

Outline detection : Sobel x 100 times