Parallel implementation of Multi-view Video Decoder for Reduction in Power Consumption

Xiang Jun Zhao
Dept. of Electronic, Information and Communication Engineering,
Konkuk University

Yeon-Man Jeong
Dept. of Information and Telecommunication,
Gangneung-Wonju National University

Yong Beom Cho
Dept. of Electronic, Information and Communication Engineering,
Konkuk University

ABSTRACT
Mobile platform based multi view video applications have gained significant attention due to increase in processing power of mobile processors. Significant performance improvements have been reported by using H.264/AVC based video encoding-decoding procedures. Multi-view coding (MVC) is an extension of H.264/AVC scheme employed for high performance compression of multi view videos. The increase in spatial/temporal information has increased the difficulties in real-time decoding. In existing implementations, multi-processor based architectures have been employed to achieve real-time processing in H.264/AVC. In this paper, a parallel MVC decoding scheme is presented. The implementation is based on pipelined architecture of Cortex-A8 processor and graphics processing unit (GPU). The decoding procedure is divided into block based and pixel based operations. The GPU is employed for processing pixel based operations while the CPU performs the block based sequential operations. The implementation results shows that the proposed scheme can achieve same peak signal-to-noise ratio (PSNR) performance with more than 29% increase in decoding speed. Hence, the proposed scheme can achieve real-time performance and can be used in mobile platform multi-video processing applications.

General Terms
Speed-up, power, architecture

Keywords
GPU (graphics processing unit), Multi-view Video coding Decoder, motion compensation parallel architecture, power estimation

1. INTRODUCTION
Next generation display units have allowed the display of 3D video content with optimal display complexity and reduced power consumption. On the other hand, the processing of these video contents causes significant implementation and power consumption concerns. Multi-view Video Coding (MVC) [1] has been proposed to achieve real time decoding capability enabling its usage in 3D television (3DTV) [2] and FVV (free viewpoint video) [3]. Although MVC achieves 20-50% reduction in bit rate as compared to existing systems, the increase in computational complexity is its major bottleneck. Employing implementation techniques such as Signal Instruction Multiple Data (SIMD) extensions, various architectures have been proposed. However, due to large area requirement and significant power consumption these architectures cannot be employed in energy sensitive mobile applications.

Graphics processing unit (GPU) based architectures have been proposed to achieve real-time decoding for single view point video schemes [4][5][6]. It has been shown that the single view-point data encoding and decoding process can be segregated in to block based operations and per-pixel operations. The pipelined architecture between CPU and GPU can be employed for pre-fetching the block processing operations in CPU, while the GPU performs the per-pixel operations. The work load sharing is made possible due to temporal co-relation in video frames.

In this paper, the MVC decoding architecture based on parallel processing in CPU and GPU is presented. The temporal and spatial co-relation between the frames in multi-view stream is used for deciding the work-load sharing between the CPU and GPU. The cortex-A8 and GPU based architecture for mobile platform is used for implementation. The simulation results show that by using the proposed architecture, the decoding speed is increased by more than 29% while the power consumption is reduced by approximately 20%.

In the following sections a brief discussion on general architecture of MVC decoder is presented. In section 3, the proposed GPU based parallel architecture for MVC decoder is presented. Experimental results and analysis are presented in section 4, followed by conclusion.

2. MOTION COMPENSATION IN MVC
Fig .1 shows the typical motion compensated video decoder of MVC, which consists of entropy decoding, inverse quantization (IQ), inverse discrete cosine transform (IDCT), in-loop de-blocking filter, intra prediction and motion compensation (MC).

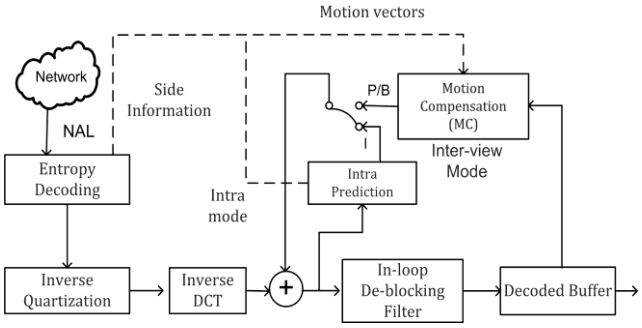

Fig 1: MVC decoder block diagram

MC uses the motion vectors (MV), generated by reconstructed intra or inter pictures, to form a prediction frame. The motion vector establishes the temporal and spatial correlation between neighboring frames of video sequences. The de-blocking filer is employed to remove the transition effects visible at the edges of macro blocks. The residual signal from the IDCT is added to the motion compensated prediction to form a decoded picture. This feedback loop determines the bit-rate and the complexity of the decoder.

Table 1 shows the profiling results of MVC decoder running on Cortex-A8 processor. It can be seen that the motion compensation is most computation extensive operation and hence, constitutes a significant portion of decoding time.

Table 1. MVC decoder profiling of building block

Block	(%)
Entropy Decoding	9
Motion Compensation (MC)	30
De-blocking Filter	11
iDCT	23
IQ & Intra Prediction	17
Others	10

The processes in MVC can be classified as those which are performed with in a block, like DCT/IDCT, and those which are performed on all the pixels identically, like computation of MV and MC. In [4], it was established that direct mapping of video decoding in 2D and 3D graphics engines is not possible. It was shown that since the graphics processors are designed to handle edge and vertex based information, it will be more prudent to perform per-pixel based operations in GPU. Similarly, the block based operations require sequential processing and hence, must be performed in CPU.

3. PROPOSED ARCHITECTURE

3.1 Video coding on programmable GPU

The GPU uses the position information in 3D space through vertices and triangles. While three edges constitute vertices, a combination of triangle forms a quad. The information of vertices is used to from a projection on a plane which is then converted to pixel based raster. The color information is obtained either by calculation, or from look-up memory. The resultant is either sent to display or rendered in to a texture for use in further processing.

The process of rasterization and vertex mapping is performed by executing small subroutines called shaders on GPU [8]. Vertex transformation and pixel rasterization are done by vertex and pixel shaders respectively. The advantages of these shaders are that they enable custom calculations to be performed on pixels, thereby allowing the implementation of non-graphic applications on GPU. The interaction of CPU subroutines and shaders determine the work-load shared between the CPU and GPU.

Fig. 2. shows the programmable Graphics Pipeline, in which the vertex and fragment processing (rasterization) is broken out into programmable units. The 3D data or coordinated are projected on the screen along with the color information in the vertex stage, while the color of each texture is computed in the fragment stage. By programming the per-pixel operation of video decoding in per-vertex shader, operations like target positioning from motion vector can be performed. Using the texture based operations the motion compensated perdition of image can be formed. Hence, MV based computation and thereby MC computation can be performed in GPU.

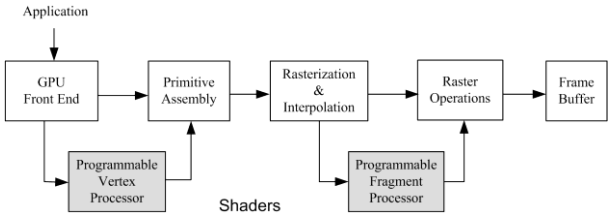

Fig 2: Programmable GPU pipeline architecture

3.2 Proposed MVC decoder architecture

In [4], it was shown that for single view-point videos stream the MC is performed in GPU. The shader modules were used for the MC while the block based operations like IDCT were performed in the CPU.

In terms of multi-view video, the frames are correlated both in terms of space and time. However, the frames are delivered in sequential group of pictures (GOP). The pixel and fragment shaders can be employed for performing the MV based estimation and MC for a given view angle. The shaders can retain the reference frame information in sub-routine memory to perform the MV based computation across various view-points.

Fig. 3 shows the proposed parallel architecture that consist CPU and GPU.

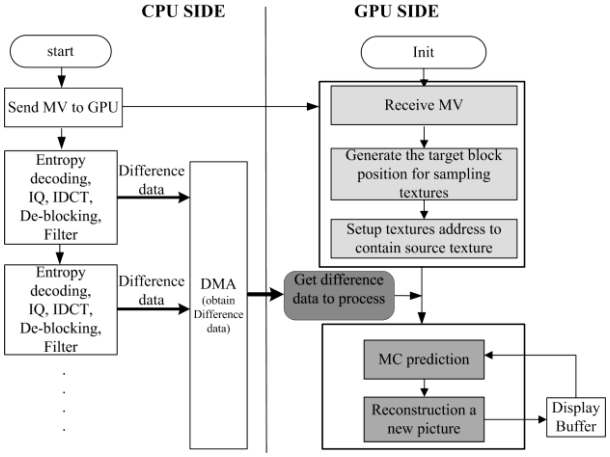

Fig 3: Optimized parallel MVC decoder model

The GPU is connected as co-processing element initiated by the MV computation unit in CPU. The incoming bit stream is process by the CPU to generate the MV. The MV is then passed on to GPU which generated the motion compensated prediction of image. The CPU and GPU are pipelined to exploit the parallelism between them. During the process of motion compensated image perdition the CPU uses the frame data to perform block based operations such as IDCT. The residual information is passed on to GPU through DMA for generating the decoded frame. To optimize the work-load sharing between the CPU and GPU, the buffer size for storing CPU intermediate result is kept to retain at least 4-5 frames. Hence, while the GPU computes the motion compensated prediction, the CPU computes the intermediate date essential

for generating the decoded frame. To co-ordinate the data dependencies DMA is employed as the intermediate buffer between the ARM core and GPU. In the proposed scheme, the intermediate DMA buffer is also used to store the frame information between the different view-points.

3.3 Motion compensation on the GPU

As shown in Fig. 3, the MC computation is initiated after storing the intermediate results to DMA [9]. The process of motion compensation is divided in to vertex processor (vertex shader) and fragment processor (pixel shader), to divide the work-load. The motion vectors are transferred directly to vertex shader. The texture coordinates of vertices or pixels are translated by vertex processor based on motion vector values. The texture information is then used by the pixel shader to sample the texture information available in GPU frame buffer and compute the motion compensated predication and render the target texture in desired position.

The motion vector information for each macro block is independent of each other. However, same motion vector information is used for all the pixels in a macro-block. The predicted position of the macro block is determined only by the four vertices and the operation for each vertex of pixel is same. This is also based on the motion translational model being employed for motion compensation. For every macroblock to be motion-compensated, the target block positions and the source (reference) texture addresses is computed by vertex shader. MC can be performed at different data precisions: integer pixel level, subpixel level, half-pel and quarter-pel. To handle different MC types and resolutions, divide-and-conquer method is employed similar to [4]. Blocks belong to the same category are grouped and processed together.

Fig. 4 shows the functions performed by pixel shader. The pixel shader itself consists of various subroutines. The pixel shader 1 (PS1) prepares the padded and the decomposed reference texture (T2) from the reference texture T. The texture T2 is then used by next pixel shader, PS2, for motion compensation process. The difference texture data, T0, obtained from CPU is added to motion compensated texture, T3, by pixel shader 3, PS3.The pixel shaders can also be employed for obtaining the frame reference information to process the multi-view frames.

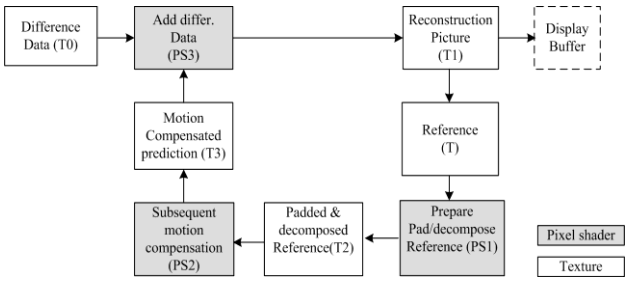

Fig 4: Data flow of pixel shaders and textures

4. EXPERIMENTAL RESULTS

The proposed design for MVC decoder is based on JMVC 8.2 which is the extension of H.264/AVC standard with multi-view high profile. The design was used to perform extensive test on embedded system with Samsung S5PC100 consisting of the Cortex-A8 based CPU and the 3D Accelerator. Intermediate buffer with 4-frames was used to achieve optimum work load sharing between CPU and GPU thereby, increasing the processing speed. For testing the proposed parallel architecture, we used YUV420 sequence in QCIF (176×144) resolution from slow to high motion. The testing sequences employed are Akiyo (slow moving, simple), container (medium moving) and Coastguard (fast moving, detail). Each sequence has at least 240 frames for each view and frame rate is set as 30 frames per second.

Table 2 shows that the comparison of proposed MVC decoding with the original decoding scheme. By comparing the frames (decoded)-per-second, it can be seen that the proposed decoding increases the decoding speed by more than 29 % for various view-point videos.

Fig. 5 shows the relation between bit-rate and PSNR [9] under different quantization parameters (22, 26, 30, 36) of Akiyo file. It is observed that with proposed MVC decoder architecture, MVC decoder maintains the same video quality while achieves a better bit-rate.

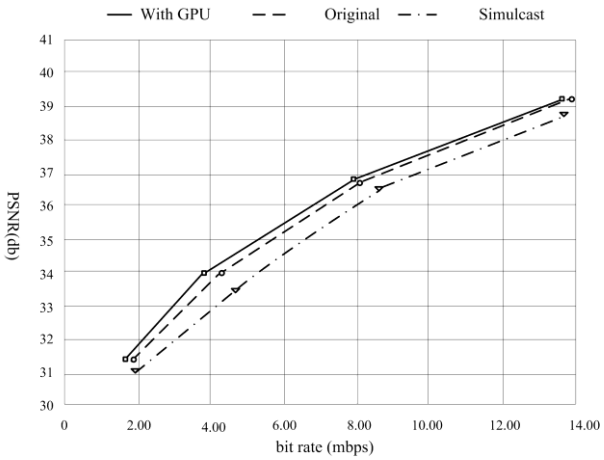

Fig 5: PSNR/bit rate chart of Akiyo file

To measure the power consumption of the multi-view video decoder that includes CPU and GPU core, the LabVIEW Tool (Measurement and Automation Explorer) was employed. Fig. 6 shows the comparison of power consumption in CPU based and proposed CPU+GPU based architecture. Due to work load sharing and reduced computation time, the power consumption in the proposed scheme is reduced by 20-25% as compared to existing decoder working on CPU only.

5. CONCLUSION

In this paper a parallel decoding methodology for multi-view coding is presented. The proposed scheme employs the pipelined CPU-GPU architecture to share the decoding computations. The decoding process is segregated in to independent processes, which run in parallel across CPU and GPU. The proposed scheme was implemented in CPU-GPU environment for mobile platform. The implementation results show that the proposed scheme can achieve same decoding performance in terms of quality with improved decoding speed and reduced power consumption. The optimization of speed and the power consumption enables the use of proposed scheme in energy sensitive mobile applications.

Table 2. Comparison of our parallel architecture to general architecture

Test Sequence	Views #	Original			With GPU			Speed-up
		Speed (fps)	PSNR (dB)	Bitrate (kb/s)	Speed (fps)	PSNR (dB)	Bitrate (kb/s)	
Akiyo (Slow moving, Simple)	2	23.21	36.28	124.40	30.16	35.75	122.40	1.31
	3	22.01	38.85	123.90	28.87	37.95	122.23	1.33
	4	20.37	34.34	119.73	27.02	34.20	118.12	1.32
Container (Medium moving)	2	22.50	37.15	127.02	29.05	35.12	123.50	1.29
	3	20.81	36.27	120.35	27.47	37.45	120.06	1.32
	4	19.89	34.52	118.86	25.86	33.56	116.89	1.30
Coastguard (Fast moving, detail)	2	20.72	36.87	122.37	27.76	36.00	122.37	1.34
	3	19.95	34.51	118.45	26.54	33.02	117.62	1.33
	4	19.10	36.20	123.39	25.09	36.19	123.08	1.30

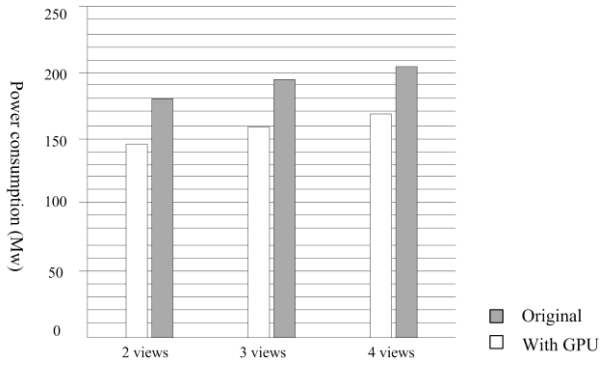

Fig 6: Power consumption of proposed architecture with general architecture

6. ACKNOWLEDGEMENT

This research was supported by the ETRI SW-SoC R&BD Center.

7. REFERENCES

[1] Y.Fu, Y.Guo, "Multi-View Video Summarization", IEEE Trans on Multimedia, Vol. 12, No. 7, Nov. 2010.

[2] N. A. Dodgson, "Autostereoscopic 3D Displays", IEEE Computer, Vol. 38, No. 8, pp.31-36, 2005.

[3] M. Tanimoto, "FTV (Free Viewpoint Television) Creating Ray-Based Image Engineering", International Conference on Image Processing, Vol. 2, pp.25-28, 2005.

[4] Guobin Shen, Guang-Ping Gao, Shipeng Li, Heung-Yeung Shum, and Ya-Qin Zhang, "Accelerate Video Decoding With Generic GPU", IEEE Transactions on circuits and systems for video technology, Vol. 15, No. 5, Pg. 685-693, May 2005.

[5] K. Mohiuddin, P. J Narayanan, "A GPU-Assisted Personal Video Organizing System,"2011 international conference, 2011.

[6] R. R. S´anchez, J. L. Mart´ınez,,"A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding," 2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.

[7] P. Colantoni, N. Boukala, and J.D. Runga, "Fast and accurate color image processing using 3-D graphics cards," Proc. of 8th of Germany Int. Fall Vision Modeling and Visualization Workshop.

[8] R. Fernando and M.J. Kilgard, "The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics." Addison-Wesley Longman Publishing, 2003.

[9] M. Karaorman, G. Dang, Ezell Young, Using DMA with Framework Components for 'C64x+, Texas instruments, 2007.