Parallel Image Encryption Algorithm Using Partitioned Cellular Automata on Graphic Processor Unit

1th Mahmood Fazlali
Cybersecurity and Computing Systems Research Group
University of Hertfordshire
Hatfiled, UK
M.Fazlali@herts.ac.uk

2th Amirhossein Alihoseini
Dept. of data and Computer Sciensce, Faculty of Mathematical sciences
Shahid Beheshti University
Tehran, Iran
A.Alihosini@sbu.ac.ir

3th Ebrahim Zarei
Dep. of Computer Science, Khansar Campus
University Of Isfahan
Isfahan, Iran
E.Zarei@khc.ui.ac.ir

4th Myasar Tabani
Cybersecurity and Computing Systems Research Group
University of Hertfordshire
Hatfiled, UK
M.Tabani@herts.ac.uk

5th Seyedali Pourmoafi
Cybersecurity and Computing Systems Research Group
University of Hertfordshire
Hatfiled, UK
S.Pourmoafi@herts.ac.uk

***Abstract*—Image encryption is a reliable method for securely transmitting images over a network. The time required to encrypt and decrypt an image in online applications is also very important. Although cellular automata cryptography is an appropriate technique for parallelizing and accelerating cryptographic methods, its capacity cannot be demonstrated only in multi-core platforms. Thus, it is needed to parallelize cellular automata cryptography on Graphic Processor Units (GPUs) in order to significantly decrease the encryption/decryption time. In this paper, we propose a new parallel algorithm for two-dimensional cellular automata cryptography that is implemented on GPU. The proposed algorithm uses multiple threads at once to accelerate the bit-level permutation and substitution operations by taking into account the capacity of cellular automata in parallel processing. According to the study experimental findings, the proposed algorithm performs faster on GPU compared to a multicore platform while maintaining the same level of security in comparison to the serial algorithm.**

Keywords— Partitioned cellular automata, Image cryptography, Parallel cryptography, Graphic Processing Unit (GPU)

I. INTRODUCTION (*HEADING 1*)

The need for quick, secure image storage, display, and transfer is unavoidable given the 20th century's rapid advancement of information technology and the pervasive use of digital images in communication and information networks [1, 2, 3]. Image cryptography is different from textual cryptography algorithms due to some inherent intrinsic of images. We can encrypt images using traditional text-based cryptography algorithms like AES, DES, and Blowfish. These algorithms are designed to encrypt data in binary format, which includes image files. However, it is important to note that image files are larger than regular text files, and thus the encryption and decryption time may be longer. Additionally, the size of the encrypted image file may be larger due to the addition of padding [3, 4].

There are several image cryptography algorithms that make use of Cellular Automata (CA), chaotic functions, DNA computing, and other techniques [5, 6, 7, 8]. Because of their complex behavior, randomness, and unpredictable nature as well as their simplicity in implementation, cellular automata is one of the nonlinear dynamic systems that are frequently used in cryptography [9, 10]. The combination of the cellular automata-based image encryption technique with parallel processing techniques is a good candidate for meeting the expanding computational needs of image cryptography due to the large data capacity of images. A Partitioned Cellular Automata (PCA) is a regular (arranged) network of cells, with each cell subdivided into multiple components. For updating the state of each cell, the PCA-based image encryption employs two functions: (1) Function P, which replaces each part of the cell with one of its neighboring cells. (2) Function F, which modifies the state of all cells. The following algorithm first receives the image and the encryption key, then encrypts the image using functions P and F after some repetitions.

Because the computational complexity of image cryptography using PCA is so high, parallelizing it can result in a significant increase in computing speed. However multicore platform cannot fulfill the potential parallel capabilities in PCA. The Graphics Processing Unit (GPU) has a high capacity for parallelizing with the use of different threads on GPU cores, and its speed is significantly faster than the Central Processing Unit (CPU) cores.

In this paper, we propose a parallel algorithm based on PCA that runs on GPUs. Each cell's computation is assigned to one thread for this purpose. As a result, all cells are computed concurrently by different threads on GPU to reduce the execution time (encryption/decryption time). So the main contribution of this paper is presenting a new parallel image encryption based on PCA and paralleling the proposed method on GPU. The study measures the speedup of the proposed scheme and shows it is quick for online applications.

The rest of this paper is structured as follows. Section II present a literature review. Section III describes the proposed parallel image encryption algorithm. Section IV demonstrates the performance and security analysis to the proposed algorithm. Finally, Section V concludes this study.

II. RELATED WORKS

There are numerous image cryptography techniques based on chaotic functions, DNA computing, and Cellular Automata [11]. One class of nonlinear dynamic systems known as chaotic systems exhibits important characteristics

like initial condition dependence, unpredictability, and system definability while behaving erratically. These systems are therefore ideally suited for image cryptography. Rostami et al. [1] propose a sample method based on chaotic functions that makes use of logistic maps. This technique divides the image into blocks of 16×16 pixels, and simultaneously encrypts each block. Order plays a crucial role in chaotic functions. This is why blocking is used in the above method. Due to floating-point calculations, this method takes longer to encrypt an image than cellular automata. cellular automata-based image cryptography techniques come in a variety of forms. For instance, cellular automata are used in some algorithms to permute the image's pixels and sabotage the relationship between them [12, 13].

Researchers in [3] uses recursive cellular automata for image cryptography. Initially, the image is divided into blocks of 16 × 16 pixels, and then, a pseudo-random permutation function is applied on the blocks. Next, the blocks are cipher simultaneously using GPU. Debasis and Abishak in [14] proposed an encryption algorithm based on Recursive Cellular Automata (RCA). In this method, a circular transfer chart is used so that the image has the capability of being decrypted. Using cellular automata, this approach can have a parallel implementation. However, the proper parallel algorithm is not proposed.

A technique for sharing cryptography with cellular automata has been put forth by Hernandez et al. [15], in which multiple files are ciphered simultaneously using GPU. The above-mentioned method receives several images at once and encrypts them all at once. For image cryptography, Zhang and colleagues [16] have also employed two-dimensional cellular automata based on stream cyphers. In the above method, the sender and receiver first share the initial information over a secure channel before they begin sending and receiving data. Because the encryption and decryption procedures are less complicated with stream cyphers than with block cyphers, they operate more quickly.

In [17], a secured lightweight cryptosystem is designed based on lookup table operations that reduce computational overhead, resource requirement and power consumption compared to traditional security mechanisms. In this context, one-dimensional elementary cellular automaton has been combined with Henon chaotic map to design a cryptosystem, which can produce unprecedented results in cryptography. However it is not enough Fast. Researchers in [18] tried to overcome this weakness by using FPGA hardware platform but it needs a new hardware and extra cost. It is worth mentioned that cryptography helps protect data against malware, and malware detection systems may use cryptography to identify and verify the integrity of files [19].

Partitioned Cellular Automata in two dimensions were used for image encryption by Wang and coworkers [9]. Due to its flexibility and ease of use, this method is utilized for images with various color depths. Additionally, it is capable of bing parallelized. The aforementioned algorithm's processing time is insufficient for online communications and images with large amounts of data. In order to improve running time, a parallel algorithm based on PCA is proposed in this paper using the programming language "CUDA" on GPUs to overcome this weakness.

.

III. Proposed Parallel Image Encription Algorithm

A. Partitioned Celular Automata

PCA is a D-dimensional network with K neighbors, defined as quintuplet P = (ZK, Q, S, f, #) in which:

- Z is the set of integer numbers
- Q is a set of states of each cell
- S = {s1, s2, . . ., sm} is the set of all the neighbors such that si ∈ ZK (i = 1, . . ., m)
- f: Q → Q defines the rules of PCA
- # ∈ Q is a quiescent state such that f (#) = #

Suppose α(r+1)(x) demonstrates the state of a cell in coordinate x after r reiterations of function f which can be calculated as follows:

$$\alpha^{(r+1)}(x) = f\left(p_1\left(\alpha^{(r)}(x+s_1)\right), \ldots, p_1\left(\alpha^{(r)}(x+s_m)\right)\right) \quad (1)$$

where p_i is a function with rules of $p_i(x) = x.ei$ which indicates dot product of x and ei, and $ei = [a_1, a_2, \cdots, a_m]$ are basic vectors in an m−dimensional space. In PCAs, there are two approaches for processing the cells in edges:

1) The cells at the beginning and the end of each row (or each column) are considered adjacent; (2) The grid is surrounded by an outer layer of cells in the fixed state of zeros. Besides, there are two types of boundary conditions in PCA: (1) Von Neumann neighborhood consists of four orthogonally surrounding neighbors.

2) Moore neighborhood comprises four Von Neumann neighborhoods and four diagonally surrounding neighbors.

Paper [9] uses a two-dimensional PCA which has a suitable structure for digital images. The proposed PCA is uniform and includes cyclic boundary conditions. Therefore, the rules for all cells are the same. Moor adjacency is used in the proposed PCA because it can be suitable for different color depths.

B. The Proposed Parallel Image Encryption by Partitioned Celular Automata for GPU

The method for parallelizing image cryptography using PCA on GPUs is shown in Figure 1. Also Algorithm 1 displays the encryption procedure's steps as pseudocode. Following is a summary of the encryption algorithm's overall steps:

1. The inputs—a M × N plain image and a 4 × 4 key—are first received and stored in the GPU's global memory.
2. In the GPU's global memory, a matrix named matrix key with a size of M ×N bytes is defined to hold the key.
3. The function InitiateRondKey is simultaneously run on the M/4 × N/4 threads of GPU through CUDA to generate the matrixkey. Each thread puts a 4 × 4 tile matrix equivalent to a 4 × 4 key. In this section, the external key, which is the size of 4 x 4, is taken, and in

the matrixkey, which is the size of the image, it is placed next to each other in the form of tiles.

4. After generating the matrixkey, the function F is simultaneously run on the M/4 × N/4 threads of GPU through CUDA. As shown in Algorithm 2, each thread receives one byte from the image matrix and one byte from the key matrix. Then, functions P, S-box, and XOR are executed, and finally, the output is stored.

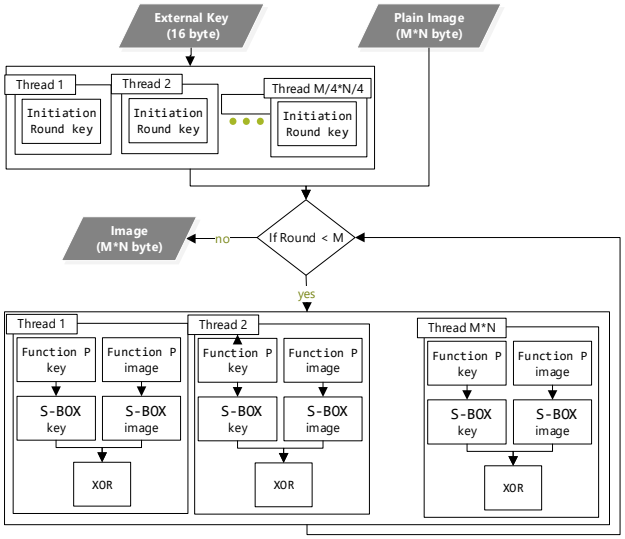

Figure 1: The flowchart of the Parallel PCA Encryption Algorithm.

5. After the repetition of step 4 for M times, the cipher image, is transferred from the GPU's global memory to RAM and the encryption process ends. The number of repetitions of step 4 in the article [9] is equal to the number of rows in the image.

Due to the large size of the matrices and lack of storage space in faster and smaller memories, we used the global memory of the GPU to implement the suggested algorithm. On the other hand, it is not possible to use constant memory because the values of the matrices change. The S-box function's value is fixed for all images, so rather than implementing it, we simply included its ready matrix in the code. We need a constant matrix to implement the S-box, and we store it in GPU constant memory to speed things up.

Algorithm 1: PCA Encryption Algorithm

Input: image Arrays of Byte, key Arrays of Byte
dImage ← copyHostToDevice(image)
dkey ← copyHostToDevice(key)
matrixkey ← cudaMalloc M*N of Byte
for i ← 1 **to** M/4 **do**
  for j ← 1 **to** N/4 **do** in parallel
    InitiationRoundKey(dkey[i][j], matrixkey)
  end
end
cudaDeviceSynchronize()
for i ← 1 **to** M **do**
  for j ← 1 **to** N **do**
    for k ← **to** M **do** in parallel
      Function_F(dImage[j][k], matrixkey[j][k])
    end
  end
  cudaDeviceSynchronize()
end
Outimage ← copyDeviceToHost(dImage)
Output: Outimage Arrays of Byte

Algorithm 2: Function_F in PCA encryption

Input: Byte Image, Byte Key
copyImage ← Function_p(Image)
copyKey ← Function_p(Key)
Image ← S_box(copyImage)
Key ← S_box(copyKey)
Image ← Xor(Image, Key)
Output: Image, Key

It should be noted that Function P performs a permutation between bits of the pixels in the image using Eq. 2, and function F uses Sbox table from the AES algorithm.

$$I'_t(i.j) = I_t\left((i, s_{t,1})\ mod\ M, (j, s_{t,2})\ mod\ N\right)$$
$$s_1 = (1,-1), s_2 = (1,0), s_3 = (1,1),$$
$$s_4 = (0,-1), s_5 = (0,1), s_6 = (-1,-1), \quad (2)$$
$$s_7 = (-1,0), s_8 = (-1,1)$$
$$s_t = (s_{t,1}, s_{t,2}), t = 1,\ldots,8$$

Algorithm 2 demonstrate the details in function F in the proposed PCA encryption algorithm.

Now we need a parallel decryption method to be run on GPU. Here the reverse of the encryption is performed. The steps of decryption algorithm can be summarized as follows:

1) First, the M × N image and the 4 × 4 key are received as the inputs and stored in the GPU's global memory.

2) The matrix key (matrix of size M × N bytes) is defined in the GPU's global memory for storing the key.

3) The function InitiateRondKey is simultaneously run on the M/4 × N/4 threads of GPU through CUDA to generate the matrix key.

4) To start the decryption operation, the matrix key of the last stage of encryption must be generated, and then the decryption operation must be started. To do this, the P and S-box functions must be executed consecutively M times on the key matrix, and the output of the last step will be the desired key matrix. To increase the speed of this part, each repetition is done in parallel by M × N threads of GPU. Note that the repetitions must be done serially, but in each repetition, there is the ability to parallelize.

5) After generating the key matrix in the last stage of the decryption process, functions XOR, P, and S-box are run simultaneously on M × N threads. Each thread receives one byte from the key matrix and one byte from the image matrix, and then runs functions XOR, P, and S-box, and finally stores the output.

After the repetition of step 5 for M times, the resulting image is transferred from GPU's global memory to RAM and the decryption process ends.

IV. Performance and Security Analysis

The speed and the security of the proposed algorithm were assessed through a number of different experiments in this section. First, the execution time is the main factors used to assess the performance of the proposed method. Then, the security and toughness of the proposed parallel PCA were also examined.

A. Performance Analysis

In this section, the performance analysis of the proposed algorithm is presented on two GPUs and a CPU. The main specifications of the platforms used for the experiments are shown in Table 1.

Table 1: The Specification of the Running Platform

CPU	Intel i7
GPU	GeForce GT 740M, GeForce GTX 940M
OS	Windows 10 x64
CUDA Version	9. 0.176

A random 4×4 matrix of integer numbers between 0 and 255 has been generated and considered as the secret key. The proposed approach can encrypt the plain image into the noise-style cipher image.

The proposed method is then applied to the plain image of Lena (1024×1024 pixels in size). The execution times of running algorithms are shown in Tables 2. The first column of the table identifies whether the operation is encryption or decryption. The serial implementation of the proposed algorithm is equal to Wang et al. Algorithm [9]. The number of threads used for the operation is displayed in the second column. The execution times of each operation using a CPU are shown in the third column, and the execution times of each operation using two different types of GPUs are shown in the last two columns. To improve the accuracy of the execution time, each experiment was performed 10 times and the average execution time was reported in these tables.

Table 2: Encryption and Decryption's Execution Time on CPU and GPUs/ Picture Size 1024 × 1024

Mode	Thread	CPU	GPU	
		8Core	GT 740M	GTX 960M
Encryption	1	3.318(s)	15.478(s)	0.091(s)
	2	1.710(s)	7.829(s)	2.877(s)
	4	0.984(s)	4.041(s)	1.5331(s)
	8	1.067(s)	2.108(s)	0.716(s)
	16	-	1.128(s)	0.372(s)
	32	-	0.637(s)	0.222(s)
	64	-	0.351(s)	0.159(s)
	128	-	0.218(s)	0.158(s)
	256	-	0.229(s)	0.160(s)
	512	-	0.259(s)	0.161(s)
	1024	-	0.303(s)	0.164(s)
Decryption	1	4.935(s)	24.853(s)	8.920(s)
	2	2.514(s)	12.713(s)	4.607(s)
	4	1.678(s)	6.489(s)	2.380(s)
	8	1.811(s)	3.347(s)	1.274(s)
	16	-	1.761(s)	0.676(s)
	32	-	0.964(s)	0.383(s)
	64	-	0.531(s)	0.267(s)
	128	-	0.343(s)	0.264(s)
	256	-	0.341(s)	0.255(s)
	512	-	0.361(s)	0.257(s)
	1024	-	0.393(s)	0.259(s)

As you can see in Tables 2, there are some instances where the time is not calculated and is empty in the third column. This is due to the lengthened runtime caused by thread counts that are higher than permitted. Threads can be run simultaneously on twice as many cores as CPUs. The ideal number of threads to operate effectively on CPU cores is eight where the encryption/decryption has its minimum execution time and let overhead.

As shown in the table, by increasing the number of threads running on the CPU cores, from one thread to four threads, the execution time is reduced. Afterwards, the execution time is increased. This means that using four threads on CPU is the saturation point and the best option for paralleling the algorithm on CPU where the parallelization overhead is less than the obtained acceleration.

The table's fourth and fifth columns display the algorithm's execution time on two GPUs for various thread counts per block. The algorithm's execution time in encryption decreases as the number of threads increases and is significantly faster than parallel CPU implementation. When 128 threads are used, the encryption algorithm performs best on both GPUs. For both GPUs, 256 threads and more take longer to execute than 128 threads. This is because running this many threads has a high synchronization overhead and cost of thread creation. This means that the plain image Lena, with a size of 1024 × 1024, should not be encrypted using a granularity of more than 128 threads. The same scenario is happened for the decryption part and using 256 threads on GPU is the best option.

Table 3: Encryption and Decryption's Execution Time on CPU and GPUs/ Picture Size 8192 × 8192

Mode	Thread	CPU	GPU	
		8Core	GT 740M	GTX 960M
Encryption	1	167.442(s)	-	-
	2	85.299(s)	-	185.743(s)
	4	46.400(s)	-	98.968(s)
	8	36.945(s)	132.892(s)	45.202(s)
	16	-	70.291(s)	22.439(s)
	32	-	38.672(s)	13.535(s)
	64	-	20.589(s)	9.596(s)
	128	-	12.778(s)	9.516(s)
	256	-	12.553(s)	9.336(s)
	512	-	14.490(s)	9.712(s)
	1024	-	16.904(s)	9.844(s)
Decryption	1	282.144(s)	-	-
	2	145.294(s)	-	293.837(s)
	4	81.891(s)	-	154.921(s)
	8	70.939(s)	211.627(s)	81.592(s)
	16	-	109.992(s)	42.447(s)
	32	-	59.173(s)	22.951(s)
	64	-	31.687(s)	15.817(s)
	128	-	19.632(s)	15.449(s)
	256	-	19.599(s)	15.132(s)
	512	-	20.280(s)	15.271(s)
	1024	-	22.292(s)	14.805(s)

Table 3 displays the proposed algorithm's execution time for the 8192×8192 pixel plain image Lena. The table illustrates how the algorithm execution time is decreased by increasing the number of threads running on the CPU cores. Both encryption and decryption algorithms follow this pattern. The acceptable parallelization overhead in comparison to the parallel acceleration is the cause of this. The first GPU's GPU-implemented encryption algorithm performs best when 256 threads are active. Additionally, using 256 threads produces the best results for the GPU-implemented encryption algorithm on the second GPU. The decryption algorithm for the plain image Lena, which has a size of 8192×8192, follows the same principle. The best number of threads to run the decryption algorithm on both GPUs is 256, and parallel execution of threads on CPU cores thus reduces the algorithm's execution time.

B. Key Space Alanysis

The key space in cryptography refers to the collection of all feasible keys used to initialize the cryptographic scheme. In cryptography, a key space's size determines how secure an encryption scheme is against a brute-force attack. As of right now [20, 21], cryptography schemes with key spaces larger than 2100 1030 can withstand brute-force attacks. The key space size for the suggested image encryption scheme will be 2128, which is larger than 2100. As a result, the proposed key space is sufficiently large to fend off a brute-force attack.

C. Kye Sensivity Analysis

To analysis the key sensitivity in the encryption and decryption process, we can compute the Number of Bit Change Rate (NBCR). For two images B_1 and B_2, the NBCR is expressed as:

$$\mathrm{NBCR}\,(B1, B2) = \mathrm{Ham}\,(B1, B2)/\,\mathrm{Len} \times 100\% \quad (3)$$

Where Ham(B_1; B_2) is the hamming distance between two images B_1 and B_2 and Len is the bit length of B_1 and B_2. According to Eq. 3, if the NBCR(B1, B2) is 50%, then two images B_1 and B_2 are completely different. The ideal value of NBCR is about 50%. We can analyze the sensitivity of each bit of the K_1 as follows: (1) Change one bit of the secret key K_1 to obtain secret key K_2 ; (2) In the encryption process, to test the key sensitivity, encrypt the plain image P using the secret keys K_1 and K_2 to obtain the cipher images C_1 and C_2 and then calculate the NBCR(C_1,C_2); (3) In the decryption process, to test the key sensitivity, decrypt the cipher image C_1 using the secret keys K_1 and K_2 to obtain the decrypted images D_1 and D_2 and then calculate the NBCR(D1,D2); (4) Iterate the steps 1, 2 and 3 for all the 128 bits in the secret key K_1.

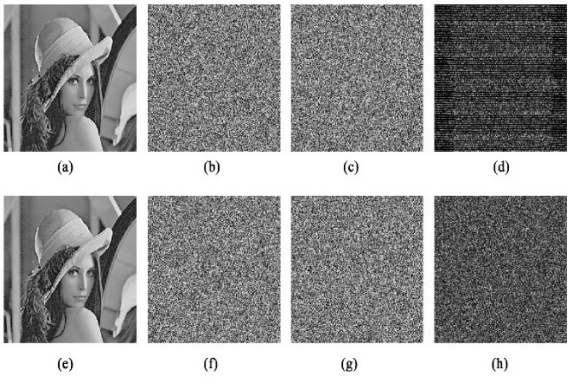

Figure 2: (a) The plain image P; (b) The cipher image C1 = EncK1(P); (c) The cipher image C2 =EncK2(P); (d) The different between C1 and C2; (e) The decrypted D1 = DecK1(C1); (f) The decrypted D2 = DecK2(C1); (g) The decrypted D3 = DecK3(C1); (h) The different between D2 and D3.

The secret key should be changed with extreme care in a good image encryption scheme. In other words, you can correctly decrypt the plain image if you use the right secret key. Several experiments have been conducted to evaluate the encryption schemes' key sensitivity. To achieve this, we generate a 128-bit secret key K_1 at random, change one bit of K_1, and then generate K_2 and K_3 secret keys that are similar to K_2. We create two cypher images, C1 and C2, and then encrypt the plain image P, depicted in Figure 2(a), using the secret keys K_1 and K_2 (c).

Fig. 2 displays the decrypted image C1, which was done with the right secret key K1 (e). The decrypted image C_1 is shown in Fig. 2(f), (g), and (h). These noisy and dissimilar images were created using two incorrect secret keys, K_2 and K_3. The experimental results demonstrate that even a small modification to the encryption schemes' secret keys results in a completely different ciphered image. As a result, the secret key plays a critical role in the proposed image encryption scheme. The proposed image encryption scheme's key sensitivity analysis results are shown in Fig. 3. The NBCR of the two obtained cypher images in the encryption and decryption processes are 50% on average by changing one bit of the secret key. In other words, the two obtained cypher images and the two obtained decrypted images are completely different, and the proposed encryption scheme has a secret key that is extremely sensitive to changes of one bit.

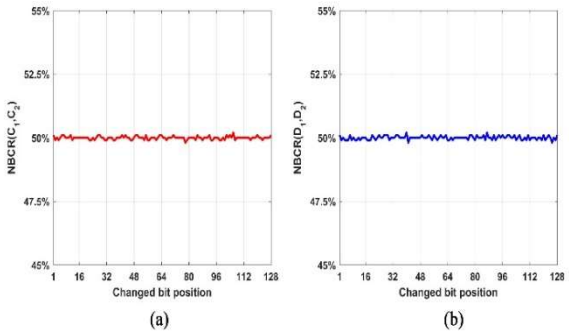

Figure 3: Key Sensitive Analysis of the proposed Algorithm.

D. Correction Analysis

The correlation coefficient in a digital image reflects the relationship between its pixels. Usually, the correlation between adjacent pixels in the plain image (in different direction vertical, horizontal, and diagonal) is high. An ideal encryption scheme should reduce the correlation between adjacent pixels in the cipher image to an acceptable level to prevent the statistical attacks. The ideal value of correlation for cipher image is close to zero. In this paper, we selected 1000 pairs of adjacent pixels in different directions from plain image and cipher image randomly, and compute correlation coefficients for all of them. Table 4 shows the result of the correlation coefficient of plain image Lena of sizes 1024×1024, and 8192×8192 the correlation coefficient of the corresponding chipper images. The results demonstrate that the correlation coefficients of the plain images in different sizes are quite high, but the correlation coefficients of the cipher image are extremely low and close to zero. The proposed encryption scheme could efficiently decrease the high correlations between adjacent pixels of the plain image.

Table 4: The correlation coefficient of adjacent pixels in the plain images and their corresponding chipper images

Image Size	Image Type	Horizontal Direction	Vertical Direction	Diagonal Direction
1024 × 1024	plain	0.9810	0.9891	0.9670
	cipher	-0.0204	0.0153	-0.0122
8192 × 8192	plain	0.9997	0.9956	0.9950
	cipher	0.0052	0.0011	-0.0351

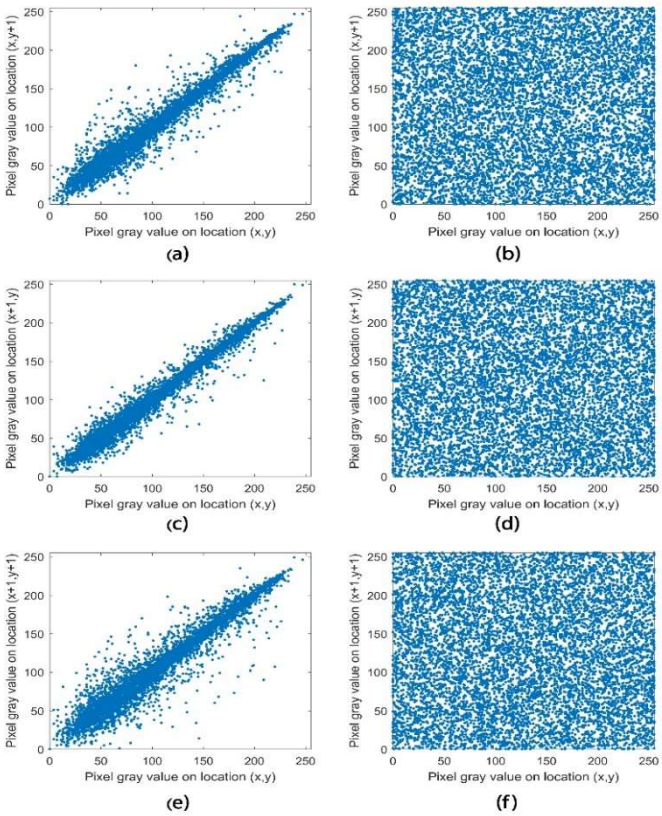

Figure 4: Correlations of two adjacent pixels. (a) Vertical direction of the Lena plain-image 1024×1024, (b) Vertical direction of the Lena cipher-image 1024×1024, (c) Horizontal direction of the Lena plain-image 1024 × 1024, (d) Horizontal direction of the Lena cipher-image 1024×1024, (e) Diagonal direction of the Lena plain-image 1024×1024 and (f) Diagonal direction of the Lena cipher-image 1024 × 1024.

Fig. 4 shows the results of the correlation of two adjacent pixels of the plain image Lena and the cipher image Lena of size 1024 × 1024. The results demonstrate that the correlation coefficients of the plain images in different sizes are quite high, but the correlation coefficients of the cipher image are extremely low and close to zero. The proposed encryption scheme could efficiently decrease the high correlations between adjacent pixels of the plain image.

E. Differencial Attack

In this attack, the attacker may create a slight change in the original image and find the encrypted image of it with the original encrypted image and find a meaningful relationship between them. To test the resistance to the attack, two coefficients of NPCR and UACI are used. The NPCR value for the Lena is close to 100% and the UACI value is close to 33.5% showing the sensitivity of the method to the original image changes is showing.

V. CONCLUSION

The paralleling capabilities of GPU and CPU were used in this paper to propose a new parallel image encryption algorithm based on two-dimensional cellular automata. The implementation of C++ on CPU and GPU using Visual Studio, OpenMP, and CUDA were all examples of the paralleling style. The permutation and substitution operations for each pixel of the image are accelerated in order to accomplish this task using parallel threads running on GPU. According to the performance data, the proposed algorithm on a GPU can process plain images of sizes 1024 ×1024, and 8192×8192 up to 27, and 28 times faster than its sequential counterpart. Additionally, the security analysis demonstrates that the proposed algorithm is secure against statistical and differential attacks. It also comes with a sufficiently large key space and being extremely sensitive to even the smallest change to the secret key and plaintext.

REFERENCES

[1] M. J. Rostami, A. Shahba, S. Saryazdi, H. Nezamabadi-pour, A novel parallel image encryption with chaotic windows based on logistic map, Computers & Electrical Engineering 62 (2017) 384–400.

[2] Q. Zhou, K.-w. Wong, X. Liao, T. Xiang, Y. Hu, Parallel image encryption algorithm based on discretized chaotic map, Chaos, Solitons & Fractals 38 (4) (2008) 1081–1092.

[3] F. K. Mohamed, A parallel block-based encryption schema for digital images using reversible cellular automata, Engineering Science and Technology, an International Journal 17 (2) (2014) 85–94.

[4] M. Machkour, A. Saaidi, M. Benmaati, A novel image encryption algorithm based on the two-dimensional logistic map and the latin square image cipher, 3D Research 6 (4) (2015) 36.

[5] E. Z. Zefreh, S. Rajaee, M. Farivary, Image security system using recursive cellular automata substitution and its parallelization, in: 2011 CSI International Symposium on Computer Science and Software Engineering (CSSE), IEEE, 2011, pp. 77–86.

[6] X. Wei, L. Guo, Q. Zhang, J. Zhang, S. Lian, A novel color image encryption algorithm based on dna sequence operation and hyper-chaotic system, Journal of Systems and Software 85 (2) (2012) 290–299.

[7] R. Bhardwaj, D. Bhagat, two level encryption of grey scale image through 2d cellular automata, Procedia Computer Science 125 (2018) 855–861.

[8] J. Jin, an image encryption based on elementary cellular automata, Optics and Lasers in Engineering 50 (12) (2012) 1836–1843.

[9] Y. Wang, Y. Zhao, Q. Zhou, Z. Lin, Image encryption using partitioned cellular automata, Neurocomputing 275 (2018) 1318–1332.

[10] A. Bakhshandeh, Z. Eslami, an authenticated image encryption scheme based on chaotic maps and memory cellular automata, Optics and Lasers in Engineering 51 (6) (2013) 665–673.

[11] P. Fang, H. Liu, Ch. Wu, M. Liu, A survey of image encryption algorithms based on chaotic system, The Visual Computer 39 (1) (2023) 1975-2003.

[12] T. Chen, M. Zhang, J. Wu, C. Yuen, Y. Tong, Image encryption and compression based on Kronecker compressed sensing and elementary cellular automata scrambling, Optics & Laser Technology 84 (2016) 118–133.

[13] X. Wang, D. Luan, A novel image encryption algorithm using chaos and reversible cellular automata, Communications in Nonlinear Science and Numerical Simulation 18 (11) (2013) 3075–3085.

[14] D. Das, A. Ray, A parallel encryption algorithm for block ciphers based on reversible programmable cellular automata, arXiv preprint arXiv:1006.2822 (2010).

[15] A. Hernandez-Becerrjl, M. Nakano-Miyatake, H. M. Perez-Meana, A. Bucio, M. Ramirez-Tachiquin, A parallel authenticated encryption sharing scheme based on cellular automata, in: Proceedings of the World Congress on Engineering and Computer Science, Vol. 1, 2014.

[16] X. Zhang, C. Wang, S. Zhong, Q. Yao, Image encryption scheme based on balanced two-dimensional cellular automata, Mathematical Problems in Engineering 2013 (2013).

[17] Kumar, A., Raghava, An efficient image encryption scheme using elementary cellular automata with novel permutation box, Journal of Multimedia Tools and Applications 80 (1) (2021) 21727–21750.

[18] George Cosmin Stănică, Petre Anghelescu, Cryptographic Algorithm Based on Hybrid One-Dimensional Cellular Automata, in Mathematics 11 (6) (2023) 1481.

[19] M. Fazlali, P. Khodamoradi, F Mardoukhi, M. Nosrati MM. Dehshibi, Metamorphic malware detection using opcode frequency rate and decision tree, International Journal of Information Security and privacy. (IJISP) 10 (3) (2016), 67-86.

[20] L. Y. Zhang, C. Li, K.-W. Wong, S. Shu, G. Chen, cryptanalyzing a chaos-based image encryption algorithm using alternate structure, Journal of Systems and Software 85 (9) (2012) 2077–2085.

[21] H. Zhu, X. Zhang, H. Yu, C. Zhao, Z. Zhu, A novel image encryption scheme using the composite discrete chaotic system, Entropy 18 (8) (2016) 276.