Parallelization of a Color DCT Watermarking Algorithm using a CUDA-based Approach

Alireza Ahmadi Mohammadabadi
Department of Computer Engineering
Razi University
Kermanshah, Iran
alireza.ahmadi.computer@gmail.com

Abdolah Chalechale
Department of Computer Engineering
Razi University
Kermanshah, Iran
chalechale@razi.ac.ir

***Abstract*— Image watermarking in DCT domain has a high computational complexity especially for color and high resolution images, where usage of them has been significantly grown. To address this issue, in this article, a data-parallel color DCT watermarking approach is proposed and implemented on GPU using CUDA. Also, in this work, before embedding, the color watermark is compressed using a modified method to get less distortion. CUDA implementation of 8×8 DCT offers 12x-43x speedup with GT 540M and 94x-105x speedup with GTX 580, for different image sizes. In case of embedding procedure, the speedup obtained by GT 540M is between 7x and 26x, and the speedup obtained by GTX 580 is between 46x and 73x, for various case studies. Furthermore, in case of extracting procedure, GT 540M leads to a speedup between 10x and 29x, and GTX 580 leads to a speedup between 75x and 80x, for various case studies.**

Keywords—Color images; CUDA; DCT; GPU; Parallel programming; Watermarking;

I. INTRODUCTION

Image watermarking is an efficient solution for authentication and copyright protection of images in popular communication environments like Internet which is susceptible to illegal usages [1]. The basic procedure of image watermarking is to hide some secret data as watermark into a cover image as host, verifying ownership of the image by detecting the watermark. The watermark can be inserted into spatial domain of the host image by changing the gray-levels of some pixels. However the spatial domain algorithms suffer from vulnerability against signal processing attacks such as JPEG compression or filtering.

In the last decades, most of the reported watermarking methods are concentrated in the embedding in transform domain, such as Discrete Cosine Transform (DCT), Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT) and so on [1-4]. The DCT is used most widely for transforming the multimedia data to the frequency domain in most of the compression standards such as JPEG, MPEG, JVT, ITU's H.261 and H.263, etc [3]. As the conclusion, frequency domain especially DCT is more popular, efficient and applied in the watermarking algorithm. In the same time, DCT watermarking has more time complexity rather than spatial domain.

978-1-5090-3586-1/16/$31.00 ©2016 IEEE

On the other hand, with ever-increasing use of color images, color image watermarking has become more important, so that, in recent years, color image watermarking has been becoming one of the hot research topics [2, 4-5]. Compared with gray-level image, the color one takes advantages of higher capacity and fidelity, which is because the color perception depends not only on the luminance but also on the chrominance. However, due to color space transforms and further processing requirement, the color watermarking leads to further increase the time complexity.

Therefore, it is required to use newer methods as well as platforms with higher computing power to cope with these time consuming applications. In recent years, due to power and technology restrictions, the role of parallelism as well as multi- and many-core processors for higher performance and speedup in various programs has become more and more important.

Graphics Processing Unit (GPU) as a highly parallel, multithreaded and many-core architecture can be applied by user for computationally intensive processes. To address the issue, NVIDIA Corporation introduced Compute Unified Device Architecture (CUDA) as a general purpose parallel computing architecture with a new parallel programming model and instruction set architecture [6]. In fact CUDA is an extended model of standard C language for parallel computing that allows the user to program own algorithms on GPU easily. Comprehensive information about parallel programming with CUDA can be found in [6], [7].

Our group has already presented parallelized and CUDA-based computations for image retrieval algorithms. In our approaches, CUDA threads efficiently mapped on different image blocks to extract the features based on color [8], texture [9], edge histogram [10] and so on [11]. Furthermore, in [12], GPU is used to accelerate a DCT-based steganography algorithm using an OpenCL implementation.

In this work, we select a color DCT watermarking algorithm proposed by Su et al. [4] to parallelize and efficiently implement on GPU using CUDA. Su's algorithm is one of most recent watermarking researches in the DCT domain that outperforms earlier color watermarking schemes in terms of robustness and invisibility. However, the algorithm has high computational complexity due to its time consuming steps such as color space transforming for host image, one-level DCT performing in 8×8 blocks for both host and watermark images, and two-level DCT performing in 4×4 blocks.

This paper unfolds as follows. GPU architecture and CUDA programming model are briefly described in Section II. Our parallelization strategies and modifications on Su's algorithm [4] are presented in Section III. Section IV brings experimental results and speedups obtained by our CUDA implementation. Concluding remarks are drawn in Section V.

II. NVIDIA GPU Architecture and CUDA Platforme

The GPU has a set of multiprocessor cores and each multiprocessor containing several scalar processors is capable of executing a very high number of threads concurrently. So each multiprocessor has Single Instruction, Multiple Threads (SIMT) architecture, and consequently a large number of threads run the same instruction, operating on different data elements in parallel. Which means Single Instruction, Multiple Data (SIMD) programming model can be realized by CUDA to implement the massive data-parallel programs on GPU. In fact for parallel processing of large data sets in CUDA architecture, many threads run concurrently and each thread processes a portion of the data.

The GPU has three different memory types: global memory, constant memory, and texture memory. Since a lot of concurrent threads have to read from and write to the memory at the same time, the GPU memory bandwidth is much higher than CPU. Global memory is slowest memory type and is used to data communication between host (CPU) and device (GPU).

In a CUDA program, a function that must be executed on GPU is called kernel. The same process on *N* different data elements is written in forms of a kernel to execute by *N* threads in parallel. The kernel is invoked by the host to run on the device. Programmer determines the number of threads by setting the number of threads per thread block and the number of blocks per grid. All threads within a block can cooperate among themselves by sharing data through some shared memory and synchronizing their execution to coordinate memory accesses. Each thread block is executed by only one multiprocessor, though the multiprocessor may be applied to execute one or several blocks. The dimensions of the grid and the thread blocks as well as the number of the elements in each dimension must be determined so to map to the data which should be processed by the GPU, achieving a high performance. However, constraints of the GPU model in terms of maximum the number of threads in each block and the number of processors in the system are limiting factors.

The multiprocessor partitions its thread blocks into groups of 32 parallel threads called warps. The threads composing a warp, have a property that should be considered in CUDA programming. When diverging a thread caused by data-dependent conditional branch, the warp including the thread executes each branch path serially. Consequently, the threads that are not on that path, are disabled. However, after completing paths, the threads converge back to the same execution path [6]. So in writing kernel code, data-dependent conditional branch should be avoided as possible to get more performance.

III. The Proposed CUDA-based Approach

To efficiently utilize many-core architecture of the GPU for the data-independent image processing applications, it is necessary that an appropriate organization of CUDA threads is taken, and input data is correctly mapped on the threads, so that a higher pixel-level parallelism is achieved. In fact, how to organize CUDA blocks and threads as well as map data on them has a great impact on overall performance. In this research, it is attempted that CUDA blocks and threads is matched with different parts of the image as possible. It is notable that the Su's watermarking algorithm [4] is slightly modified in order to get lower complexity and higher quality. In the following, the proposed parallelization strategies are separately described for DCT, color space transform and different steps of the watermarking algorithm.

A. DCT

DCT can be accounted as a core for the studied watermarking algorithm and can be a bottle neck. So, it is necessary that an efficient data parallel approach is utilized to implement DCT operation on GPU. To perform 2D-DCT on an image, the image is partitioned into 8×8 non-overlapping blocks and each of them is transferred by DCT operator as expressed by the following equation [4].

$$C(u,v)=\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos(\frac{\pi(2x+1)u}{2N})\cos(\frac{\pi(2y+1)v}{2N}) \quad (1)$$

$$\alpha(u)=\begin{cases}\sqrt{1/N}, & u=0\\ \sqrt{2/N}, & u\neq 0\end{cases}$$

Here, f(x,y) and C(u,v) are input and output matrices , respectively. Also, below equation depicts *N*×*N* 2D IDCT.

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cos(\frac{\pi(2x+1)u}{2N})\cos(\frac{\pi(2y+1)v}{2N}) \quad (2)$$

To efficient map input data on CUDA threads, dimensions of each CUDA block is set to 8×8. Also, dimensions of the grid are exactly determined based on number of 8×8 blocks in the image. For example, if size of input image is 256×512, then number of 8×8 independent blocks which are used for DCT operation is 32×64. Thus, size of the grid is set to 32×64, and size of the CUDA block is set to 8×8. Consequently, one thread is considered for each pixel, so that the thread can compute one output DCT coefficient. In this case, coordinate of the output DCT coefficient is exactly equal to thread coordinate and so mapping procedure is easy. If the coordinate of the output DCT coefficient is (*u*,*v*), then the coordinate is calculated as follows.

$$\begin{aligned} u &= \text{blockIdx.x}\times 8+\text{threadIdx.x}\\ v &= \text{blockIdx.y}\times 8+\text{threadIdx.y}\end{aligned} \quad (3)$$

Using this approach, computational complexity of 2D-DCT is reduced from $O(N^4)$ to $O(N^2)$. However, generally, for the threads considered for an ordinary image, there is not a sufficient number of cores available on the current GPUs. Also, the operating frequency of the GPU cores is generally lower than CPU. Furthermore, the data communication overhead reduces the GPU performance. Nevertheless, it is expected the GPU implementation offers a significant speedup due to the data-parallel nature of DCT application and its efficient mapping on CUDA threads.

In case of 4×4 2D-DCT, since a 4×4 region in the upper-left corner of 8×8 blocks is processed, size of gird is determined similar to 8×8 2D-DCT; however size of block is set to 4×4. Also, the coordinate of the DCT coefficient corresponding to the thread is obtained by equation 3. But, only the pixels placed in the 4×4 region are processed.

B. Color Space Transform

In the studied color image watermarking algorithm, the host image must be transformed from RGB color space to YIQ color space and vice versa. Transforming equations can be written as [4]:

$$\begin{aligned} Y &= 0.299\,\mathrm{R} + 0.587\,\mathrm{G} + 0.114\,\mathrm{B} \\ \mathrm{I} &= 0.596\,\mathrm{R} + (-0.275\,\mathrm{G}) + (-0.322\,\mathrm{B}) \\ \mathrm{Q} &= 0.211\,\mathrm{R} + (-0.523\,\mathrm{G}) + 0.312\,\mathrm{B} \end{aligned} \qquad (4)$$

$$\begin{aligned} R &= 1.000\,\mathrm{Y} + 0.956\,\mathrm{I} + 0.621\,\mathrm{Q} \\ \mathrm{G} &= 1.000\,\mathrm{Y} + (-0.272\,\mathrm{I}) + (-0.647\,\mathrm{Q}) \\ \mathrm{B} &= 1.000\,\mathrm{Y} + (-1.106\,\mathrm{I}) + (-1.703\mathrm{Q}) \end{aligned} \qquad (5)$$

For color space transform, each pixel in each output color component is a function of corresponding three pixels in input color components. So, if size of the input image is $M\times N\times 3$, then values of $M\times N\times 3$ pixels should be computed for the output image, in which computation of each output pixel is independent of others. Consequently, it is suggested that $M\times N\times 3$ threads are considered for calculation of $M\times N\times 3$ output pixels, in which each thread is responsible for one output pixel. Note that the total number of threads must be $M\times N\times 3$; however different thread organizations may be used. But, it is more efficient that the thread organization to be matched on size of the problem. Therefore, size of the grid is considered as 1×3, and size of each block is considered as $M\times N$. Consequently, each output color component is obtained by the threads of one CUDA block. The output pixels from the three color components are obtained by three different equations. But, all threads of the grid execute the same instructions. To solve this problem, according to the index of related CUDA block (blockIdx), the appropriate equation is selected. Therefore, all of threads within one CUDA block execute the same code and consequently there is no warp that its threads diverged from each other, showing superiority of proposed CUDA threads organization.

C. Embedding Procedure

Here, the proposed parallel implementation of the embedding procedure on GPU is explained step by step by means of the following pseudo code.

1- Host image is transferred from the main memory (CPU side) to the GPU memory. Then, the kernel explained in previous sub-section is invoked to transform the host image from RGB color space to YIQ color space.

2- The DCT kernel is invoked to perform 8×8 2D-DCT on component Y.

3- The DCT kernel with a new thread organization is again invoked to perform 4×4 2D-DCT, getting two-level DCT transferred coefficients of component Y.

4- Similar to the host image, one-level DCT is performed on the watermark image using CUDA parallel threads.

5- DCT transformed watermark image is compressed by a new kernel. Our compressing method is slightly different from one used in [4]. In Su's method [4], according to Zig-Zag order, the six coefficients (the DC coefficient and the first five AC ones) are selected and other coefficients are discarded. In the remaining coefficients, the first two are encoded with 16 bit binary, and the others are encoded with 8 bit binary, respectively. So, 64 bits are required to encode an 8×8 block in DCT domain. By DCT performing on 8×8 blocks of standard images used in most image processing algorithms, it can be found that the absolute value of DC coefficient has generally the greatest value among coefficients, the first two AC coefficients are generally have smaller values with approximately similar range, and other three AC coefficients have generally much smaller values. So, it is proposed to consider 14 bits for DC coefficient, 11 bits for the first two AC coefficients and 9 bits for the next three AC coefficients in Zig-Zag order. Thus, using this method, 63 bits are required to encode an 8×8 block in DCT domain. Also, experimental result shows that the proposed compress coding offers a higher PSNR than Su's compress coding (refer to Section IV).

After compressing procedure, a sequence of watermark bits is generated (64 bits per 8×8 block of the host watermark). In our GPU implementation, number of all threads is equal to number of 8×8 blocks of the DCT coefficients matrix and consequently each thread is responsible for generating watermark bits from one 8×8 block. The generated sequence is transmitted to the Main memory.

6- In the CPU side, the sequence of watermark bits is copied three times into a larger sequence. So, a sequence containing three similar sets of watermark bits which must be transferred to the GPU memory. Then, a kernel is invoked to embedding watermarks bits. In [4], it is described that 9 DCT coefficients of each 8×8 block are selected for embedding watermark information (refer to [4] for more information about embedding and extracting formula). So, for the GPU implementation, size of the grid will be equal to number of 8×8 blocks. Also, for each CUDA block, 9 threads are considered to embed a watermark bit from the sequence.

7- Parallel execution of 4×4 inverse 2D-DCT on resultant of the previous step should be performed using GPU cores, similar to step 3.
8- Also, 8×8 inverse 2D-DCT is performed.
9- Finally, the watermarked component Y along with component I and Q must be transformed to RGB color space, achieving the watermarked color image. The GPU implementation of this step is similar to step 1. The watermarked color image is written back to the main memory of CPU.

Thus, the proposed GPU implementation of the embedding algorithm has 9 kernels with various thread organizations, so that these kernels must be invoked by the CPU in serially.

D. *Extracting Procedure*

Following pseudo code depicts the proposed GPU implementation of extracting procedure. The strategies applied to most of steps have been explained in previous sub-section. Therefore, extra descriptions about the GPU implementation are prevented.

1- The watermarked image is transferred from the main memory to the GPU memory to transform from RGB color space to YIQ color space.
2- 8×8 2D-DCT is performed.
3- Also, for the two-level-DCT, 4×4 2D-DCT is performed.
4- Similar to the embedding kernel, size of the grid is equal to number of 8×8 blocks. Also, there are 9 threads per CUDA block to extract watermark bits. Generated watermark bits sequence is transferred to the main memory.
5- The sequence containing three sets of the watermark bits is split into three other sequences which must be transferred to the GPU memory. A kernel is invoked to recognize final sequence, based on majority principles. The total number of threads for this kernel is equal to number of watermark bits. Therefore, each thread yields one watermark bit as output, using three input bits.
6- In this step, a kernel should be invoked to reconstruct de-compressed matrix using the sequence. This matrix should be containing 8×8 blocks, so that each block is reconstructed using 64 bits taken from the sequence. So, number of all threads is equal to number of 8×8 blocks of the watermark image, so that each thread is responsible for reconstructing one 8×8 block, as previously described.
7- Finally, inverse 2D-DCT is performed on 8×8 blocks, to get extracted watermark image.

Thus, the proposed GPU implementation of the extracting procedure has 7 kernels with various thread organizations, so that these kernels must be invoked by the CPU in serially. In both procedures, it is attempted to get higher thread-level parallelism, efficient utilizing the GPU cores and achieving higher performance.

IV. Experimental Results

CUDA 6.5 was used to implement the presented data-parallel watermarking algorithm on the GPU. NVIDIA's Geforce GT 540M (1GB global memory and 64 cores with frequency of 1.3 GHz) and Geforce GTX 580 (2GB global memory and 512 cores with frequency of 1.4 GHz) were used as two GPUs with different number of cores. To compare performance of the parallel implementation with serial implementation on a General Purpose Processor (GPP), the watermarking algorithm was coded by using single thread C language and implemented on a PC with CPU Intel Pentium 4 running at 3.1 GHz.

In order to evaluate performance of the modified compressing method, the benchmark images depicted by Fig.1 were compressed by Su's method [4] and the proposed method described in the previous Section. Table I shows PSNR values for the benchmark images. PSNR is used to evaluate perceptual distortion of the compressing methods. It is observed that, the proposed method improves the PSNR value up to 21.99%, compared to Su's method [4], for the 5 benchmark images. Fig. 2 shows Lena as host, and Peugeot logo as watermark after the code compressing procedure. As can be seen, the logo due to its lower quality is more affected by both compression methods. However, the proposed method has reduced this distortion significantly.

To analysis the efficiency of the presented parallel approach, 8×8 2D-DCT was performed on different image sizes varying between 64×64 and 2048×2048 using the CPU and the two model GPUs, in which the results have been listed in Table II. It is notable that for more accurate calculation of the execution time, each work load is performed with 2000 iterations and the average time is considered as result.

Fig. 3 illustrates the achieved speedup by the GPUs for different image sizes. It can be seen, by the parallel implementation on GPU, up to 43x speedup for GT 540M as well as up to 105x speedup for GTX 580 are achieved, for different image sizes. The interesting note is significant reduction in speedup obtained by GT 540M, when the image size increases from 256×256 to 512×512. Even more interesting, there is not this speedup degradation for GTX 580. This is due to the fact that number of cores in GT 540M is only 64 and consequently, for high resolution images, inevitably, many of GPU threads are serially executed, reducing the performance.

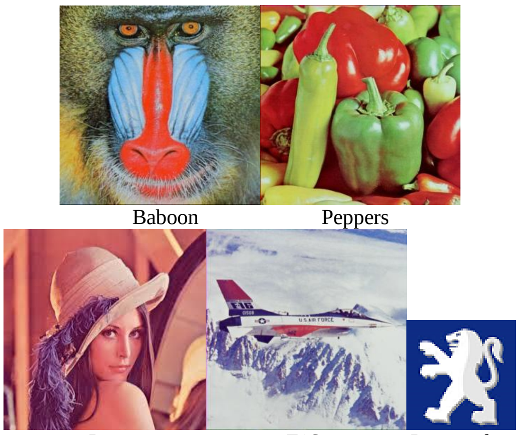

Fig.1. Benchmark images used in this work

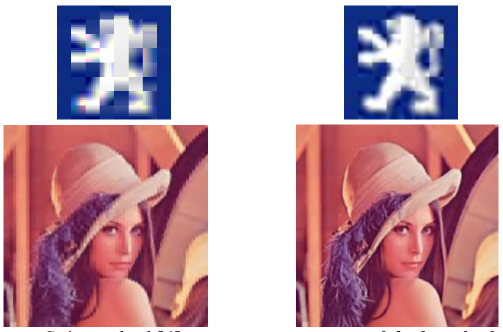

Fig.2. Compressed images using two methods

TABLE I. PERFORMANCE EVALUATION OF THE TWO IMAGE COMPRESSION METHODS USING PSNR

Image	**Size**	**PSNR (db)**		
		Su's method [4]	***Proposed method***	***Improvement percent***
Logo	64×64×3	60.2487	64.3611	6.82%
F16	256×256×3	62.4902	69.6833	11.51%
Lena	512×512×3	60.1162	76.9536	28.01%
Peppers	512×512×3	60.1177	73.3373	21.99%
Baboon	512×512×3	61.0714	65.5031	7.26%

TABLE II. EXECUTION TIME (SECOND) OF 8×8 DCT OPERATION

Image size	**Platform**		
	CPU	***GPU GTX 540M***	***GPU GTX 580***
64×64	0.025059	0.000587	0.000239
128×128	0.101197	0.002338	0.000956
256×256	0.34125	0.00780	0.00332
512×512	1.32481	0.10427	0.01317
1024×1024	5.17327	0.42391	0.05156
2048×2048	20.67854	1.67512	0.21956

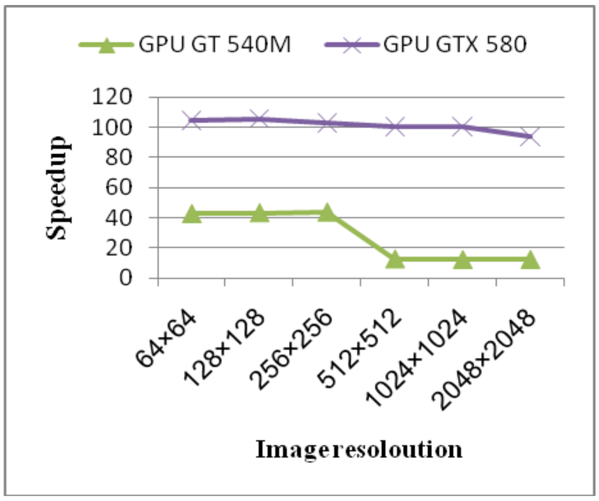

Fig. 3. Speedup of GPUs over the CPU, for 8×8 DCT operation

To evaluate the proposed parallelization method, various case studies with different image resolutions images (for host and watermark) are considered which are listed in Table III.

Table IV exhibits the execution time of the embedding procedure for the case studies on different target platforms. Furthermore, Fig. 4 displays the speedup obtained by the utilized GPUs. The achieved speedup for GT 540M is between almost 7x and 26x for various case studies. Moreover, the more power full GPU, (i.e. GTX 580) lead to a speedup between 46x and 73x.

Also, in case of the extracting procedure, Table V lists the execution times for the various case studies. Meanwhile, attained speedups are illustrated in Fig. 5. GT 540M offers a speedup between 10x and 29x, and GTX 580 offers a speedup between 75x and 80x, for the various case studies. It can be observed, the speedup is reduced when higher resolution images.

V. CONCLUSION AND FUTURE WORK

For different steps of the color DCT watermarking algorithm proposed by Su et al. [4], different parallelization approaches and efficient CUDA thread organizations were designed in this work. Furthermore, the compression method for color watermarks was modified, so that the PSNR values are improved up to 21.99%.

Implementing the proposed parallel approach on GPU GT 540M with 64 cores can reach up to 43x, 26x and 29x speedups, and GTX 580 with 512 cores can reach up to 105x, 73x and 80x speedups, for 8×8 DCT operation, embedding and extracting procedures, respectively. Data independent and the parallel pixel nature of the DCT operation and most steps of the watermarking algorithm leads to efficient utilize the many core architecture of the GPU, getting a significant speedup.

As future work, we investigate to design new hardware and parallel architectures for the DCT core and the watermarking algorithm. Also, with implementing the customize architectures on FPGA; we can compare performance of FPGA and GPU, as accelerator platforms.

Table III. Case studies used for the watermarking algorithm

Case study number	Host image resolution	Watermark image resolution
1	128×128	16×16
2	256×256	32×32
3	512×512	64×64
4	1024×1024	128×128
5	2048×2048	256×256

TABLE IV. EXECUTION TIME (SECOND) OF THE EMBEDDING PROCEDURE

Case study number	Platform		
	CPU	*GPU GTX 540M*	*GPU GTX 580*
1	0.186721	0.007284	0.002538
2	0.731552	0.028884	0.009817
3	2.90308	0.0127640	0.043675
4	11.55045	1.467805	0.207061
5	45.93954	6.361712	0.997195

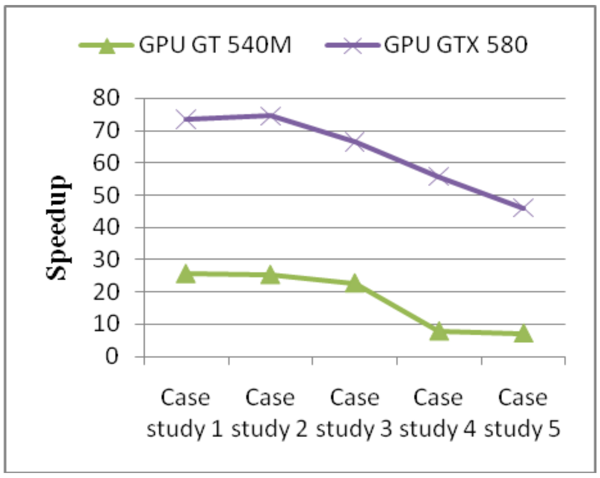

Fig. 4. Speedup of GPUs over the CPU, for the embedding procedure

TABLE V. EXECUTION TIME (SECOND) OF THE EXTRACTING PROCEDURE

Case study number	Platform		
	CPU	*GPU GTX 540M*	*GPU GTX 580*
1	0.115280	0.003899	0.001429
2	0.451655	0.015567	0.005615
3	1.79234	0.062553	0.022567
4	7.13116	0.656372	0.089960
5	28.37651	2.863219	0.377547

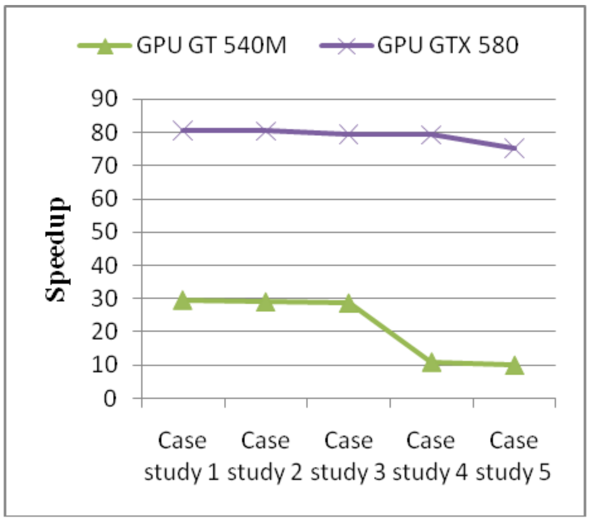

Fig. 5. Speedup of GPUs over the CPU, for the extracting procedure

REFERENCES

[1] S. Fazli and M. Moeini, “A robust image watermarking method based on DWT, DCT, and SVD using a new technique for correction of main geometric attacks,” Optik - International Journal for Light and Electron Optics, vol. 127, no. 2, January 2016, pp. 964–972.

[2] T.K. Tsui, X.-P. Zhang and D. Androutsos, “Color image watermarking using multidimensional fourier transforms”, IEEE Transaction on Information Forensics and Security, vol. 3, no. 1, March 2008, pp. 16–28.

[3] S.D. Lin and C.-F. Chen, “A robust DCT-based watermarking for copyright protection”, IEEE Transactions on Consumer Electronics, August 2000, vol. 46, no. 3, pp. 415-421.

[4] Q. Su, Y. Niu, X. Liu and T. Yao, “A Novel Blind Digital Watermarking Algorithm for Embedding Color Image into Color Image”, Optik - International Journal for Light and Electron Optics, vol. 124, no. 18, September 2013, pp. 3254–3259.

[5] M. Barni, F. Bartolini and A. Piva, “Multichannel Watermarking of Color Images”, IEEE Transaction on Circuits and Systems for Video Technology, vol. 12, no. 3, March 2002, pp. 142–156.

[6] NVIDIA Corporation, NVIDIA CUDA C Programming 5.0, 2012.

[7] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[8] H. Heidari, A. Chalechale and A. Ahmadi, “Parallel implementation of color based image retrieval using CUDA on the GPU”, International Journal of Information Technology and Computer Science (IJITCS), vol. 6, no. 1, December 2013, pp. 33-40.

[9] H. Heidari, A. Chalechale and A. Ahmadi, “Parallel implementation of texture based image retrieval on The GPU”, International Journal of Image, Graphics and Signal Processing, vol. 5, no. 9, July 2013, pp. 36-42.

[10] A. Ahmadi, A. Chalechale and H. Heidari, “Parallelized computation for Edge Histogram Descriptor using CUDA on the Graphics Processing Units (GPU)”, 17th CSI International Symposium on Computer Architecture and Digital Systems (CADS 2013), Tehran, 2013, pp. 9-14.

[11] A. Ahmadi, A. Chalechale and H. Heidari, “GPU implementation of edge histogram descriptor and color moments fused features for efficient image retrieval”, The CSI Journal on Computer Science and Engineering, vol. 9, no. 2, 2013, pp. 22-33.

[12] A. Poljicak, G. Botella, C. Garcia, L. Kedmenec, M. Prieto-Matias, “Portable real-time DCT-based steganography using OpenCL”, Journal of Real-Time Image Processing, special issue, July 2016, pp. 1-13.