CUDA-based JPEG2000 Encoding Scheme

Jeong-Woo LEE, Bumho KIM, and Ki-Song YOON

ETRI (Electronics and Telecommunications Research Institute), Korea

jeongwoo@etri.re.kr, mots@etri.re.kr, ksyoon@etri.re.kr

***Abstract*—JPEG2000 is the international standard for image compression. The rich feature set and the state of the art image compression performance make JPEG2000 an attractive alternative for many applications. Especially JPEG2000 is used in the area for digital cinema and medical image. Although the JPEG2000 provides high compression rates and error tolerance, it is burden for both encoding and decoding. To improve the 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 encoding scheme by using GPUs. We present CUDA algorithms that perform DCDM decomposition, multi-component transform, 2D discrete wavelet transform, and quantization completely on a CUDA device, which brings us significant performance gain on a general CPU without extra cost. In addition, we present CUDA algorithm for performing the color conversion from RGB to XYZ.**

***Keywords*—JPEG2000, CUDA, GPU, Parallel Processing, NVIDIA**

I. Introduction

JPEG2000 [1] is the latest international image compression standard created by a joint committee of ISO/IEC and ITU. JPEG2000 was developed to address the needs of many applications through its wide set of features. In the summer of 2004, Digital Cinema Initiatives (DCI) selected JPEG2000 as the compression format to be used for digital distribution of motion pictures.

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]. 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 [4]. As GPUs are rapidly developing, however, they are being steadily used in various fields. It should be clear that GPUs are designed as numeric computing engines, and will not perform well for certain tasks where CPUs are better designed; therefore, one should expect that most applications will use both CPUs and GPUs, executing the sequential parts on the CPUs and numerically intensive parts on the GPUs. When adapting parallel arithmetic to very large image processes, the higher resolution an image is at, the more the GPUs contribute to high-speed processing. CUDA was offered from NVIDIA to program along GPGPU principles [5]. 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.

The rest of this paper organized as follows. In section II, we present a brief overview of JPEG2000 encoding system. Section III introduces the architecture of GPUs and CUDA. Section IV expresses our proposed GPU-accelerated JPEG2000 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 Encoding Process

Figure 1 shows the process outline of JPEG2000 encoding system.

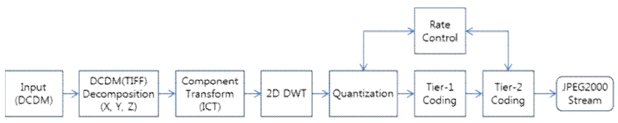

Figure 1. JPEG2000 encoding process

In the first process, denoted as DCDM decomposition, the components stored in *RGBRGB…* order are decomposed into R, G, and B planar components. JPEG2000 does not use the *RGB* to *YCbCr* de-correlation transform (followed by sub-sampling) since the multi-resolution nature of the wavelet transform may be used to achieve the same effect. All planar data with full resolution are compressed independently. Next, the components are optionally transformed into X, Y, and Z components. Hereafter, the processes occur for each component signal independently.

In the next process, denoted as a DC level shift, the components represented by unsigned values are level shifted to have zero DC values. The 2D-DWT divides signals into subbands, and the coefficients represent frequency features. The quantized wavelet coefficients in the code-blocks are encoded using coefficient bit modeling and arithmetic coding. This process is called tier-1 coding in JPEG2000. Tier-1 coding is essentially a bit-plane coding technique commonly used in wavelet-based image coders [6]. In tier-1 coding, code-blocks are encoded independently. If necessary, the generated bitstream can be truncated and grouped for rate control. The JPEG2000 stream is completed by adding the header information.

III. CUDA Parallel Processing

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.

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. CUDA-based JPEG2000 Encoding method

A. DCDM decomposition

Components stored in a TIFF or DPX data format, the components should be converted into X, Y, and Z planar components. In addition, they must be transferred into global memory on the GPU.

Figure 2 shows the memory copy method from DCDM to GPU memory. The size of input image is aligned to size conforming to coalesced memory access of CUDA. The memory controller of CUDA GPU tries to coalesce memory loads and stores issued by 32 threads (a warp) into as few memory transactions as possible. Only older GPUs coalesce memory transaction within half-warps (16 threads). The *width* and *height* parameters represent the real size of an image x-axis and y-axis, respectively. The parameters *stride_x* and *stride_y* represent the maximum size of an image x-axis and y-axis. The parameters *stride_x* and *stride_y* are set to 2,048 and 1,080 for the 2K, and 4,096 and 2,160 for the 4K, respectively. To maintain the coalesced memory access, the height of the image in GPU memory is changed. Note that the data structure can be fully used in CUDA, because the parameter stride_x is in multiples of 32, which is the warp size in CUDA.

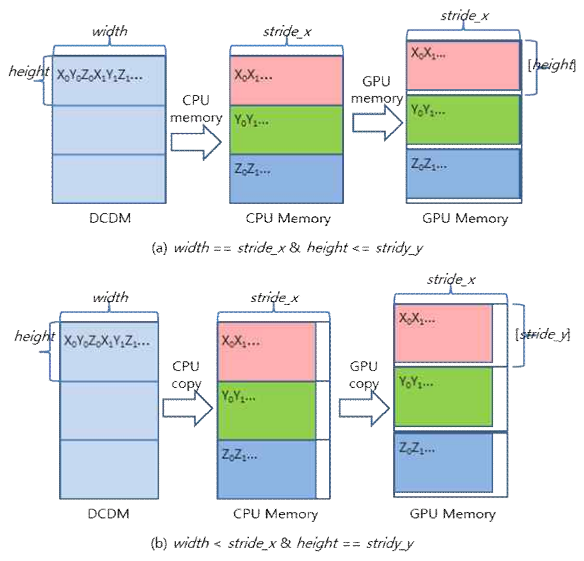

Figure 2. Memory copy from CPU to GPU

The first step of the computation is to fetch image data from global memory into fast shared memory. It is crucial here to comply with coalesced global memory access. For the coalesced reading access in the decomposition kernel function, the data in the global memory are loaded into the shared memory with a length of (*blockDIM* * 3), as shown in Figure. 3. According to the thread index number, each component is decomposed into the composite data allocated in the shared memory using modulo operation. Because the composite data for X, Y, and Z planar data are stored sequentially, all stores are coalescent. Finally, all data are stored back into the global memory.

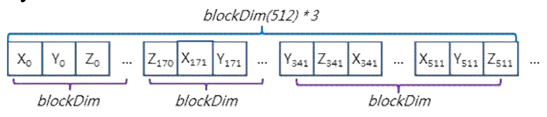
blockDim(512) * 3
X_0 Y_0 Z_0 ... Z_{170} X_{171} Y_{171} ... Y_{341} Z_{341} X_{341} ... X_{511} Y_{511} Z_{511} ...
blockDim
blockDim
blockDim

Figure 3. Coalesced reading access

B. Component transform

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. Similar to the decomposition and color conversion process, DC-shift and irreversible color transform 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*.

C. 2D-DWT

Figure 4 shows the 2D-DWT method used on CPUs or GPUs. Figure 4(a) shows the conventional 2D-DWT algorithm operated on CPUs. To calculate the 2D-DWT for an image, the lifting processes should be performed for all levels [7]. 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.

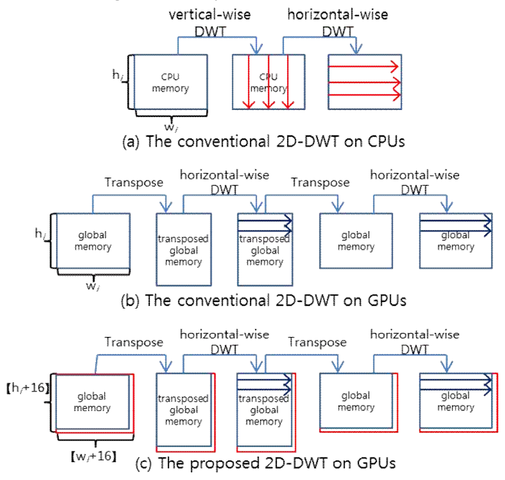

Figure 4. 2D-DWT on CPUs and GPUs

To avoid this problem, the transposed matrix has been used as shown in Figure 4(b). 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, an image is decomposed from 0 to last 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 as shown in Figure 4(c).

D. Quantization

After the 2D-DWT, all of the resulting subbands are quantized, which means that the precision of the wavelet coefficients is reduced. Quantization is the main source of information loss and aids in achieving compression. The quantizer maps several values that are in the range of some interval to one fixed value. It should be noted that the quantization step size can be chosen differently for every subband. That is, the quantization kernel function must be applied to each subband separately. After quantization, the data are transferred from global memory in the GPU to CPU memory. It should be noted that the operations on GPUs and CPUs are executed independently.

E. EBCOT

Embedded Block Coding with Optimal Truncation (EBCOT) is the fundamental and computationally very demanding part of the compression process of the JPEG2000 algorithm. Load balancing between the CPU and the GPU is a key performance factor. In this paper, we therefore assigned the EBCOT task to the CPU part.

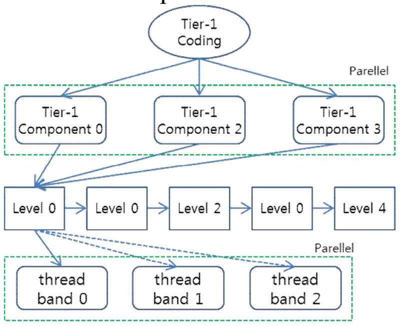

Figure 5. Tier-1 Coding using threads

F. Streaming

A stream function permits the GPUs to manage a one kernel function and one data transfer part simultaneously. Figure 6 shows our JPEG2000 encoder using a stream function. In this case, three components are processed. 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 decomposition kernel function, however, synchronization for the threads must be performed because it uses the data in the other components. The EBCOTs for the components are executed on a CPU simultaneously. It should be noted that the functions on the GPU and CPU are performed independently, as shown in Figure 6.

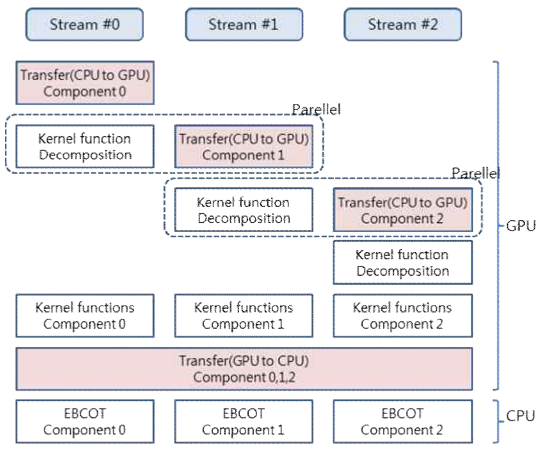
Stream #0
Stream #1
Stream #2
Transfer(CPU to GPU) Component 0
Parellel
Kernel function Decomposition
Transfer(CPU to GPU) Component 1
Parelle
Kernel function Decomposition
Transfer(CPU to GPU) Component 2
Kernel function Decomposition
GPU
Kernel functions Component 0
Kernel functions Component 1
Kernel functions Component 2
Transfer(GPU to CPU) Component 0,1,2
EBCOT Component 0
EBCOT Component 1
EBCOT Component 2
CPU

Figure 6. Stream function

V. Simulation Results

To evaluate the performance of the proposed algorithms, we consider images with 2K and 4K resolution. The proposed schemes are implemented in the reference software, called "JasPer" [8], which is defined in Part 5 of the JPEG2000 standard. 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 Geforce GTX 580. For the GPU implementation, we used CUDA as the development environment.

TABLE 1. Test image specification

TestID	Profile	Size	bits/sample
1	2K	2048x1080	8, 10
2	2K	1920x1080	8, 10
3	4K	4096x2160	8, 10
4	4K	3840x2160	8, 10

Table 1 shows the test image specifications and corresponding test ID. For each profile, the reference use samples that each sample has 8 bits per sample. Otherwise, the proposed algorithm is adapted to an image that has 10 bits per sample.

TABLE 2. Execution time of JasPer and the proposed algorithm

	TestID	Time(sec)		
		ICT	2D-DWT	Q
JasPer (CPU)	1(8bit)	0.0045	0.0766	0.0126
	2(8bit)	0.0058	0.0648	0.0179
	3(8bit)	0.0151	0.4954	0.0414
	4(8bit)	0.0194	0.4193	0.0540
Proposed Algorithm (GPU)	1(10bit)	0.0005	0.0055	0.0005
	2(10bit)	0.0005	0.0053	0.0005
	3(10bit)	0.0017	0.0192	0.0017
	4(10bit)	0.0016	0.0184	0.0017

Table 2 shows the encoding time of Jasper and the proposed algorithm for each test set. 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.

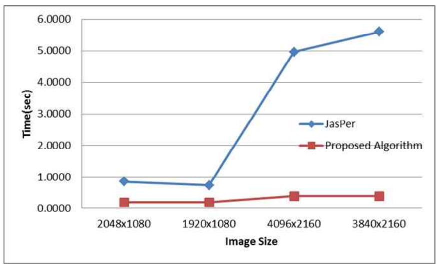

Figure 7. Comparison of JasPer and the proposed algorithm.

Figure 18 shows a comparison of the reference software and the proposed implementation for each resolution. Compared with the reference software, the proposed algorithm for each module provides about a ten to fifteen-fold better performance. It should be noted that the proposed algorithm for a whole image provides about a twenty-fold better performance.

VI. Conclusion

In this paper we described the development of CUDA implementation of JPEG2000 encoding. Specifically, we proposed a new method to maintain the coalesced global memory access, even though the width and height of an image is not a multiple of 16, which is half of the warp size in CUDA. In addition, we have proposed a new JPEG2000 algorithm that considers GPUs and multi-CPUs. Compared with other solutions, our JPEG2000 solution also provides high-speed encoding 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] ISO/IEC 15 444-1: Information Technology—JPEG 2000 Image Coding System—Part 1: Core Coding System, 2000.
[2] G. K. Wallace, "The JPEG still picture compression standard," IEEE Trans. Consum. Electron., vol. 38, no. 1, pp. 18–34, Feb. 1992.
[3] 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.
[4] 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.
[5] J. Sanders and E.Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley, 2011.
[6] J. M. Shapiro, "Embedded image coding using Zerotrees of wavelet coefficients," *IEEE Trans. Signal Process*., vol. 41, no. 12, pp. 3445–3462,Dec. 1993.
[7] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," SIAM Journal on Mathematical Analysis, vol. 29, no. 2, pp. 511-546, 1998.
[8] 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.

Jeongwoo Lee 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).

Bumho Kim received MS degree in information technology from Information Communication University in 2002. He is currently working in Electronics and Telecommunications Research Institute (ETRI).

Ki-Song Yoon received the Ph.D. degree in Computer Science from New York City University in 1993. From 1993, he was a principal member of Electronics and Telecommunications Research Institute (ETRI).