APBT-JPEG Image Coding Based on GPU

Chengyou Wang*, Rongyang Shan and Xiao Zhou
School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai
Weihai 264209, China
[e-mail: wangchengyou@sdu.edu.cn, sdusry@163.com, zhouxiao@sdu.edu.cn]
*Corresponding author: Chengyou Wang

Received December 25, 2014; revised February 7, 2015; accepted March 4, 2015; published April 30, 2015

Abstract

In wireless multimedia sensor networks (WMSN), the latency of transmission is an increasingly problem. With the improvement of resolution, the time cost in image and video compression is more and more, which seriously affects the real-time of WMSN. In JPEG system, the core of the system is DCT, but DCT-JPEG is not the best choice. Block-based DCT transform coding has serious blocking artifacts when the image is highly compressed at low bit rates. APBT is used in this paper to solve that problem, but APBT does not have a fast algorithm. In this paper, we analyze the structure in JPEG and propose a parallel framework to speed up the algorithm of JPEG on GPU. And we use all phase biorthogonal transform (APBT) to replace the discrete cosine transform (DCT) for the better performance of reconstructed image. Therefore, parallel APBT-JPEG is proposed to solve the real-time of WMSN and the blocking artifacts in DCT-JPEG in this paper. We use the CUDA toolkit based on GPU which is released by NVIDIA to design the parallel algorithm of APBT-JPEG. Experimental results show that the maximum speedup ratio of parallel algorithm of APBT-JPEG can reach more than 100 times with a very low version GPU, compared with conventional serial APBT-JPEG. And the reconstructed image using the proposed algorithm has better performance than the DCT-JPEG in terms of objective quality and subjective effect. The proposed parallel algorithm based on GPU of APBT also can be used in image compression, video compression, the edge detection and some other fields of image processing.

Keywords: Parallel computing, GPU, image coding, all phase biorthogonal transform (APBT), discrete cosine transform (DCT)

This work was supported by the promotive research fund for excellent young and middle-aged scientists of Shandong Province, China (Grant No. BS2013DX022) and the National Natural Science Foundation of China (Grant No. 61201371).

1. Introduction

Image compression has become an important research area for many years due to increasing demand on transfer and storage of data. The most widely used and successful image coding standard should be joint photographic experts group (JPEG) standard, also known as ITU-81 [1] first issued in 1992. Because JPEG encoding can be implemented quickly, around 80% of the images on the Internet observe JPEG standard. The JPEG system mainly makes up of discrete cosine transform (DCT) [2] and Huffman coding, DCT has been really developed during the recent years, and applied in the international standards for image and video compression, like JPEG [3], MPEG-2 [4], MPEG-4 [5], H.264/AVC [6] and H.265/HEVC [7], and DCT is also widely used in many other fields of image processing. DCT still takes an important place in image processing. With the development of orthogonal transform, DCT has become quite mature, and two-dimensional DCT is the core of JPEG coding. However, DCT is not the best choice in image coding, because block DCT transform coding has serious blocking artifacts when the image is highly compressed at low bit rates. The all phase biorthogonal transform (APBT) [8] which is based on Walsh-Hadamard transform (WHT), DCT and inverse discrete cosine transform (IDCT) proposed by Hou et al. is a new transform for image compression instead of DCT, which solves the problem of blocking artifacts in DCT, and APBT uses the uniform quantization step instead of the complex quantization table in DCT which makes APBT save the storage space of quantization table.

In wireless multimedia sensor networks (WMSN), image compression and transmission [9] are widely used. When the image is collected by camera, it will be transferred on the internet and the source image needs to be compressed usually for reducing the occupation of the bandwidth. But the compression algorithm always has high time complexity and the real-time of WMSN is a very important issue. Parallel algorithm provides an effective way to improve the efficiency and real-time.

At present, the operating frequency of processor has hit a clock rate limit at around 4 GHz, but for current technology, if the clock rate continues to increase, more heat and electric bill rather than efficiency will be got. People are not able to improve the efficiency of computation by improving the frequency of processor, so parallel computation becomes more and more popular. Some researches of parallel computing are based on many-core processors [10, 11], in which Yan and Zhang proposed a parallel framework to decouple motion estimation (ME) for different partitions on many-core processors, and compared with serial execution, their work achieves more than 30 and 40 times speedup for 1920×1080 and 2560×1600 video sequences. However, with the increasing compute capability of GPU, GPU is not only a graphic card, but also it is being used in the field of computing. Although GPU's operation frequency is lower than CPU, GPU's Flops (floating-point operations per second) is much higher than CPU, because GPU has more ALU. GPU can launch thousands of threads at the same time, so GPU is more suitable for parallel computing. Currently parallel computing based on GPU has been widely used in scientific research.

Compute unified device architecture (CUDA) toolkit which is released by NVIDIA makes parallel computation based on GPU easier than before. Parallel algorithm implemented by CUDA can get 10 times acceleration easily than serial algorithm. CUDA and massively parallel GPU hardware is changing how we think about computation. No longer limited to performing one or a few operations at a time, CUDA programmers write programs that perform tens of thousands of operations simultaneously. In 2011, Tokdemir and Belkasim [12] used parallel DCT algorithm in data compression, and they got a satisfying result in efficiency. Liu and Fan [13] designed parallel program for DCT in 2012, they used parallel DCT algorithm in JPEG coding, and the parallel DCT algorithm gains 20 times acceleration than serial DCT algorithm in the experiment, but they did not make all parts of JPEG algorithm run on GPU. Holub and Srom [14] used GPU-accelerated DXT in low-latency network transmissions of HD, 2K, and 4K video in 2013.

The system of basic JPEG based on DCT is shown in **Fig. 1**. The encoder mainly contains four parts: DCT, quantization, Zig-zag ordering and Huffman coding. At the beginning of JPEG coding, the source image is divided into 8×8 sub-images, then every sub-image is processed by DCT. After DCT, every sub-image gets an 8×8 matrix, it includes 64 DCT coefficients. The upper left corner coefficient is called DC coefficient, the other 63 coefficients are AC coefficients. DC coefficient plays an important role in reconstructed image; AC coefficients have a low influence on reconstructed image compared with DC coefficient. So some AC coefficients can be dropped by quantization table for removing redundancy. The coefficient matrix of every sub-image, DC coefficient is coded by differential pulse code modulation (DPCM), and AC coefficients are put in a one-dimensional array with Zig-zag ordering, after that Huffman coding is used for image compression.

Accordingly, the decoder also has four parts: Huffman decoder, inverse Zig-zag ordering, inverse quantization, and inverse DCT. It has the reverse order with encoder.

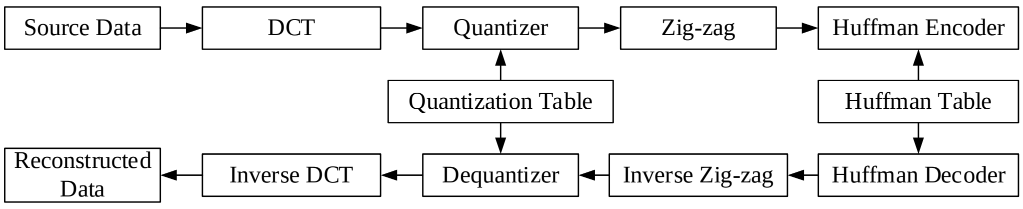


Fig. 1. DCT-JPEG system

JPEG can compress the source color image data (luminance component and chrominance component) from the different color spaces (also referred to as color model or color system) like RGB, YCbCr, etc. Wherein, RGB model almost includes all of the colors perceived by the human eye, and the RGB model is widely used in digital color. However this model is not suitable for graphical analysis, since the R, G and B components are highly relevant. When changing the brightness, the three components will be amended accordingly. Therefore, most of the color image compression schemes transform the highly correlated RGB color space into a decorrelated color space like YUV, YCbCr, etc. In JPEG system, the color space is YCbCr, so it is necessary to convert the RGB to YCbCr. The equation used in color space conversion is given by:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = -0.1687R - 0.3313G + 0.5B + 128 \\ Cr = 0.5R - 0.4187G - 0.0813B + 128 \end{cases} \tag{1}$$

where R, G, B are the red, green and blue components in RGB color space, and Y, Cb, Cr are the luma component, the blue-difference and red-difference chroma components in YCbCr color space.

The rest of this paper is organized as follows. Section 2 introduces CUDA. Section 3 starts with a brief review of DCT and APBT. Section 4 is the design of parallel algorithm of

APBT-JPEG system. Experimental results of the proposed method are presented in Section 5. Conclusions and remarks on possible further work are given finally in Section 6.

2. CUDA

CUDA is an easy-to-use programming interface which is added to graphics card by NVIDIA in 2007. C-like language is used in CUDA to design parallel program, so developers can rewrite the serial code which programed before in C language to parallel code. With CUDA, parallel code can gain higher computational efficiency than serial one. In order to get higher efficiency CUDA program, it is necessary to know some basic knowledge about GPU. **Fig. 2** shows that the architecture of CUDA has three main components. It contains CUDA libraries, CUDA runtime API and CUDA device API.

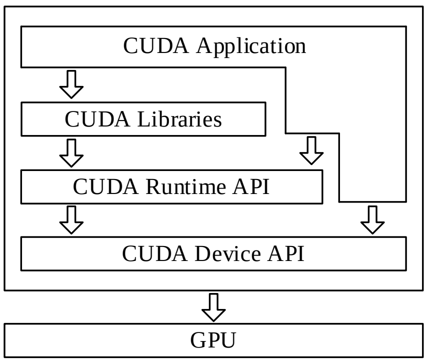


Fig. 2. The architecture of CUDA

Fig. 3 displays the programming model of CUDA. CPU could be seen as host and GPU should be seen as device or the co-processor of CPU which has a memory. In this system, it can have one host and multiple devices. Under this model, host and device work together and fulfill their proper function. The CPU is responsible for preparing data and driving kernel, while the GPU is focused on the implementation of highly threaded parallel processing tasks. CPU and GPU have their own memory and they cannot visit each memory directly [15]. Data need to be copied from host to device at the beginning of the program and copied back at the end of the program; it will have some latency in the CUDA application. But it will be solved in the future, because the CUDA toolkit will support unified memory in next version.

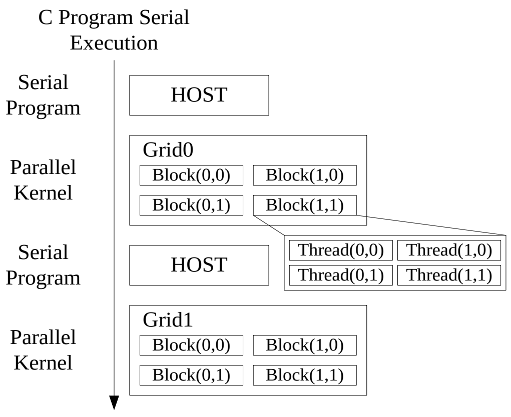


Fig. 3. CUDA program model

After developers analyze the algorithm, they give the part of program which is needed parallel computing to GPU. The function which runs on GPU for parallel computing is called kernel. Kernel is not a complete program; it is part of CUDA programs which runs on GPU and is used to compute what you need. The kernel function and the serial processing in host-side compose a complete CUDA program (as shown in **Fig. 3**).These programs will be executed with the order of the sentences in the program. Host-side code is mainly used to prepare data and run the kernel on GPU.

In CUDA application, kernel is organized as grid. Every kernel has one grid, which is made up of blocks, and every block has many threads. Generally, the number of threads in every block has the relation with GPU. There are two levels of parallelism in a kernel: parallelism between blocks in the grid and parallelism between threads in the block. Each thread executes the kernel one time according to the serial order of the instruction in the program [15]. Threads in the same block can communicate through shared memory, so developers would have more room for their program.

3. DCT and APBT

3.1 Discrete Cosine Transform (DCT)

The conventional two-dimensional DCT transform is made usually by two one-dimensional DCT transform on row and column directions separately [1]. $\boldsymbol{X}$ is the data of an $N \times N$ image block, and $\boldsymbol{C}$ represents DCT matrix with size of $N \times N$ respectively. After two-dimensional DCT transform, transform coefficients block $\boldsymbol{Y}$ can be denoted by

$$\boldsymbol{Y} = \boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}, \tag{2}$$

$$\boldsymbol{C}(i,j) = \begin{cases} \sqrt{\dfrac{1}{N}}, & i = 0,\ j = 0,1,\cdots,N-1, \\ \sqrt{\dfrac{2}{N}} \cos\dfrac{i(2j+1)\pi}{2N}, & i = 1,2,\cdots,N-1,\ j = 0,1,\cdots,N-1. \end{cases} \tag{3}$$

where $\boldsymbol{C}^{\mathrm{T}}$ is the transpose matrix of $\boldsymbol{C}$.

Since DCT is an orthogonal transform, i.e. $\boldsymbol{C}^{\mathrm{T}} = \boldsymbol{C}^{-1}$, we use

$$\boldsymbol{X} = \boldsymbol{C}^{-1}\boldsymbol{Y}(\boldsymbol{C}^{\mathrm{T}})^{-1} = \boldsymbol{C}^{-1}\boldsymbol{Y}(\boldsymbol{C}^{-1})^{\mathrm{T}} = \boldsymbol{C}^{-1}\boldsymbol{Y}C, \tag{4}$$

to reconstruct the image, where $\boldsymbol{C}^{-1}$ is the inverse matrix of $\boldsymbol{C}$.

3.2 All Phase Biorthogonal Transform (APBT)

On the basis of all phase digital filtering , three kinds of all phase biorthogonal transforms based on the WHT, DCT and IDCT were proposed and the matrices of APBT were deduced in [8]. Similar to DCT matrix, it also can be used in image compression to transform the image from spatial domain to frequency domain.

Taking all phase discrete cosine biorthogonal transform (APDCBT) for example, the process of two-dimensional APBT is introduced as follows. $\boldsymbol{X}$ is the data of an $N \times N$ image block, and $\boldsymbol{V}$ represents APDCBT matrix with size of $N \times N$ respectively. After two-dimensional APDCBT transform, transform coefficients block $\boldsymbol{Y}$ can be denoted by

$$\boldsymbol{Y} = \boldsymbol{V}\boldsymbol{X}\boldsymbol{V}^{\mathrm{T}}, \tag{5}$$

$$\boldsymbol{V}(m,n)=\begin{cases}\dfrac{N-m}{N}, & n=0,\ m=0,1,\cdots,N-1,\\[2mm] \dfrac{1}{N^2}\left[(N-m)\cos\dfrac{mn\pi}{N}-\csc\dfrac{n\pi}{N}\sin\dfrac{mn\pi}{N}\right], & n=1,2,\cdots,N-1,\ m=0,1,\cdots,N-1.\end{cases} \tag{6}$$

where $\boldsymbol{V}^{\mathrm{T}}$ is the transpose matrix of $\boldsymbol{V}$. We use

$$\boldsymbol{X}=\boldsymbol{V}^{-1}\boldsymbol{Y}(\boldsymbol{V}^{-1})^{\mathrm{T}}, \tag{7}$$

to reconstruct the image, where $\boldsymbol{V}^{-1}$ is the inverse matrix of $\boldsymbol{V}$.

4. The Design of Parallel Algorithm of APBT-JPEG System

The important part of designing parallel program is breaking down task. These sub tasks are processed parallel for improving the computational efficiency. At the beginning of APBT-JPEG, the first step is the preprocessing of source image, which should be divided into 8×8 sub-images, every sub-image has 64 pixels. In the conventional serial algorithm of APBT-JPEG, every sub-image is processed serially, and each pixel is independent of each other. In CUDA program model, it has two parallel levels. The first one is parallelism between blocks in grid, and the second one is parallelism between threads in block. **Fig. 4** shows the mapping relation between image component and grid. The image is mapped to the grid, sub-images are mapped to the blocks in the grid, and pixels in each sub-image are mapped to the threads in the block.

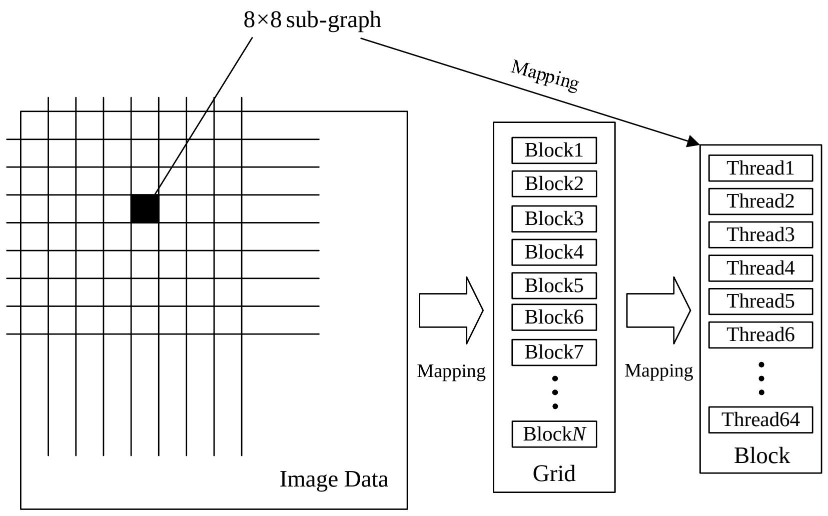


Fig. 4. Mapping relation between image component and grid

The whole task of APBT-JPEG needs to be divided into N parts. Every part is processed in one block, and every block has 64 threads. If the size of the source image is not integral multiple of 64, it can be filled by zeros at the end of source data.

$$N=\frac{W\times H}{64}, \tag{8}$$

where N is the number of blocks in the grid, W is the width of source image, and H is the height of source image.

4.1 Parallel Algorithm of APBT and Quantization

The algorithm of APBT is similar to the conventional DCT. Through DCT or APBT, the energy of image is concentrated in the top left corner. DC coefficient is in the upper left corner, and the others are AC coefficients. After quantization, the image can be compressed easily.

The essence of DCT, APBT and their inverse transform is the matrix multiplication, as shown in Eqs. (9) and (10).

$$\boldsymbol{Y} = \boldsymbol{T}\boldsymbol{X}\boldsymbol{T}^{\mathrm{T}}, \tag{9}$$

$$\boldsymbol{X} = \boldsymbol{T}^{-1}\boldsymbol{Y}(\boldsymbol{T}^{-1})^{\mathrm{T}}, \tag{10}$$

where $\boldsymbol{X}$ is the two-dimensional image matrix, $\boldsymbol{Y}$ is the transform coefficients block, $\boldsymbol{T}$ is the transform matrix. In the parallel algorithm of APBT, every thread executes vector multiplication twice, then every thread gets an APBT coefficient.

APBT removes the correlation between pixels in each sub-image; it provides necessary conditions for image compression. After APBT, every block gets 64 APBT coefficients. The first coefficient, which locates in the upper left corner, is DC coefficient; the other 63 coefficients are AC coefficient. In parallel algorithm of APBT, blocks are executed parallel and threads are executed parallel in the macro. Every thread gets an APBT coefficient and then processes the data of APBT coefficient parallel. After APBT, every APBT coefficient is quantified by uniform quantization table and they are rounded to integer which shows in Eq. (11),

$$F_q = \mathrm{round}\left(\frac{F}{Q}\right), \tag{11}$$

where F_q is the data after quantization, F is the data before quantization, Q is uniform quantization step. In conventional JPEG system, it contains two kinds of quantization tables: chrominance quantization table and luminance quantization table, but in APBT-JPEG, it uses uniform quantization. In the serial algorithm of APBT and quantization, the pixels of source image are processed in order. But in the parallel algorithm, each thread executes the program at the same time, so the data is processed parallel. In this way, the efficiency of parallel algorithm has a high improvement.

4.2 The Design of Parallel Huffman Coding

After threads execute the code of APBT and quantization completely, the block will get an 8×8 coefficient matrix. Every thread puts its data in a one-dimensional array with the scanning order of Zig-zag in **Fig. 5.**

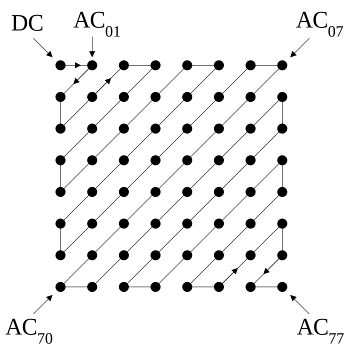


Fig. 5. Zig-zag ordering

In JPEG system, entropy coding contains Huffman coding and arithmetic coding, and in the baseline JPEG, it only uses Huffman coding. APBT and quantization are prepared for image compression, and the image data is further compressed by entropy coding.

In parallel APBT-JPEG system, there are two levels for parallelism in Huffman entropy. The first level is block-level parallelism, which is insufficient to achieve good performance as we witnessed during the development process. The second level is intra block parallelism; it utilizes the fact that both run-length encoding (RLE), which compresses long sequences of zeros after quantization of APBT-transformed data, and lookup of Huffman codes can be done in parallel. At first DC coefficient is coded by DPCM as shown in Fig. 6.

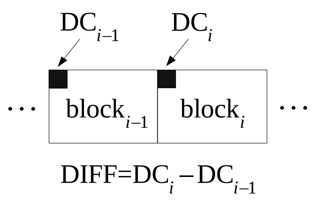


Fig. 6. The DPCM of DC coefficient

The first thread is called thread zero, when the difference between two adjacent blocks is calculated by thread zero. The next task of thread zero is coding the difference. The DC coefficient can be described by two symbols. A is the size of the difference and B is the amplitude of the difference. While thread zero gets A, it will query the DC Huffman table (as shown in Table 1) for Huffman code, and B is coded by VLI. The thread zero puts the Huffman code of B in the end of the Huffman code of A, which constitutes the Huffman code of DC coefficient.

Table 1. The luminance Huffman table of DC coefficient

Size	**Length**	**Code**
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Fig. 7 shows that Huffman code of AC coefficient can also be described as two symbols: symbol A contains run-length and the size of non-zero AC coefficient, which is coded by RLE. Symbol B is the amplitude of AC coefficient.

Symbol A	Symbol B
(RUNLENGTH, SIZE)	(AMPLITUDE)

Fig. 7. The Huffman code of AC coefficient

In parallel algorithm of Huffman coding, it is very difficult to get the run length, so the issue of getting run-length is a technical problem, because threads which run on GPU are concurrent, we cannot count it one by one, like in serial algorithm. In parallel algorithm, we propose a solution to solve this problem, we use sorting algorithm to get run-length in this paper. There are many sorting algorithms available, some can be implemented easily and efficiently on the GPU but many of them are not so suitable. In this parallel algorithm, odd-even sort is used in this paper (**Fig. 8**). Firstly, comparing the element at the even with the higher adjacent element at the odd. If the odd element is larger than the even element, the elements are swapped. Next, start from the odd element and repeat the above step. The list has been sorted until no swaps.

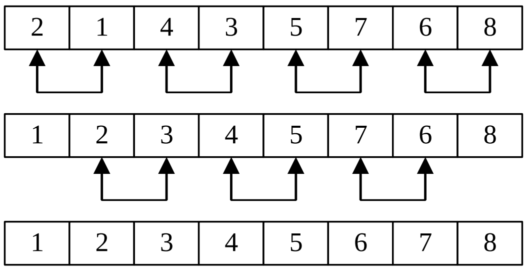


Fig. 8. Odd-even sort

We use an array which size is 64 in the share memory to achieve this algorithm. Share memory is shared in block, each block has their own share memory, and the share memory can only be visited by threads in the same block. When Zig-zag ordering, every thread compares its quantized APBT coefficient with zero, if the coefficient is not equal to zero, we put the number of the thread in the array. If the coefficient is equal to zero, we put the number which is above 64 in the array in order to facilitate the sorting, because the biggest number of thread is 63. When the array is sorted, the difference of two adjacent numbers in the array is run-length (as shown in **Fig. 9**). The maximum number of the array is at the end of the array which is described as EOB.

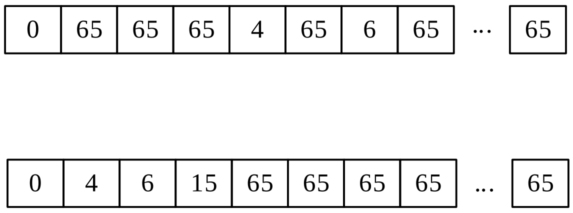


Fig. 9. The mean to get run-length

When the run-length of the AC coefficient and the size of AC coefficient are got, the Huffman code can be obtained by querying the luminance Huffman table of AC coefficient. The amplitude of AC coefficient is also coded by VLI. The highest bit of VLI is the sign bit. If the amplitude is above zero, the sign bit is “1” and the binary of the amplitude is the symbol B. If the amplitude is below zero, the sign bit is “0” and the ones-complement code of amplitude is the symbol B.

While the encoder is completed, the parallel algorithm of inverse APBT and quantization is similar to parallel APBT and quantization. In Huffman coding, we know that EOB is at the end of quantized APBT coefficients in each block. So at beginning of decoding, we need to pre-process the data of JPEG image on CPU. We extend the array to 64 elements by EOB. Therefore the parallel Huffman decoder has the same structure with Huffman coder. So it can be designed easily.

5. Experimental Results and Analysis

In the experiment, the result shows that the efficiency of parallel APBT-JPEG is much higher than serial APBT-JPEG. Peak signal to noise ratio (PSNR) is chosen to measure the performance of reconstructed image in this paper. The CPU used for the experiment is Intel i3 3.10GHz with 6GB DDR3 memory. The GPU used for the experiment is GTX480 with 384 CUDA cores.

$$\mathrm{PSNR} = 10\log_{10}\left[\frac{255^2 MN}{\sum_{i=1}^{M}\sum_{j=1}^{N}[I_{\mathrm{in}}(i,j) - I_{\mathrm{out}}(i,j)]^2}\right](\mathrm{dB}) \tag{12}$$

where I_{in} and I_{out} stand for the original image and the reconstructed image respectively. M and N represent the height and width of the test image.

In order to test the performance of the proposed algorithm, simulation is conducted by applying to gray image Lena (8bits/pixel, 512×512). From **Fig. 10**, there are four reconstructed images of Lena. We can know that: **Fig. 10(a)** and **Fig. 10(c)** are the reconstructed images of DCT-JPEG; **Fig. 10(b)** and **Fig. 10(d)** are the reconstructed images of APBT-JPEG. Compared **Fig. 10(a)** with **Fig. 10(c)**, it can be seen that the reconstructed image using parallel algorithm based on GPU has the same subjective effect with the serial one which runs on CPU, and comparing **Fig. 10(b)** with **Fig. 10(d)**, we can conclude the same conclusion. Compared **Fig. 10(a)** with **Fig. 10(b)**, the reconstructed image of APBT-JPEG has a better subjective quality than DCT-JPEG.

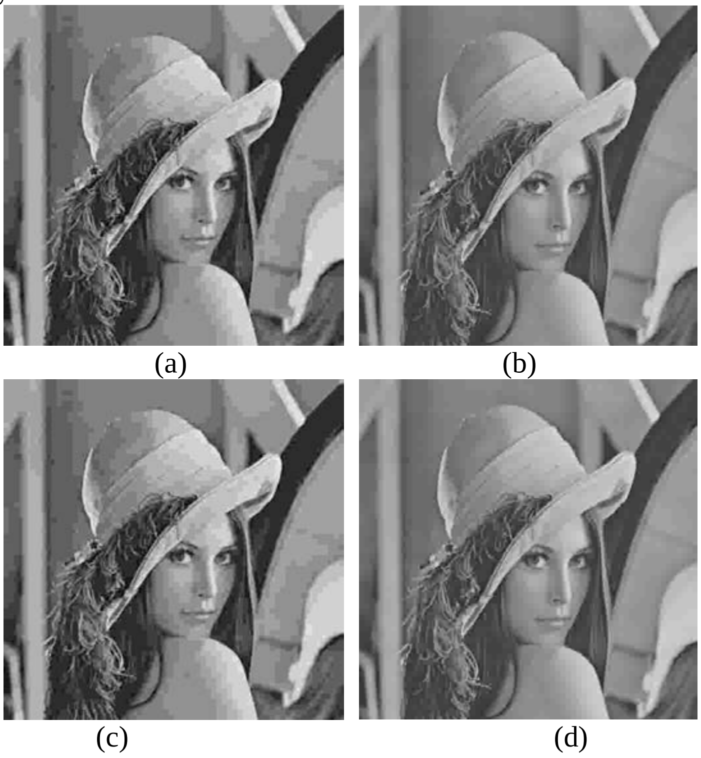


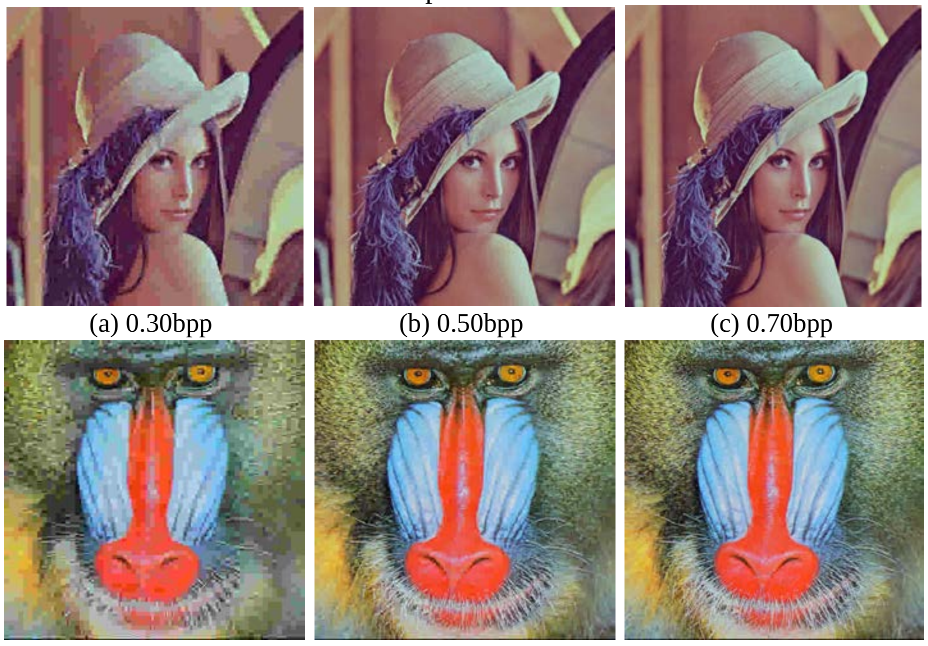
Fig. 10. The reconstructed images of Lena (0.20bpp): (a) DCT-JPEG on GPU, (b) APBT-JPEG on GPU, (c) DCT-JPEG on CPU, (d) APBT-JPEG on CPU

Table 2 shows the PSNR of reconstructed images which obtained from the parallel algorithm of DCT-JPEG and the parallel algorithm of APBT-JPEG which runs on GPU. In **Table 2**, the PSNR of APBT-JPEG is higher than DCT-JPEG which means the objective quality of APBT-JPEG is better than DCT-JPEG. From **Table 2** and **Fig. 10**, the reconstructed image using the parallel algorithm of APBT-JPEG has the better performance with the DCT-JPEG in terms of objective quality and subjective effect.

Table 2. The PSNR of reconstructed images in different algorithms

Bit rate/bpp	DCT-JPEG PSNR/dB	APBT-JPEG PSNR/dB
0.20	28.52	28.82
0.25	30.42	30.66
0.35	32.82	32.90
0.45	34.15	34.27
0.60	35.57	35.73
1.00	37.63	38.10
1.25	39.08	39.20

We also apply the parallel APBT-JPEG to color image. In the experiment, we use the images Lena (24bits/pixel, 512×512) and Mandrill (24bits/pixel, 512×512). In **Fig. 11**, there are some reconstructed images at different bit rates, and the reconstructed images of APBT-JPEG based on GPU have the same performance with APBT-JPEG based on CPU.

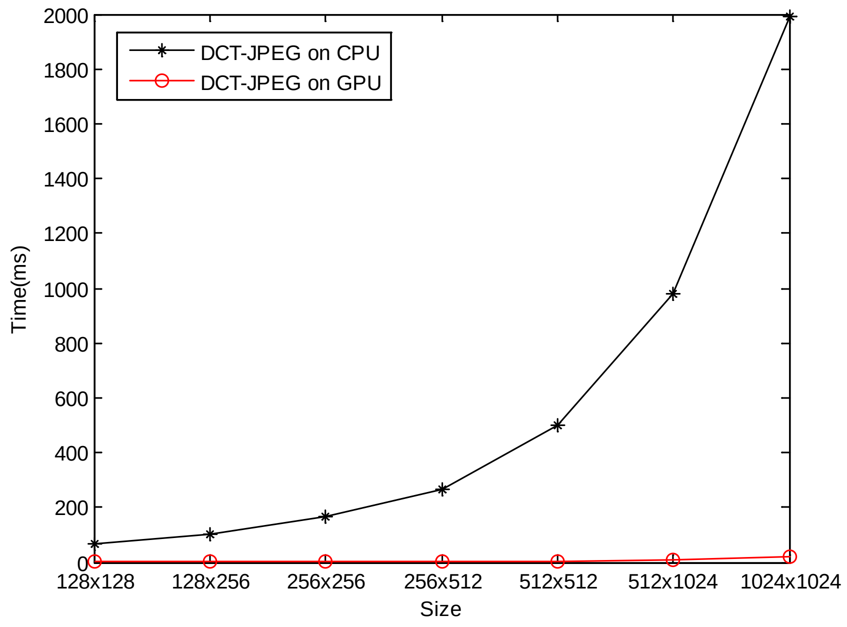

(a) 0.30bpp (b) 0.50bpp (c) 0.70bpp

(d) 0.30bpp (e) 0.50bpp (f) 0.70bpp

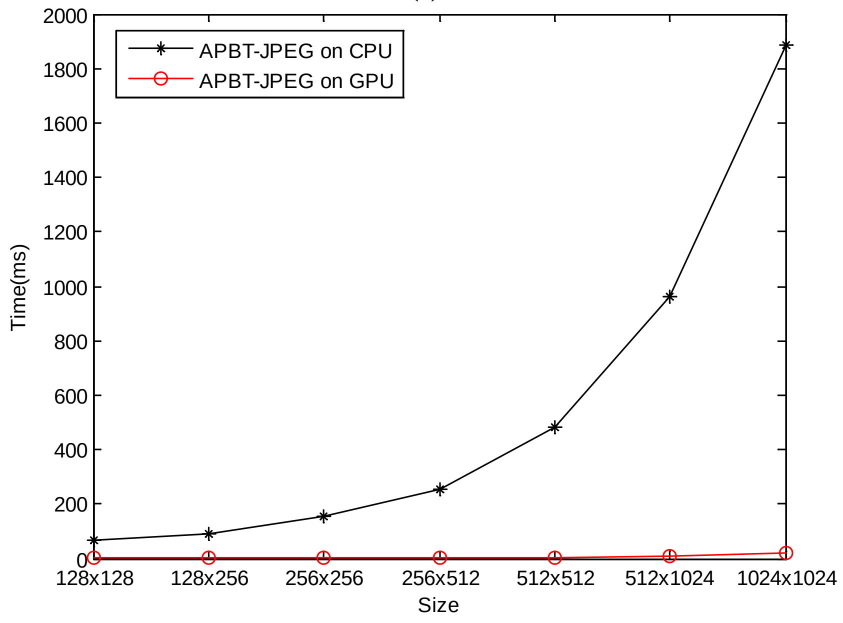
Fig. 11. The reconstructed images based on parallel APBT-JPEG

The different sizes from 128×128 to 1024×1024 of Lena are used to test the efficiency of parallel algorithm. We compare the runtime on GPU with the runtime on CPU. From the experimental results in **Fig. 12**, we can know the efficiency of parallel DCT-JPEG and APBT-JPEG algorithm is much higher than serial algorithms, so the efficiency of parallel computing is very impressive in general.

From the experimental results in **Fig. 12** and **Table 3**, we can know the efficiency of parallel APBT algorithm is much higher than serial APBT algorithm. The parallel APBT that runs on GPU could gain at least 100 times acceleration, and the maximum speedup ratio can reach more than 140 times, so the computational efficiency of parallel computing is very impressive in general. Parallel computing based on GPU can process dozens of images at the same time, and the efficiency of all phase biorthogonal transform is greatly improved.



(a)



(b)

Fig. 12. The running time of DCT-JPEG and APBT-JPEG in different platforms: (a) DCT-JPEG on CPU and GPU, (b) APBT-JPEG on CPU and GPU

Table 3. The running time of DCT-JPEG and APBT-JPEG in different platforms

time / size	DCT-JPEG(ms)		APBT-JPEG(ms)	
	On CPU	**On GPU**	**On CPU**	**On GPU**
128×128	70	0.482	66	0.472
128×256	101	0.781	93	0.775
256×256	167	1.347	153	1.336
256×512	268	2.551	256	2.540
512×512	499	4.387	484	4.382
512×1024	981	9.174	964	9.161
1024×1024	1992	18.853	1886	18.775

6. Conclusion

On the basis of above discussion, it can be concluded that the parallel algorithm of APBT-JPEG is proposed in this paper. Compared with the conventional serial algorithm, the parallel APBT-JPEG that runs on GPU could gain at least 100 times acceleration, and the maximum speedup ratio can reach more than 140 times. The algorithm of parallel APBT-JPEG can get the same reconstructed image with serial algorithm. Compared with DCT-JPEG algorithm, at low bit rates, the reconstructed image has better objective quality and subjective effects. So the efficiency problem of conventional APBT algorithm and the blocking artifacts in DCT can be solved by the parallel APBT algorithm. Therefore, parallel algorithm of APBT is very helpful to improve the real-time of WMSN and the improvement the quantity of reconstructed image.

In the future, we will research the video compression based on APBT, and use parallel algorithm to accelerate the speed of video compression.

References

[1] ISO/IEC, “Information Technology -- Digital Compression and Coding of Continuous-tone Still Images—Part 1: Requirements and Guidelines,” *ISO/IEC 10918-1 | ITU-T Rec. T.81*, 1994. Article (CrossRef Link).

[2] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE Transactions on Computers*, vol. 23, no. 1, pp. 90-93, 1974. Article (CrossRef Link).

[3] ISO/IEC, “Information Technology -- Digital Compression and Coding of Continuous-tone Still Images -- Part 1: Requirements and Guidelines,” *ISO/IEC 10918-1: 1994 | ITU-T Rec. T. 81*, 2011. Article (CrossRef Link).

[4] ISO/IEC, “Information Technology -- Generic Coding of Moving Pictures and Associated Audio Information -- Part 2: Video,” *ISO/IEC 13818-2: 2013*, 2013. Article (CrossRef Link).

[5] T. Ebrahimi and C. Horne, “MPEG-4 natural video coding -- an overview,” *Signal Processing: Image Communication*, vol. 15, no. 4-5, pp. 365-385, 2000. Article (CrossRef Link).

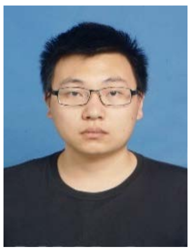
[6] Joint Video Team of ITU-T and ISO/IEC, “Information Technology -- Coding of Audio-Visual Objects -- Part 10: Advanced Video Coding,” *ITU-T Rec. H.264 | ISO/IEC 14496-10: 2012*, 2014. Article (CrossRef Link).

[7] ISO/IEC, “Information Technology -- High Efficiency Coding and Media Delivery in Heterogeneous Environments -- Part 2: High Efficiency Video Coding,” *ISO/IEC 23008-2: 2013*, 2013. Article (CrossRef Link).

[8] Z. X. Hou, C. Y. Wang, and A. P. Yang, “All phase biorthogonal transform and its application in JPEG-like image compression,” *Signal Processing : Image Communication*, vol. 24, no.10, pp. 791-802, 2009. Article (CrossRef Link).

[9] X. H. Zhao, Z. L. Wang, and K. K. Zhao, “Research on distributed image compression algorithm in coal mine WMSN,” *International Journal of Digital Content Technology and its Applications*, vol. 5, no. 18, pp. 283-291, 2011. Article (CrossRef Link).
[10] C. G. Yan, Y. D. Zhang, J. Z. Xu, F. Dai, J. Zhang, Q. H. Dai, and F. Wu, “Efficient parallel framework for HEVC motion estimation on many-core processors,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 12, pp. 2077-2089, 2014. Article (CrossRef Link).
[11] C. G. Yan, Y. D. Zhang, J. Z. Xu, F. Dai, L. Li, Q. H. Dai, and F. Wu, “A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors,” *IEEE Signal Processing Letters*, vol. 21, no. 5, pp. 573-576, 2014. Article (CrossRef Link).
[12] S. Tokdemir and S. Belkasim, “Parallel processing of DCT on GPU,” in *Proc. of the Data Compression Conference*, pp. 479, 2011. Article (CrossRef Link).
[13] D. Liu and X. Y. Fan, “Parallel program design for JPEG compression encoding,” in *Proc. of the 9th International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 2502-2506, 2012. Article (CrossRef Link).
[14] P. Holub, M. Srom, M. Pulec, J. Matela and M. Jirman, “GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video,” *Future Generation Computer Systems*, vol. 29, no. 8, pp. 1991-2006, 2013. Article (CrossRef Link).
[15] NVIDIA Corporation: NVIDIA CUDA programming guide. http://docs.nvidia.com/cuda/.

Chengyou Wang received his B.E. degree in electronic information science and technology from Yantai University, China in 2004, and his M.E. and Ph.D. degree in signal and information processing from Tianjin University, China in 2007 and 2010 respectively. Now he is an associate professor in the School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai, China. His current research interests include digital image/video processing and analysis, multidimensional signal and information processing.



Rongyang Shan received his B.E. degree in communication engineering from Shandong University, Weihai, China, in 2014. Now he is pursuing his M.E. degree in signal and information processing in Shandong University, Weihai, China. His current research interests include parallel computing, digital image processing and analysis.

Xiao Zhou received her B.E. degree in automation from Nanjing University of Posts and Telecommunications, China in 2003, her M.E. degree in information and communication engineering from Inha University, Korea in 2005, and her Ph.D. degree in information and communication engineering from Tsinghua University, China in 2013. Now she is a lecturer in the School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai, China. Her current research interests include wireless communication technology, digital image processing and analysis.