A Parallel **Watermarking** application on a GPU

García-Cano C. Edgar,
– Rabil Bassem S. – Sabourin Robert
Universidad Nacional Autónoma de México - Universidad de Quebec – Canada.

Para citar este artículo / To reference this article / Para citar este artigo García, C. E, Rabil, B. S, Sabourin, R (2012). A parallel watermarking application on a GPU. Ingenio Magno. Vol.3, pp. 6 - 15. Universidad Santo Tomás Tunja - CIIAM.

Recepción: 2012-07-13 I Aceptación: 2012-09-10

Resumen— Debido al gran volumen de información que fluye a través de Internet, las marcas de agua se utilizan ampliamente para proteger la autenticidad e integridad de la información. La inserción y la extracción de marcas de agua se pueden hacer en el dominio espacial o de otros dominios de frecuencia, como la Transformada Discreta del Coseno (DCT) y la Transformada Discreta Wavelet (DWT). La inserción y la extracción en dominios como DCT tienen un gran costo computacional en comparación con los métodos espaciales. Sin embargo, el proceso de marcas de agua en el dominio de la frecuencia tiene mejores resultados en calidad y robustez debido al uso de coeficientes no correlacionados. En este trabajo, se propone utilizar una unidad de procesamiento gráfico (GPU) para reducir el costo computacional de la inserción y extracción de los bits de la marca de agua en el dominio de DCT. Se propone, para tomar ventaja de los bloques generados después de la DCT, asignar la misma configuración de bloques en la GPU. También se hace uso de los diferentes tipos de memoria, como la constante y compartida, para optimizar el uso de los recursos del GPU. Los experimentos evalúan el desempeño de la marca de agua en la GPU, y muestran que el algoritmo que se ejecuta en la GPU es hasta 6 veces más rápido en comparación con el ejecutado en el CPU, aun tomando en consideración el tiempo que lleva transferir datos desde la memoria RAM a la memoria de la GPU.

Palabras clave— CUDA, GPU, watermarking, authenticate the information, copyright, invisible watermark.

Abstract— Due to the vast volume of information flowing on the Internet, watermarking is widely used to protect information authenticity and integrity. Watermarking embedding and extraction can be done in spatial domain or other frequency domains like Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). The embedding and extraction process for domains like DCT have higher computational cost compared to spatial based methods. However the frequency domain watermarking results in better watermark quality and robustness due to the use of uncorrelated coefficients. In this paper, it is proposed to utilize a Graphics Processing Unit (GPU) to reduce the computational cost of the insertion and extraction of the watermark bits using DCT domain. It is also proposed to take advantage of the blocks generated after the DCT to map them into the same configuration of blocks on the GPU. The proposed approach uses different types of memory, the constant and shared, in order to optimize the GPU's resources. Experiments evaluate the performance of the watermarking in the GPU, and show that the algorithm running in the GPU is up to 6 times faster compared to CPU implementation, even considering the time taken for transferring data from the RAM memory to the GPU memory.

Keywords— CUDA, GPU, watermarking, authenticate the information, copyright, invisible watermark.

I. Introduction

Nowadays with the use of Internet it is important —as common users— to have a way to protect our information such as documents, videos, images, music, etc. In the industry it is important to protect the authenticity of their documents with copyright in order to secure its information and the one from consumers or customers; here is where the watermarks make sense as a way to protect it.

Computers have been improved in order to compute a huge quantity of information; however, it is kind of difficult to have easy access to supercomputers or clusters to process it; that is why the massive parallelism on Graphics Processing Units (GPU) for general-purpose problems has arrived as a cheap and feasible solution to accelerate the process. The Compute Unified Device Architecture (CUDA architecture) created by NVIDIA, has arrived with the objective of accelerate general-purpose problems taking advantage of the massive parallelism in its new paradigm (NVIDIA, 2012).

Thread level parallelization model is based on the idea of having as many threads as possible working continually in order to take advantage of the GPU resources. Using this scheme, the proposed system takes one image, splits it in accordance with the DCT algorithm, and maps it to the GPU's global memory. From here, different GPU execution configurations are used to execute the operations needed for the watermarking algorithm, depending on the requirements of each one; besides the shared memory is applied to avert over-reading into the global memory.

The main objective of the proposed system is to handle a huge quantity of digital information; specifically images in gray scale JPG format, to authenticate the information using a watermarking algorithm, and to process it faster sing modest computational resources and utilizing a GPU to accelerate this process.

The paper is organized as follows: section 2 gives an explanation about what watermarking is, its different types, the metrics to evaluate the quality of a watermarked image, and previous work in watermarking images. In section 3, the GPU watermarking proposed system is presented. Section 4 explains the experimental study and results. Finally, the conclusions are presented in section 5.

EL proyecto de investigación consiste en la búsqueda y selección adecuada y prueba de técnicas, herramientas, protocolos y demás recursos tecnológicos que permitan modelar el sistema teniendo en cuenta los recursos temporales, físicos, servicios humanos y el impacto social.

II. Related Work

In images, a watermark is a pattern inserted in the image that helps to copyright it. There are two ways to apply the watermarking in images. The first one is called visible watermark. The characteristic of this type is that you can see the watermark over the image like a logo or a sign; it is common to see images on Internet where you can see logos or signs over them, or in TV shows it is usual to see the broadcaster's logo. The second one is the invisible watermarking. This type cannot be perceived by human eyes, and to insert or extract it is necessary to use electronic devices (AlpVision, 2012).

There have been other works in which GPUs have been used in watermarking, like the ones of Brunton & Zhao, (2006), Zhao & Yang (2011), and Vihari & Mishra (2012). The first one was applied for real-time video watermarking, while the other two were applied to image watermarking. Zhao & Yang used features extracted from the low and middle frequency domain of DCT coefficients to embed them into the high frequency domain, and Vihari & Mishra's work based on Huffman Coding for encoding the copyright data, and to embed it they made use of the Modified Auxiliary Carry Watermarking method.

A. Watermarking Metrics

In the digital framework, watermarking algorithms that make use of information hiding techniques have been developed, and hiding capacity has naturally been used as a metric in evaluating their power to hide information (the maximal amount of information that a certain algorithm can "hide" keeping the data within allowable distortion bounds). In literature, the next metrics had been used in order to evaluate the quality of the watermarking (Ramesh, Shanmugam, & Gomathy, 2011). This paper is focused on the parallel implementation of a watermarking algorithm on GPUs, and the most commonly used metrics to evaluate it. This work could be extended further in the future for more metrics.

1) Watermark Fidelity

The fidelity represents the similarity of the watermarked image with the original image. Peak Signal to Noise Ratio (PSNR) is commonly used to evaluate image degradation or reconstruction fidelity (National Instruments , 2012). It is defined for two images I and K of size MxN as:

$$PSNR(I,K) = 10\ log_{10} \frac{255^2}{\sqrt{MSE(1,K)}} \quad (1)$$

Where I is the original image, K is a reconstructed or noise approximation, 255^2 is the maximum pixel value in image I, and MSE is a mean squared error between I and K.

$$MSE\ (I,K) = \frac{1}{M}\frac{1}{N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} \| I(i,j) - K(i,j) \|^2 \quad (2)$$

PSNR is expressed in decibel scale. In image reconstruction typical values for PSNR vary within the range [30, 50]. A PSNR value of 50 and higher calculated from two images that were processed on diverse devices with the same algorithm indicates that the results are practically identical.

2) Watermark Robustness

The robustness represents the resistance of the watermark against attacks like compression, rotation, scaling, etc., done on the watermarked image. The Normalized Correlation (NC) is used to measure the robustness between the original watermark and the extracted watermark. When different attacks have been applied to a watermarked image, the NC is calculated between the embedded watermark W(i,j) and the extracted watermark from the attacked image W'(i,j), where both watermarks have the same dimensions $M_{wx}N_{w.}$

$$NC = \frac{\sum_{i=1}^{Mw}\sum_{j=1}^{Nw}[W(i,j)\ W'(i,j)]}{\sum_{i=1}^{Mw}\sum_{j=1}^{Nw}[W(i,j)]^2} \quad (3)$$

B. Watermark Embedding/Extraction Algorithm

In order to develop the application, it was decided to implement a watermarking algorithm proposed by Shieh, Huang, Wang, & Pan, (2004). This algorithm is used because it is a blind algorithm, which means that it is not necessary to have the original cover to extract the watermark. This algorithm is based on the Discrete Cosine Transformation (DCT). The DCT divide the picture in blocks of 8x8 with well-defined frontiers, this feature helps to map every block of the image in a block on the GPU to process over it.

Initially to insert the watermark, the image X of size MxN to be watermarked is split into 8 x 8 blocks to perform DCT on these blocks, generating the matrix $Y_{(m,n)}(k)$ This resultant matrix has the upper left corner, as DC coefficient and the rest of the matrix are the AC coefficients, where the DCT coefficients are zigzag ordered as seen in figure 1.

$Y_{(m,n)}(0)$	$Y_{(m,n)}(1)$	$Y_{(m,n)}(5)$	$Y_{(m,n)}(6)$	$Y_{(m,n)}(14)$	$Y_{(m,n)}(15)$	$Y_{(m,n)}(27)$	$Y_{(m,n)}(28)$
$Y_{(m,n)}(2)$	$Y_{(m,n)}(4)$	$Y_{(m,n)}(7)$	$Y_{(m,n)}(13)$	$Y_{(m,n)}(16)$	$Y_{(m,n)}(26)$	$Y_{(m,n)}(29)$	$Y_{(m,n)}(42)$
$Y_{(m,n)}(3)$	$Y_{(m,n)}(8)$	$Y_{(m,n)}(12)$	$Y_{(m,n)}(17)$	$Y_{(m,n)}(25)$	$Y_{(m,n)}(30)$	$Y_{(m,n)}(41)$	$Y_{(m,n)}(43)$
$Y_{(m,n)}(9)$	$Y_{(m,n)}(11)$	$Y_{(m,n)}(18)$	$Y_{(m,n)}(24)$	$Y_{(m,n)}(31)$	$Y_{(m,n)}(40)$	$Y_{(m,n)}(44)$	$Y_{(m,n)}(53)$
$Y_{(m,n)}(10)$	$Y_{(m,n)}(19)$	$Y_{(m,n)}(23)$	$Y_{(m,n)}(32)$	$Y_{(m,n)}(39)$	$Y_{(m,n)}(45)$	$Y_{(m,n)}(52)$	$Y_{(m,n)}(54)$
$Y_{(m,n)}(20)$	$Y_{(m,n)}(22)$	$Y_{(m,n)}(33)$	$Y_{(m,n)}(38)$	$Y_{(m,n)}(46)$	$Y_{(m,n)}(51)$	$Y_{(m,n)}(55)$	$Y_{(m,n)}(60)$
$Y_{(m,n)}(21)$	$Y_{(m,n)}(34)$	$Y_{(m,n)}(37)$	$Y_{(m,n)}(47)$	$Y_{(m,n)}(50)$	$Y_{(m,n)}(56)$	$Y_{(m,n)}(59)$	$Y_{(m,n)}(61)$
$Y_{(m,n)}(35)$	$Y_{(m,n)}(36)$	$Y_{(m,n)}(48)$	$Y_{(m,n)}(49)$	$Y_{(m,n)}(57)$	$Y_{(m,n)}(58)$	$Y_{(m,n)}(62)$	$Y_{(m,n)}(63)$

Figure 1. The matrix of the zigzag ordered DCT coefficients. Each $Y_{(m,n)(k)}$ is a frequency band where the watermark bits could be inserted.

The transformed matrix $Y_{(m,n)}(k)$ is then used to get the ratio matrix between the DC and the AC coefficients R(i) using (4).

$$R(i) = \sum_{m=1}^{M/8} \sum_{n=1}^{N/8} \left(\frac{Y_{m,n}(0)}{Y_{m,n}(i)} \right), i \in [i, 63] \quad (4)$$

Then the polarities matrix $P_{(m,n)}(i)$ is calculated applying (5).

$$P_{(m,n)}(i) = \begin{cases} 1 & if \left(Y_{(m,n)}(i) \cdot R(i) \right) \geq Y_{(m,n)}(0), i \in F; \\ 0 & otherwise \end{cases} \quad (5)$$

Next, assuming that the binary watermark is W(m,n), the watermarked DCT coefficient Y ' is obtained using (6).

$$Y'_{(m,n)}(i) = \begin{cases} Y_{(m,n)}(i) & if\ P_{(m,n)}(i) = W_{(m,n)}(i) = 0, i \in F; \\ \left(Y_{(m,n)}(0) / R(i) \right) + 1 & if P_{(m,n)}(i) = 0, W_{(m,n)}(i) = 1, i \in F; \\ Y_{(m,n)}(i) & if\ P_{(m,n)}(i) = W_{(m,n)}(i) = 1, i \in F; \\ \left(Y_{(m,n)}(0) / R(i) \right) - 1 & otherwise \end{cases} \quad (6)$$

After that, the watermarked image X_c is obtained by using the inverse DCT for Y'.

When extracting the watermarks, the original image X is not required in our algorithm. However, the optimized watermarked image might be subjected to some intentional or unintentional attack, and the resulting image after the attack is represented by X". We calculate the DCT of the watermarked image after attacking Y". We then reproduce the estimated reference table R' from the attacked X" by following the operations in (7), and we are able to extract the watermark $W'_{(m,n)}$,

$$W'_{(m,n)}(i) = \begin{cases} 1 & if \left(Y''_{(m,n)}(i) \cdot R'(i) \right) \geq Y''_{(m,n)}(0), \forall i; \\ 0 & otherwise \end{cases} \quad (7)$$

$$W'_{(m,n)} = \bigcup_{m=0}^{M/M_{W-1}} \bigcup_{n=1}^{N/N_{W-1}} \left(\frac{Y_{m,n}(0)}{Y_{m,n}(i)} \right), i \in F \quad (8)$$

This watermarking algorithm has many computationally complex processing steps just for one image; however in real life it is crucial to be able to work with a huge quantity of files, and to process them as soon as possible to satisfy the consumers. In order to solve these requirements, CUDA architecture becomes a solution to accelerate the process at a low cost. Figure 2 shows the general process steps of the algorithm proposed by Shieh et al. (2004) to embed and extract the watermark.

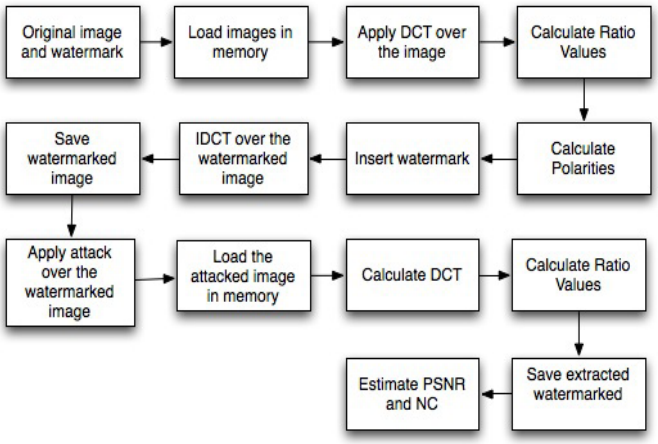

Figure 2. Steps of the embedding and extraction algorithm proposed by Shieh et al. (2004).

III. GPU Based watermarking

The system is based on the idea of parallelizing all the operations involved in the watermarking algorithm using CUDA architecture, in order to accelerate the process.

Using an image of size 512x512 as an input, it is possible to divide it in 64x64 blocks as in the DCT. The 64x64 matrix is easily mapped to the same number of blocks in the GPU, and the configuration of the threads will depend on the type of operation to be executed. The limitation with the image is the size, due to the fact that the GPU

has a limitation in the data quantity that it can store in the different type of memories.

Respecting the thread configuration, for example, to compute one operation that requires comparing a [512x512] watermarked image with the original one; in a sequential form, if every comparison consumes 1 second to be executed 262,144 seconds would be required to complete the comparison. Using a GPU, 64x64 blocks are generated, and each of them have 64 threads, in total there are 262,144 threads working in parallel doing one comparison, consuming just 1 second to complete all of them (theoretically).

CUDA architecture is based on block components called streaming multiprocessor (SM), and depends on the number of them; you can create applications that could run on hardware with different prices and performances. The scheduler is the responsible of assigning one or more blocks of threads on each SM, depending on how many blocks and threads a SM can support; this is called compute capability. It ought to be taken in account that the number precision representation changes with the different compute capabilities, which can be seen reflected in the outcomes.

The code that is going to run in parallel on the device needs to be written in functions called kernels. These functions indicate to the compiler that the code will run on the GPU. The kernels are executed in the order in which they were launched, and if the SMs are free to work in their next duty. The way the kernel has to be coded to take advantage of the parallelism depends on execution configuration; this is where blocks and threads are set out in order to have done its work (Sanders & Kandrot, 2010). Therefore, the configuration of the blocks and threads for an application on a GPU must be carefully analyzed.

Other point of consideration in the use of the GPUs is the memory treatment. In this application the global memory was used to put up the image and the watermark data, the ratio and polarities matrices. This memory is used to carry the data from the host (RAM memory) to the device (GPU memory) and vice versa. The problem of using it is the long time it spends in the transfer depending on the amount of data. Another type of memory used in this application was the shared memory. This memory is used just inside the blocks and it is not visible between others, which, unlike the global memory is visible for all the blocks. The shared memory is faster than the global memory; the problem with it is the size, the handling and the overall synchronization with the threads.

Following with the previous explanation, a 128x128 binary watermark image to be inserted into a 512x512 gray scale image is considered. It is necessary to load the image into the GPU memory and to apply the DCT. In order to take advantage of the parallelism, a library with this function provided by CUDA was used (Obukhov & Kharlamov, 2008). After applying the DCT to the 512x512 image, a matrix of 64x64 blocks representing the image is obtained. Each block is divided at the same time into 8x8 frequency bands where the watermark will be inserted. The configuration of 64x64 blocks is maintained in the GPU for all the operations, each block in the GPU represents one block of the image after the DCT; what differs in the GPU is the configuration of the threads that depends on the need of the operation to be executed. For instance, in the calculation of the NC there were required just 4 threads to do the comparisons, but in the case of the MSE 64 threads working at the “same time” were required.

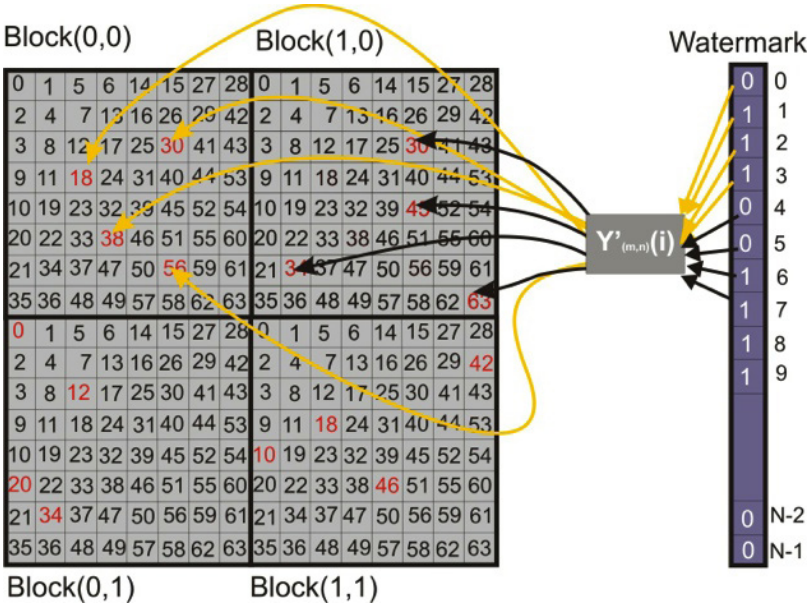

Figure 3. The watermarked image, and the watermark mapping to GPU.

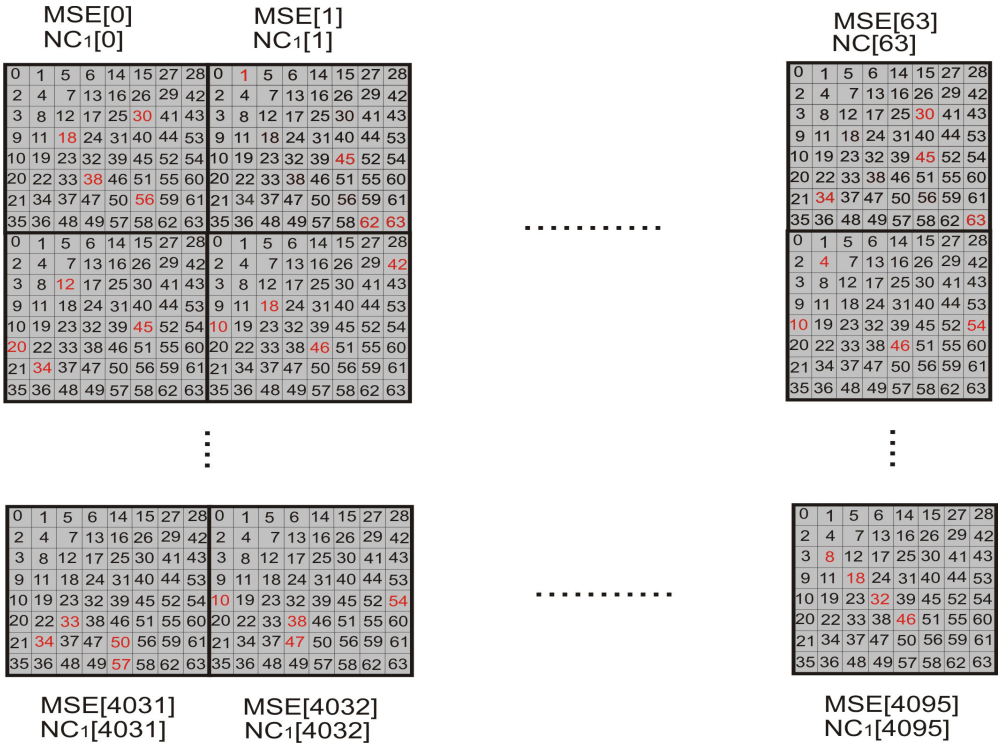

Figure 4. Image block organization.

As shown in figure 3, the data of the watermark image are inserted uniformly in the original image and in different frequency bands in order to have the new image with the hidden watermark. These bands are randomly selected and they could be different from block to block resulting with different MSE and NC values from test to test. NVIDIA provides a library used to generate the random numbers in the GPU, and to keep them in the constant memory in order to be used by any process (NVIDIA, 2010).

Finally, to complete the process it is important to implement a barrier to be sure that the routines have finished their work in the GPU; using synchronization helps to be sure that the process is going to be finished. On the other hand, CUDA runtime system is in charge of assigning the resources used for the different blocks to be executed. Putting together these features, CUDA provides the flexibility to run the implementation with different hardware resources, being aware of the memory and cores limitations of low-cost GPUs, which could affect the performance of the application. Figure 5 shows the steps to execute the embedding/extraction algorithm on the GPU.

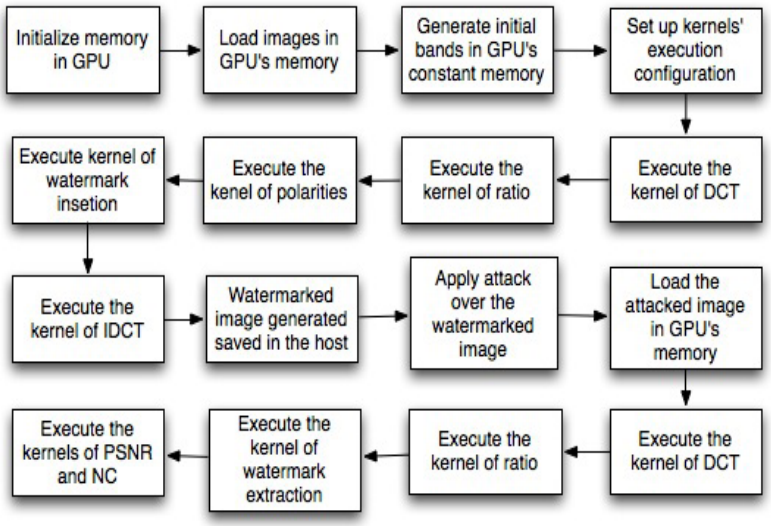

Figure 5. Steps of the embedding and extraction algorithm proposed by Shieh et al. (2004) on the GPU.

IV. Experimental Study

In order to propose a simpler way to measure the fitness and the robustness spending the shortest time possible, the MSE was taken from the PSNR and the NC was changed. When measuring the MSE in each block, just 64 comparisons are needed and they are executed at the "same time" in the other blocks. In the sequential process there are needed

512x512 evaluations, one after another for a 512x512 image size. The same case was applied for the NC, where it is computed for each block, instead of being calculated for the whole image as in a sequential form.

The NC and the MSE are computed for each 8x8 block as shown in figure 4. This was done with the purpose of dividing the data in the GPU as efficiently as possible. In order to calculate the fidelity, it is necessary just to compare block by block how much the original image changes in contrast with the watermarked one. If the MSE value is zero, then it means that the block has not changed at all. As you can see, it is not necessary to calculate the PSNR if it is possible to obtain the same image fidelity calculation by just using MSE.

In the case of NC (for robustness), a variation of it was calculated. The bitwise operations are faster than a multiplication, which is why applying one of them reduces the runtime. In order to reduce the runtime in the evaluation of the NC, the logical operation "exclusive disjunction", also called exclusive or was used. The NC value must be close to zero between the original watermark W and the extracted watermark W', to prevent the loss of the watermark image information.

Figure 4 shows in big scale how the blocks of the image after the DCT are organized. For each 8x8 block, the MSE and the NC are calculated. If the MSE and the NC values are close to zero, it is an indication that there is a good frequency band set to insert the watermark image into the corresponding 8x8 blocks.

A. Test and results

The tests were executed in one server which features are shown in table 1. The operating system used in the server is 64 bits (x86_64) Ubuntu 10.10 running a 2.6.35-30-generic Linux kernel. The server uses a 4.0 CUDA driver version, 3.2 CUDA runtime version, and a 1.3 CUDA capability version. For the GPU implementations, the NVCC compiler of the CUDA development toolkit has been used.

Table 1 Servers' features.

Server name	CPU	GPU
Geogpus	8 Intel Xeon E5677 with: • 4 cores, 3.47GHz • 12 GB RAM	1 Tesla C1060 with: • 240 cores, 1.3 GHz • 4 GB RAM

The CUDA host code was compiled by NVCC using the system's GCC version 4.4.2.

1) Input data

In order to test the implementations, the grey scale 512x512 of "Barbara" test image shown in figure 6 is used as cover image to embed the watermark logo shown in figure 7.

Figure 4. Original image (Barbara).

Figure 5. Watermark image

Table 2 shows the outcomes of executing sequential and CUDA implementations using Barbara image. The first tables show the results of executing five experiments, and taking the runtime for each function involved in the insertion and extraction algorithm.

These experiments were executed with the aim of comparing the runtimes between the implementation in C++ and the one in CUDA C, based on the idea that the operations executed in the GPU must be faster than the ones computed in the CPU.

The results obtained from the GPUs in both servers are faster than the ones collected from the CPU. At this point, the results seem to fit in the idea that the GPU is faster than the CPU. It should be noted that the functions are not considering the load and download of the data to and from the GPU.

Table 2 Runtime for functions involved in the insertion/extraction algorithm (using Barbara image).

GEOGPUS (sequential)

	DCT (ms)	RATIO (ms)	POLARITIES (ms)	IDCT(ms)	INSERTION (ms)	EXTRACTION (ms)
1	0.18646	0.00388	0.00032997	0.018899	0.001003	0.000041962
2	0.18629	0.003881	0.00033188	0.01913	0.0010269	0.000041008
3	0.18733	0.0038848	0.00033116	0.018871	0.0010052	0.000041008
4	0.18609	0.003866	0.00032997	0.018895	0.001003	0.0000422
5	0.1882	0.00385	0.00032997	0.018888	0.0010071	0.000040054
	0.18687400	0.00387236	0.00033059	0.01893660	0.00100904	0.00004125

GEOGPUS (CUDA)

	DCT (ms)	RATIO (ms)	POLARITIES (ms)	IDCT (ms)	INSERTION (ms)	EXTRACTION (ms)
1	0.0002141	0.0019349	0.000000051212	0.000051975	0.00014806	0.000082493
2	0.00021601	0.001907	0.000000050902	0.000051975	0.00010586	0.000085115
3	0.00021505	0.0016931	0.000000050592	0.000051022	0.0001049	0.000081062
4	0.00021482	0.001937	0.000000049901	0.000051022	0.00014687	0.000082803
5	0.00021505	0.0016671	0.000000049806	0.00005175	0.00010395	0.000082016
	0.00021501	0.00182782	0.00000005	0.00005159	0.00012193	0.00008270

Complexity reduction (sequential / CUDA)

869.16	2.12	6,548.59	367.03	8.28	0.50

The line in black shows the acceleration of GPU implementation compared to CPU for different processes involved in the watermark embedding/extraction.

Table 3 Runtime of the insertion and the MSE, and the extraction and the NC operations (using Barbara image).

GEOGPUS (sequential)

	INSERTION OP.	MSE	EXTRACTION OP.	NC
1	0.23152	0.002527	0.22152	0.0061359
2	0.22963	0.0025148	0.22486	0.0055099
3	0.23099	0.0025282	0.22162	0.0056429
4	0.22983	0.0025229	0.22303	0.0060408
5	0.22824	0.002522	0.22373	0.0061831
	0.23004200	0.00252298	0.22295200	0.00590254

GEOGPUS (CUDA)

	INSERTION OP.	MSE Total	MSE	EXTRACTION OP.	NC Total	NC
1	0.019511	0.012955	0.0000088215	0.038799	0.00579	0.0000081062
2	0.019452	0.012958	0.0000097752	0.037034	0.0064609	0.0000078678
3	0.019466	0.014669	0.0000088215	0.038534	0.0058	0.0000078678
4	0.019507	0.012967	0.000010014	0.037186	0.005774	0.0000081062
5	0.019937	0.013699	0.000010967	0.037502	0.0058062	0.0000081062
	0.01957460	0.01344960	0.00000968	0.03781100	0.00592622	0.00000801

Complexity reduction (sequential / CUDA)

11.75	0.19	609.77	5.90	1.00	736.82

Table 3 shows the runtime of the complete procedure to insert and extract a watermark involved in Shieh algorithm. In these experiments the upload and download of the data are considered. GPU does not seem much superior considering the results of the last tables. The MSE and the NC functions (see MSE and NC in the table) executed on the GPU without considering the data transfer seems to be fast, but considering the data transfer are more expensive than the ones executed in the CPU (see MSE Total and NC Total in the table).

The line in black shows the time spent to compute the complete watermarking process. The times are lower than the ones in table 2, due to the fact that the upload and download of data on the GPU are considered. The implementation with CUDA maintains the dominance in the minor runtime in respect to the C++ implementation, except for the MSE operation that spends more time in the GPU.

Table 4 Represents the average of the time in tables 2 and 3, and the acceleration in the process. The row A exemplifies the difference in runtime of the operations when the transfer of data to and from the GPU is not taking in account. Row B shows the same runtime operation but considering the data transfer.

	CPU (ms)	GPU (ms)	Acceleration (times)
A	0.21106384	0.00229910	91.80
B	0.46141950	0.07676142	6.01

Table 4 shows the acceleration, that in the case of the results in table 2, the GPU is 91.80 times faster than the CPU, but taking into account the upload and download of the data on the GPU, the process is just 6.01 times faster in the GPU than in the CPU. The measured accuracy for fitness evaluation is evaluated to be 5% due to changes of numerical representation between GPU and CPU implementation.

This paper is not focused on how to improve or optimize the watermarking algorithm, but in to accelerating the process. In order to calculate the accuracy of the GPU implementation compared to CPU results, there were generated the random bands where the watermark was going to be inserted. These sequences of bands were used to test the GPU and CPU code implementations.

V. Conclusion

With the vast volume of information flowing on the Internet, watermarking is widely used to protect this information authenticity. The need for copyright a huge quantity of digital files, spending the less possible amount of time and avoiding the loss information were the reasons to propose the use of an accelerated version of the watermarking algorithm proposed by Shieh et al. (2004), using CUDA architecture.

The use of a GPU for accelerating the operations involved in the algorithms of insertion and extraction of the watermarking algorithm was a challenge, since it is a parallelism paradigm. There is not a standard configuration for the blocks, threads or the memory treatment in the GPU. That is why the analysis and design of the procedures are a requirement needed to take full advantage of the parallelism. In order to use parallel programming in a GPU, it is necessary to shift from a sequential to a parallel thinking, strictly learning how to divide a huge problem into small ones —divide and conquer— attempting to obtain the best performance.

As shown in the experiments, the runtime of the functions are fastest in the GPU without considering the data transfer. Considering it, sometimes the function spends more time than the sequential execution. In the case of the Shieh algorithm, the required equations to be parallelized were analyzed to get the best performance on the GPU. For the calculation of the MSE and NC there was not an improvement of the performance compared with the sequential version. The execution of the functions is fast, but the transfer of the data to the GPU and back slows down the performance. For this reason it is necessary to seek for another solution for the transaction of the data.

On the other hand, the use of optimization techniques such as Genetic Algorithms, Bioinspired Algorithms, etc. is widely recommended in order to improve the outcomes. These algorithms based on population have the ease to be parallelizable and, combined with the embedding and extracting algorithm would provide a robust application to verify the authenticity of digital image files (García-Cano, 2012).

VI. Acknowledgements

The first author, student at the Posgrado en Ciencia e Ingeniería de la Computación of the Universidad Nacional Autónoma de México, wants to express his gratitude to the support received from CONACYT (scholarship number 37617) and PAPIIT (project number IN400312). This work is also supported by BancTech Inc.

REFERENCIAS

1. AlpVision. (2012). Digital watermarking. Retrieved from http://www.alpvision.com/watermarking.html
2. Brunton, A., & Zhao , J. (2006). Real-time video watermarking on programmable graphics hardware. Information Engineering and Computer Science (ICIECS), 1312 - 1315 .
3. Farber, R. (2011). CUDA Application Design and Development (1st ed.). USA: Morgan Kaufmann.
4. García-Cano, E. (2012). A parallel bioinspired watermarking algorithm on a GPU. Posgrado en Ciencias e Ingeniería de la Computación, UNAM.
5. Mohanty, S., Pati, N., & Kougianos, I. (2007). A Watermarking Co-Processor for New Generation Graphics Processing Units. International Conference on Consumer Electronics, 2007 .
6. National Instruments . (2012). Peak Signal-to-Noise Ratio as an Image Quality Metric. Retrieved from http://www.ni.com/white-paper/13306/en
7. NVIDIA. (2010). CUDA Toolkit 4.2 CURAND Guide. (N. Corporation, Ed.)
8. NVIDIA. (2012). NVIDIA CUDA C Programming Guide, version 4.2. (N. Corporation, Ed.)
9. Obukhov, A., & Kharlamov, A. (2008). Discrete Cosine Transform 8x8 Blocks with CUDA. USA: NVIDIA Corporation.
10. Ramesh, S., Shanmugam, A., & Gomathy, B. (2011, February). Comparison and Analysis of Self-Reference Image with Meaningful Image for Robust Watermarking Algorithm based on Visual Quality and Fidelity. International Journal of Computer Applications, 15(5).
11. Sanders, J., & Kandrot, E. (2010). CUDA by example: An Introduction for General Purpose GPU Programming.
12. Shieh, C.-S., Huang, H.-C., Wang, F.-H., & Pan, J.-S. (2004, March). Genetic watermarking based on transform-domain techniques. Pattern Recognition, 37(3).
13. Vihari, P., & Mishra, M. (2012). Image Authentication Algorithm on GPU. 2012 International Conference on Communication Systems and Network , 874 - 878 .
14. Zhao , L., & Yang, J. (2011, March). A High Performance Image Authentication Algorithm on GPU with CUDA. (M. E. Press, Ed.) I. J. Intelligent Systems and Applications, 2, 52-59.