JACET Journal of Advances in Computer Engineering and Technology, 3(2) 2017

Parallelization of Rich Models for Steganalysis of Digital Images using a CUDA-based Approach

Mahmoud Kazemi[1], Meysam Mirzaee[2], Reza Isfahani[3]

Received (2016-10-17)
Accepted (2017-03-17)

***Abstract* — There are several different methods to make an efficient strategy for steganalysis of digital images. A very powerful method in this area is rich model consisting of a large number of diverse sub-models in both spatial and transform domain that should be utilized. However, the extraction of a various types of features from an image is so time consuming in some steps, especially for training phase with a large number of high resolution images that consist of two steps: train and test. Multithread programming is a near solution to decreasing the required time but it's limited and it 'snot so scalable too. In this paper, we present a CUDA based approach for data-parallelization and optimization of sub-model extraction process. Also, construction of the rich model is analyzed in detailed, presenting more efficient solution. Further, some optimization techniques are employed to reduce the total number of GPU memory accesses. Compared to single-thread and multi-threaded CPU processing, 10x-12x and 3x-4x speedups are achieved with implementing our CUDA-based parallel program on GT 540M and it can be scaled with several CUDA cards to achieve better speedups.**

***Index Terms* — CUDA, GPU, Parallelization, Rich models, Steganalysis.**

1- ICT Research Center, University of Imam Hussein, Tehran, Iran. (mkazemi@ihu.ac.ir)
2,3- ICT Research Center, University of Imam Hussein, Tehran, Iran

I. INTRODUCTION

With ever-increasing growth of electronic information and communications, it is important to design methodologies for enhancing the security of exchanged information. One of these methodologies is stegonagraphy that is a procedure to hide some secret data into a carrier multimedia such as image, audio or video. In fact, the steganography is derived from the Greek words "stegos" meaning "cover" and "grafia" meaning "writing" defining it as "covered writing" [1]. The most important goal of the stegonagraphy is secret communication. Other similar technology for embedding data into multimedia is watermarking, however the major aim of the watermarking is protection of intellectual property through the embedded data which is usually a signature to signify origin or ownership of the multimedia [2].

As carrier multimedia, digital images are the most popular because of their frequency on the Internet. One the most important property of the image stegonagraphy is imperceptibility. It means that human visual system should not be able to recognize significant differences between original image and the image after embedding secret information.

On the contrary, steganalysis is a procedure to estimate existence of the secret data in the image. For keeping invisibility, there is no specific sign demonstrating the embedded data and consequently, it seems the steganalysis is an impossible task. But statistical analysis of the images without the embedded data shows that there is a significant correlation between adjacent pixels [3]. Therefore, absence of the correlation may demonstrate existence an embedded secret message.

In fact, steganalysis is a process of reverse engineering. For this, a huge set of basic statistical criteria called feature are extracted from the questionable image [4]. The features are utilized in a two phases process called learning. The phases are called train and test, respectively.

In the training phase, a large number of clear and stego images are analyzed, to recognize the features which are modified due to embedding data. So, the values of the features are computed for a set of clear and stego images. The results (images, features and values) are considered as training set and delivered by a classifier. In the training phase, some parameters are generated to utilize for classification in the test phase. The strategy for selecting the features and classifier is very critical to attain the best results.

Various algorithms are given in literature to extract the useful features and classification of the images into clear or stego categories [3-8]. Fridrich and Kodovský have presented a fully comprehensive approach for steganalysis of the images in the spatial domain [9]. In the approach, first a rich model of noisy residuals is constructed. The model is based on the difference of a pixel and its adjacent pixels. The difference of the pixels is considered as a noise and histogram of the adjacent differences are investigated for clear and stego images. Then, Ensemble classifier is used for the classification process. In Fridrich' scheme, a wide variety of forms for selection of the adjacent pixels with various weights are introduced, so that each of them is applicable to identify a specific stegonagraphy algorithm. Therefore, a large number of residual matrices are generated and then co-occurrence histograms of the residuals are calculated in vertical and horizontal direction and next, the co-occurrence matrices are symmetrized. The philosophy of the symmetry is that a large number of sub-model are generated, so that a lot of them have zero value and thus they should be symmetrized. After symmetry, some useful features are extracted from the image. Fridrich and Kodovský have demonstrated that the framework is efficiently works when a secret message is embedded in the spatial domain of an image by steganographic algorithms like HUGO [10], edge-adaptive algorithm by Luo et al. [11], and optimally-coded ternary ± 1 embedding. So, the approach is also called Spatial Rich Model (SRM). As mentioned, SRM is a comprehensive approach and for each stegonagraphy algorithm, some sub-models are selected to recognize the stego image. On the other hand, the diversity of models causes a so time consuming task for the learning phase. In fact, universality and comprehensiveness of the Fridrich's approach results in an acceptable performance, however the extraction of the features leads to a computationally intensive process, especially when the resolution and number of learning images are increased.

To accelerate time consuming algorithms, parallel systems can be considered as an exciting option. Graphics Processing Unit (GPU) as a highly parallel, multithreaded and many-core architecture can be applied by user for heavy computational algorithms. 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 [12]. 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 [12, 13].

It is notable that the data-level parallelization should be performed for implementing an algorithm on GPU, getting acceptable performance. Image processing algorithms due to their parallel nature are suitable cases, however it is important to design an efficient parallelization approach based on the GPU architecture [14-20]. In this paper, we provide some parallelization techniques and a CUDA based implementation for SRM steps.

The paper is structured as follows. In section II, the proposed scheme for data parallelization of SRM is explained. Also, the details of our GPU implementation and optimization techniques are provide in Section III. Speedup results by the presented parallel approach are given in Section IV. Finally, the paper is concluded in Section V.

II. THE PROPOSED PARALLELIZATION SCHEME FOR GENERATING SUB-MODELS

In SRM algorithm proposed by Fridrich [9], 4 steps are required to generate 34671 features or sub-models from a gray-scale image. By analyzing the SRM algorithm (refer to Section 4 for more details), it can be found that steps 1 and 2 are dominant parts of the process. Furthermore, the steps 1 and 2 are dependent on the image sizes and their execution time increases with higher resolution images. However the output of step 2 (co-occurrence values) can be a vector with a constant length and consequently step 3 and 4 are completely independent of the image sizes and consequently, compared to the total execution time of the SRM, the execution time of the steps 3 and 4 can be ignored, especially for high resolution images. Thus, we need to propose parallelization techniques only for two first steps of the SRM. Our proposed techniques for an gray-scale image of size M×N are described in the following sub-sections.

A. Extraction of a Residual Vector R Containing 458×M×N Elements From an Image

In the first step, 458 residual matrices each of size M×N are generated from and input image matrix of size M×N. In other word, for each pixel, 458 different values are calculated as residuals. Different form for selection of adjacent of pixels, different orders and quantization values results in such diversity. In fact, a residual of a pixel can be defined as quantized difference of the pixel value from some its adjacent pixels (refer to [9] for more details). Statistic information of the residuals can lead to detect the existence a secret message in the image. To calculate a specific residual, the process is the same for different pixels. For example, to calculate the first kind of residual, the value of each pixel is subtracted from its right neighbor (see Fig. 1), and then truncation and quantization are performed. The process should be repeated for all pixels of the image. So, the first step can be so time consuming and we propose a pixel-level parallelization for the step. In other word, one CUDA thread is defined for each pixel and consequently there are M×N parallel threads, so that each thread calculates 458 different residuals for each pixel. Totally, 458 residual matrices are generated and we put their elements into a vector. In the proposed scheme, all residuals generated from all pixels are placed together in the vector R. Fig. 2 exhibits the placement of different residuals in vector R.

B. Computing Horizontal and Vertical Co-occurrences

In this step, two horizontal and vertical co-occurrence matrices are generated for each residual matrix. For this, number of occurrences for different possible patterns of the elements in a residual matrix is computed along the horizontal and vertical directions and then the result is normalized. According to the algorithm proposed by Fridrich [9], the elements of residual matrices are integer values between -2 and 2. On the other hand, the considered patterns are 4 symbols (e.g. -2 1 2 0), and consequently there are 625 different patterns. In this saturation, for horizontal co-occurrence, we should calculate that how many times each pattern is happened along horizontal direction. Similar procedure is performed to compute the vertical co-occurrence matrix. As a result, for each residual matrix, one horizontal co-occurrence matrix and one vertical co-occurrence matrix are constructed, so that we consider a 625-element vector to place the elements of each co-occurrence matrix. Since, the computation of co-occurrence arrays is a same process for all residuals, a residual-level parallelization scheme is proposed for the step 2. It means that in this step, 458 CUDA threads are considered and each thread is responsible for calculation of horizontal and vertical co-occurrences of a residual matrix.

Also, in the step 2, the horizontal co-occurrence matrices form the vector C_H containing 458×625 elements and the vertical co-occurrence matrices form the vector C_V containing 458×625 elements. Fig. 3 shows the placement of horizontal co-occurrences in C_H or vertical co-occurrences in C_V. In the step 2, each thread processes its corresponding residuals (M×N elements of vector R) and calculates 625 elements of vector C_H and 625 elements of vector C_V.

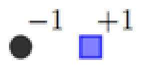

Fig. 1. The first kind of residual

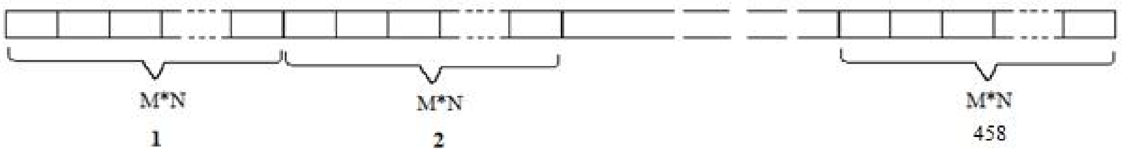

Fig. 2. Placement of different residuals of an image of size M×N

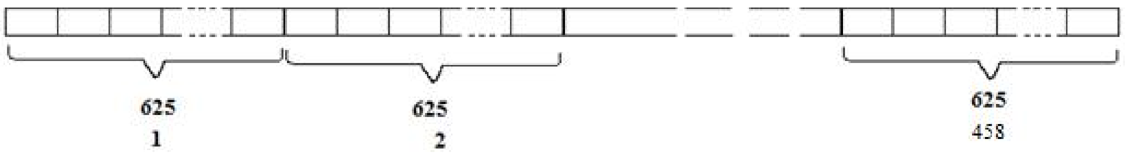

Fig. 3. Placement of co-occurrence values in a vector

As a result, by performing step 2, the vectors C_H and C_V (each of size 458×625) are produced. The obtained values should be symmetrized in steps 3 and 4. The number of elements of C_H and C_V are constant for each input image with arbitrary size, and consequently the execution time of the step 3 and 4 are not increased, with high resolution images.

III. CONSIDERATIONS AND OPTIMIZATION TECHNIQUES FOR OUR GPU IMPLEMENTATION

As mentioned, many processing cores are provided by the modern GPUs. However, the utilization of the GPU cannot guaranty reduction of the processing time in all cases. The main factors enhancing performance of a GPU-based process can be listed as follows:

• There is a potential to invent an efficient data-level parallelization approach.

• The data communications between the parallel threads must be reduced, as possible.

• Also, the total number of memory accesses must be reduced and instead, registers can be utilized as possible. The modern GPU architectures provide a significant number of registers for threads to store date which are frequently used. The registers are very faster than global memory of the GPU. More utilization of the registers can lead to enhance the performance.

• Finally, superiority of the GPU is revealed when the process is CPU bound instead of memory bound.

With respect to the mentioned factors, we provide some solutions to enhance the performance of the GPU program as follows.

• In the first step, 458 different kinds of the residuals can be calculated for each pixel. On the other hand, for each pixel, it is necessary to read some neighbor values from the memory. The fact imposes a huge number of repetitive memory accesses, leading significant performance degradation. To resolve the issue, we propose that at begin of residual extraction process, the values of the required neighbor pixels are read from the global memory and then registered to be used in the calculations.

• If consecutive memory demands of the running threads exceed capability of the GPU, then the kernel has been stopped and then a message is appeared for the user (Display driver stopped responding and has recovered). In these cases, we are forced to reduce the number of parallel threads. It means the degree of parallelization should be reduced and the kernel can be called serially to process all data. The maximum number of parallel threads is dependent on specifications of the GPU such as memory bandwidth and also number of memory accesses required for each thread. In step 2 of the SRM algorithm, there are 458 parallel threads, such that each thread attempts to read M×N elements from the global memory and write 625 elements to the global memory. This amount of memory accesses (458 × (M×N×625)) is not supported, even by state of the art GPUs. In this situation, we should reduce the number of parallel threads that run the kernel calculating co-occurrences and then kernel is serially invoked in a loop to process all of residuals. As the number of parallel threads decrease, the more number of iterations is needed. In this work, we propose that the number of parallel threads is defined parametric via determining CUDA Grid and CUDA block sizes. As a result, we can adjust the degree of parallelization in respect to available GPU. Consequently, our scheme can be adapted with various models of GPUs.

IV. EXPERIMENTAL RESULTS

The proposed parallelization scheme for the SRM algorithm was implemented on NVIDIA's GPU using CUDA 7.5. Our GPU model is Geforce GT 540M that contains 1GB global memory and 64 CUDA cores with frequency of 1.3 GHz. To compare performance of the parallel implementation with serial implementation on a General Purpose Processor (GPP), the SRM algorithm was coded by using single thread C language and implemented on a PC with CPU Intel Pentium 4. Moreover, a multi-threaded version is implemented on intel core-i7 processor using OpenCL framework. Note that all programs were executed with 200 iterations and average results are reported.

Timing analysis for the single-thread serial implementation of the SRM algorithm is performed and the detailed results are provided in Table I.

Table I: Execution time of different steps of the SRM algorithm (C implementation)

Image size	Step1 (second)	Step2 (second)	Step3 (second)	Step4 (second)
256×256	14.671	5.692	0.005	0.392
512×512	56.052	22.64	0.005	0.392

As can be seen in Table I, the execution times of steps 3 and 4 are not variable with different sizes of the input image and also ignorable compared to the first steps. Furthermore, the execution times of two first steps are proportional to the square of image size and consequently they are so time consuming when the image size is increased.

Table II lists the execution times of the serial and the proposed parallel implementation of the SRM for steps 1 and 2. Note that in Table II, data transfer overhead is not considered and the results are only pure processing times.

As can be shown in Table II, using our CUDA implementation, for step 1 (as the most time consuming step) 101x-114x times speedups are obtained. Also, 4 times speedup is achieved for step 2, where the number of parallel thread has been limited. This reduction in speedup demonstrates the impact of memory bottleneck on the achieved performance. For further performance evaluation of the proposed CUDA-based approach, the execution times of the multi-threaded CPU implementation are also provided in Table II. The proposed CUDA-based approach attains almost 49.9x-53.5x and 2x-2.3x speedups over the multi-threaded CPU implementation for the steps 1 and 2, respectively.

Furthermore, the performance of a CUDA program can be affected by the dimensions of CUDA grid and CUDA block. To implement the proposed parallel scheme on Geforce GT 540M, optimal dimensions for steps 1 and 2 (leading results listed in Table II) are given in Table III.

Here, M and N are the image sizes. It can be found that it is feasible to run M×N parallel threads for the step 1. However, in our implementations on Geforce GT 540M, M and N are less than or equal to 512. Furthermore, the used GPU is capable to execute only 128 threads for step 2, where a huge number of memory accesses is required.

In our scheme, for computing rich models, an image matrix should be transmitted from CPU memory to the GPU memory and then steps 1 and 2 should be performed by the GPU cores. Next, the results are written back to the CPU memory, performing steps 3 and 4. Considering execution times of 4 steps as well as the data transmitting overhead, the total required times and acquired actual speedups for constructing rich models are listed in Table IV.

Table IV reveals that the actual speedups obtained by our proposed scheme are about 10x and 12x for gray-scale images of size 256×256 and 512×512 respectively. Also, in compared with multi-threaded CPU implementation, our scheme reaches up to 5x-6.2x speedups for gray-scale images of size 256×256 and 512×512 respectively.

In fact, with reduction of parallelization degree

Table II: Time comparison of serial and parallel implementations of steps 1 and 2 of the SRM algorithm

Image size	Step 1 (second)					Step 2 (second)				
	Single thread on CPU	Multi threaded on CPU	Proposed parallel scheme on GPU using CUDA	Speedup GPU over single-thread CPU	Speedup GPU over multi-threaded CPU	Single thread on CPU	Multi threaded on CPU	Proposed parallel scheme on GPU using CUDA	Speedup GPU over single-thread CPU	Speedup GPU over multi-threaded CPU
256×256	14.67	6.85	0.128	114.61	53.52	5.692	2.93	1.42	4.01	2.06
512×512	56.05	27.46	0.55	101.91	49.93	22.64	13.01	5.61	4.04	2.32

Table III: Optimal dimensions of CUDA grid and CUDA block in this work

Step 1		Step 2	
grid	block	grid	block
M/8 - N/16	8-16	4-1	8-4

Table IV: Total execution time (including overheads) of serial and proposed parallel implementations to construct rich models

Image size	Single thread on CPU (second)	Multi threaded on CPU (second)	Proposed parallel scheme on GPU using CUDA (second)	Speedup GPU over single-thread CPU	Speedup GPU over multi-threaded CPU
256×256	20.76	10.19	2.04	10.18	4.99
512×512	79.19	40.87	6.59	12.01	6.2

in step 2 (due to memory access constraint), the overall performance of the parallel scheme is degraded, significantly. However, due to the excellent results of the step 1, the proposed scheme is efficient for computing rich models which is a computationally intensive process.

V. CONCLUSIONS AND FURTHER WORKS

Steganalysis of digital images requires a learning process, where thousands stego or clear images are analyzed, to extract the features that can show existence of a secret message in an image. When the number and dimensions of the images are increased, significant growth is seen in the processing time. Furthermore, a comprehensive approach for steganalysis extracts more than 34000 features for each image [9].

To address the illustrated problem, a parallelization scheme for the algorithm introduced in [9] was presented in this article.

We discussed that the algorithm has 4 steps, such that execution times of two last steps are ignorable compared to two others. Thus, we offered a pixel-level parallelization scheme for step 1, so that M×N parallel threads are defined to compute the residuals. To optimize CUDA implementation of the step, we suggested that before calculating residuals, each pixel and its adjacents are registered, significant reducing in the total number of memory accesses. Furthermore, we offered a residual-level parallelization scheme for step 2, so that 458 parallel threads are defined to compute the co-occurrences. Also, we demonstrated that in the implementation, due to memory bandwidth constraints, we should use the limited number of parallel threads for the step 2.

Experimental results showed that our

proposed parallelization scheme offers more than 101x speedup for the step 1 and about 4x speedup for the step 2. In total, with considering data transmission overheads, our parallel implementation achieves 10x and 12x speedups for gray-scale images of size 256×256 and 512×512 respectively. Moreover, our proposed CUDA-based approach is superior to multi-threaded CPU implementation.

The utilized GPU model for our implementations in this work was Geforce GT 540M. It is evident that with state of the art and more power full GPUs, a significant growth in achieved speedups are expected, especially for the step 2, where the number of parallel threads can be increased.

The steganalysis methodology presented in [9] is efficient for steganography algorithms that address spatial domain. For future work, we will introduce parallelization methods to accelerate a steganalysis algorithm that address frequency domain.

Further, a hardware and parallel architecture will be designed for the steganalysis process. The architecture can offer a significant throughput and will be implemented on FPGA.

ACKNOWLEDGMENT

The author wish to thank Dr. Zeinolabedin Norouzi and Dr. Hamed Mehrara for their guidelines, suggestions and technical supports during the work.

REFERENCES

[1] Moerland, T., “Steganography and Steganalysis”, Leiden Institute of Advanced Computing Science, www. liacs.nl/home/ tmoerl/privtech.pdf.

[2] S. Fazli, M. Moeini, “A robust image watermarking method based on DWT, DCT, and SVD using a new technique for correction of main geometric attacks,” Optik, Vol. 127, No. 2, 2016, PP. 964-972.

[3] H. Farida and S. Lyu, “Steganalysis Using Higher-Order Image Statistics”, IEEE Transactions on information Forensics and Security, February 2006, Vol. 1, PP. 111-119.

[4] Bohme, Rainer. Advanced Statistical Steganalysis. s.l. : Springer, 2009.

[5] Z. Xia, L. Yang, X. Sun, W. Liang, D. Sun and Z. Ruan, “A Learning-Based Steganalytic Method against LSB Matching Steganography”. Changsha, 410082, China : Hunan University, 2011.

[6] J. Fridrich and J. Kodovský, “Quantitative Steganalysis Using Rich Models.”, .USA : Proc. SPIE 8665, Media Watermarking, Security, and Forensics 2013, March 22, 2013.

[7] V. Holub, J. Fridrich, and T. Denemark, “Random Projections of Residuals as an Alternative to Co-occurrences in Steganalysis.”, Department of ECE, SUNY Binghamton, NY, USA : Proc. SPIE 8665, Media Watermarking, Security, and Forensics 2013, March 22, 2013.

[8] Q. Liu, A. H. Sung, “Feature Mining and Neuro-Fuzzy Inference System for Steganalysis of LSB Matching Stegonagraphy in Grayscale Images.” .New Mexico Tech, Socorro, NM 87801, USA : s.n., 2007.

[9] J. Fridrich and J. Kodovský, “Rich Models for Steganalysis of Digital Images”, IEEE Transactions on Information Forensics and Security, vol. 7, no. 3, pp. 868 – 882, June 2012.

[10] T. Pevný, T. Filler, and P. Bas, “Using high-dimensional image models to perform highly undetectable steganography.” In R. Böhme and R. Safavi-Naini, editors, Information Hiding, 12th Interna tiona l Workshop , volume 6387 of Lecture Notes in Computer Science, pp. 161–177, Calgary, Canada, June 28–30, 2010. Springer-Verlag, New York.

[11] W. Luo, F. Huang, and J. Huang, “Edge adaptive image steganography based on LSB matching revisited”, IEEE Transactions on Information Forensics and Security, vol. 5, no. 2, pp. 201–214, June 2010.

[12] NVIDIA Corporation, NVIDIA CUDA C Programming Guide 4.1, 2011.

[13] General Purpose GPU Programming (GPGPU) Web site, http: //www.gpgpu.org, 2010.

[14] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, and T.J. Purcell, “A Survey of General-Purpose Computation on Graphics Hardware,” Computer Graphics Forum, vol. 26, no. 1, pp. 80-113, Mar. 2007.

[15] I. K. Park, N. Singhal, M. H. Lee, S. CHo, and C. W. Kim, “Design and Performance Evaluation of Image Processing Algorithms on GPUs” IEEE Transactions On

Parallel and Distributed Systems, vol. 22, no. 1, pp. 91–104, January 2011.

[16] H. Heidari, A. Chalechale and A. Mohammadabadi, "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.

[17] H. Heidari, A. Chalechale and A.A. Mohammadabadi, "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.

[18] A.A. Mohammadabadi, 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.

[19] A.A. Mohammadabadi, 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.

[20] H. Heidari, A. Chalechale, A.A. Mohammadabadi, "Accelerating of color moments and texture features extraction using GPU based parallel computing", 8th Iranian Conference on Machine Vision and Image Processing (MVIP), Zanjan, 2013, pp. 430-435.