

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Parallel processing of chaos-based image encryption algorithms

Permalink

<https://escholarship.org/uc/item/6zc2n027>

Author

Raman, Ashwin

Publication Date

2016

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Parallel processing of chaos-based image encryption algorithms

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Ashwin Raman

Thesis Committee:
Professor Jean-Luc Gaudiot, Chair
Professor Nader Bagherzadeh
Professor Chen-Yu Phillip Sheu

2016

TABLE OF CONTENTS

	Page
List of Figures	iii
List of Tables	v
Acknowledgments	vi
Abstract of the Thesis	vii
1. Introduction	1
1.1. Motivation	1
1.2. Organization of Thesis	2
2. Chaos Based Image Encryption	4
2.1. Chaotic Maps	5
2.1.1. Arnold Cat Map	6
2.1.2. Henon Map	7
2.1.3. Duffing Map	9
2.1.4. Cross Chaotic Map	10
3. Graphics Processing Units	12
3.1. NVIDIA - GPU architecture	14
3.2. CUDA – Programming model	15
4. Serial Implementation of Chaos based encryption algorithm	19
4.1. Image format	19
4.2. Proposed encryption algorithm using chaotic maps	20
4.3. Running Time and Security analysis	23
4.3.1. Security Analysis technique: Randomness test – NPCR and UACI	23

4.3.2. Experimental Results	25
5. Parallel Processing of Proposed Algorithm	34
6. Conclusion and Future Work	39
6.1. Future Research	40
References	41

LIST OF FIGURES

	Page
Figure 2.1a Matrix notation of Arnold Cat Map	6
Figure 2.1b Arnold Cat Map transformation	7
Figure 2.2a Lorenz Equations	7
Figure 2.2b Henon Map Equations	8
Figure 2.2c Henon Map after several iterations	9
Figure 2.2d Henon Map initial point	9
Figure 2.3a Duffing Map after several iterations	10
Figure 2.3b Duffing Map Equation	10
Figure 2.4 Cross Chaotic Map Equation	11
Figure 3.1 GPU acceleration framework	13
Figure 3.2 GPU vs CPU memory hierarchy	13
Figure 3.3 GeForce Modern GPU Architecture	14
Figure 3.4 TFLOPS trend for NVIDIA GPUs	15
Figure 3.5 CUDA Thread Organization	17
Figure 3.6 CUDA processing flow	17
Figure 3.7 CUDA compiling C application	18
Figure 4.1 PPM image format	19
Figure 4.2 Arnold Cat Map Serial implementation	20
Figure 4.3a Key generation algorithm	21
Figure 4.3b Encryption process	22
Figure 4.3c Decryption process	23

Figure 4.4	NPCR formula	24
Figure 4.5	UACI formula	25
Figure 4.6a	512*512 input image	27
Figure 4.6b	2048*2048 input image	29
Figure 4.6c	4096*4096 input image	31
Figure 5.1	Parallel encryption of 4096*4096 images using Cross chaotic and Arnold cat map	36

LIST OF TABLES

	Page
Table 3.1	CUDA function declarations 18
Table 4.1	Encryption results for 512*512 image size - single encryption step 27
Table 4.2	Encryption results for 512*512 image size - multiple encryption steps 28
Table 4.3	Encryption results for 2048*2048 image size - single encryption step 29
Table 4.4	Encryption results for 2048*2048 image size - multiple encryption Steps 18
Table 4.5	Encryption results for 4096*4096 image size - single encryption step 31
Table 4.6	Encryption results for 4096*4096 image size - multiple encryption Steps 32
Table 5.1	Encryption results for 4096*4096 image size – serial and parallel Processing 35
Table 5.2	Encryption time for 4096*4096 image pixels without key generation Time 37

ACKNOWLEDGMENTS

I would like to express my earnest gratitude to my advisor Professor Jean-Luc Gaudiot for all his support, guidance and encouragement that helped me throughout my research work. I would like to thank my committee members, Professor Nader Bagherzadeh and Professor Chen-Yu Phillip Sheu for lending their precious time and valuable guidance during this process.

I would also like to acknowledge other members of the Parallel Systems & Computer Architecture Lab (PASCAL) research group for their persistent help and constructive feedback to help me finish my thesis.

ABSTRACT OF THE THESIS

Parallel processing of chaos-based image encryption algorithms

By

Ashwin Raman

Master of Science in Computer Engineering

University of California, Irvine, 2016

Professor Jean-Luc Gaudiot, Chair

Previous researches have shown that image encryption could be done using techniques like DES, IDEA, RES but the new and effective way for fast and secure encryption using chaos-based cryptography is the most preferred encryption technique. Chaos-based encryption algorithms are a combination of multiple chaotic maps and the same process can be repeated for multiple cycles for higher security. But, as the number of steps to process an image increases the processing time increase too. Another, reason for increased processing time is the number of pixels being encrypted. Since, chaos-based algorithms are considered as ideal for encrypting images in real time applications, higher security and lesser response time are essential.

This research primarily focusses on examining encryption techniques using two-dimensional chaotic maps and comparing the encryption algorithms security level and response time for different image sizes. After serial implementation, parts of encryption and decryption process that can be parallelized are evaluated and implemented using GPU and CUDA programming and then quantitative results are compared with the serial implementation.

1. INTRODUCTION

1.1 Motivation

Due to the rapid advancement in digital information and communication, security has become an essential part of digital media. Images, videos and speech are being shared and distributed in various fields like public use it for bank transactions or business communications, government use it to share secret confidential data, and in the medical field, it is used to account patients reports. All these require user authentication, reliability and accuracy of data and encryption techniques are useful tools to provide that required security. Consumer electronics like mobile phones use wireless network to share and receive images and videos which have limited bandwidth that definitely needs multimedia security.

In real time applications, time to compress or decompress and encrypt or decrypt the images are major impediments and hence it becomes difficult to handle a large amount of data. Hence, it is important to decide the right encryption algorithm depending upon the requirements and resources. It is important to understand that traditional text encryption algorithms cannot always be used for multimedia encryption, because images and videos have larger, redundant data and pixel values are highly correlated with each other, hence using text encryption algorithms like AES, IDEA, RES will take large computational time, power and will require more space to process. Unlike, text encryption decrypted images are acceptable even if we have minor inconsistencies as compared to the input image until those differences are minimal and not noticeable.

In the past two decades, there have been several image encryption algorithms proposed, which can be broadly classified into three major group's position permutation [1], value transformation [2] and visual transformation based algorithms.

1.2 Organization of Thesis

The remainder of the thesis is structured into five chapters. Chapter 2 discusses chaos-based image encryption algorithms. Chapter 3 discusses parallel processing using Graphic Processing Units and CUDA programming. Chapter 4 evaluates the cryptanalysis of the two-dimensional chaos-based encryption algorithms discussed in chapter 2. Chapter 4 compares the parallel implementation of encryption and decryption process and does a detailed analysis. Chapter 6 concludes the thesis with a summary and potential future work.

Chapter 1 discusses chaotic maps and their key properties. Also, the close relationship between chaos theory and cryptography which will explain why chaos-based image encryption are often preferred over traditional encryption techniques. Then a detailed explanation of the four two-dimensional chaotic maps **Arnold Cap Map, Henon Map, Duffing Map and Cross Chaotic Map** used in this thesis.

Chapter 2 gives a brief overview of Graphic processing units and how it can be used in the parallel processing of encryption algorithms, a summary of NVIDIA's GPU architecture and how to use CUDA for parallel programming.

Chapter 3 evaluates the security properties of the different arrangements using the chaotic maps explained in earlier chapters. The security analysis is done using commonly used quantities like the **number of changing pixel rate** (NPCR) and the **unified averaged**

changed intensity (UACI). Then the response time for these arrangements is calculated for both encryption and decryption process in a serial implementation.

Chapter 4 is in succession to Chapter 3 where the serial implementation of image encryption algorithms are analyzed and parallel processed using GPUs and then the rate of speedup is determined.

Chapter 5 summarizes the thesis work and gives remarks on the future scope.

2. CHAOS BASED IMAGE ENCRYPTION

All systems can be broadly classified as deterministic, stochastic (probabilistic) or chaotic systems of which chaotic systems are most unpredictable. Chaotic maps are often used in the study of dynamical systems which exhibit behavior that is highly sensitive to initial conditions and even small perturbations can yield widely diverging outcomes. Still these systems are deterministic because based on the initial condition future behavior can be predicted, hence, their behavior could be called as **deterministic chaos**.

There is a close relationship between chaotic systems and cryptography which makes chaos based algorithms a natural candidate for image encryption. The two basic properties of a good cipher are confusion and diffusion and both these are important features of chaotic systems too. For real-time applications encryption schemes which take lesser computational time but wouldn't compromise with the desired security are suitable. And chaos-based encryption technique is a good amalgamation of high speed, security, complexity and less power consumption. [3] provides further details about the relation between chaotic systems and cryptographic algorithms.

The basic principle of chaos-based encryption is to use the dynamical systems to generate a sequence of numbers that are pseudo-random in nature and these sequences could be used as a key to encrypt input image. For given parameters [4] two initial conditions can deviate exponentially into two different trajectories. These parameters can be used for encryption and decryption and keys can be chosen from these conditions. Due to these chaotic parameters and initial condition we could generate a large key space which further enhances the security. Because of the random behavior, the output seems random to the attacker whereas only the sender and receiver know that the system is well defined.

Also, these algorithms are easier and cheaper to be embedded onto small chips, which make them a good fit for **cellular systems**.

2.1 Chaotic Maps

Chaotic maps can be represented using continuous and discrete time parameters. The maps are usually iterative functions with the general representation of $f: X \rightarrow X$. The process of recurrently calling the same function, where the result generated from the initial condition is fed to the same function again and the process is continued. The output from these chaotic maps exhibits **fractal**-like property. Fractals are expanding symmetry which are repeating patterns, hence, it won't be incorrect to say that chaotic maps have periodicity.

Before discussing the chaotic maps, it is important to understand the composition of images. They are composed of distinct entities known as pixels which are represented using $m \times n$ matrix (m rows and n columns) of pixels.

The number of bits used to represent pixel tells the range of colors an image can represent. The color scale can be divided into two parts 8 bits per pixel and 24 bits per pixel. An 8 bits pixel is able to represent 2^8 or 256 different colors of grays, rather the pixel values represent the intensity of black and white colors. On the other hand, color pixels are composed of three colors red, green and blue. But, it requires 24 bits to represent color pixels which increase the range of color variation to 2^{24} or 16,777,216 different colors.

Even though we get a wide range of colors but it gives a sense the amount of processing almost increases by three times as compared to grayscale images. These representations are also known as true color. There are more advanced coloring schemes

for pixels which fall under deep color with 30 or 36 or 48 bits, but they are out of the scope of this thesis.

The following sections explain in detail about the chaotic maps in discrete time domain used in this thesis.

2.1.1 Arnold Cat Map

Arnold Cat maps which was named after Vladimir Arnold. He used an image of a cat to display the effect of this chaotic map. In this mapping technique, images go through a transformation that randomizes the original image pixels. It is a good example of hyperbolic toral automorphism where a torus is given by a square matrix. Figure 2.1a shows the matrix notation of the mapping transformation.

$$\Gamma \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \bmod n$$
$$\Gamma \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x + y \\ x + 2y \end{bmatrix} \bmod n$$

Figure 2.1a: Matrix notation of Arnold Cat Map [5]

Some key features of this mapping technique are it is area preserving that is the transformed image requires the same area as the actual image, it can be even deduced as the determinant of the matrix is 1. Also, if the image is iterated several times the original image reappears. [6] gives an overview of the properties of Cat map and basic principles of

chaos-based systems. Figure 2.1b shows how the linear map changes the unit square and how the pieces are rearranged after modulo operation.

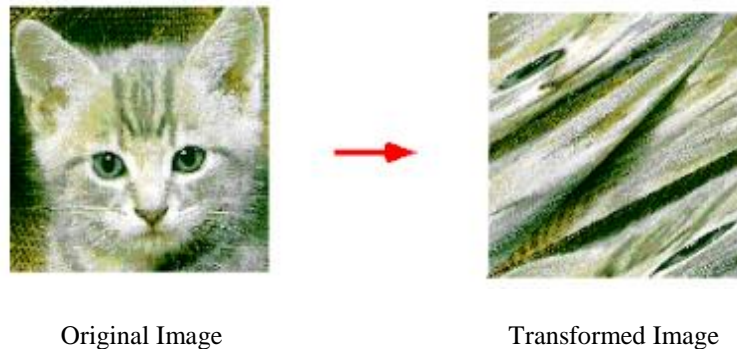


Figure 2.1b: Arnold Cat Map transformation

Even though the usefulness of Arnold Cat Map is limited, but it illustrates the power of the science behind the chaos-based system.

2.1.2 Henon Map

Henon Map is a discrete dynamic system, which was developed by Michel Henon as a simplified version of Lorenz model. In 1963, Edward Lorenz examined three first-order differential equation which was attracted to a strange attractor. It had chaotic behavior for several parameter values and initial conditions. These equations as shown in figure 2.2a are non-linear, deterministic and three dimensional. Because, of its simplicity they were widely used in areas like electric circuits, motors, chemical reactions.

Keeping the same essential properties as the Lorenz systems, Henon developed a model which was more accurate, faster and can be more mathematically analyzed.

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z.\end{aligned}$$

Figure 2.2a: Lorenz Equations [7]

Henon carried out the experimentation using the formula defined in figure 2.2b for initial conditions $a=1.4$ and $b=0.3$ and based on an initial (X_0, Y_0) condition, the sequence generated a Henon attractor, which was diverging to infinity or was converging to a strange attractor. Figure 2.2c shows the Henon attractor after several successive iterations starting from (X_0, Y_0) . [8] explains in detail the properties of Henon map and how it is derived using Lorenz systems.

The value of 'a' and 'b' was decided after carrying out the experimentation for a wide range of arrangements and it was found that not all of them have a strong attractor. Also, as discussed earlier chaotic equations depend largely on the initial condition, as the subsequent x and y coordinates are determined using them. For a given 'a' and 'b' values two unstable initial points are deduced to be $x_0=0.631354477$ and $y_0=0.189406343$ which is derived using the calculations shown in figure 2.2d. Points close to this point either converge or diverge towards a fixed point or strong attractor.

$$\begin{aligned}x_{n+1} &= 1 - ax_n^2 + y_n \\ y_{n+1} &= bx_n\end{aligned}$$

Figure 2.2b: Henon Map Equations [9]

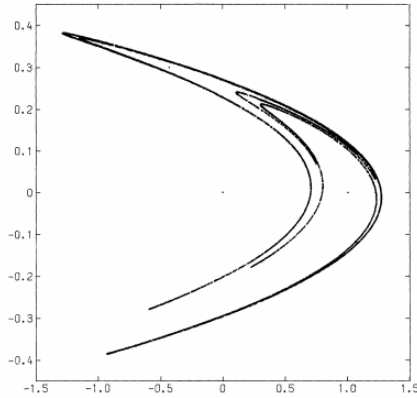


Figure 2.2c: Henon Map after several iterations [9]

$$x = \frac{\sqrt{609} - 7}{28} \approx 0.631354477,$$

$$y = \frac{3(\sqrt{609} - 7)}{280} \approx 0.189406343$$

Figure 2.2d: Henon Map initial point [9]

2.1.3 Duffing Map

Another famous discrete-time dynamical system which exhibits chaotic behavior is Duffing map, figure 2.3a shows the x and y mapping equation. Similar, to Henon map, it takes input (X_n, Y_n) to generate (X_{n+1}, Y_{n+1}) and hence it is critical to decide the right value of 'a' and 'b' so that the behavior is chaotic. For $a=2.75$ and $b=0.2$ [10], Duffing map produces the plot shown in figure 2.3b. Duffing map is derived from a discrete version of Duffing equation, which is often used for calculating oscillator's displacements, velocity, and acceleration. There are no specific research papers which analyze the output for different initial conditions, so for this research work $x_0=0.1933$ and $y_0=0.8087$ was derived using a random number generator.

$$x_{n+1} = y_n$$

$$y_{n+1} = -bx_n + ay_n - y_n^3$$

Figure 2.3a: Duffing Map Equation

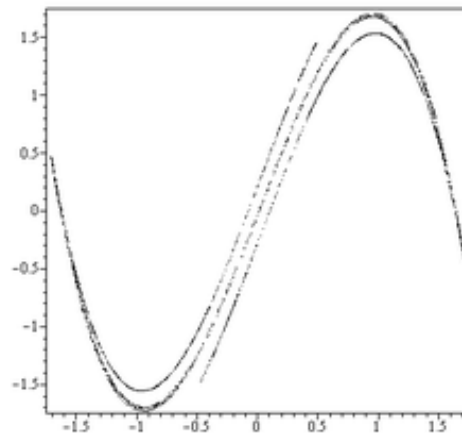


Figure 2.3b: Duffing Map after several iterations

2.1.4 Cross Chaotic Map

Cross chaotic map is an amalgamation of two chaotic maps, Logistic, and Chebyshev, the equation is being referenced from [11]. Both, these algorithms are one dimensional and non-linear dynamic systems and in order to reduce doing the multipart calculations, it is more efficient to combine these chaotic maps as shown in figure 2.4 and achieve better security level by using the resultant map in two dimensions.

As per the evaluation in [12], for values, of $\mu=2$ and $k=6$ the system produces great dynamic behavior. Similar to Duffing map the points $x_0=0.1933$ and $y_0=0.8087$ generated using a random generator is used as initial points for the cross chaotic map.

$$x_{n+1} = f(x_n) = \cos(k \cdot \cos^{-1} x_n) \quad -1 \leq x_n \leq 1 \quad k \in \mathbb{Z}^*$$

Chebyshev Map

+

$$x(n+1) = \mu x(n)[1-x^2(n)]$$

Logistic Map

=

$$\begin{aligned} x_{i+1} &= 1 - \mu \cdot y_i \cdot y_i \\ y_{i+1} &= \cos(k \cdot \cos^{-1} x_i) \end{aligned}$$

Figure 2.4: Cross Chaotic Map Equation

3. GRAPHICS PROCESSING UNITS

Graphics Processing Units have become an important part of current computational systems. This progress has been possible because of the stagnation in traditional CPU clock speed and more people have started shifting focus on using GPUs for general purpose computing. In the past decade, there has been a significant improvement in the GPU performance and programmability. It has made GPU appropriate for certain types of applications with specific characteristics with large computation requirements and considerable parallelism.

In 1999, the term Graphic processing unit was coined when Nvidia introduced the first GPU GeForce 256. At the advent GPUs were designed for real-time graphics which saw its use in fields of physics, medical imaging etc. Since it was built specifically for graphic applications, it was programmed using OpenGL programming language. But looking at the bigger scope where GPU computing could be used, new programming languages such as CUDA and OpenCL were developed that made GPUs more programmable and a new subfield of research is known as GPGPU or general purpose computing on GPU found its ways into fields as varied as hydrology, seismography, machine learning, image processing, stock market price estimation and statistics. [13] has information about how GPU has evolved over the years and about the different fields it is being used.

GPU-accelerated computing uses GPUs along with Central Processing Units (CPU) to accelerate applications that have ample parallelism. These performance gains are best realized when the computation intensity is high and elements are largely independent. GPU can provide the speed-up by offloading the compute-intensive portions of application from CPUs as shown in figure 3.1. GPU has more transistors dedicated for data processing

than for cache and control as in CPU, shown in figure 3.2. Another thesis [11] discusses in detail about GPU computation, architecture, and memory hierarchy.

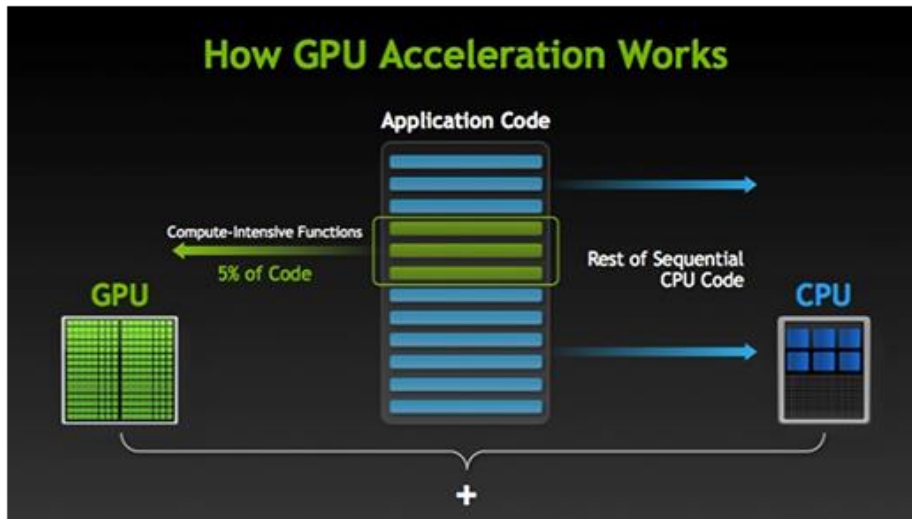


Figure 3.1: GPU acceleration framework [14]



Figure 3.2: GPU vs CPU memory hierarchy [15]

CPU and GPU use different procedures to achieve parallelism, CPU makes the workload run faster by running one task per thread by using caches, instruction or data prefetching, including branch prediction and concurrency control. Then to further improve

the performance add task parallelism by using multi-core architecture. Whereas, GPU makes the same workload run faster by using 1000s of threads run parallel and use pipelining.

GPU can't be considered as replacements to CPUs as it is not a universal solution to every computation. There are more serial tasks as compared to parallel, where CPUs performance is better. Also, with multicore CPUs being developed, some lightweight parallelism can be handled easily handled with CPUs more efficiently.

3.1 NVIDIA - GPU architecture

Figure 3.3 shows a basic CUDA capable GPU which is organized into a set of blocks the size of which has varied with generations.

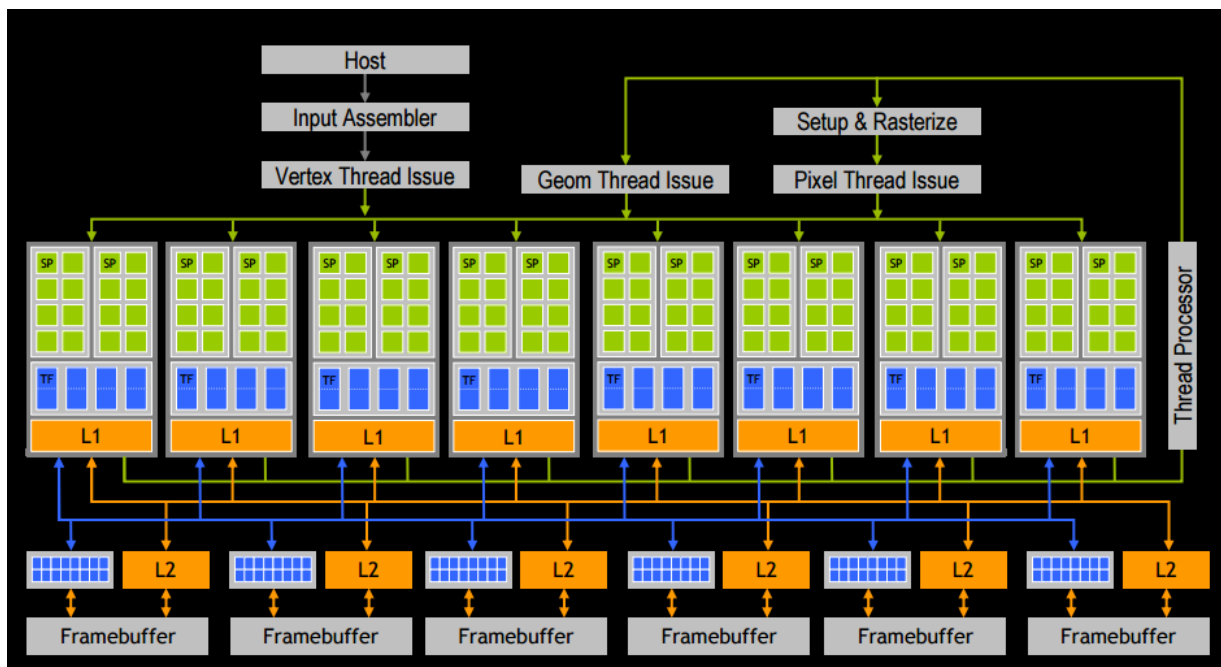


Figure 3.3: GeForce Modern GPU Architecture [16]

The single-precision floating point performance has improved significantly over the years, as compared to CPU, figure 3.4 shows the TFLOPS (trillion floating point operations per second) trend for GPU and CPUs. Because, of this rapid advancement in GPU computing it is adopted in prominent research projects like autonomous vehicles, recommendation engines, speech recognition, image detection etc.

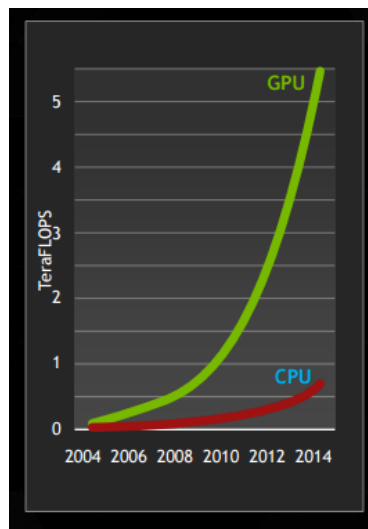


Figure 3.4: TFLOPS trend for NVIDIA GPUs

3.2 CUDA - Programming model

CUDA is the most commonly used parallel programming model and computing platform invented by NVIDIA. CUDA makes it easy to use high-level languages like C, C++ and Fortran on GPU, hence a single code can have some part that runs sequentially on CPU and some part that runs on GPU. Earlier Graphic Processing Units were exclusively used to render graphics, but over the course of time, GPU programming has improved by introducing several new extensions and functions.

Before, discussing in detail about CUDA programming in detail it is necessary to understand specific terms. First, GPU threads are not equivalent to CPU threads. GPU threads are lightweight, which reduces the complexity of context switching and they are the most elementary component that processes data.

GPU accelerated system comprises of two components host and device. Device means one or more GPUs used for parallelism, and the host is mostly the CPU hosting the application. The serial code executes on the host using CPU threads, the parallel code executes on one or many devices using GPU threads.

All GPU processes are termed as Kernel functions, which are executed by an array of threads. To manage thousands of threads run easily, they can be grouped into blocks which are further grouped into grids. Hence, kernel functions are executed as a grid of blocks of threads as shown in figure 3.5. All threads are organized in a block uses threadIdx indexes and blocks use blockIdx to be organized inside a grid, which is predefined variables in CUDA.

Figure 3.6 explains the basic processing steps for CUDA programming. **Copy processing data** from main memory to GPU memory, next CPU **instruct the processing** steps the GPU needs to run. GPU **executes the program parallel on data in each core** and then the parallel processed **results are copied back to the main memory**.

Kernel function called by the CPU can only access GPU memory, function declarations are defined with specific identifiers to distinguish if executed on and callable from host or device. Table 3.1 shows the functions declarations using qualifiers.

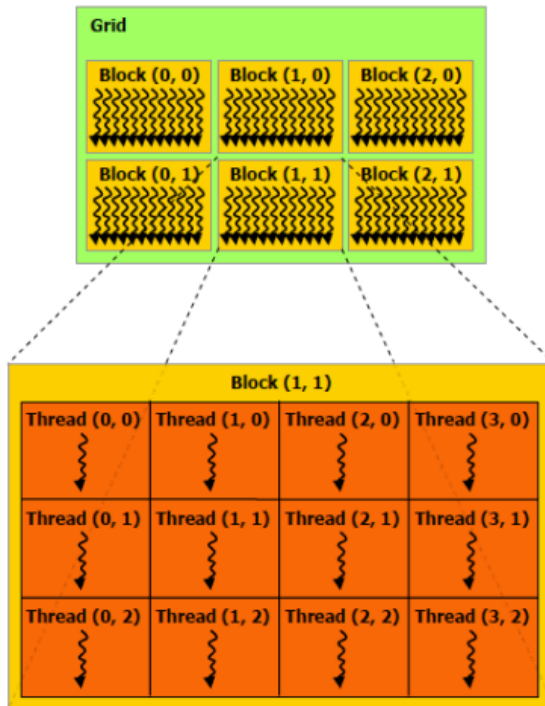


Figure 3.5: CUDA Thread Organization [17]

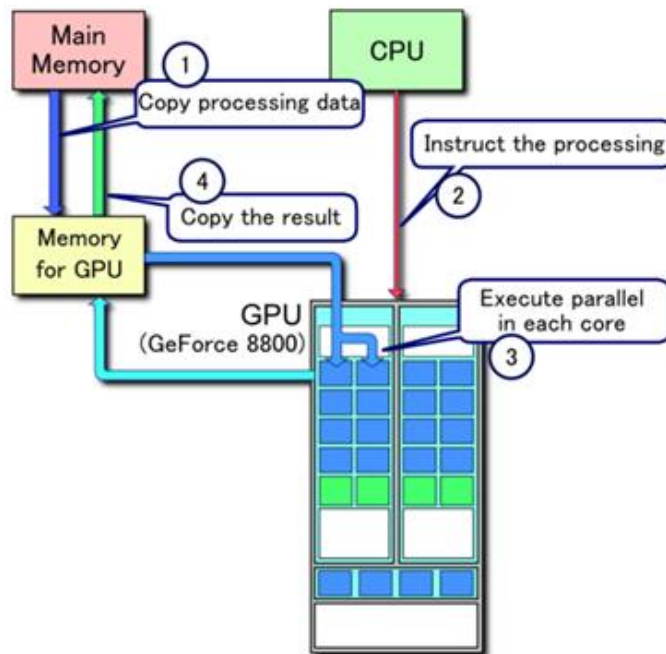


Figure 3.6: CUDA processing flow [18]

The CUDA compilation trajectory is very complicated, but the basic CUDA compilation is as shown in figure 3.7. The program is preprocessed for CUDA compilation and converted into CUDA binary (cubin). The input program is processed for CPU code into corresponding host binary file, both are linked together to obtain an appropriate executable binary file. Detailed information on CUDA programming model could be referenced from the CUDA programming guide [19].

	Called from:	Executed on:
<code>__global__ void kernelFunc()</code>	CPU (host)	GPU (device)
<code>__device__ float gpuFunc()</code>	GPU (device)	GPU (device)
<code>__host__ float cpuFunc()</code>	CPU (host)	CPU (host)

Table 3.1: CUDA function declarations

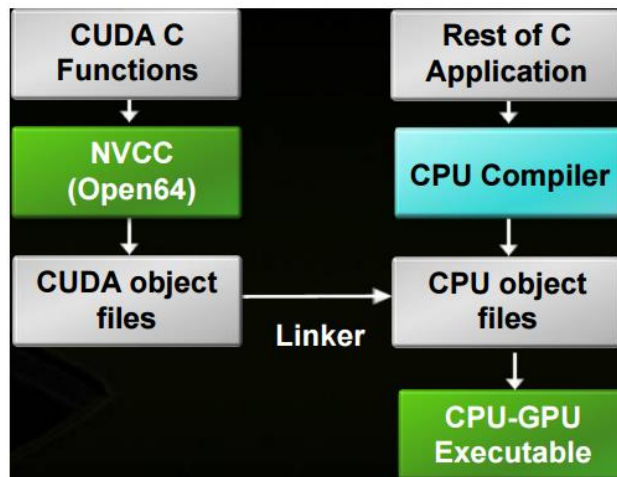


Figure 3.7: CUDA compiling C application

4. SERIAL IMPLEMENTATION OF CHAOS BASED ENCRYPTION ALGORITHM

4.1 Image format [20]

Having discussed 2d chaotic maps in chapter 2, this chapter will analyze the serial implementation of the four 2d chaotic maps used in this thesis. For this purpose a program using C language is developed for the different maps and their combinations.

Input image used is of **ppm** format, which is a non-compressed, lowest common denominator of the image and consumes more space than other compressed formats. It is an intermediate image format before being converted to a compressed and efficient image format.

Each ppm image file has the header data in format as shown in figure 4.1. It comprises of a “magic number” which can be P3 or P6. P3 image pixels have ASCII text format and P6 image pixels are stored in byte format. In this thesis, the image of type P6 is considered. The next line is for the image width and height respectively in ASCII decimal. The last part is for the maximum allowable pixel value if the value is 255 pixels can range from 0 to 255. These three lines comprise the information header for the ppm file. There can also be comments added using “#” sign then the comment text. After the header, the next set of values represent the pixel values in ASCII format. Pixel values are read from left to right which is stored in triplets of red, green and blue values. The pixel values can be 1 or 2 bytes long, depending upon the maximum limit set in the information header.

```
P6
1024 788
# A comment
255
```

Figure 4.1 PPM image format

4.2 Proposed encryption algorithm using chaotic maps

This section talks over about the serial implementation of two-dimensional chaotic maps using a C program.

For Arnold Cat map, it takes the pixel values and shuffles them using simple matrix multiplication as shown in figure 4.2 where the red, green and blue pixel values are transformed individually and to maintain the newly mapped index within the image size it is being modulated with the image width.

The limitation of using Arnold Cat map is that image width and height must be same for the transformation to work and decrypted and the original image to be same.

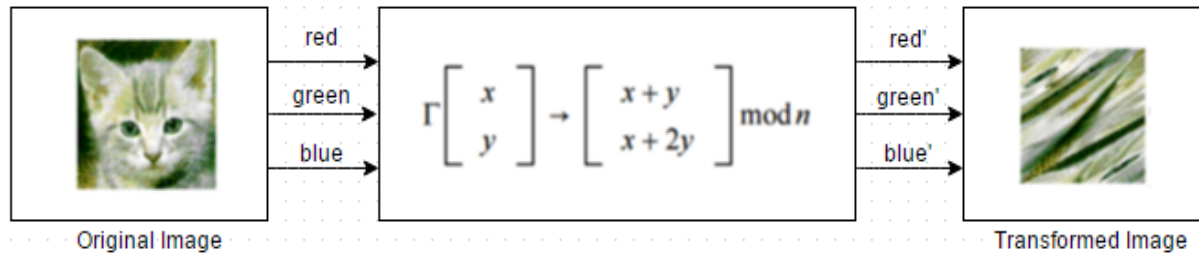


Figure 4.2 Arnold Cat Map Serial implementation

For the other three 2d chaotic maps the steps for the algorithm are different and the chaotic model formula changes, figure 4.2 shows the two step process of the algorithm's implementation.

The first step is to use the chaotic maps to run a recursive process where X_{n+1} and Y_{n+1} pseudorandom values are calculated using X_n and Y_n as shown in figure 4.3a. For starting the algorithm initial conditions x_0 and y_0 are explicitly defined, even the other variables used in the formula are predefined as discussed in Chapter 2 based on the state

when the system shows chaotic behavior. This process can run for any value of n, but these values are going to be used as a key for encrypting the pixel values, hence they need to be restricted by some constraint. The size of the key stream is a very important parameter when choosing an image encryption algorithm for real-time application. In this approach, the above process is repeated for N*N times, where N is the width and height of the image. Due, to this we will have one pseudorandom value for encrypting one pixel, also, since the key length depends on the image size it would be unique for a particular image size.

This key generation logic is based on the similar approach used in [21]. Next, the N*N image is broken into individual red, green and blue pixel values into a 2-d matrix. Every individual pixel value is then XORed using the key generated in the previous step as shown in figure 4.3b.

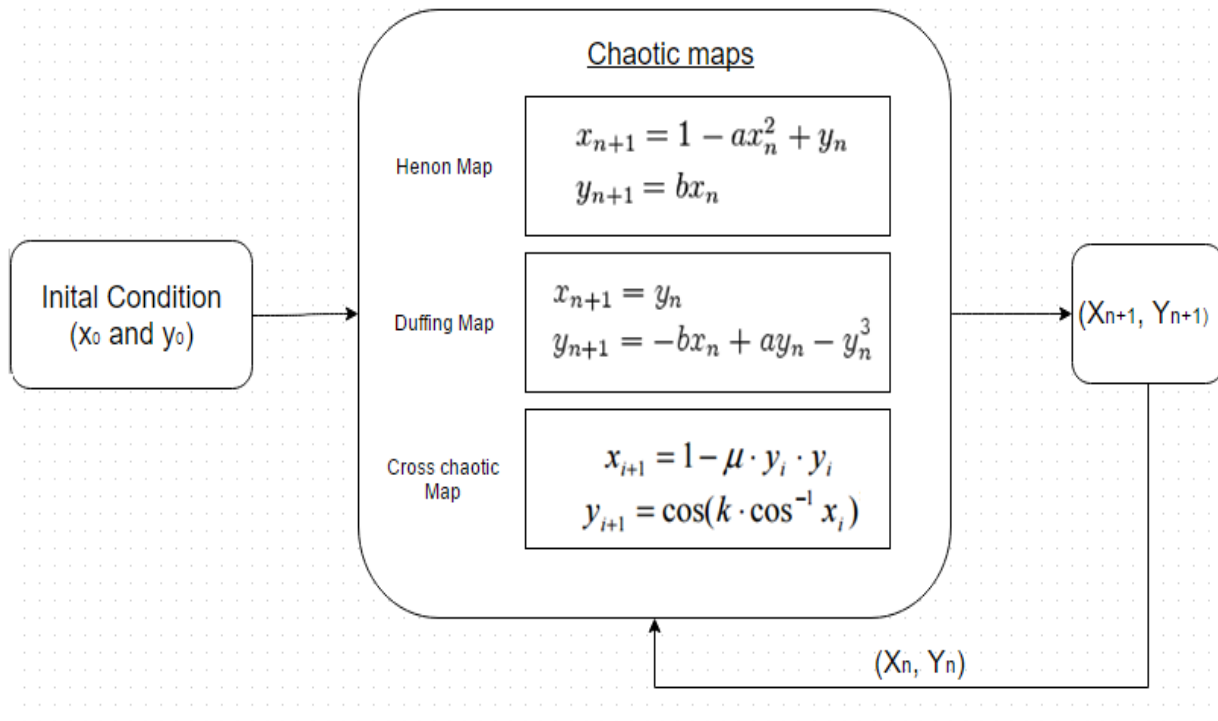


Figure 4.3a Key generation algorithm

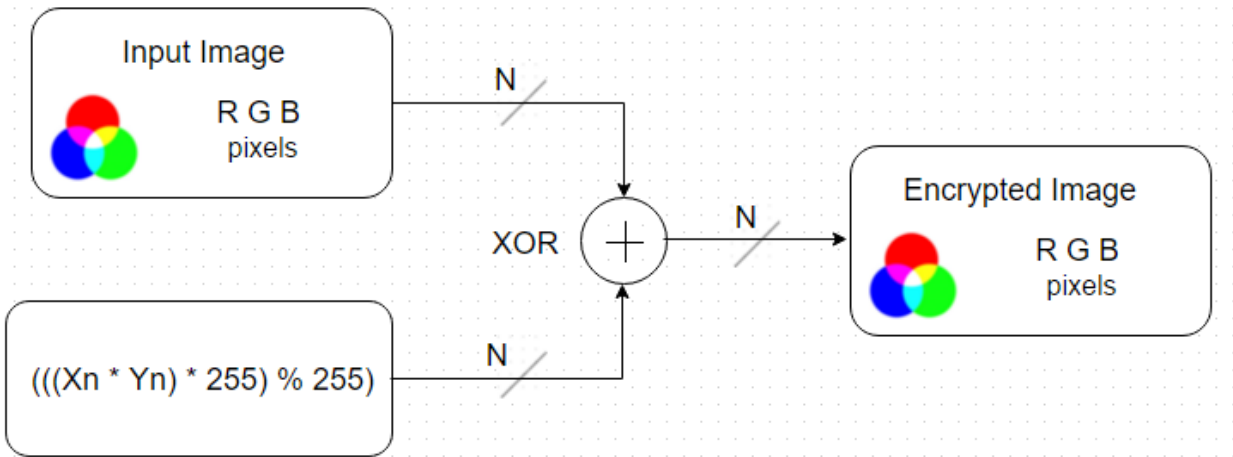


Figure 4.3b Encryption process

The key generated has x and y , but for XOR operation, we only need one value. Hence, the key value is obtained by multiplying the X_n with Y_n . In the three chaotic maps used the value of X_0 and Y_0 are considered between $(0, 1]$. Due to which most key values are limited to that range. To make the key values to be spread in the pixel range from $[0, 255]$. These result obtained after multiplying X_n and Y_n are even multiplied with 255 and later mod with 255 to keep the value within the maximum pixel level. The obtained output pixels forms the encrypted image.

The decryption process is fairly simple because it is just repeating the same steps with the encrypted image to get the original image. Arnold cat map can be decrypted by reversing the assignment. This is the reason to use an equal width and height image so that the area is preserved and no pixel value is lost in the decryption process. In the case of Henon, Duffing and Cross chaotic maps the key generation algorithm will remain similar, but instead of XORing, the input pixels and key, the encrypted image pixels and key will be XORed to get the decrypted image pixels as shown in figure 4.3c. Because, XOR cipher is

like an additive cipher, which implies an input can be encrypted by applying the bitwise XOR function with key and when reapplying the same XOR operation will even remove the cipher key.

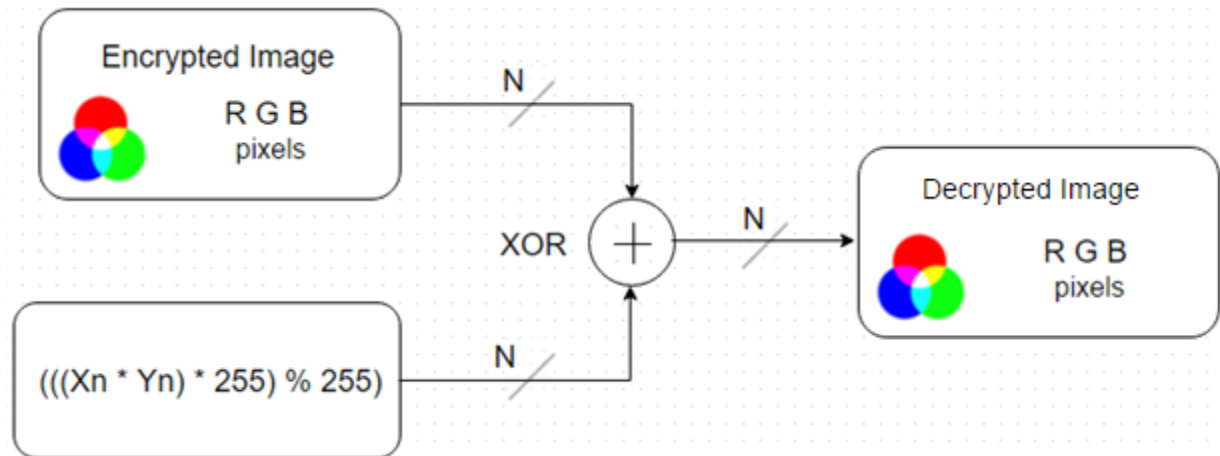


Figure 4.3c Decryption process

4.3 Running time and security analysis

4.3.1 Security analysis technique: Randomness test – NPCR and UACI

In the past decade, there have been several cryptosystems developed and they are all being evaluated using cryptanalysis of differential attack. Cryptanalysis became important after Eli Biham and Adi Shamir published a paper [22] which did a security analysis of Data Encryption Standard (DES) and various other ciphering techniques. Since then it is an important evaluation criterion to test a new algorithm. But, it is primarily used for block ciphers, can be used for stream ciphers or hash functions too. It calculates the independence between input information and resultant output, and testing if the algorithm exhibit non-random behavior which allows the secret key to extract easily.

But, these evaluations were based on binary representation which is very different than the image encryption input. An image is a two-dimensional matrix of data with a lot of redundancy and a large amount of data as compared to text input. All these features compel to look for a better randomness test.

In image encryption, algorithms strength to resist different attacks is usually evaluated using two most common quantities **number of changing pixel rate** (NPCR) and the **unified averaged changed intensity** (UACI). It is often regarded that a high NPCR and UACI value means a higher security level, but it is still not clear if that is always applicable, because there are some odd use cases where even with higher NPCR and UACI values the security level is not high.

As shown in figure 4.4 NPCR value focusses on the entire number of pixels where C^1 and C^2 are images before and after encryption respectively and $C^1(i, j)$ and $C^2(i, j)$ are one-pixel value at i and j coordinate. T denotes the total number of pixels. If the pixel values of input and output images are same then the bipolar array D has a value 0 and if the values are different then the value is 1. The overall NPCR value ranges between $[0, 1]$.

$$D(i, j) = \begin{cases} 0, & \text{if } C^1(i, j) = C^2(i, j) \\ 1, & \text{if } C^1(i, j) \neq C^2(i, j) \end{cases} \quad (1)$$

$$\text{NPCR: } \mathcal{N}(C^1, C^2) = \sum_{i, j} \frac{D(i, j)}{T} \times 100\% \quad (2)$$

Figure 4.4 NPCR formula

Another commonly used parameter for randomness test is UACI which is used to calculate the averaged difference between the input and encrypted image pixels, shown in figure 4.5. Here $C^1(i, j)$ and $C^2(i, j)$ denote the same as NPCR pixel values, here F denotes the largest supported pixel value, for this thesis it will be 255. Hence, even UACI value ranges from $[0, 1]$.

$$\text{UACI: } \mathcal{U}(C^1, C^2) = \sum_{i,j} \frac{|C^1(i, j) - C^2(i, j)|}{F \cdot T} \times 100\%$$

Figure 4.5 UACI formula

Based on the experimental findings in [23] comparing two encryption outputs based on their test scores quantitatively is not accurate. It is noticeable that NPCR values are often close in the range of 99-100%. Hence, it is preferable to have a high NPCR value, but the differences are not very significant. But, UACI values calculated using numerical and experimental results it is clear that many image encryption methods fail UACI test because of either too low or too high scores. For different image size, we have different theoretical UACI critical values anything outside that range is fails the randomness test.

4.3.2 Experimental Results

These experiments are performed on different image sizes using the above-mentioned encryption techniques, based on which the encryption and decryption times, NPCR and UACI values are calculated. These experimental results show that to achieve higher security processing time increases substantially too.

If we combine the different chaotic maps as a combination of encryption steps the processing time will be almost equivalent to the summation of the individual encryption process. Hence, it is critical to maintaining a balance between achieving better security and running time of the algorithm. Since chaotic maps are used for image encryption in real-time applications it is important to improve the processing time.

Image encryption and decryption time, NPCR and UACI values are dependent upon two factors image or pixel size and chaotic map used. In differential attacks, if a minor change in the input image can bring significant change in the cipher image then the differential attack is difficult and NPCR and UACI are the used to measure them.

In case, of encrypting using Arnold Cat Map it is relatively faster than the other chaotic maps, but the NPCR and UACI are smaller, which shows that the degree of security can be weaker when we compare with other chaotic maps which have more complex quadratic, cubic and trigonometric key generation algorithms and encryption technique.

As, mentioned earlier NPCR and UACI values are often considered with certain upper and lower bound, hence in certain odd cases if we have a higher NPCR and UACI values can't be called as better encryption algorithm unless this trend is consistent across multiple input images with different pixel sizes.

The following experiments are performed using one chaotic map or by combining Henon, cross chaotic and duffing maps with Arnold Cat Map. Based, on the results, it is evident that as the image size increases the encryption and decryption time increases much faster, which is a bottleneck when faster response time is expected.

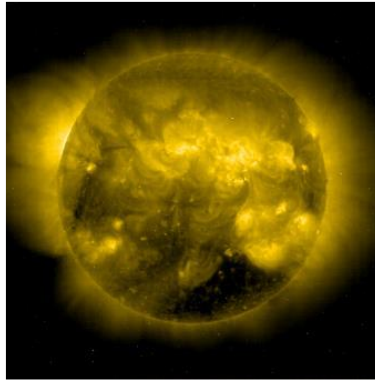


Figure 4.6a 512*512 input image [24]

Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
Arnold Cat Map	11	11	90.2584080	31.592369
Henon Map	37	34	98.690414	30.126399
Cross Chaotic Map	86	71	98.745346	35.752609
Duffing Map	45	33	99.385071	37.220723

Table 4.1 Encryption results for 512*512 image size - single encryption step

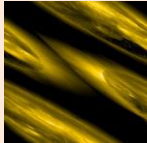
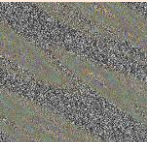
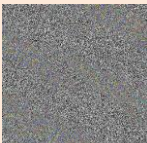

Encrypted Image + Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
 Arnold Cat Map	11	11	90.2584080	31.592369
 Arnold Cat Map + Henon Map	50	48	99.541092	36.888738
 Arnold Cat Map + Cross Chaotic Map	109	90	99.533844	36.731957
 Arnold Cat Map + Duffing Map	50	47	99.602509	36.754821

Table 4.2 Encryption results for 512*512 image size - multiple encryption steps



Figure 4.6b 2048*2048 input image [25]

Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
Arnold Cat Map	337	207	99.295521	23.552930
Henon Map	574	595	98.669314	26.934233
Cross Chaotic Map	1194	1234	98.743224	32.040857
Duffing Map	671	600	99.398541	33.488615

Table 4.3 Encryption results for 2048*2048 image size - single encryption step


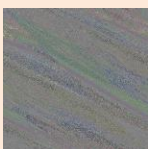

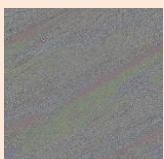
Encrypted Image + Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
 Arnold Cat Map	337	207	99.295521	23.552930
 Arnold Cat Map + Henon Map	786	800	99.577570	31.871477
 Arnold Cat Map + Cross Chaotic Map	1390	1363	99.599671	32.699511
 Arnold Cat Map + Duffing Map	799	809	99.613428	33.114105

Table 4.4 Encryption results for 2048*2048 image size - multiple encryption steps



Figure 4.6c 4096*4096 input image

Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
Arnold Cat Map	710	946	99.292421	23.546194
Henon Map	2259	2183	98.690414	26.947997
Cross Chaotic Map	4801	4687	98.736620	32.036465
Duffing Map	2681	2384	99.400342	33.524418

Table 4.5 Encryption results for 4096*4096 image size - single encryption step


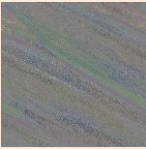


Encrypted Image + Encryption Technique	Encryption Time (ms)	Decryption Time (ms)	NPCR	UACI
 Arnold Cat Map	710	946	99.292421	23.546194
 Arnold Cat Map + Henon Map	3069	3207	99.580973	31.889175
 Arnold Cat Map + Cross Chaotic Map	5536	5770	99.602526	32.745837
 Arnold Cat Map + Duffing Map	3202	3266	99.615389	33.144419

Table 4.6 Encryption results for 4096*4096 image size - multiple encryption steps

On comparing the NPCR and UACI values cross chaotic and duffing maps are more efficient than the others also when the encryption process comprises of multiple

encryption steps using a combination of Arnold Cat map with Henon, Duffing and cross chaotic maps the results are further improved, but this increases the total encryption time even more. There is a direct co-relation between the image size, level of security and encryption time.

5. PARALLEL PROCESSING OF PROPOSED ALGORITHM

Important properties of image encryption algorithm are redundancy and a large quantity of data. But, often these image pixels are independent of each other, and hence, can be managed independently. The response time for the image encryption algorithm implemented in the previous step can be improved if the encryption process can be parallelized using graphic processing units (GPU).

Chaos-based encryption algorithm has two phases, first is key generation using the two-dimensional chaotic maps, followed by XOR operation using those key to encrypt each pixel value. The key generation process is recursive where (X_{n+1}, Y_{n+1}) key values are dependent upon (X_n, Y_n) key values, which makes it difficult to parallelize the key generation process, hence the initial focus is on parallelizing the encryption phase and to improve the response time.

To perform the parallel processing the GPU used is **Tesla M2090**. It will have the following configuration of threads,

$$1 \text{ block} = \mathbf{256 \text{ threads}}$$

If the image is 2048×2048 which is the number of pixels, therefore, we need 2048×2048 threads running in parallel based on which the number of blocks required is 16384.

$$2048 \times 2048 / 256 = \mathbf{16384 \text{ blocks}}$$

If the serial key generation and serial encryption processes are separated into two phases, and their individual time is being considered it can be seen that the key generation and encryption XOR operation can be equally time-consuming. We could reduce the XOR

time by parallelizing it using GPU and parallel programming. Table 5.1 shows the encryption and decryption time excluding the key generation time.

The parallel encryptions are performed on large image sizes to show the scale of improvement as compared to serial processes. Based on the above-mentioned GPU configurations the speed up averages at **60-80**, which is really a good improvement and if these encryption processes are on multiple images it would be even higher.

Encryption Technique	Serial Encryption Time (ms)	Serial Decryption Time (ms)	Parallel Encryption Time (ms)	Parallel Decryption Time (ms)
Henon Map + Arnold Cat Map	1265	1519	22.388128	24.660032
Cross Chaotic Map + Arnold Cat Map	1794	1601	22.388256	24.665600
Cross Chaotic Map + Arnold Cat Map	1432	1792	22.436319	24.686369

Table 5.1 Encryption results for 4096*4096 image size – serial and parallel processing

So, the only bottleneck is the key generation algorithms, as the performance improvement would not be substantial if the key generation process can't be parallelized. But unlike other key generation algorithms for text input where the key used is highly

dependent upon the input text. In the image encryption process, the key generation algorithm depends upon the image size, predefined initial input conditions and the chaotic map equation and not the pixel values.

Using this property of key generation the overall time for encrypting multiple images can be improved. Figure 5.1 shows four 4096*4096 size images using one key generation step and that encrypts all the four images.

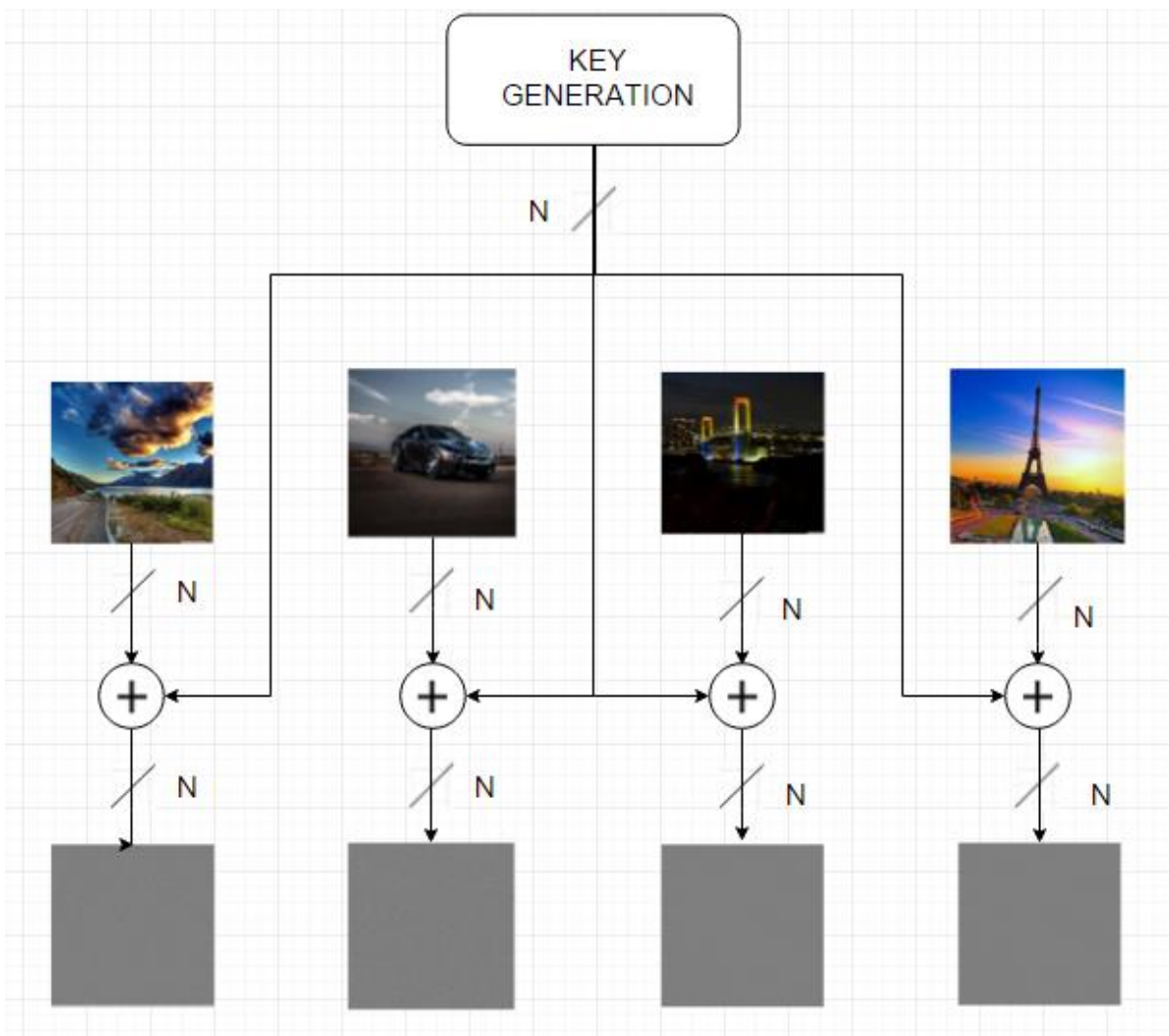


Figure 5.1 Parallel encryption of 4096*4096 images using Cross chaotic and Arnold cat

map




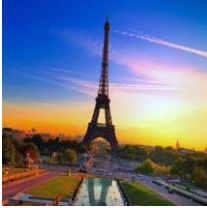
Encryption Technique		Parallel Encryption Time (ms)
Cross Chaotic Map + Arnold Cat Map		22.417984
		22.414848
		22.387552
		22.374945

Table 5.2 Encryption time for 4096*4096 image pixels without key generation time

[26][27]

Table 5.2 shows the image encryption time for the large size images but with single key generation step, the key generation takes almost **2310 ms**. These results are being obtained by using cross-chaotic maps which have the highest encryption and decryption time among the other chaotic maps.

Hence, the speedup based on the encryption time for four images using serial and parallel processing is calculated as follows:

Cross Chaotic + Arnold Cat Map serial encryption of 4096*4096 image = **5536 ms**

(from table 4.3)

For 4 images the encryption time = $5536 * 4 = \mathbf{22144 \text{ ms}}$

Encryption time for parallel processing four images of same sizes =

$(22.417984 + 22.414848 + 22.387552 + 22.374945) = \mathbf{89.595329 \text{ ms}} \approx \mathbf{89.6 \text{ ms}}$

Total encryption time = key generation time + encryption time = $89.6 + 2310 = \mathbf{2399.6 \text{ ms}}$

Speedup = $\mathbf{22144 / 2399.6 = 9.23 \text{ times}}$

6. CONCLUSION AND FUTURE WORK

From the above-discussed encryption schemes, some principles could be used for more robust and efficient image encryption algorithms. The primary focus of this thesis is to understand chaos-based image encryption technique still in its progressing phases, but which has a lot of scope in various fields. In the past decade, there has been several new image encryption techniques developed or enhancements for the existing techniques being done. Similarly, this research work does an analysis of the two-dimensional chaotic maps and how they are implemented as encryption algorithms.

The main focus of this thesis is to do a run-time analysis of these algorithms using different levels of security and image sizes. It was found that even though existing algorithms are secured, but for these algorithms to be used for real-time applications it is necessary to even improve the processing time of these encryption algorithms. There is a direct correlation between the encryption steps and the average encryption and decryption time.

But, because of the property of image pixels being separate entities they can be encrypted independently and hence this research uses Graphic processing units (GPU) to parallel process these image pixels and then profile their behavior based on different parameters. The results were improved substantially, because the time complexity is reduced from quadratic $O(n^2)$ to constant $O(1)$ time, Even though using GPU, we need to transfer the data from host system to the GPU device which is an overhead, but this varies from machine to machine, there are also efficient ways of making it less time consuming and above all when compared with the overall time improvement this overhead time is minuscule.

But, the main bottleneck is the encryption key generation algorithm, as chaos--based encryption algorithms are based on the principle of feedback system where current values are dependent upon the previous values. Hence, they can't be parallelized, to overcome this issue, multiple images are being encrypted using single key generation step. When implemented using four images of 4096 by 4096 pixels and a single key generation step speedup of ~10 times was achieved.

6.1 Future Research

During the research of this thesis, several interesting problems were faced which can be solved as future work. A summary of some additional research are as follows:

- Study other image encryption algorithms like a hash function, Advanced Encryption Standard (AES) based, complex chaotic maps with three and four dimensions and other authentication schemes. Similarly, compare their results and parallel process them to find the most efficient image encryption technique.
- Improve the key generation algorithm so that it can be parallelized which would improve the response time substantially without requiring multiple images to use the same key stream.

REFERENCES

- [1] Sinha, A., & Singh, K. (2003). A technique for image encryption using digital signature. *Optics communications*, 218(4), 229-234.
- [2] Panchal, D., Jani, C., & Panchal H., "An Approach Providing Two Phase Security of Images Using Encryption and Steganography in Image Processing." *International Journal of Engineering Development and Research*. Vol. 3. No. 4 IJEDR, 2015.
- [3] Mao, Y., & Chen, G. (2005). Chaos-based image encryption. *Handbook of Geometric Computing*, 231-265.
- [4] Lawande, Q. V., Ivan, B. R., & Dhodapkar, S. D. (2005). Chaos based cryptography: a new approach to secure communications. *BARC newsletter*, 258(258).
- [5] Wikipedia contributors. "Arnold's cat map." *Wikipedia, The Free Encyclopedia*. *Wikipedia, The Free Encyclopedia*, 9 Aug. 2015. Web. 2 Mar. 2016.
- [6] Peterson, G. (1997). Arnold's cat map. *Math45-Linear algebra* <http://online.redwoods.cc.ca.us/instruct/darnold/maw/catmap.htm>.
- [7] Wikipedia contributors. "Lorenz system." *Wikipedia, The Free Encyclopedia*. *Wikipedia, The Free Encyclopedia*, 8 Jan. 2016. Web. 2 Mar. 2016.
- [8] Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, 50(1), 69-77.
- [9] Wikipedia contributors. "Hénon map." *Wikipedia, The Free Encyclopedia*. *Wikipedia, The Free Encyclopedia*, 12 Dec. 2015. Web. 2 Mar. 2016.
- [10] Wikipedia contributors. "Duffing map." *Wikipedia, The Free Encyclopedia*. *Wikipedia, The Free Encyclopedia*, 10 Oct. 2013. Web. 2 Mar. 2016.

- [11] Maotai, Z., & Sha, J., "Simulation Results." Communications and Information Processing International Conference, ICCIP 2012, Aveiro, Portugal, March 7-11, 2012, Revised Selected Papers. Part II. Berlin: Springer, 2012. 139-41. Print.
- [12] Wang, L., Ye, Q., Xiao, Y., Zou, Y., & Zhang, B. (2008, May). An image encryption scheme based on cross chaotic map. In Image and Signal Processing, 2008. CISP'08. Congress on (Vol. 3, pp. 22-26). IEEE.
- [13] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU computing. Proceedings of the IEEE, 96(5), 879-899.
- [14] NVIDIA GPU computing, 2011, <http://www.nvidia.com/object/what-is-gpu-computing.html>.
- [15] Möckel, M. (2015). High Performance Propagation of Large Object Populations in Earth Orbits. Logos Verlag Berlin GmbH.
- [16] Rege, A. (2008). An Introduction to Modern GPU Architecture. En ligne].
- [17] Romero, M., Urra, R., CUDA Programming Information and Resources http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm
- [18] "CUDA." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 16 Feb. 2016. Web. 1 Mar. 2016.
- [19] Nvidia, C. U. D. A. (2011). NVIDIA CUDA C Programming Guide Version 7.0 (2015).
- [20] Bourke, P., PPM image files, <http://paulbourke.net/dataformats/ppm>.
- [21] Kumar, S., Sinha, B., & Pradhan, C. (2015). Comparative Analysis of Color Image Encryption Using 2D Chaotic Maps. In Information Systems Design and Intelligent Applications (pp. 379-387). Springer India.

- [22] Biham, E., & Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1), 3-72.
- [23] Wu, Y., Noonan, J. P., & Aghaian, S. (2011). NPCR and UACI randomness tests for image encryption. *Cyber journals: multidisciplinary journals in science and technology, Journal of Selected Areas in Telecommunications (JSAT)*, 31-38.
- [24] Delaboudiniere, J. P., Artzner, G. E., Brunaud, J., Gabriel, A. H., Hochedez, J. F., Millier, F., ... & Kreplin, R. (1995). EIT: extreme-ultraviolet imaging telescope for the SOHO mission (pp. 291-312). Springer Netherlands.
- [25] [Untitled image of mountain road scenery]. Retrieved from <http://wallpapersforipad.com/hd/amazing-ipad-wallpapers/>
- [26] [Untitled image of rainbow bridge Tokyo]. Retrieved from <http://www.thousandwonders.net/Rainbow+Bridge#pictures>
- [27] [Untitled image of eiffel tower]. Retrieved from <http://travellerguidance.com/wallpapers-eiffel-tower-in-paris-hd-2560x2048-406068-eiffel/>