GPU-based JPEG2000 Decoding Scheme for Digital Cinema

Jeong-Woo LEE, Bumho KIM, Jungsoo LEE, and Ki-Song YOON
ETRI(Electronics and Telecommunications Research Institute), Korea
jeongwoo@etri.re.kr, mots@etri.re.kr, jslee2365@etri.re.kr, ksyoon@etri.re.kr

***Abstract*—For the digital cinema system specification released by Digital Cinema Initiatives, it was decided to use 2K or 4K images encoded by the JPEG2000 standard. JPEG2000 provides high compression rates and error tolerance, but it is a burden for both encoding and decoding. To improve the decoding performance, a parallel computing architecture called CUDA has been receiving a lot of attention recently. In this paper, we attempt to realize a real-time JPEG2000 decoding scheme for digital cinema using multiple CPU cores and GPUs. We present CUDA algorithms that perform inverse quantization, inverse 2D discrete wavelet transform and inverse irreversible color transform on a CUDA device, which brings us significant performance gain on a general CPU without extra cost.**

***Keywords*—JPEG2000, CUDA, Digital Cinema, GPU, Parallel Processing, 4K, coalesced memory access**

I. INTRODUCTION

Digital Cinema Initiatives (DCI) established the specifications of Digital Cinema [1], in which the quality of the digital playback and display of a feature film is commensurate with that of 35 mm film release prints. The specifications require images with a 4,096 x 2,160 pixel resolution at 24 fps, or a 2,048 x 1,080 pixel resolution at 48 or 24 fps, the pixel arrays of which are called 4K and 2K, respectively. To maintain the quality of the master data, a visually lossless encoder is required; JPEG2000 is therefore employed to compress the images.

Compared with a traditional codec like JPEG [2], which uses the discrete cosine transform as an orthogonal transform, JPEG2000 adapts the wavelet transform, which considers the temporal locality of the signals [3], [4]. JPEG2000 divides an image into tiles, and processes the tiles independently. For this reason, JPEG2000 has higher compression rates and error tolerances than conventional codecs. Since JPEG2000 requires massive processing, however, it seems impossible to realize its real-time software codec for high-resolution images such as 4K on a general CPU.

Graphics Processing Units (GPUs), which include some multi-processor units, have taken over the graphic processing in current computer architectures. In the past, GPUs were only used for accelerating the production of a rendered image [5]. As GPUs are rapidly developing, however, they are being steadily used in various fields [6]. In this paper, we attempt to take advantage of the GPU processing with the CUDA model to improve the performance of the JPEG2000 encoding algorithm for digital cinema.

The rest of this paper organized as follows. In section II, we describe the use of JPEG2000 as a compression method within the DCI specifications. Section III introduces the architecture of GPUs and CUDA. Section IV expresses our proposed GPU-accelerated JPEG2000 decoding method. Simulation results are presented to evaluate the proposed method in section V, and finally, some concluding remarks are given in section VI.

II. JPEG2000 FOR DIGITAL CINEMA SYSTEM

The block diagram of a digital cinema system is illustrated in Figure 1 [7]. As depicted in the figure, a digital cinema system can be divided into four stages: mastering, distribution, playback, and projection. At the mastering stage, the movie is compressed, encrypted, and packaged for delivery to theaters. The data are then distributed to the exhibition site, where they are decrypted, uncompressed, and played back. The encryption and decryption are optional and may be skipped.

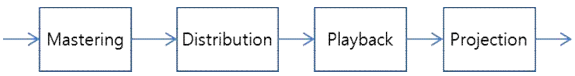

Figure 1. Digital cinema system

The DCI standard defines the size of each cinema frame to be as large as 4,096 x 2,160 pixels. Since large sizes make the distribution of uncompressed data impractical, as mentioned before, the DCI selected JPEG2000 as the compression format. There are two JPEG2000 profiles for digital cinema. The 2K digital cinema profile describes the parameter set for a 2K DCP, whereas the 4K digital cinema profile describes the parameters for a 4K DCP.

The output of the digital cinema post-production process is referred to as the Digital Cinema Distribution Master (DCDM). The image data in the DCDM are compressed using JPEG2000. There are two image structures defined in DCDM: 2K resolution (up to 2160x1080 pixels) and 4K resolution (up to 4096x2160 pixels). The bit depth of each color component is 12 bits. The frame rate is set to 24 Hz. In addition, a frame rate of 48 Hz is also allowed for 2K content.

The DCDM is required to use a common standardized file format for each element. The DCDM image file format is mapped into TIFF format. In reality, however, the DPX and

Cineon formats should be supported for the DCDM image file format. In general, the bit depth of the color component in DPX is 10 bits. In addition, the maximum number of horizontal and vertical pixels shall be constrained to fit within either a 2K or 4K resolution. For example, an image pixel size of 2,048 x 858 and 1,920 x 1,080 is possible. However, an image pixel size of 1,920 x 858 is not.

III. Parallel Processing

General-purpose computations on a GPU are generally conducted during numerical simulations. General-purpose computing on graphics processing units is the utilization of a graphics processing unit (GPU) to perform computation in applications traditionally handled by the central processing unit (CPU). CUDA was offered from NVIDIA to program along GPGPU principles [8]. A function executed on a GPU is called a kernel function, and we therefore need to program the kernel function and data transfer injunction between the CPUs and GPUs to parallelize the computing.

A thread is a processing unit, and is programed by developers using CUDA. The developers need to designate the number of threads, and the threads are managed in a hierarchical structure called a grid and block. A grid principally corresponds to a GPU device. Starting from CUDA compute capability 2.0, the maximal dimension of the grid is 65,535 x 65,535 x 65535 blocks and the gpu can hold up to 65536 x 65536 x 65536 kernels. The number may be even higher with compute capability 3.x. In the last CUDA version, the maximum number of threads per block is 1024. To operate these threads efficiently, the developers also have to maximize the number of threads within the restriction of the GPUs.

The GPU calculations are processed in a warp having 32 threads. Hence, the number of parallel calculations should be designed in multiples of 32. Note that a warp is applied within the same SP. CUDA has some memory types that match the hardware memories. Global memory can be accessed from all threads, but the reference time is slow. The capacity of global memory is several GBytes. At the first part of program, the host (CPU) must transfer the data to global memory to process on the GPUs. Only threads within a same block access shared memory, and the reference time is about 200-times faster than global memory. Thus, shared memory must be used to improve the performance as soon as possible. However, the size of the shared memory is small and restricted. Consequently, all data are transferred to global memory, and the smaller data, which are processed by each block, are only deployed in shared memory. It should be emphasized that we have to use these memories in a suitable way for efficient calculations.

A stream function permits the GPUs to manage a single kernel function and single data transfer pair simultaneously. Using a stream function, the data transfer of the second stream starts just after the data transfer of the first stream is finished; the kernel function of the first stream and the data transfer of the second stream then work at the same time.

The global memory bandwidth is used most efficiently when simultaneous memory access by threads with half of the warp size is guaranteed. That is, the global memory access by 16 threads is coalesced into a single memory transaction as soon as the words are accessed by all threads. It should be noted that we should maintain coalesced global memory access for reads and stores to improve the performance in the GPU kernels.

IV. GPU-accelerated JPEG2000 decoding method

Load balancing between the CPU and the GPU is a key performance factor for the JPEG2000 decoding. Here, we decide which part of the work will be done in the GPU and which part will be left to the CPU.

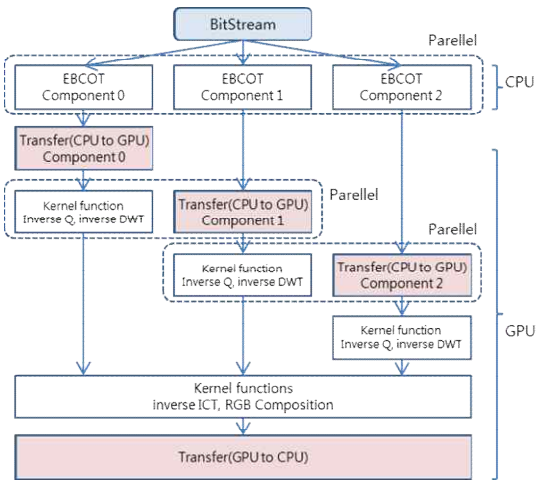

Figure 2. JPEG2000 Decoding in GPU and CPU

We assign the EBCOT task to the CPU and the other parts to the GPU as shown in Figure 2. Each process of EBCOT for component 0 to component2, and band 0 to band 2, is performed simultaneously. As mentioned before, a stream function can permit the GPUs to manage a one kernel function and one data transfer part simultaneously. Using the stream function, the data transfer of the second stream starts just after the data transfer of the first stream is finished; the kernel function of the first stream and the data transfer of the second stream thus work at the same time. In the inverse ICT kernel function, however, synchronization for the threads must be performed because it uses the data in the other components. It should be noted that the functions on the GPU and CPU are performed independently.

A. Inverse Quantization

After the EBCOT, all of the resulting subbands are dequantized. The inverse quantization step size can be chosen differently for every subband. That is, the inverse quantization kernel function must be applied to each subband separately. As a result, the inverse quantization process has no coalescent loads or stores at all in some cases.

B. Inverse DWT

The intensive computation of DWT from multilevel filtering/down-sampling, which is called the filter bank scheme (FBS), does not introduce a serious problem when the data size is small. However, this will become a significant bottleneck in real-time applications when the data size is large. Consequently, an efficient implementation of DWT operated on CPU, known as a lifting scheme (LS), was proposed [9]. In this paper, the lifting-based DWT algorithm is used to calculate the inverse 2D-DWT. For optimization, shared memory was also used, and the coalesced memory access was maintained as much as possible.

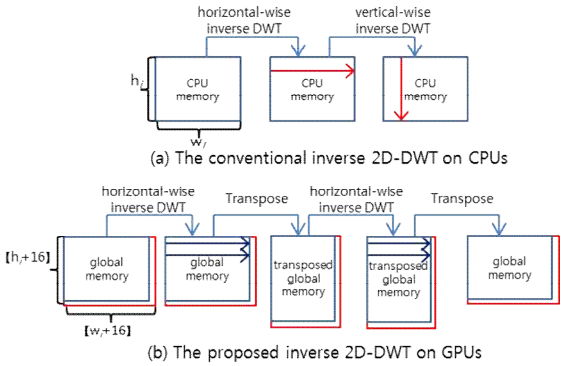

Figure 3. Inverse 2D-DWT on CPUs and GPUs

Figure 3 shows the inverse 2D-DWT method used on CPUs or GPUs. Figure 3(a) shows the conventional inverse 2D-DWT algorithm operated on CPUs. The parameters w_i and h_i represent the width and height of an image at the i^{th} level of approximation, respectively. To calculate the inverse 2D-DWT for an image, the lifting processes should be performed for all levels between 0 and 4. After the whole horizontal line is loaded into shared memory and the lifting steps for the horizontal line are first performed, the whole vertical line is loaded into shared memory, and the lifting steps for the vertical line are performed. Finally, all data are stored back into global memory. In this case, the vertical transform has no coalescent reads and writes at all, because the successive pixels in one column are transferred into shared memory, which degrades the performance significantly.

To avoid this problem, the transposed matrix has been used as shown in Fig. 10(b). The lifting-based inverse wavelet transform is applied to each horizontal line of the image. Two different lifting kernels were identified in this method: a lifting-based wavelet transform and a matrix transposition. The horizontal block size should be a multiple of 16, such that coalesced access is not broken by a thread block misalignment. As mentioned before, however, a 2K image is decomposed from 0 to 4 levels. As a result, the width and height of an image at each level may not be a multiple of 16. Therefore, the kernel function for the transposing matrix has no coalescent reads and stores in certain blocks. To support the coalesced access for reads and writes in the transpose kernel, we utilize another global memory for the transposed matrix, which stores the result of the transpose kernel. The width and height of the global memory are set to the multiples of 16. The coalesced memory access for the reads and writes is achieved because of the sequentially computation order. In addition, there is no conditional statement for checking the boundary of the data compared to the matrix transpose example from the CUDA development kit.

Lifting-based implementation is very suitable for GPU processing because every calculation on one line depends only on the lines above. On the other hand, it seems to be necessary to have whole samples in one single thread block, because synchronization is necessary after each lifting step. To adapt the four lifting steps to the subband of each layer, the samples in a subband must be joined as shown in Figure 4. The join and lifting processes are also performed in one kernel.

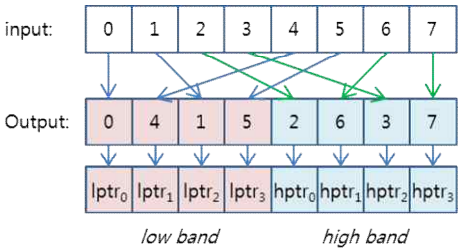

Figure 4. Join Operation

C. *Inverse ICT and Clipping*

For optimization, coalesced memory access should also be achieved during this process. The sequential computation order allows a coalesced memory access, that means, thread one processes pixel one, thread two processes the pixel that is saved directly after pixel one, and so forth.

The inverse irreversible color transform and clipping process are both also realized in one kernel. To take advantage of the parallelization, every pixel has its own thread. Even if the x-axis length of the image is less than the multiples of the number of threads, the read and store processes satisfy the coalesced access because the length of the x-axis is extended to the value of stride_x.

V. Simulation Results

To evaluate the performance of the proposed algorithms, we consider images with 2K and 4K resolution for DCI JPEG2000 profiles. The proposed schemes are implemented in the reference software, called "JasPer" [10], which is defined in Part 5 of the JPEG2000 standard.

TABLE 1. Test Environments

Options	Values
Processor	W5590 x 2
Cores	8 Cores
Clock frequency	3.33GHz
GPU	NVIDIA GTX680 x 2
CUDA version	5.0

Table 1 summarizes the hardware and software parameters of our evaluation environment and their corresponding values. In our experiments, we used two CPUs with an Intel Xeon

w5590 at 3.33GHz. The GPU platform used for evaluation purposes was an NVIDIA GTX 680. For the GPU implementation, we used CUDA as the development environment.

Table 2 shows the encoding time of Jasper and the proposed algorithm for the digital cinema profile. As shown in Table 2, the proposed method is about 20 times faster the reference software. The reference software fully used dual CPU with 8 cores. Otherwise, the proposed algorithm just used single GPU.

TABLE 2. DECODING TIME OF JASPER AND THE PROPOSED ALGORITHM (SEC)

Image Size	Jasper	Proposed Algorithm
2048x1080 (2K Profile)	0.6414	0.0163
4096x2160 (4K Profile)	2.5974	0.0307

Table 3 shows the proposed JPEG2000 decoding performance for the cinema 4K profile. We used Intel Xeon 2-way CPUs and two GPUs. As shown in Table 3, the proposed method supports 30 fps. Our method can be adapted to a digital cinema decoding server, whose minimum frame rate is at least 24fps.

TABLE 3. PERFORMANCE THE PROPOSED ALGORITHM (FPS)

Decoder #	1	2	4	6	8	10	12
fps	10.1	16.4	18.5	27	31	32.5	31

VI. CONCLUSION

CUDA is a new hardware and software architecture for issuing and managing computations on a GPU. It promises to simplify the development of applications that take full advantage of current and future powerful GPUs. In this paper we described the development of a GPU implementation of JPEG2000 decoding, intended for supporting digital cinema service. Specifically, we proposed a new method to maintain the coalesced global memory access.

The real-time JPEG2000 decoding for large data sizes such as 4K (4,096 x 2,160) and 8K (8,192 x 4,320) are not possible because the global and shared memories of the GPUs are limited. In this paper, we have therefore proposed a new JPEG2000 algorithm that considers GPUs and multi-CPUs. Compared with other digital cinema solutions, our JPEG2000 solution also provides high-speed mastering and playback service.

ACKNOWLEDGMENT

This work was supported by MSIP (Ministry of Science, ICT and Future Planning) (10041539 High Compression, Low Loss Content Creation /Distribution /Display Technology Development for 8K-Video Service).

REFERENCES

[1] DCI Std., Digital Cinema System Specification ver. 1.2, DCI, 2008.
[2] G. K. Wallace, “The JPEG still picture compression standard,” IEEE Trans. Consum. Electron., vol. 38, no. 1, pp. 18–34, Feb. 1992.
[3] ISO/IEC 15 444-1: Information Technology—JPEG 2000 Image Coding System—Part 1: Core Coding System, 2000.
[4] C. Christopoulos, A. Skodras, and T. Ebrahimi, “The JPEG2000 stillimage coding system: an overview,” IEEE Trans. Consum. Electron., vol. 46, no. 4, pp. 1103–1127, Nov. 2000.
[5] D. Ko, J. Lee, S. Lim, et al., "Construction and Rendering of Trimmed Blending Surfaces with Sharp Features on a GPU," ETRI Journal, vol. 33, no.1, Feb. 2011, pp. 89-98.
[6] M. Ciznicki, K. Kurowski, and A. Plaza, “Graphics processing unit implementation of JPEG2000 for hyperspectral image compression,” Journal of Applied Remote Sensing, vol. 6, 2012.
[7] A. Bilgin and M. Marcellin, “JPEG2000 for Digital Cinema,” SMPTE Motion Imaging Journal, vol. 114, pp. 202-209, 2005.
[8] J. Sanders and E.Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley, 2011
[9] W. Sweldens, “The lifting scheme: A construction of second generation wavelets,” SIAM Journal on Mathematical Analysis, vol. 29, no. 2, pp. 511-546, 1998.
[10] M. D. Adams and F. Kossentini, “JasPer: a software-based JPEG-2000codec implementation,” in Proc. IEEE Int. Conf. Image Processing, vol. 2, Oct. 2000, pp. 53–56.

Jeong-Woo Lee received the B.S. degree in information and telecommunication engineering from Jeonbuk National University, Jeonju, Korea, in 1996, and the M.S. degree in information and communications engineering from Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, in 1998. He received the Ph.D. degree in the Information and Communications Department from GIST in 2003. He is currently working in Electronics and Telecommunications Research Institute (ETRI). His research interests include digital video coding algorithms, implementations for H.264 and HEVC, rate control algorithms for video coding, scalable video compression, and gpu-based coding algorithms.

Bumho Kim received the BS degree in computer science from Sogang University in 2000 and MS degree in information technology from Information Communication University in 2002, respectively. Currently, he is a senior researcher in the Creative Content Research Lab. at ETRI, Daejeon, Korea. His research interests include multimedia, video codec, digital cinema, and digital contents distribution.

Jungsoo Lee received his B.S. and M.S. degrees from Jeonbuk University, Korea in 1995 and 1997, respectively and his Ph.D. degree in Electronic Engineering from Hanyang University, Seoul Korea in 2005. From 2000 to 2005, he was a senior member of MarkAny Research Institute. Currently, he is a senior member of Electronics and Telecommunications Research Institute(ETRI). His research interests are digital watermarking, fingerprinting, image processing, digital rights management, digital cinema and digital signage.

Ki-Song Yoon received his M.S. and Ph.D. degrees in Computer Science from New York City University in 1988 and 1993 respectively. From 1993, he was a principal member of Electronics and Telecommunications Research Institute (ETRI). His research interests are digital contents distribution, digital rights management and digital cinema/signage.