# A Real-time Video Processing Implementation with Massively Parallel Computation Support

Woosuk Shin, Mingyu Kim, Sukjun Park and Nakhoon Baek
School of Computer Science and Engineering
Kyungpook National University
Daegu, Korea
oceancru@gmail.com

***Abstract*—Recently, performance of mobile device's processor has advanced dramatically. There are many researches to utilize the processor computing ability to maximize application performance. In the field of augmented reality (AR), video stream processing is an application that requires large amount of computing powers, from the real-time processing point of view. However, due to nature of mobile platforms, unnecessary overheads should be minimized to acquire higher performance. In this paper, we introduce OpenCL based video frame segmentation to minimize unnecessary overhead in video processing applications, with dynamically changing the local work group size. Our prototype system show at most 2.3 times speed-up compared to well-known, de-facto image processing library OpenCV.**

***Keywords—real-time video processing, mobile computing, massively parallel processing***

## I. INTRODUCTION

There have been many researches to enhance performance of mobile processors. It enabled high performance graphics applications to render real-time result on mobile devices. Therefore, AR or Virtual Reality (VR) environments can be implemented on a various mobile platforms. In AR applications, the application program needs to recognize surrounding environments in real-time to show meaningful information to right place. To recognize surrounding environment, AR applications usually utilizes surrounding video, which can be captured with cameras. Acquired videos are processed to extract meaningful information (pattern) from them. Then, extracted patterns can be utilized to generate marker for tracking or classification information as explained in [1] and [2]. There are many research results including [3] to find meaningful patterns (objects) from single frame of video (image) in real-time. To detect such patterns, conventional image processing algorithms, including convolution operations, are widely used.

Those convolution operations typically requires multiple random accesses the pixels of frames, causing page faults. In the case of conventional CPUs, page fault can be optimized by predicting executions, whereas mobile processors has adopted to consume less energy. Though OpenCV image processing library provides parallel versions of their algorithms, converting data types from the decoded video streams to OpenCV acceptable data types are necessary to process each frame. Also, embedding such a large library in the executable image of mobile environment can cause reduced performance of the program. Additionally, even though these convolution algorithms are implemented to utilize parallel processors (like GPU), minimizing memory access is one of the most important factors since most mobile processors have no dedicated graphics memory areas.

## II. DESIGN

As shown in the last section, reducing pre-processing and memory access is one of the key factors in optimizing video segmenting applications. Since most video decoders generate *YV12* frames, the frames should be converted to OpenCV image type. To remove unnecessary type conversions from the decoded video streams to OpenCV image types, we use the video decoder outputs as input images, through implementing dedicated frame readers and writers.

In our design, each pixel of frames will be processed a by single thread. Each thread is executed separately by parallel processing core. A typical OpenCL devices will limit its maximum number of threads that can be simultaneously launched as a work-group. Each device has its own work group size depending on vendors and chipsets. We also design dynamic work-group sizing strategy according to the input video size to support large videos in various devices. First, we need to acquire the device maximum work size $(X_{max}, Y_{max}, Z_{max})$. If we set each frame's consecutive horizontal 64 pixels, $(64, 1, 1)$ as a basic local working group. Then, we can calculate the total threads as in (1).

$$Num_{threads} = (((FrameWidth) - 1) / 64 + 1) * (FrameHeight) \quad (1)$$

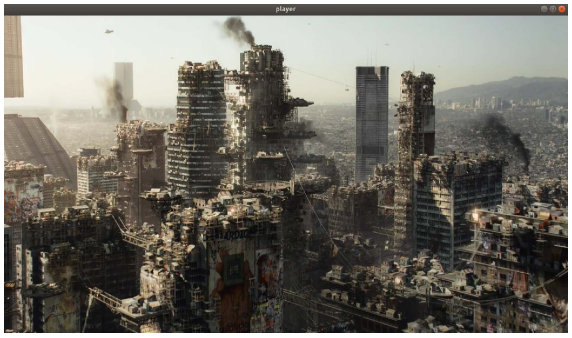
Next, we let $(Num_{threads}, 1, 1)$ as a global working group size $(x, y, z)$. If $x$ is greater than $X_{max}$, increase $y$ value by one, as subtracting 64 from $x$, until $y$ reaches $Y_{max}$. If $y$ reaches $Y_{max}$, increase $z$ by one and set $y$ to one and $x$ to $Num_{threads} / z$. Finally, we set $x$ as a multiple of 64. Through these steps, our system fully supports large size videos by dynamically resizing its work-groups, minimizing number of calls to launch (queue) work-groups.

In the memory point of view, OpenCL provides constant memory, which is faster than graphics memory (global memory). We can achieve higher performance by utilizing constant memory for small size videos. However, since we are aiming to support large size videos bigger than 16 million

pixels, constant memory can be insufficient to hold whole frames. Thus, we use global memory to read and write frames. Nonetheless, by storing each frame as textures, which is used in the graphics pipeline, we can achieve faster random accesses for the frames.

## III. Experimental Results

In our implementation, we targeted the Exynos 8890 processor as a mobile processor. The processor consists of ARM Mali-T880 GPU as the OpenCL driver. Since the processor has the maximum work-group size of *(256, 256, 256)*, it supports up to 4K resolution videos which a is standard high resolution video. Among many convolution kernels, we choose the Sobel kernel (operator) [4] to detect edges, for ease of implementation and visualization. We process each frame pixel with a single thread, by multiplying Sobel kernels to adjacent pixels. Sum of two absolute values of each gradient contains edge information of the pixel. By applying this kernel to all pixels, we can get results as shown in Fig. 1.



(a) An original frame from video stream


(b) Edge detected frame

Fig. 1. An example of our Sobel edge detection implementation

We evaluated the average frame rate per second (fps) of the edge-detection algorithm implemented with full software, OpenCV and our method (OpenCL). For input videos, we used same video encoded in different resolutions. We decoded video using the ffmpeg library, since each video is encoded in the H.264 format.

Table I shows that our method performs at most 2.3 times faster than the OpenCV-based implementation on UHD (2160p) video using Exynos 8890 processors. Our system achieved average of 1.8 times faster than OpenCV-based implementation. Additionally, for 4K video resolutions, our system shows 23 frames per second.

TABLE I. The average frame rates of different implementations

| Resolution | Frames Per Second (FPS) | | |
|---|---|---|---|
| | ***CPU (Single thread)*** | ***OpenCV*** | ***OpenCL*** |
| VGA (640 * 480) | 15 | 109 | 167 |
| SVGA (800 * 600) | 12 | 104 | 160 |
| WXGA (1280 * 720) | 4 | 63 | 114 |
| FHD (1920 * 1080) | 2 | 30 | 57 |
| UHD (3840 * 2160) | 1 | 10 | 23 |

## IV. Conclusion

In this paper, we implemented prototype real-time video processing system using massively parallel processing. To minimize unnecessary overheads on converting image formats, our system utilize video decoder output frame directly. We also designed a method to dynamically adjust work-group sizes for mobile devices. Our prototype implementation shows that it out-performs on mobile processors, up to 4K resolution videos. In the future, we are aiming to use CPU and GPU simultaneously by using dynamic work-load balancing to get better performance.

## Acknowledgment

This work has supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grand No. NRF-2019R1I1A3A01061310). This work was also supported by the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (21A20131600005).

## References

[1] J. Dim, and T. Takamura, "Alternative Approach for Satellite Cloud Classification: Edge Gradient Application," Advances in Meteorology, vol. 2013, pp. 1-8, December 2013.

[2] M. Hirzer, "Marker detection for augmented reality applications," Technical Report, Computer graphics & vision, Graz University of Technology, October 2008.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp.779-788.

[4] C. I. Gonzalez, P. Melin, J. R. Castro, and O. Castillo, Edge Detection Methods Based on Generalized Type-2 Fuzzy Logic, 1st ed. Springer, 2017, pp.11-16