Palestine Polytechnic University

College Of IT and Computer Engineering

Computer System Engineering Department

Project title: Using MPI in Image Chaos Encryption

Student Name:-

Islam Ayman Amar

Supervised by:-

Dr. Mohammed Abutaha

Contents

List of Figures

List of Tables

Key words

MPI: message passing interface

Parallel chaos crypto system

Parallel chaos generator

Pthread

DES

AES

RSA

Triple DES

Salsa 20/r

Rabbit

Shared memory

Distributed memory

IoT : Internet of Things

OpenMp

Acknowledgments

I would like to express my deep gratitude to Dr. Mohammed Abutaha, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Eng. Basil Tamimi, for his great advice and assistance in keeping my progress on schedule.

Finally, I want to thank my parents and my cousin Dr. Maher Maghalseh for their support and encouragement throughout my study.

Abstract

In order to secure images through transmission at the internet and local network, there is need to encrypt these images by using an algorithm that has good characteristics such as randomness and non-periodic that all available in chaotic algorithms. However, almost system of these days is a real-time system that in which time plays an important factor to achieve availability of the secured image at the right time. From that point and the need of increasing performance and efficiency of encryption in this research we used distributed parallel programming model especially message passing interface (MPI) for these purposes. The main goal of this study aimed at developing a parallel encryption algorithm that can be used for encryption image and video in efficient manner.

Chapter 1

Introduction

1.1 Introduction

The rapid communication technology and exchanging information through the internet by using transfer control protocol as plaintext leads to the need for using encryption to secure sensitive data during the transition. The word “encryption” comes from the Greek word “kryptos”, which means hidden or secret. The use of encryption in the past has first appeared in 1900 B.C as the art of communication itself. Moreover, the encryption developed to convert a message into unreadable groups of figures to protect the message secrecy while it was carried from one place to another. After many years cryptography has been developing in the ways of encryption to add features to sensitive data like confidentiality, integrity, authenticity known with CIA. Nowadays cryptography includes three aspects which are: symmetric, asymmetric cipher and protocols. The majority of encryption schemes from ancient times until 1976 were symmetric ones that used the same key for encryption and decryption.

Encryption is the process that converts normal text (plaintext) to unreadable text called (ciphertext).In contrast, the decryption is an operation that converts unreadable text (ciphertext) to readable ones which are plain text.

In 1976 asymmetric cryptography was proposed by Diffie, Hellman, and Merkle[11]. Asymmetric cryptography consists of two keys; public key for encryption and private key for decryption. Anyone can use the public key to encrypt the specific data while private keys are only available to the authorized person for decrypting the data.

The majority of today's protocols are hybrid schemes that use both symmetric ciphers for encryption and message authentication. Moreover, asymmetric ciphers used for key exchange and digital signature.

The symmetric ciphers encompass stream cipher and block ciphers. Stream cipher algorithm encrypts bits individually as clarified in Figure 1. It seems usually small, fast and common in embedded devices such as GSM Phones. While block ciphers which encrypt entire block (several bits) which mostly used and common in internet application as shown in Figure 2.

The first symmetric encryption algorithm is the Data Encryption Standard (DES) which uses the 56-bit key. While the advanced encryption standard (AES) is considered more reliable because of using 128- bit, 192-bit, 256-bit key.

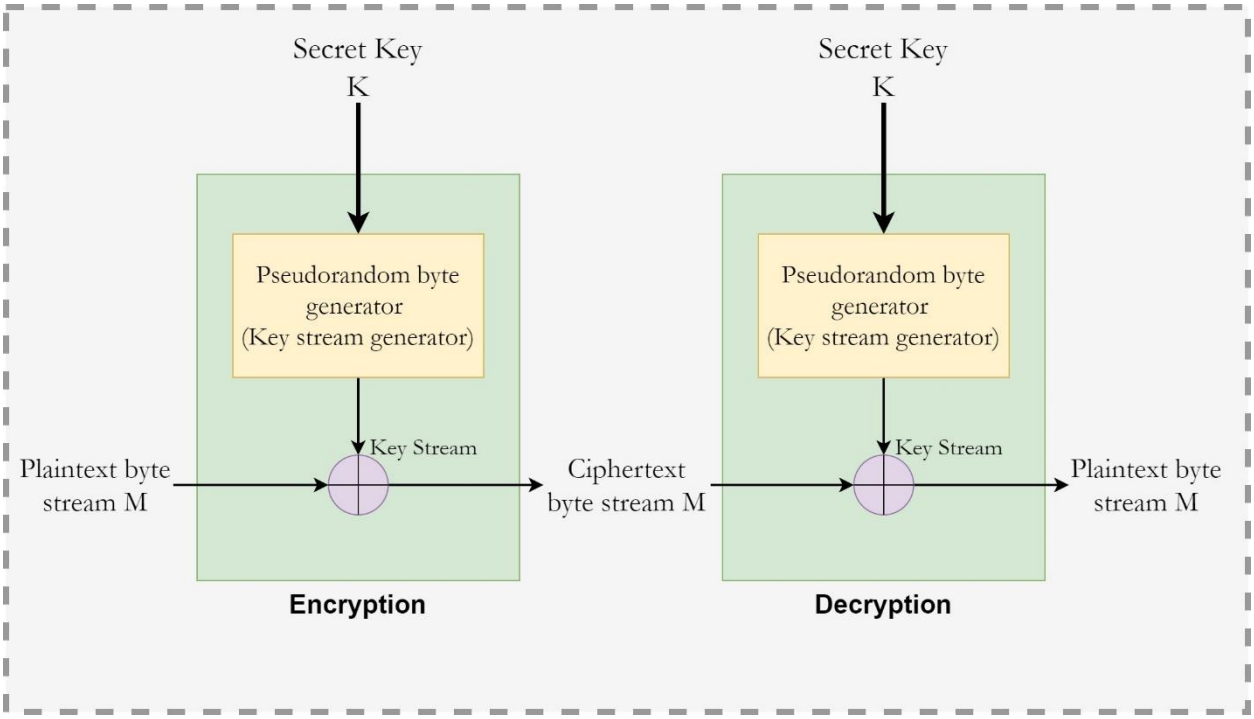


Figure 1: Stream Cipher encryption structure

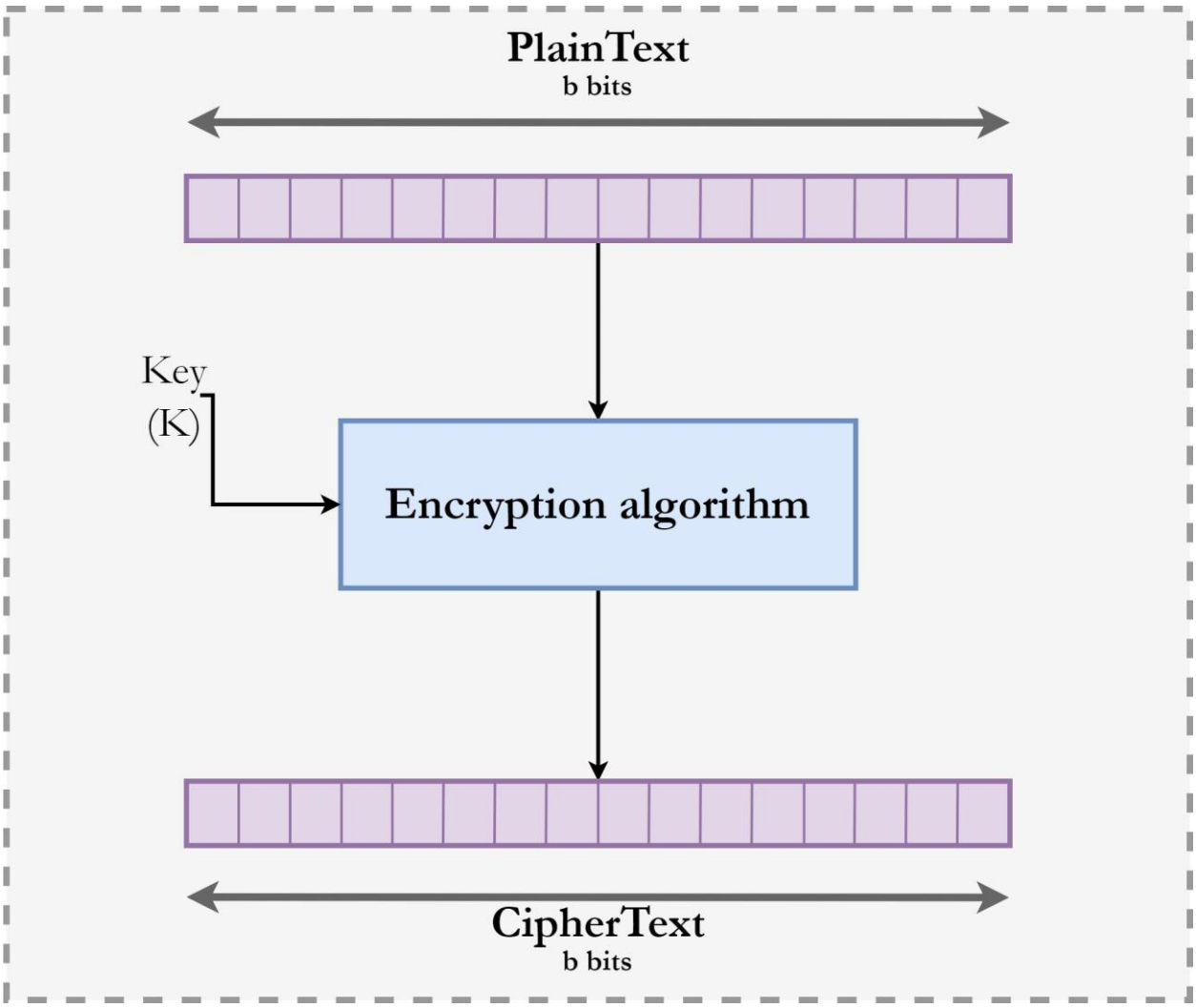


Figure 2: Block Cipher encryption structure

1.2 Using Of Encryption

Encryption was almost used by governments and large companies until the beginning of 1970. When the Diffie- Hellman key exchange and RSA algorithm was published [21] [22]. In the 1990s both public key and private key encryption deployed in a web browser and servers to protect sensitive data.

Nowadays, encryption becomes an important crucial factor in data transmission. Many application in our life are dealing with sensitive data that need to be encrypted while storing or communicating the sensitive data with other devices through network.

In addition, Encryption was used in devices such as modems, smartcards and SIM cards that rely on protocols like SSH and SSL to protect sensitive data through transmission.

1.3 Advantages of encryption

Sensitive data must be encrypted to prevent unauthorized third parties from accessing the data such as man in the middle attack. Modern encryption algorithm plays an important role in security assurance of website and social media. Besides, it provides confidently for these key element:

1. Privacy: tends to be the most important thing in these days, because of that, encryption plays an important part in this filed such as end-to-end encryption that provides privacy for all users
2. Authentication: verification of the original message
3. Integrity: Proof that the content of the message has not been changed during the transition
4. Nonrepudiation: the sender of the message cannot deny sending the message.

1.4 Encryption algorithms

Encryption algorithms are a mathematical procedure used to perform encryption using two inputs which are plaintext and a secret key. To protect sensitive data and keep it confidential there is a need for effective encryption algorithms. Here are some commonly used algorithms:

Symmetric Algorithm

1.4.1 Block cipher algorithms:

1.4.1.1 DES

DES is the most popular block cipher for most of the last 30 years. It encrypts a block of size 64 bits. It was developed by IBM based on the cipher Lucifer under the influence of the national security agency (NSA) [11]. It was standardized 1977 by the National Bureau of Standards (NBS) today called the national institute of standards and technology (NIST). Nowadays considered insecure due to the small key length of 56 bit however 3DES assume to be a very secure cipher, still widely used today.

1.4.1.2 Triple-DES (3DES)

Triple DES is a protocol that consists of three passes of DES using multiple keys. It enhances the keyspace from 56 bit to 112 or 168 bit, depending on whether two or three keys are used [17]. Triple DES is computationally secure because of the underlying security of the DES and vastly increased keyspace. It was used in many application systems like a banking system.

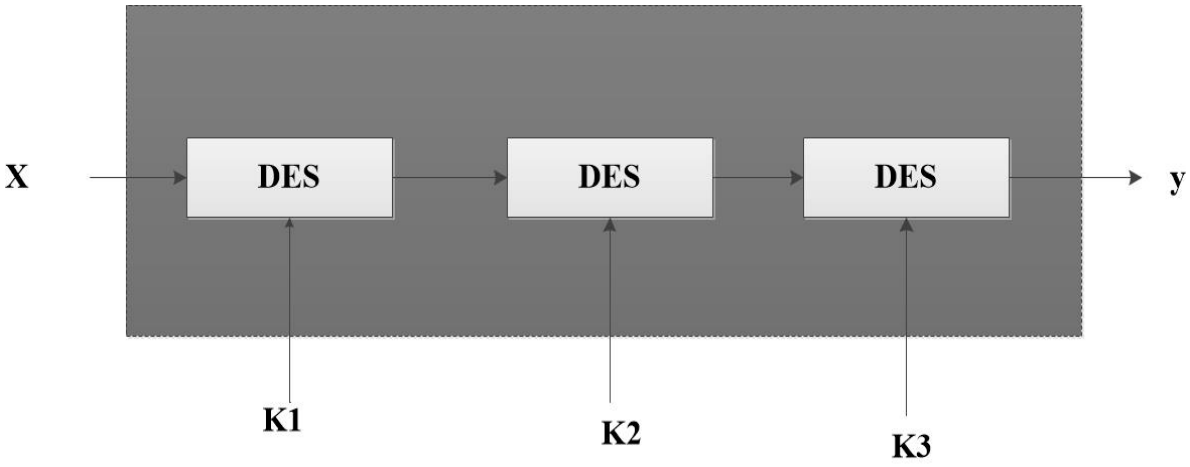


Figure 3: Triple-DES Structure

1.4.1.3 AES

AES is known as Rajindal and it stands for advanced encryption standard that was developed by the National Institute of Standards and Technology (NIST). The AES algorithm uses cryptographic keys of 128, 192 and 256 bit to encrypt and decrypt data in blocks of 128 bits as figured out in Figure 4 [17].

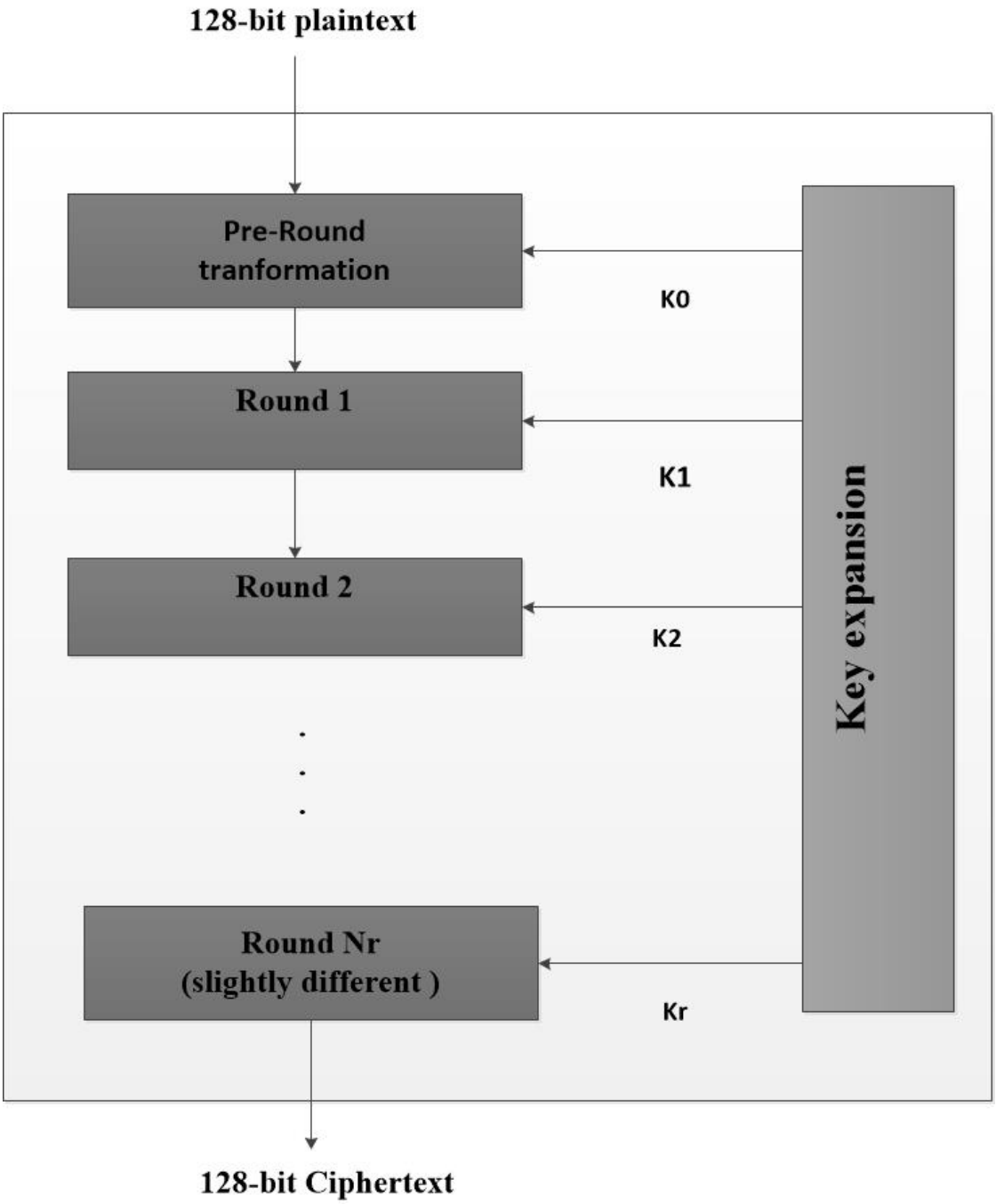


Figure 4: Advanced Encryption Standard

1.4.1.4 International Data Encryption Algorithm (IDEA)

IDEA is a symmetric key block cipher developed by Swiss Federal institute in 1990.IDEA used 128-bit key [16] .supposedly, it is more efficient to implement in software more than DES and triple-DES.

1.4.1.5 CAST

The cast algorithm is a substitution-permutation algorithm similar to DES and triple DES. The casting algorithm was developed by Carlisle Adams and Strafford traverse [12]. It supports variable key lengths from 40 to 256 bits in length. Cast algorithm has been reported to be two times faster than DES and six to nine times faster than triple DES.

1.4.1.5 Blowfish

Blowfish was developed in 1993 by Bruce Schneier as an alternative to existing encryption algorithms [13]. Blowfish is a 64-bit block cipher that has a variable key length ranging from 32 to 448 bits. It is considered to be faster than DES and IDEA.

1.4.2 Stream ciphers algorithms:

1.4.2.1 RC4

It was developed by Ron Rivest. RC4 is a stream cipher that uses variable size key However when used with a key of 128, it can be very effective [14]. RC4 is used in web browsers such as Netscape Navigator and Internet explorer.

1.4.2.2 Salsa20/r

Salsa20/r is one of stream cipher algorithms developed by Daniel J. Bernstein, where r = 8, 12, 20 which are number of iteration of round function. The Salsa algorithm can use cryptographic keys of 128, 192 and 256 bit [15]. A 64-bit nonce (unique message number), 64 bit counter and four 32 bit constants are used to construct 512 internal states. A 512-bit considered as keystream output. Each output block contains an independent combination of the key, nonce, and counter. Since there is no chaining

between blocks, the operation of Salsa20/r as the same as the operation of a block cipher in counter mode. The maximum length of the keystream produced by Salsa20/r is 2^{70} bits.

1.4.2.3 Rabbit

Rabbit is a stream cipher was developed in 2004. It considered being one of the most effective and advanced algorithms. It takes 64-bit nonce and 128 bit key as input. It generates 128-bit output after each iteration. Its cipher consists of 513 internal state which compromises eight 32 bit internal state variables, eight 32 bit counters and one counter carry bit [20]. It has security against several possible attacks: algebraic, correlation and statistical attacks. However, differential fault attacks were discovered in 2009.

1.4.3 Asymmetric algorithms:

1.4.3.1 RSA

It is an asymmetric or public key cryptographic algorithm Used to encrypt and decrypt the messages. It was proposed by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977 [21]. RSA relies on finding factors of large composite numbers.

1.5 Objectives

In this project goals was given as the following:

1. Performance enhancement

- By using a message passing interface (MPI) as a parallel distributing system.
- Decrease the number of a cycle that needed to encrypt and decrypt the image
- Reduce the average encryption time (ET)
- Increase the average encryption throughput

2. Assuring a secure and strong encryption system

- Resist brute force attack by generating key has a 299-bit size.
- Passing the NIST test which is a statistical package test that consists of 188 tests and subtests
- The generated sequence of chaos generators supposed to be uncorrelated.

3. Development of a portable and user-friendly system

- Adapt an android application to encrypt images.
- The system has a good interface for making interaction and communication with people more easier.

4. Achieving the confidentiality and integrity of our encrypted images

- Confidentiality: the images just only deciphering if the receiver has the key.
- Integrity: the content of the encrypted images be the same during the transmission

1.6 Conclusion

In this chapter, we make a review for some historical information about encryption and Its fundamental definition. The importance of encryption such as, using encryption and its impact on our daily life has been summarized. Besides the advantages of encryption also have been detailed. Furthermore, some of the famous encryption algorithms were also described in this chapter. In the next chapter, we will give a brief and short description. In the first section, we will talk about parallel programming. In the second one we will handle the techniques used in parallel programming. Finally, we will explore the relation of parallel techniques with our proposed project.

Chapter 2

Parallel programming models and message passing interface (MPI)

2.1 Introduction

The need for high speed and high performance of hardware system in many aspects like Artificial intelligence, IoT, Real-time application, video conferencing, video on demand that led programmers to search and rely on another way of programming that pave the way to solve problems which concurrent programming consistently run into. Parallel programming considers being an alternative choice that communicates directly with hardware to provide high speed and high efficiency of the desired application. Parallel programming seems to be the best way to exploit all potentials resources and processors for offering high-speed performance. This type of programming can be used in many science aspects to solve mathematical problem.

In this chapter, we review some of the parallel programming models, multiprocessor architectures and the need for parallel programming in our proposed project.

2.1.1 Parallel programming models

In the 1980s, in order to get high speed, CPU manufacturers such as Intel, AMD increased the speed of the computer by developing the hardware architecture and increased number of transistors for each processor. This evolution transition continued until the middle of the 1990s and the beginning of the 2000s. They reach to physical limitation manner caused by disabling to cool heat of CPU when the clock rate becomes more higher. Manufacturers increase speed by building a new CPU architecture that consists of many cores and can receive multiple instructions and multiple data (MIMD). The idea of existing multi-core architecture taking into concerns the base of exploiting all these cores by using a parallel programming model.

The parallel programming model depends on hardware architecture which is shared a memory or distributed memory architecture. Both architectures share the same idea that breaks down works to such a unit and assign this unit for each core to process it separately. Shared memory

architecture shared all control unit (CU) and processing unit (PU) with one memory as shown in Figure 5. However in distributed memory, each processor unit and control unit has its local memory which completely connected by the Interconnection network as shown in Figure 6.

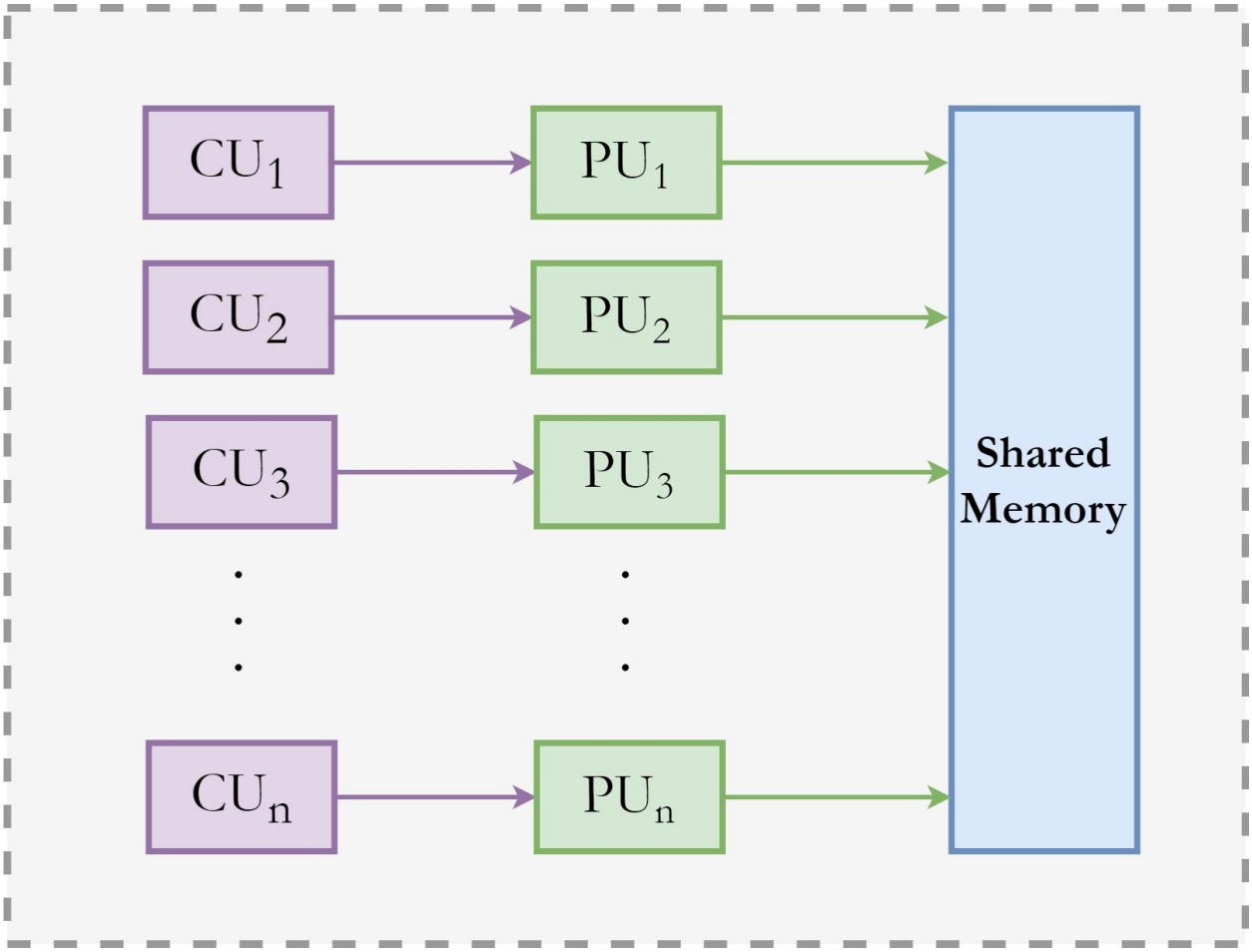


Figure 5: Shared Memory Structure

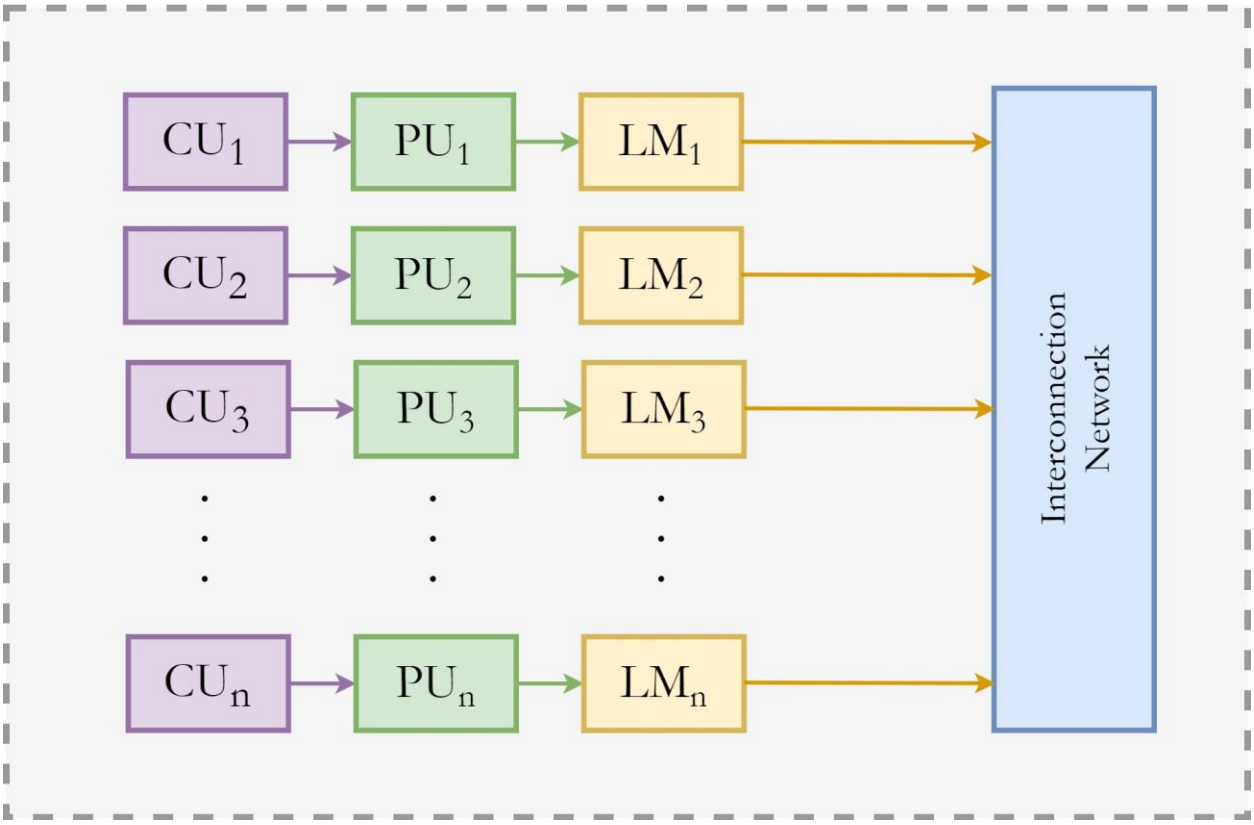


Figure 6: Distributed Memory Architecture

2.1.2 Classification of the parallel programming model

The parallel programming model classified into two aspects as shown on the following:

1. process interaction
2. problem decomposition

The process of interaction compromises shared memory, implicit interaction, and message passing.

2.2 *Shared memory*

Many programming languages and libraries support shared memory parallelism which all processor shared the same memory during processes operation such as openMP and Pthreads.

2.2.1 OpenMP

It is an application program interface (API) supported by Fortran C/C++. It takes user application, tears it down into a group of threads and runs them on a shared memory foundation. OpenMP is a cross-platform that can run in any operating system such as UNIX, Linux, and windows [18]. OpenMP is created from a combination of compiler directives, Runtime libraries, and Environmental libraries. To define OpenMP we include the Pthread header in C program: #include<omp.h>.

2.2.2 *Pthread*

Pthread is a library can be linked with C/C++ language .it is not a programming language like Python and Java.it was also known as POSIX thread. It used shared memory parallelism. To define Pthread we include the Pthread header in C program: #include<pthread> [19]. Posix thread has a standardize extension (IEEE Std 1003.1c-1995). POSIX thread does many procedures like Thread management - creating, joining threads, etc. Mutexes, condition variables and Synchronization between threads using read/write locks and barriers.

2.3 Distributed Memory

2.3.1 MPI

MPI stands for the message passing interface [10]. MPI link with other languages such as C/ C++, FORTRAN. To link MPI in C++ we define the MPI header: #include<mpi.h>. MPI is considered as one of the most important programming languages that use distributed memory Architecture. It is like another language has its own data type and its own function.In order to communicate between two processors, the first processor performs **send** operation while the other performs **receive** operation. We use mpicc command for compiling MPI in C or C++ program. To execute the program we use mpirun followed by the number of the processor in your specific program. MPI has many implementations such as MPICH, MPICH2, and OpenMPI. Now MPI is binding to other any language by wrapping MPI implementation in other languages like Java, Matlab, and Python.

2.3.1.1 Elementary MPI datatypes

MPI likes other languages have their data type. Moreover, you can create a more complex datatype for characterizing messages. Here some of the data type in MPI [10] as shown in table 1.

Table 1: MPI datatypes

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	Char

2.3.1.2 *MPI send and receive*

Send and receiving a message are two basic concepts of MPI[10]. Process P_1 wants to send a message to other processor P_2. First processor P_1 puts a message into a buffer. These buffers are known as the envelope that keeps preserving the data. After data is stored in the buffer, the message is routing from buffer to a specific location. This specific location identified by a processor rank. The processor P_2 still has to acknowledge that it has the ability to receive data from processor P_1. When it does, the data is transmitted into processor P_1. Sends and receives operation use tag to specify the message from the other.

MPI_Send function Prototype

```
MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
```

MPI_Recv function Prototype

```
MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

The first argument indicates the data in the buffer. The third and second argument describes the count and data type in the buffer. the fourth in `MPI_Send` indicates for a destination which message transmitted into however in MPI_Recv the fourth indicates the source which message comes from. the fifth and sixth describe the tag and communicator. Besides, MPI_Recv has an additional argument which is status

Table 2: MPI Send and Receive

Simple Code

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>

void main(int *argc, char *argv[])
{
        int id = 0;
        int senddata = 0;
        int recievadata = 0;
        MPI_Init(&argc ,&argv);

        MPI_Comm_rank(MPI_COMM_WORLD,&id);
        if (id == 0)
        {
                senddata = 100;
                MPI_Send(&senddata,1,MPI_INT,1,2,MPI_COMM_WORLD);
        }
        MPI_Status status;
        if (id == 1)
        {
                MPI_Recv(&recievadata, 1, MPI_INT, 0, 2, MPI_COMM_WORLD, &status);

                 printf("receive data is  %d\n", recievadata);
        }

        MPI_Finalize();

}
```

The output of MPI Send and Receive code

```
receive data is  100
```

2.3.1.3 *MPI Collective communication and synchronization*

Collective communication is the way of communication which involves all process in a communicator . in order to reach in a synchronization point which all processor must pass this point before they can execute again. MPI has a private function dedicates to synchronization: MPI_Barrier(MPI_Comm communicator)[10].

MPI_Barrier function creates a virtual barrier that prevents all processor to pass the barrier until all processors reach the barrier as shown in figure 7

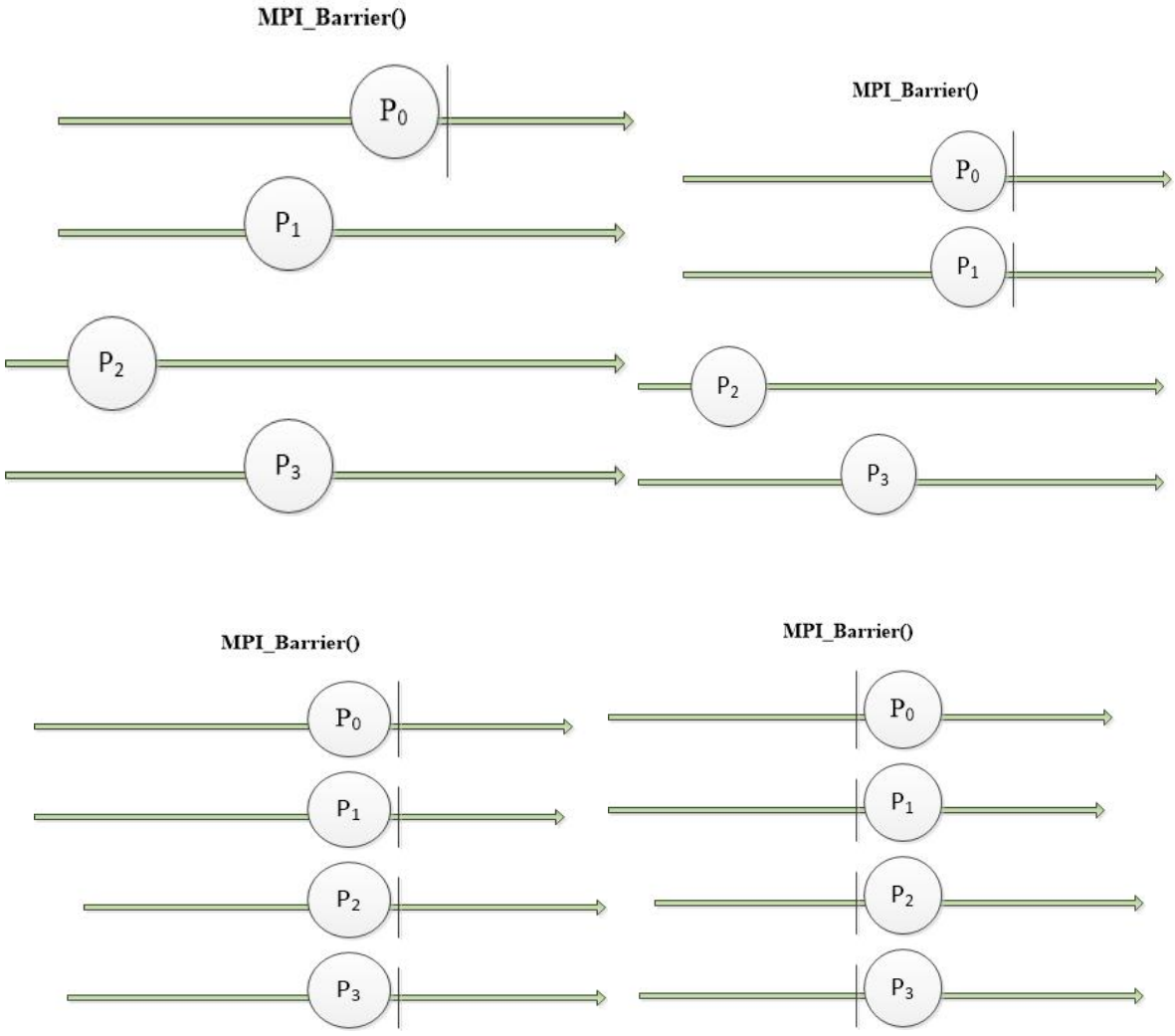


Figure 7: MPI barrier illustration

The first picture clarifies when calling MPI_Barrier() the process P_0 handle in the barrier while others do not reach it . when P1 reaches the barrier it becomes like P_0 cannot do anything because of their operation handle in the barrier. Picture 3 indicates when all processors handle before the barrier before they doing its specific operation. In this situation that means all processors reach the synchronization state which does their task at the same time.

2.3.1.4 *Broadcasting in MPI*

Broadcasting considers being the most famous collective communication techniques [10]. Which sends data from one processor to all processors in the communicator as shown in figure 8.

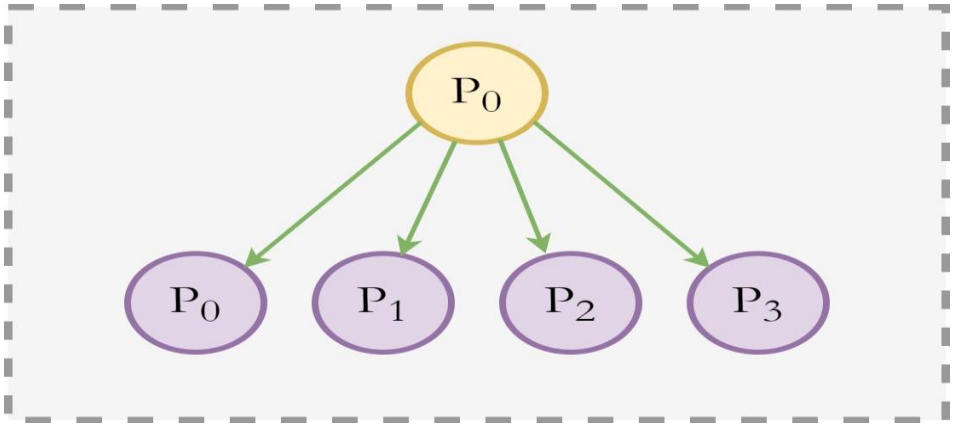


Figure 8: Broadcasting

Broadcasting can be achieved by using MPI_Bcast ()function as below :

MPI_Bcast Prototype

```
MPI_Bcast(
   void* data,
   int count,
   MPI_Datatype datatype,
   int root,
   MPI_Comm communicator)
```

When the root processor calls the MPI_Bcast function, the root processor sends **data** argument to all receiving processor. As shown from the prototype of MPI_Bcast function besides sending data as an argument we also have to dedicate the **count** and determine the suitable data type. it takes the **root** argument which is the root processor rank.

Table 3: MPI_Bcast

Simple code

```
#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>

int main( int argc , char *argv[])
{
   int rank;
   int data = 0 ;
   MPI_Init(NULL, NULL);

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank== 0 )
   {

      data=10;
   }
   printf("\nbfore bcast data in process %d:%d",rank,data);
   MPI_Bcast(&data,1,MPI_INT,0,MPI_COMM_WORLD);
   printf("\nafter bcast data in process %d:%d",rank,data);


   MPI_Finalize() ;
   return 0;
}
```

The output of the MPI_Bcast code is :

```
before cast data in process 0:10
after cast data in process 0:10
before cast data in process 1:0
after cast data in process 1:10
before cast data in process 2:0
after cast data in process 2:10
before cast data in process 3:0
after cast data in process 3:10
```

2.3.1.5 *MPI Scatter*

MPI scatter is one of the most important techniques in MPI collective communication it is similar to MPI_Bcast in the concepts but with small differences [10]. MPI_Bcast take data from root processor and distribute the same data for all processors however MPI scatter take the data represented in an array from the root and distribute all the array elements on their respective order for all available processor as shown in figure 9:

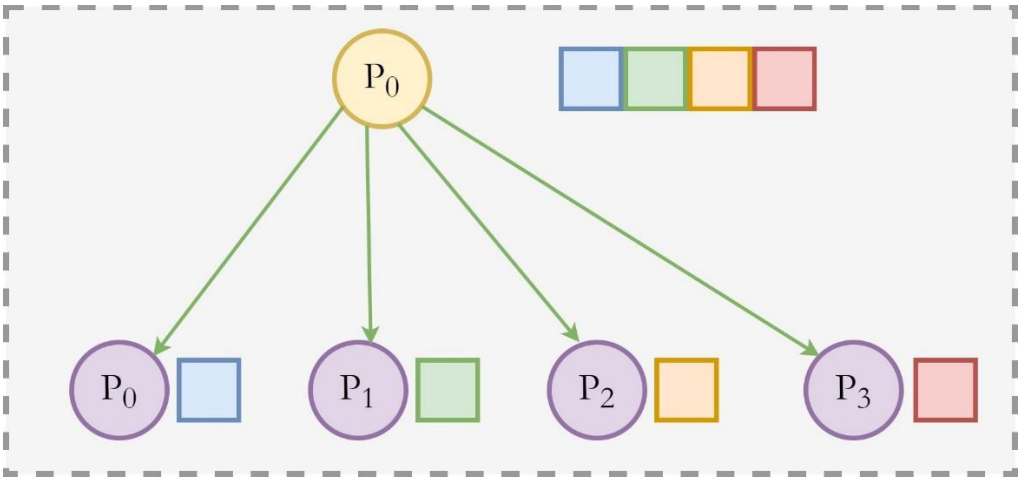


Figure 9: MPI Scatter diagram

MPI_ Scatter Prototype

```
MPI_Scatter(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

The first parameter **send_data** indicates the array stored in the root processor. The second and third parameter which are **send_count** and **MPI_Datatype** which are represent a number of data that hold by each processor and data type of each of this data . for example if the **send_count** is two so P_0 and P_1 will hold all the elements and each one of them will have two elements from the whole data. **recv_data** buffer of data that hold **recv_count** elements with the datatype of **recv_datatype**. The **root** indicates the root processor that distributes all data to other processors.

Table 4 : MPI_Scatter program

Simple code

```
#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>
 void show()
 {
    printf("islam\n") ;
     printf("mmmm\n") ;
      printf("ssss\n") ;
       printf("llll\n") ;


 }
void  main(int argc , char **argv)
{
   MPI_Init(&argc,&argv);
   int id ;
   int recivedata=0;
   MPI_Comm_rank(MPI_COMM_WORLD,&id);
   int sactterd[4]={0};
   if(id==0)
   {

   sactterd[0]=10 ;
   sactterd[1]=20 ;
   sactterd[2]=30 ;
   sactterd[3]=40 ;
   }
   MPI_Scatter(&sactterd,1,MPI_INT,&recivedata,1,MPI_INT,0,MPI_COMM_WORLD);
  // barrier
   printf("processor %d with recieve data %d\n",id,recivedata);
   printf("Hello world!\n");
}
```

Output of the MPI_Scatter code
processor 0 with recieve data 10 Hello world! processor 1 with recieve data 20 Hello world! processor 2 with recieve data 30 processor 3 with recieve data 40 Hello world! Hello world!

2.3.1.6 *MPI_Gather*

In contrast of spread element from one processor to all processor in MPI_Scatter, MPI_Gather takes elements from all processor and gather it to the root processor [10]. It is commonly used in sorting and searching.

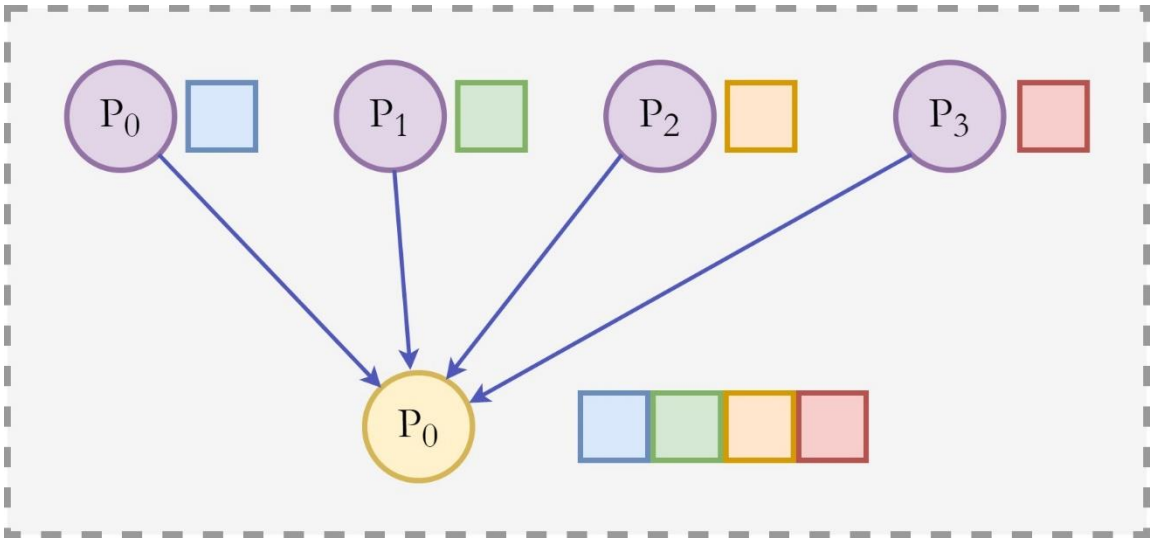


Figure 10: MPI_Gather diagram

MPI_ Gather Prototype
MPI_Gather(void* send_data, int send_count, MPI_Datatype send_datatype, void* recv_data, int recv_count, MPI_Datatype recv_datatype, int root, MPI_Comm communicator)

Table 5: MPI_Gather program

Simple code

#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>
void main(int argc , char **argv)
{
 	
MPI_Init(&argc,&argv);
int id= 0 ;
MPI_Comm_rank(MPI_COMM_WORLD, &id);
int data ;
data = (id+1)*10 ;
int gather[4];
MPI_Gather(&data,1,MPI_INT,&gather, 1,MPI_INT,0,MPI_COMM_WORLD);
if (id==0)
{
 	
for(int i =0 ;i< 4 ; i++)
{
 	
printf("%d\n",gather[i]) ;
}
}
printf("Hello world!\n");
}

Output of MPI_gather code
Hello world!Hello world! 10 20 30 40 Hello world! Hello world!

2.3.1.7 MPI_Allgather

Take an element of the distributed processor and gather it to all processors [10]. Most MPI_Allgather program followed by MPI_Bcast.

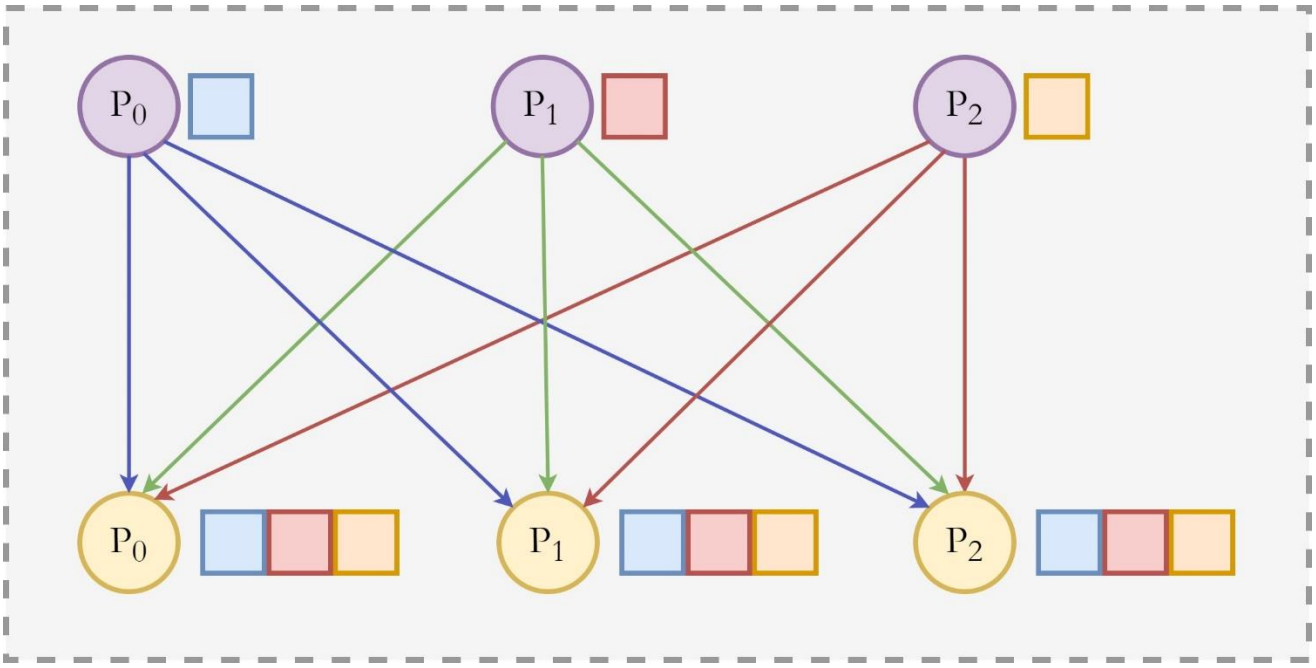


Figure 11: MPI_ Allgather diagram

MPI_ Allgather Prototype

```
MPI_Allgather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    MPI_Comm communicator)
```

Table 6: MPI_Allgather

Simple code

```
#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>
void main(int argc, char *argv[])
{       int id;
        int data = 0;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &id);
        int scatterdata[4] = { 0 };
        if (id == 0)
        {
                scatterdata[0] = 10;
                scatterdata[1] = 20;
                scatterdata[2] = 30;
                scatterdata[3] = 30;
        }
        MPI_Scatter(scatterdata, 1, MPI_INT, &data, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        int allgather[4] = { 0 };
        MPI_Allgather(&data, 1, MPI_INT, allgather, 1, MPI_INT, MPI_COMM_WORLD);
        if (id == 0)
        {
                for (int i = 0; i < 4; i++)
                {
                        printf("process %d %d\n", id , allgather[i]);
                }
                }
        if (id == 1)
        {
                for (int i = 0; i < 4; i++)
                {
                          printf("process %d %d\n", id , allgather[i]);
```

```
                }
}
        if (id == 2)
        {
                for (int i = 0; i < 4; i++)
                {

                        printf("process %d %d\n", id , allgather[i]);
                }
        }
        if (id == 3)
        {
                for (int i = 0; i < 4; i++)
                {

                         printf("process %d %d\n", id , allgather[i]);
                }
        }

        MPI_Finalize();
}
```

Output of MPI_AllGather

```
process 0 10
process 1 10
process 1 20
process 1 30
process 1 30
process 2 10
process 2 20
process 2 30
process 2 30
process 3 10
process 3 20
process 3 30
process 3 30
process 0 20
process 0 30
process 0 30
```

2.3.1.8 *MPI_Reduce*

Data reduction reduces a set of elements into a smaller set of elements by using some function. For example if we have a list of number [20,30,40,50,60] it reduces this list of numbers by using sum function: sum([20,30,40,50,60]) = 200 or using multiply function to reduce a list : multiply([20,30,40,50,60])= 720 . MPI consists of MPI_Reduce function that has the same idea of data reduction.

MPI_Reduce Prototype

```
MPI_Reduce(
    void* send_data,
    void* recv_data,
    int count,
    MPI_Datatype datatype,
    MPI_Op op,
    int root,
    MPI_Comm communicator)
```

The first parameter **send_data** indicates the array that each process wants to reduce. the second parameter **recv_data** array contains reduced data and has sized the same as the **count**. the **op** parameter indicates the operation that you would apply to your data. Here are some of the following operations in a data type that we can use it inside MPI_Reduce function.

Table 7: MPI operation

Operation	Function
MPI_MAX	Returns the maximum element.
MPI_MIN	Returns the minimum element.
MPI_SUM	Sums the elements.
MPI_PROD	Multiplies all elements.
MPI_LAND	Performs a logical *and* across the elements.
MPI_LOR	Performs a logical *or* across the elements.
MPI_BAND	Performs a bitwise *and* across the bits of the elements.
MPI_BOR	Performs a bitwise *or* across the bits of the elements.
MPI_MAXLOC	Returns the maximum value and the rank of the process that owns it
MPI_MINLOC	Returns the minimum value and the rank of the process that owns it.

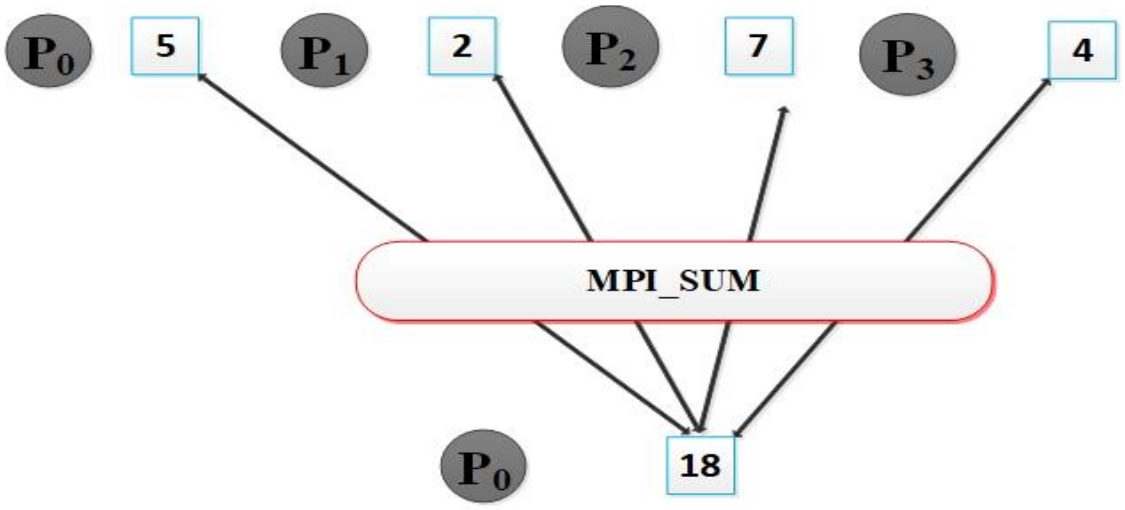


Figure 12: MPI_Reduce diagram

Also, we can use it when processors contain multiple elements to be stored in the root processor like in the following picture:

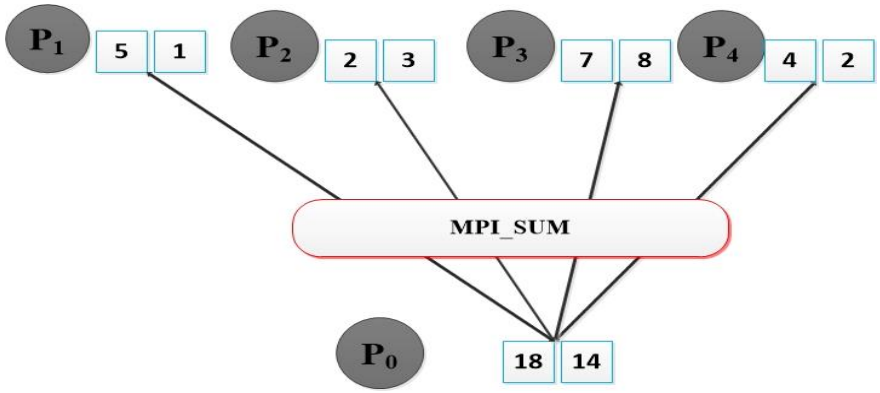


Figure 13: MPI_Reduce for multiple elements diagram

Output of the MPI_Reduce code is :

```
#include <stdio.h>
#include <stdlib.h>
#include<mpi.h>
void main(int argc, char *argv[])
{
        int id;
        int data = 0;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &id);
        int reduceddata = 0;
        data = (id + 1) * 10;
        MPI_Reduce(&data, &reduceddata, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
        if (id == 0)
        {
                printf("reduced data  %d \n",reduceddata);
        }


        MPI_Finalize();
}
```

Output of MPI_Reduce _

```
reduced data  100
```

2.4 STATE-OF_THE_ART

In order to use the best encryption system that applies the concepts of security combining them of using parallel models and libraries to increase the performance of image encryption. We have studied some researches about new encryption techniques that provide security either in sequential style or parallel style. After that, we shall discuss and review some of these studies.

Chaos-based encryption system literature review:

With rapid change and development in technology, it becomes easier to hack and penetrate all aspects of life such as medicine, military, and economic institution. Thus, there was a need to use encryption algorithm for securing sensitive data and especially image through the transition from malicious attacks. This reason encourage us to use traditional image encryption such as (DES) and (3DES). But these style and methodology of encryption don't meet the real encryption needs. Chaos has come as instead choice that repairs the problem in a traditional image encryption algorithm. Chaos-based encryption considers being pseudo randomness, ergodicity and no periodicity [7].

In spite of the power of chaos-based encryption system sometimes appear not secure enough as Diap proposed in his paper [8] that can cryptanalysis this algorithm and broke it in one chosen-plaintext attack. However, zhang suggests in his paper [9] the use of a chaotic map such as skew tent map as which makes using a ciphertext feedback mechanism to resist the chosen-plaintext attack. But zhu [7] cracked algorithm by applying chosen-plaintext combined with a chosen ciphertext attack. Abutaha et.al [6] design an efficient implementation of chaos-based stream cipher by applying two maps skew-tent map and PWLC map on the key and get the result of XORing them together

To produce the first key in chaos generator as output it will be entered as input to the second key and this operation is keep on until finished all key-round with its delay producing total keyspace 299 bit. This long key-space and using key-delay give his algorithm immunity against brute force, differential, and chosen-plaintext attack.

On the other side zue and wang [7] provide an efficient way that adds power to image encryption. This based on chaos encryption by applying SHA-256 with chaos. They take the hash value of the plain image that converted to a numeric value between [0,255] that are adding around plaintext image as padding concepts rather than external key encryption.

For achieving the purpose of combining the security side and good performance that led us to study some sample and make a comparison view of parallel programming encryption and sequential programming encryption. Here are some of the literature that related to our work:

Shung zue et.al [7] demonstrates an efficient way of encrypting images by using a chaos theorem that provides randomness and ergodicity with secure hash algorithm SHA-256.

Abutaha et.al [6] shows a new sequential model of chaotic generator by using skew-tent and piecewise linear chaotic map (PWLC) to encrypt images by generating a key that exceeds 256 bit.

Halin pan et.al [5] provides an efficient way of image encryption based on a double logistic chaotic map by combining two chaotic sequence generators as a key that used to encrypt a clear image to scramble image.

Boris and Anthony [1], they improve architecture and addressing thread safety of modern reliable messaging software and identifying and taking advantage of inherent concurrency in message-passing software itself.

Jesper larosson and William grop [4] surveyed and analyzed possible, sensible, and desirable to formulate system independent by using good MPI implementation. They discuss basic MPI terminology in their paper such as general communication, collective communication likes regular, reduction, irregular collective and communicators and topologies of MPI.

Rajeev Thakur and William gropp[2] discuss and surveyed open issue in MPI implementation such as performance, scalability, fault tolerance, support for debugging and verification, virtual to physical topology, derived datatypes, collective communication, parallel I/O, one side communication and efficient support for MPI- THREAD- MULTIPLE.

Rajeev and William gropp[3] demonstrate and analyzed test suite for evaluating the performance of MPI implementation that supports MPI THREAD MULTIPLE. They show concurrent bandwidth latency tests on MPI implementation.

2.5 Conclusion

In this chapter, we make a review of parallel programming definition and parallel programming model. The parallel programming model and its type whether shared and distributed memory have been summarized. MPI tools and functions such as point to point communication, collective communication, scattering and reducing have been detailed. Furthermore, studying some samples and make a comparison view of parallel programming encryption and sequential programming encryption were also described in this chapter.

Chapter 3

Proposed parallel Chaos crypto system

3.1 Introduction

Cryptography was used since past years until now to secure sensitive data such as military information. In recent times the amount of cryptography applications has been widely expanded as the development of technology. Cryptography considers as an important tool that achieves security concepts such as confidentiality, authenticity and Integrity. Chaos supports the security concept of keeping data secure and providing protection of it from malicious software. Chaos has many advantages and good properties such as ergodicity, pseudo randomness, Parameter variation and high dependency on its initial conditions. Chaos can be generated by a non-linear dynamic system. Steam cipher and block cipher are two famous ways of encrypted data. They are common in the idea of converting normal text (plaintext) into ambiguous and unclear text although stream cipher is more efficient than block cipher according to high encryption and decryption data rate and limited needs of physical resources such as memory, CPU. In order to combine security and performance for image encryption, there is a need to Use a distributed parallel architecture method that provides good performance and the least encryption time for the chaos-based system.

Firstly In this chapter, we will give a short description of the chaos-based generator. Then, we will propose a parallel implementation of chaos-based system. Finally, we will give methods and tools of MPI.

3.2 Internal state of chaos crypto system

The chaos-based generator according to Abutaha et.al [6] consists of several blocks. As shown in figure 2 it takes 3 input initial vector IV, secret key K and other parameters. In parallel implementation the chaos-based generator need to compute Key K and IV for three times. For that purpose IV-Setup used as IV generator that generate IV as input for three times and Key-Setup as Key generator that generate key K three times in case of parallel implementation. The internal state of chaotic generator based on two chaotic maps Skew Tent map (ST) and discrete piecewise linear chaotic map PWLC as presented in figure 14.

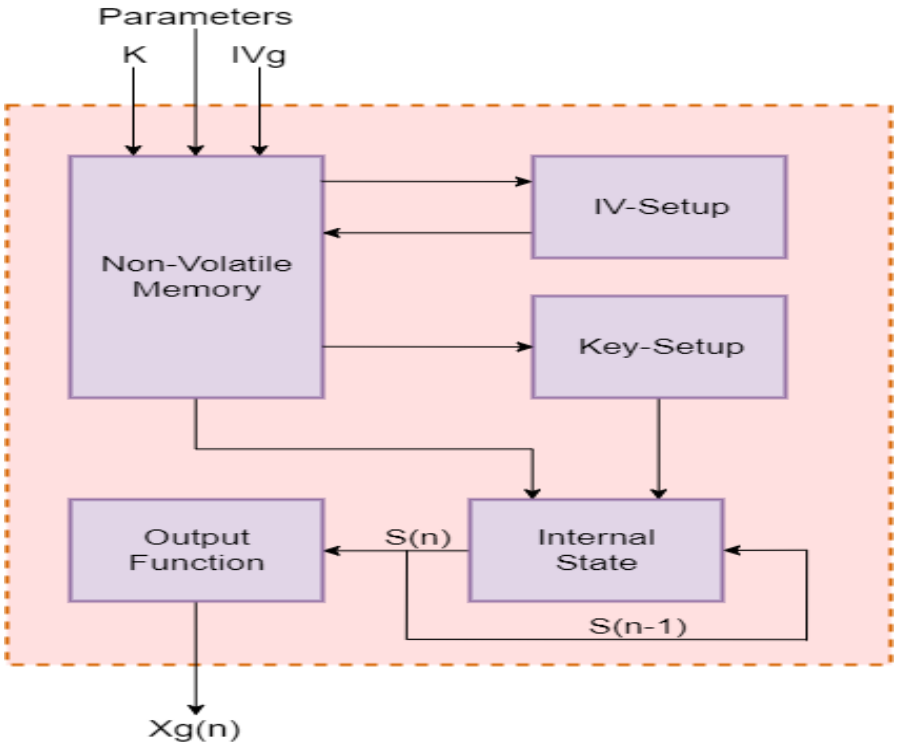


Figure 14 : Internal Architecture of Chaos Generator

Internal sate takes Nonce IVg and Key K as input to a chaotic generator internal state. K1_s, K2_s, K3_s, K1_p, K2_p, K3_p are coefficients of recursive cells. U_s is the least significant bit in IV however U_p is the most significant bit in IV. In table 3 we clarify the notations and meanings of parameters in internal state block.

Table 8: chaos generator parameter and notation

Parameter / notation	Description
K1_s	Parameter for the recursive cell with delay =1 using skew tent map
K2_s	Parameter for the recursive cell with delay =2 using skew tent map
K3_s	Parameter for the recursive cell with delay =3 using skew tent map
K1_p	Parameter for the recursive cell with delay =1 using PWLC map
K2_p	Parameter for the recursive cell with delay =2 using PWLC map
K3_p	Parameter for the recursive cell with delay =3 using PWLC map
U_s	LSB (IV)
U_p	MSB(IV)
X_s	The initial value of the skew tent map
X_p	The initial value of PWLC map
X1_p	The initial value for recursive cell in PWLC map with delay=1
X2_p	The initial value for recursive cell in PWLC map with delay=2
X3_p	The initial value for recursive cell in PWLC map with delay=3
X1_s	The initial value for recursive cell in skew tent map with delay=1
X2_s	The initial value for recursive cell in skew tent map with delay=2
X3_s	The initial value for recursive cell in skew tent map with delay=3

3.3 Proposed parallel chaos crypto system

In order to increase the efficiency and computation performance of a chaotic generator, there is a need to use of parallel programming model. This study focuses on using distributed memory architecture model such as MPI as the proposed parallel programming model because of its ability to pass a message between local processes or processes distributed across networked hosts. MPI is a way to program on distributed memory devices. This means that the parallelism occurs where every parallel process is working in its own memory space in isolation from the others. Every part of code that has written is executed independently by every process. The MPI system tell each process exactly which part of the global problem they should be working on based entirely on their process ID. This study focus in applying MPI on chaos generator in order to increase the total performance of the generated keys by applying send and receive message function for every output bit generated from both map chaotic maps (Skew tent and PWLC map .

.

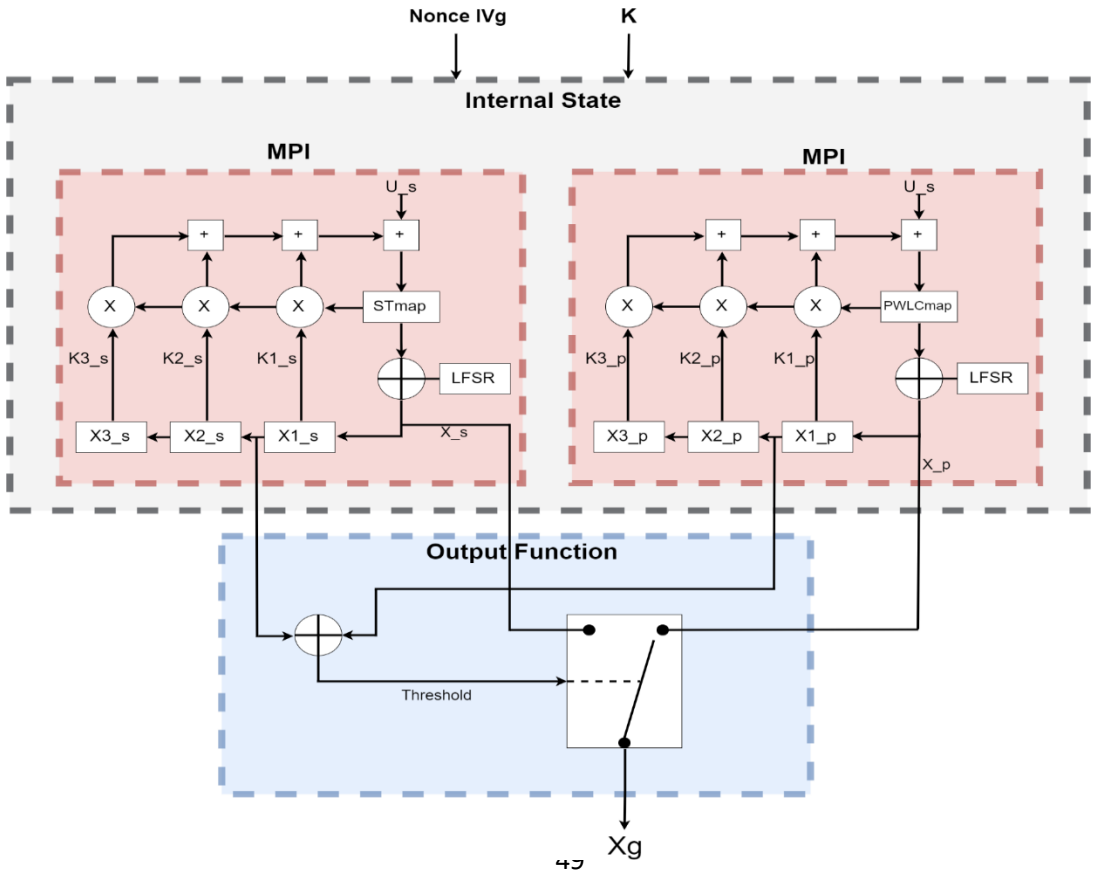


Figure 15 : Proposed parallel Chaos Generator

3.4 Methodology

We apply **Bsend** and **Breceive** function for two cells that are illustrated in figure 15

Each one of them is responsible to generate bit by entering them to skew tent map and PWLC map after doing some operations such as linear shifting and XORing with input parameter. The idea from using point to point communication is distributing data on two or more processors according to the specified computer and workstation. Instead of focusing all processing and working in one the proposed methodology using (MPI) on many cores distributing the load of bits that will increase the average bit rate and decrease Number of cycles per that calculated according to the equation 1 and 2.

$$\mathbf{BR} = \mathbf{Data\_size(mbit)/GT(ms)} \quad (1)$$

$$NCPB = CPU\ Speed\ /BR \quad (2)$$

Figure 16: shows point to point communication between two processors. Message data are copied to a system-controlled block of memory. Process 0 continues executing other tasks without waiting. When process 1 is ready, it fetches the message from the remote system buffer and stores it in the appropriate memory location. This operation must be preceded with a call to MPI_Buffer_attach function.

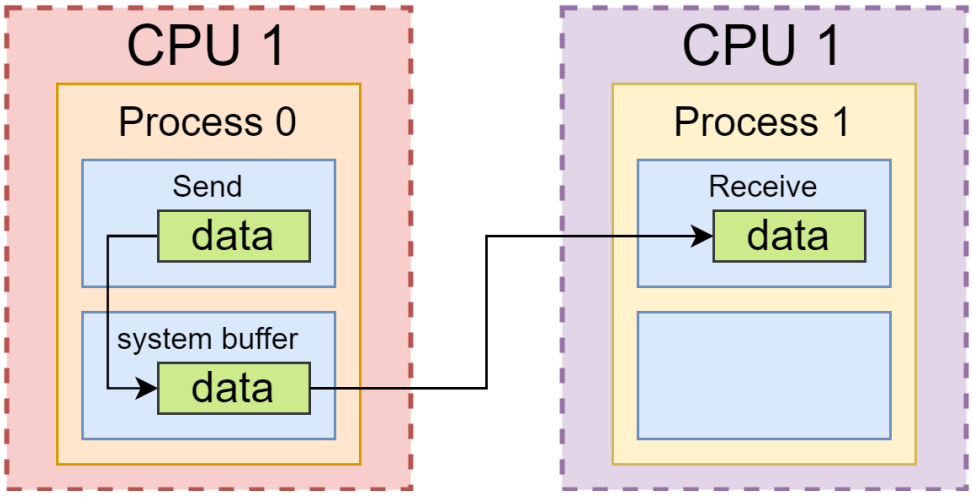


Figure 16: Point to point communication using Bsend and Breceive

3.5 Algorithm of proposed Chaos crypto system

In our case the proposed algorithm is described as the following:

Algorithm: distribute bits to processors

Input image
Convert image into bytes
Split image into 4 bytes
Initialize id ;
Initialize senddata;
Initialize recievadata;
Initialize MPI environment ;
Initialize core_num;
If id = p0 then
Input senddata
Send data from processor to p1;
Else
Input recievadata
Receive data from processor p0
End if
Finalize MPI Environment
Generate a sequence of 32 bit
Convert each 32-bit sequence into 4- byte
Distribute generated sequence into thread
XORing each 4-byte in key with the corresponding 4-byte In the image
Scattered each 32-bit sequence into processors
Output Scrambled image

The **input image is entered** to convert it to unclear or **scramble image** First we read the image and **convert** the content of the image into bytes. second, we **split** image into 4-bytes to make **XORing** with the specified key that was generated from chaos generator, third we **define** and **initialize** the **identity(id)** or rank of each processor, fourth we initialize **senddata** that express about bit generated from chaos generator wanted to send and distribute it to another processor, fifth initializing **recievadata** that express about bit that received from root processor, sixth we **initialize MPI environment**, in the seventh step we check if identity of worked processor is a root processor then input **senddata** and send it to process p1 else **input recievadata** and receive it from root processor p0, in the end, we finalize MPI environment to produce distributed sequence in threads each one of them is 4-byte. Finally, we get scrambled and unclear images by XORing these sequences with sequences from the image. In this way we splitting half number of cores for each bit. By supposing in this case that we have 2 cores the first core p0 for the first bit and second core for the second bit p0. In case of having 8 cores the distribution of processor will be as presented in Figure 17 and Figure 18.

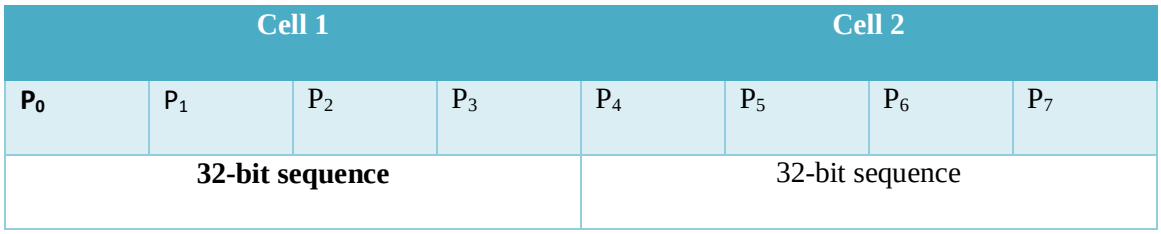


Figure 17: parallel sequence generator

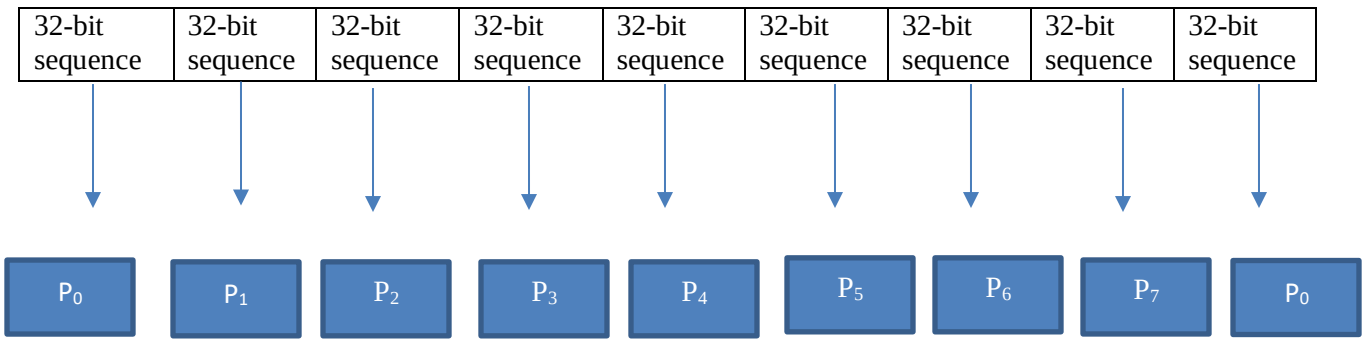


Figure 18 : scattering the generated sequence

3.6 IOT

The Internet of Things (IoT), also called the Internet of Everything or the Industrial Internet, is a new technology paradigm envisioned as a global network of machines and devices capable of interacting with each other. The IoT is recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries [23]. In order to make parallel chaos crypto system be interactive and available for each user we will make an emulation and application of parallel proposed chaotic system in raspberry pi.

3.7 Conclusion

We designed and implemented an efficiently and securely chaos-based generator, its structure is generic randomness, non-periodic and it gives production of highly secure sequences. Also, the study demonstrates the general design of this chaos generator and its specified components. Finally, we develop and improve parallel chaos crypto system by applying the distributed memory architecture model by using message passing interface implementation to improve the efficiency of using resources and decrease the needed time for encryption.

In the next chapter, we will implement a parallel chaos crypto system for an image to produce a scrambled image in an efficient way that combines good encryption from attack and at the same time have good performance. Also, there will be a comparative study that combines three sides. first sequentially applying chaos generator, second apply shared memory parallel model and finally applying MPI in chaos generator by computing the number of cycles per byte, bit rate and encryption time. Finally, we will apply and make an emulation of images has encrypted by chaos generator using MPI on the IoT environment especially in raspberry pi.

Chapter 4

Performance and security analysis of proposed parallel cryptosystem

4.1 Introduction

In today's applications the information security becomes one of the most important study subjects. For achieving the purposes of security through securing sensitive data which is confidentiality, integrity, and availability there is a need to use of classical and traditional techniques to encrypt data such as data encryption standard (DES), advanced encryption standard and the algorithm that developed by Shamir and Rivest (RSA). These algorithms based on Number theory and algebraic are more suitable for encoding text and not seem very suitable to encode large data sizes such as image and video. The chaos came as an alternative choice to fill and close this gap. In block encryption algorithms, encryption applied to fragmented data in contrast to image encryption algorithms that are pixel-based. Chaos theory provides as any other encryption algorithm two important features confusion and diffusion concepts that are proposed by Shannon. Confusion means that any change in original data led to change in encrypted data however diffusion means that no derivation of original data from encrypted data through statistical attacks. As we know that chaos has many characteristics such as system parameter, ergodicity, sensitivity to its initial conditions and any changes of it led to completely change in values that are generated from chaos. This study are ongoing on large and small data sizes to complete the operations in a shorter time and to reduce costs. Parallel computing technologies depended on distributed memory architecture model that based on message passing interface came as an alternative choice that communicates directly with hardware to provide high speed and high efficiency of the desired application . Firstly In this chapter, we will give a short description of the structure and work mechanism chaos-based generator. Then, we will discuss Performance computation of parallel proposed Chaos cryptosystem on two different workstations and cores. Finally, we will give a short description about Security analysis of chaos system.

4.2 The General architecture of Proposed Parallel chaos cryptosystem

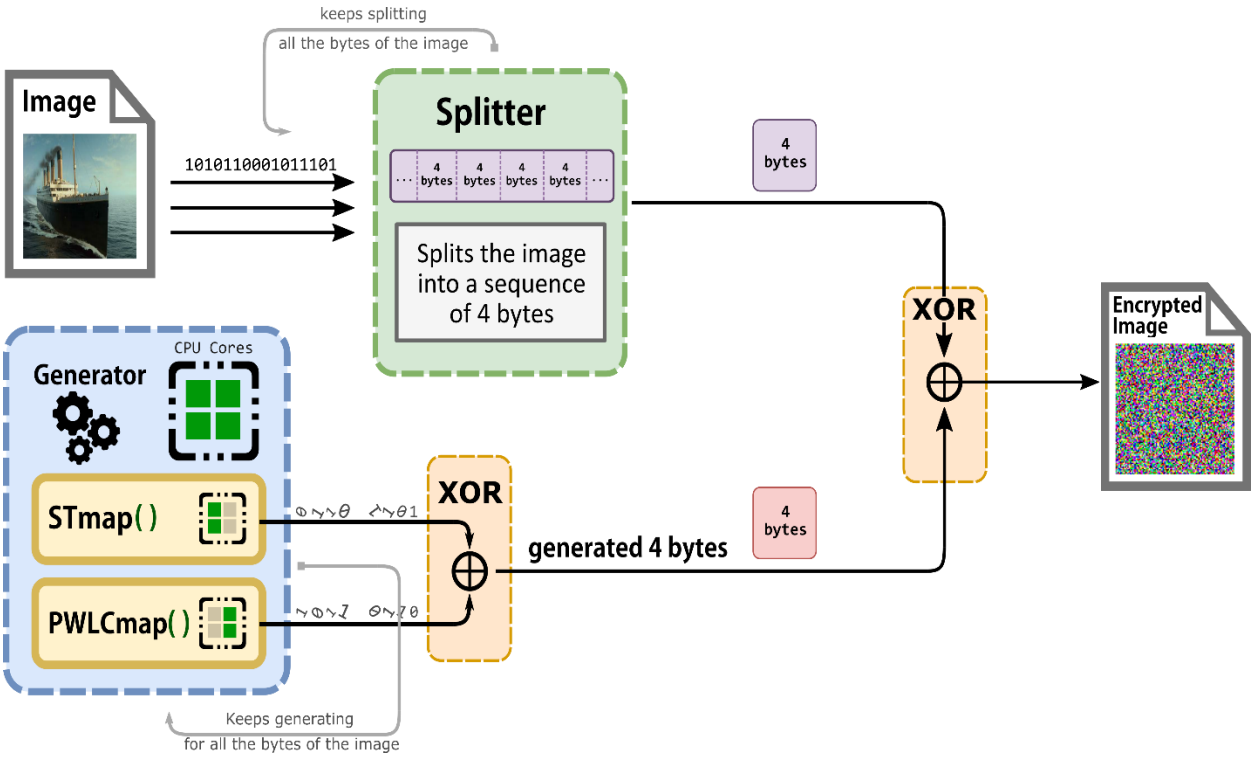


Figure 19: Proposed chaos cryptosystem

Figure 19 illustrates the general architecture, components, and Processing operations in our proposed chaos cryptosystem. It has two main components such a splitter and generator. First image enters a splitter that split the image into a sequence of 4 bytes. In the same time generator responsible to generate 4 bytes sequence through XORing both sequences came from SkewTent and PWLC map. As noted from the figure that cores distribute equally to each map. In the end every 4 bytes in generator sequence XOR with corresponding 4 bytes in splitter sequence in order to generate a scrambled image.

4.3 Performance computation of Parallel Proposed Chaos cryptosystem

In order To calculate and evaluate the performance of the parallel proposed cryptosystem based on MPI, we apply some experiments on two computers. First one that has Intel Core i3 (TM) working at 1800 GHz CPU and 4 GB Ram and the second has Intel Core i5 (TM) working at 2400 GHz CPU and 4 GB Ram. We run the parallel cryptosystem under Ubuntu 18.04 as a trusted Linux version. In these experiments, we take in concern three measurements i.e. Number of the cycle to generate one byte (cycle/byte) measurement, Generation time measurement in a microsecond, and the rate at which bit transferred in megabit per second (Mbit/s). As shown in **Equation 1, 2, 3** and **Table 9, 10, 11,12,13,14** that present and clarify the mathematical equation and result obtained from these measurements [6].

$$\mathbf{NCpB} = \frac{\mathbf{CPUSpeed(Hertz)}}{\mathbf{BR(Mbit/s)}} \quad \textbf{(1)}$$

$$\mathbf{BR} = \frac{\mathbf{Datasize(Mbit))}}{\boldsymbol{Gen\_time(s)}} \quad \textbf{(2)}$$

$$\mathbf{ET} = \frac{\mathbf{Image\_size(Mbit)}}{\mathbf{Encryption\_time(s)}} \quad \textbf{(3)}$$

As noted from **Table 9** that the number of cycles per byte generally decreasing faster in MPI comparing with sequential and parallel implementation because MPI exploits all parallelism In cores either in small and large data by distributing and balance all the load equally to processors. As shown in **Table 9** that NCBP, when the size of data is small such as 2048 byte in MPI, is 44.9 However in sequential and Pthread are 155.9 and 82950 respectively. In contrast, when applying MPI in large data such as 3145278 the NCPB in MPI is 22.2 and the NCPB in sequential and parallel are 32.0 and 26.3 respectively. So as the result has shown that using MPI in our cryptosystem consider as the fastest model either in small and large image sizes.

Table 9: Number of Cycle per byte for Proposed Cryptosystem On two Cores

Data	NCpB_Seq	NCpB_Pthread	NCpB_MPI
64 byte	242.1 cycles/B	13463.2 cycles/B	48.4 cycles/B
128 byte	184.0 cycles/B	6726.8 cycles/B	53.3 cycles/B
256 byte	114.6 cycles/B	2791.9 cycles/B	53.3 cycles/B
512 byte	127.1 cycles/B	1326.9 cycles/B	23.0 cycles/B
1024 byte	292.0 cycles/B	9879.8 cycles/B	44.0 cycles/B
2048 byte	155.9 cycles/B	8295.0 cycles/B	44.9 cycles/B
4096 byte	82.1 cycles/B	3586.4 cycles/B	27.1 cycles/B
16384 byte	44.9 cycles/B	649.9 cycles/B	24.6 cycles/B
32768 byte	37.6 cycles/B	443.6 cycles/B	24.1 cycles/B
65536 byte	34.7 cycles/B	189.8 cycles/B	23.6 cycles/B
125000 byte	34.6 cycles/B	141.4 cycles/B	24.8 cycles/B
196608 byte	34.7 cycles/B	78.0 cycles/B	23.3 cycles/B
3145728 byte	32.0 cycles/B	26.3 cycles/B	22.2 cycles/B

In contrast to **Table 9** that shows the result based on number of cycles **Table 10** has shown the average time of chaos generator in both ways sequential and parallel. The study shows that the average Generation time of parallel chaos generator based on MPI implementation has less time than sequential chaos generator and parallel chaos generator based on shared memory architecture (Pthread implementation). That means MPI exploits all its core parallelism by distributing memory for each core that runs independently from each other. The average encryption time when small data entered the generator such as 128 In MPI is 11 and in sequential and Pthread versions are 38 and 1389 cycle respectively. However, when the large data entered the cryptosystem such as 3145728 byte the average generation time in MPI is 116401 in contrast to sequential and Pthread the average generation time will be 162322 and 133493 respectively.

As known these days that time is an important factor to measure the efficiency and availability of software applications and for that reasons using MPI as a parallel model consider as the best choice to reduce time and increase availability through encrypting images.

Table 10 Generation Time For Sequential , Pthread and MPI Implementation on two cores

Data	Gen_time_Seq	Gen_time_Pthread	Gen_time_MPI
64 byte	25	1390	5
128 byte	38	1389	11
256 byte	49	1153	22
512 byte	105	1096	19
1024 byte	471	15938	71
2048 byte	503	26763	145
4096 byte	530	23142	175
16384 byte	1180	17090	646
32768 byte	1982	23400	1271
65536 byte	3665	20052	2489
125000 byte	6973	28523	5008
196608 byte	10990	24748	7391
3145728	162322	133493	116401

As clarified in **Table 11** that demonstrates the result of bit rate which is the number of bits that are processed in a unit of time that bit rate in MPI is 862.32 when the data is 512, however, when the data the bit rate in sequential and Pthread implementation is 156.04 and 14.95 respectively but the bit rate in MPI is 864.79 when data is 3145728 and the bit rate in sequential and Pthread is 620.14 and 754.07 respectively. The reason for increasing bit rate In MPI comparing with other implementation that MPI has a good communication system that allows the processor to

communicate and transfer data among each other which makes MPI have the highest encryption bit rate in both small and big data.

Table 11 : Bit Rate for Sequential, Pthread and MPI Implementation on two cores

Data	BR_Seq	BR_Pthread	BR_MPI
64 byte	81.92 Mbps/s	1.47 Mbps/s	409.60 Mbps/s
128 byte	107.79 Mbps/s	2.95 Mbps/s	372.36 Mbps/s
256 byte	173.06 Mbps/s	7.10 Mbps/s	372.36 Mbps/s
512 byte	156.04 Mbps/s	14.95 Mbps/s	862.32 Mbps/s
1024 byte	67.94 Mbps/s	2.01 Mbps/s	450.70 Mbps/s
2048 byte	127.24 Mbps/s	2.39 Mbps/s	441.38 Mbps/s
4096 byte	241.51 Mbps/s	5.53 Mbps/s	731.43 Mbps/s
16384 byte	442.03 Mbps/s	30.52 Mbps/s	807.43 Mbps/s
32768 byte	527.95 Mbps/s	44.72 Mbps/s	823.29 Mbps/s
65536 byte	571.90 Mbps/s	104.53 Mbps/s	842.11 Mbps/s
125000 byte	573.64 Mbps/s	140.24 Mbps/s	798.72 Mbps/s
196608 byte	572.45 Mbps/s	254.21 Mbps/s	851.20 Mbps/s
3145728 byte	620.14 Mbps/s	754.07 Mbps/s	864.79 Mbps/s

We apply NCpB, Bitrate, and generation time as the main measurements on a computer that has 4 processors to see the difference in these measurements when applying MPI on chaos cryptosystem. In general, this study shows that performance increase twice than on two processors that means the performance of our proposed parallel cryptosystem increasing proportionally with increasing the number per processor on workstations as In contrast of NCpB

that decreasing proportionally with increasing number of processors .**Table 12** clarified NCpB on a workstation that has 4 cores that MPI is the fastest model comparing with other models for example when data is 256 the NCpB in MPI is 31.5 in contrast to sequential and Pthread implementations that NCpB is 96.9 and 2767.7 respectively. However, NCpB in MPI is 13.2 in contrast to sequential, and Pthread is 19.2 and 16.7 respectively when data is 3145278 byte.

Table 12 : Number of Cycle per byte for Proposed Cryptosystem on Four Cores

Data	NCpB_Seq	NCpB_Pthread	NCpB_MPI
64 byte	164.7 cycles/B	9065.9 cycles/B	96.9 cycles/B
128 byte	116.2 cycles/B	4663.7 cycles/B	33.9 cycles/B
256 byte	96.9 cycles/B	2767.7 cycles/B	31.5 cycles/B
512 byte	89.6 cycles/B	1348.7 cycles/B	24.2 cycles/B
1024 byte	78.7 cycles/B	638.0 cycles/B	23.6 cycles/B
2048 byte	116.5 cycles/B	2532.9 cycles/B	21.4 cycles/B
4096 byte	58.1 cycles/B	1215.3 cycles/B	18.0 cycles/B
16384 byte	27.2 cycles/B	510.9 cycles/B	15.1 cycles/B
125000 byte	18.4 cycles/B	71.5 cycles/B	14.1 cycles/B
196608 byte	18.4 cycles/B	54.1 cycles/B	13.8 cycles/B
3145728 byte	19.2 cycles/B	16.7 cycles/B	13.2 cycles/B

When applying generation time results on a computer that has 4 cores the result as reported in **Table 13** demonstrates that generation time decreasing proportionally with an increasing number of processors. the generation time In MPI is 13 in contrast to sequential and Pthread implementations that the generation time is 40 and 1143 respectively when data is 256 However

when data is 3145728 the generation time in MPI is 69631 but the generation time in sequential and Pthread is 97584 and 80568 respectively.

Table 13: Generation Time For Sequential, Pthread and MPI Implementation on four cores

Data	Gen_time_Seq	Gen_time_Pthread	Gen_time_MPI
64 byte	17 Micro s	936 Micro s	10 Micro s
128 byte	24 Micro s	963 Micro s	7 Micro s
256 byte	40 Micro s	1143 Micro s	13 Micro s
512 byte	1114 Micro s	1114 Micro s	20 Micro s
1024 byte	130 Micro s	1054 Micro s	39 Micro s
2048 byte	376 Micro s	8172 Micro s	69 Micro s
4096 byte	375 Micro s	7842 Micro s;	116 Micro s
16384 byte	715 Micro s	13435 Micro s	398 Micro s
125000 byte	3714 Micro s	14416 Micro s	2844 Micro s
196608 byte	5838 Micro s	17148 Micro s	4387 Micro s
3145728 byte	97584 Micro s	80568 Micro s	69631 Micro s

As noted from **Table 14** that reported result of Bitrate on a computer that has 4 cores that experimental result shows that Bitrate is increasing proportionally with increasing the number of processors on a workstation. The bitrate in MPI is 1103.5 when data is 4096 in contrast to sequential and parallel implementation that bit rate is 341.33 and 16.32 respectively. For 196608 data the bit rate In MPI is 1434.06 However in Pthread and sequential is 1077.63 and 366.88 respectively.

Table 14: Bit Rate For Sequential, Pthread and MPI Implementation on four cores

Data	BR_Seq	BR_Pthread	BR_MPI
64 byte	120.47 Mbps/s	2.19 Mbps/s	204.80 Mbps/s
128 byte	170.67 Mbps/s	4.25 Mbps/s	585.14 Mbps/s
256 byte	204.80 Mbps/s	7.17 Mbps/s	630.15 Mbps/s
512 byte	221.41 Mbps/s	14.71 Mbps/s	819.20 Mbps/s
1024 byte	252.06 Mbps/s	31.09 Mbps/s	840.21 Mbps/s
2048 byte	170.21 Mbps/s	7.83 Mbps/s	927.54 Mbps/s
4096 byte	341.33 Mbps/s	16.32 Mbps/s	1103.45 Mbps/s
16384 byte	729.51 Mbps/s	38.82 Mbps/s	1310.55 Mbps/s
125000 byte	1077.01 Mbps/s	277.47 Mbps/s	1406.47 Mbps/s
196608 byte	1077.63 Mbps/s	366.88 Mbps/s	1434.06 Mbps/s
3145728 byte	1031.55 Mbps/s	1249.41 Mbps/s	1445.65 Mbps/s

Figure [19-24] show the curves that depict NCpB, Generation time and Bit Rate measurements either in two or four cores for sequential, Pthread And MPI Implementations .

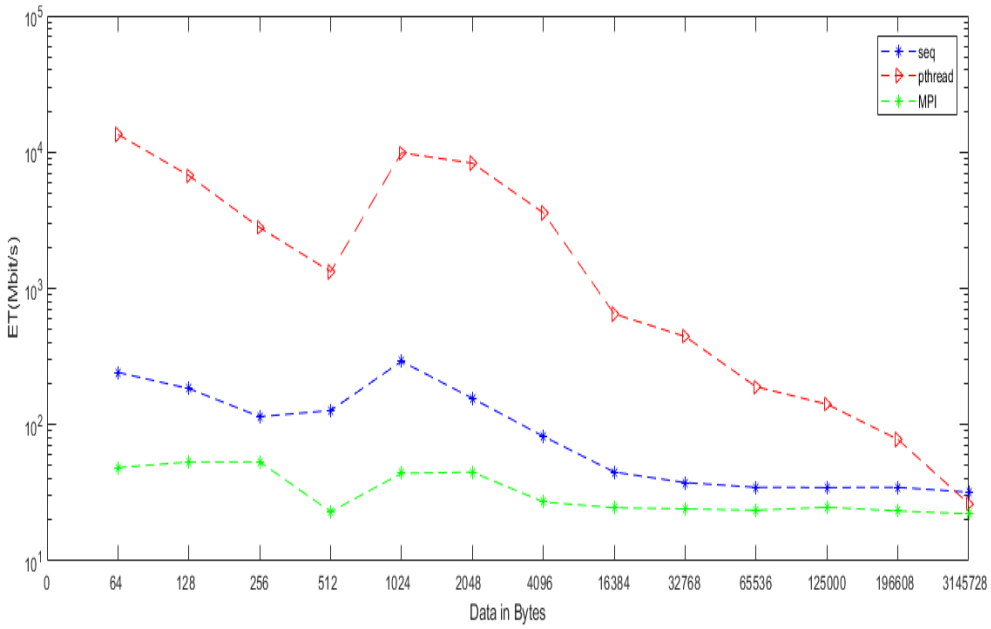


Figure 20 : NCpB For Sequential, Pthread and MPI Implementation on two cores

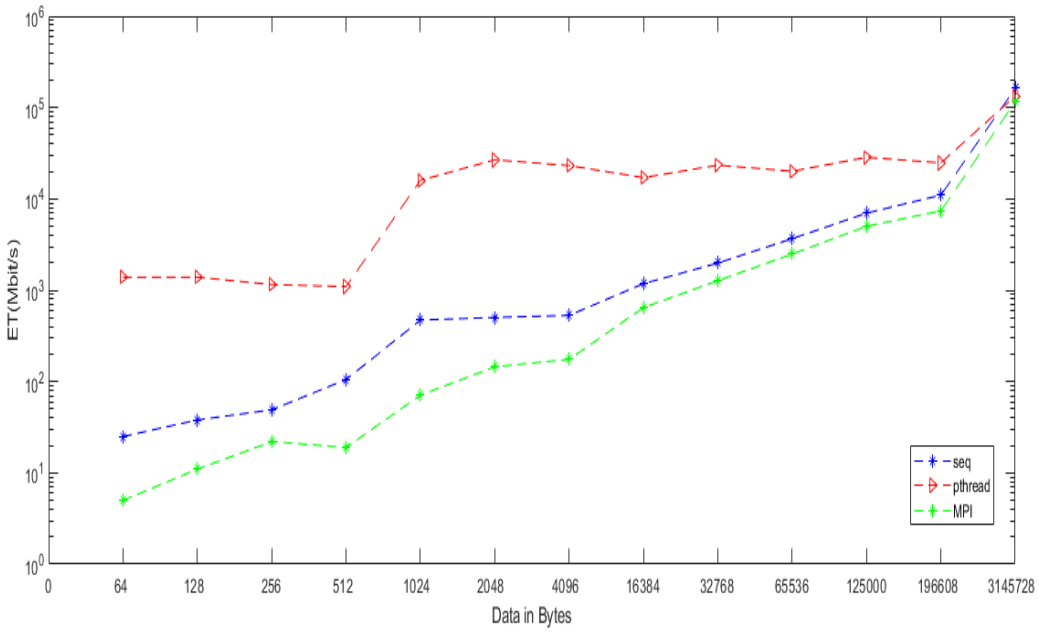


Figure 21: Generation Time For Sequential,Pthread and MPI Implementation on two cores

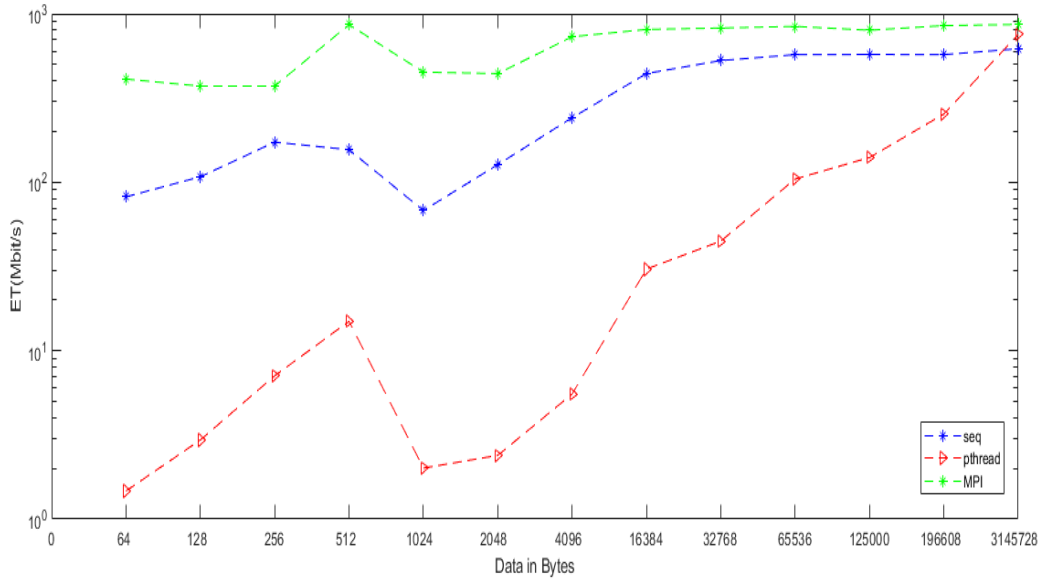


Figure 22 : Bit Rate For Sequential , Pthread and MPI Implementation on two cores

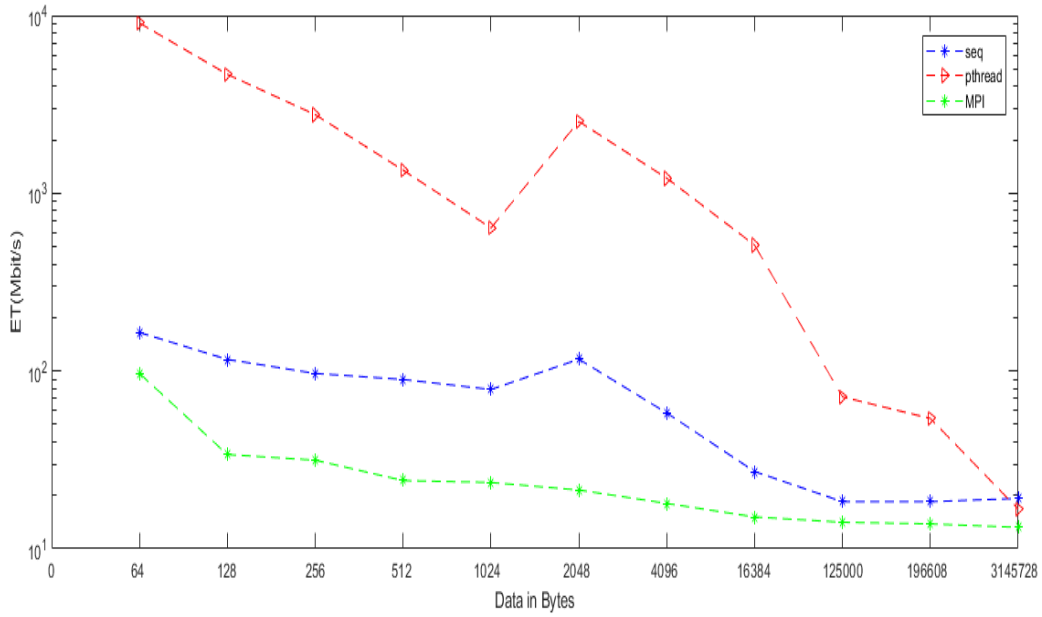


Figure 23 : . NCpB For Sequential, Pthread and MPI Implementation on four cores

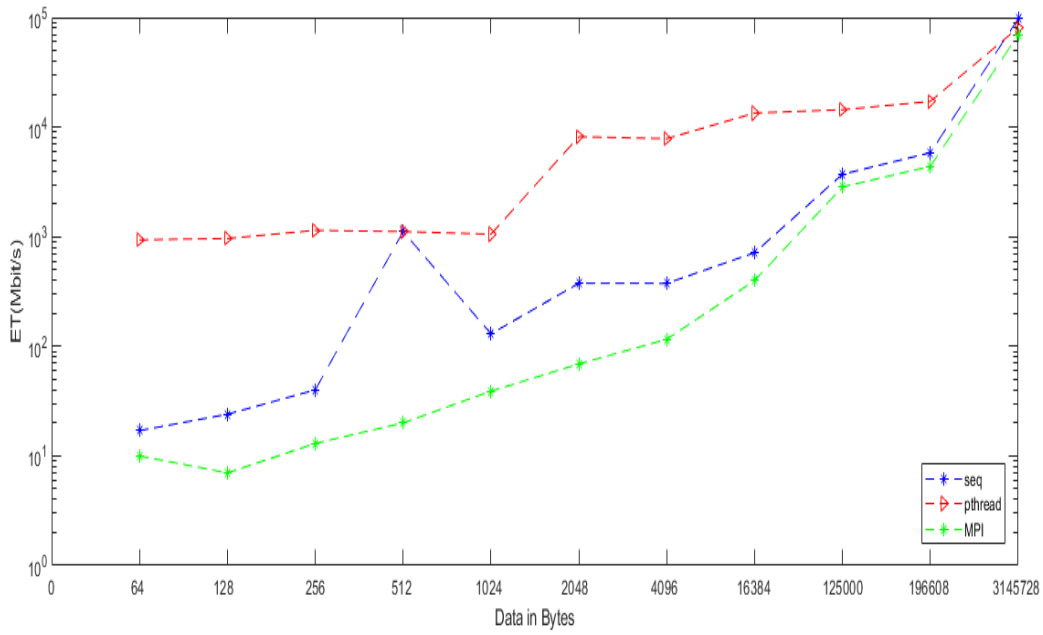


Figure 24 : Generation Time For Sequential,Pthread and MPI Implementation on four cores

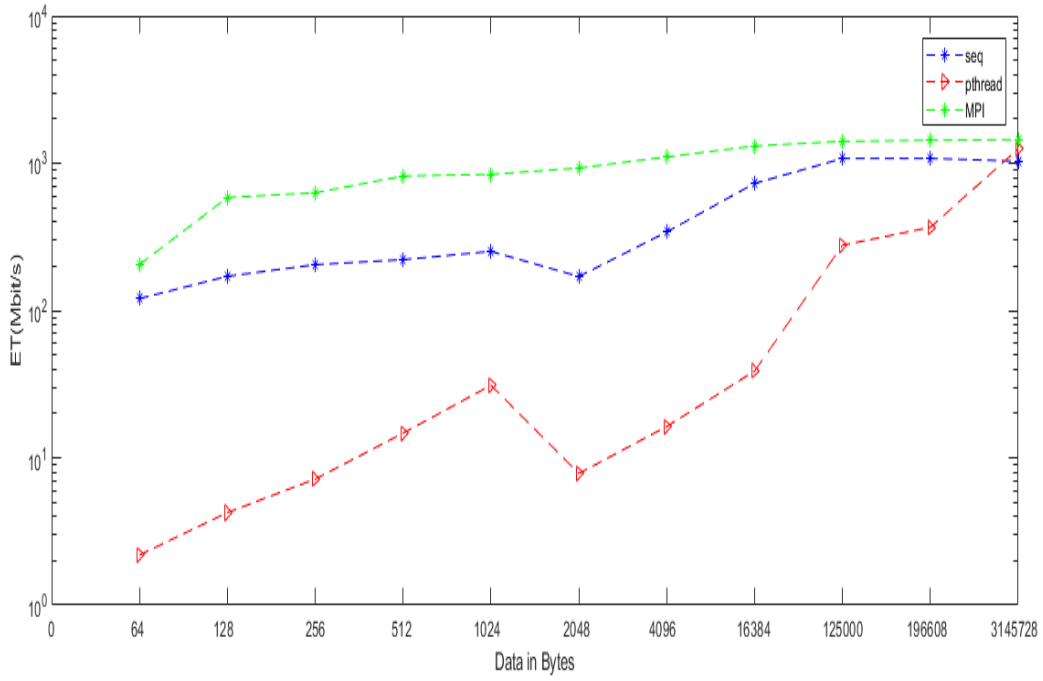


Figure 25 : Bit Rate For Sequential, Pthread and MPI Implementation on four cores

4.4 Speed Up Calculations

Speed up considered as the measurement of parallel code that show how much faster it runs in parallel. By supposing the time to run code on one processor is Time_Seq and time to run code on N processor is Time_Parallel [24] so the speedup is given as shown in **Equation 4**.

$$\boldsymbol{Speed\ Up} = \frac{\mathbf{Time\_Seq}}{\mathbf{Time\_Parallel}} \quad \textbf{(4)}$$

For measuring the enhancement of bit rate by dividing the bit rate in parallel to bit rate in sequential as clarified in **Equation 5**.

$$\boldsymbol{BitRate\_Enhancement} = \frac{\mathbf{BitRate\_Parallel}}{\mathbf{BitRate\_Seq}} \quad \textbf{(5)}$$

The enhancement of a number of cycles per byte can be given as shown in **Equation 6.**

$$\boldsymbol{NCpB\_Enhancement} = \frac{\mathbf{NCpB\_Seq}}{\mathbf{NCpB\_Parallel}} \quad \textbf{(6)}$$

By applying **Equation 4, 5, 6** on Table **[9-14]** the result will be as shown in **Figures [26-31]**.

Amdahl's Law

Amdahl's Law used to compute an upper bound on the speedup of an application without actually writing any concurrent code. Each uses the percentage of (proposed) parallel execution time (pctPar), serial execution time (1 - pctPar), and the number of threads/cores (p). A simple formulation of Amdahl's Law to estimate speedup of a parallel application on p cores is given here [25]:

$$\boldsymbol{Speed\ Up} = \frac{\mathbf{1}}{\mathbf{1 - pctPar} + \frac{\mathbf{pctPar}}{\mathbf{p}}} \quad \textbf{(7)}$$

In our case we expect 45% of a serial application's run time could be executed in parallel on 4 cores , the estimated speedup, according to Amdahl's Law, could as much 1 / (0.55 + 0.45/4) = 1.50943396226). We expect an increasing in overhead with increasing number of cores that will appear especially when number of processors will be more than 8 processors.

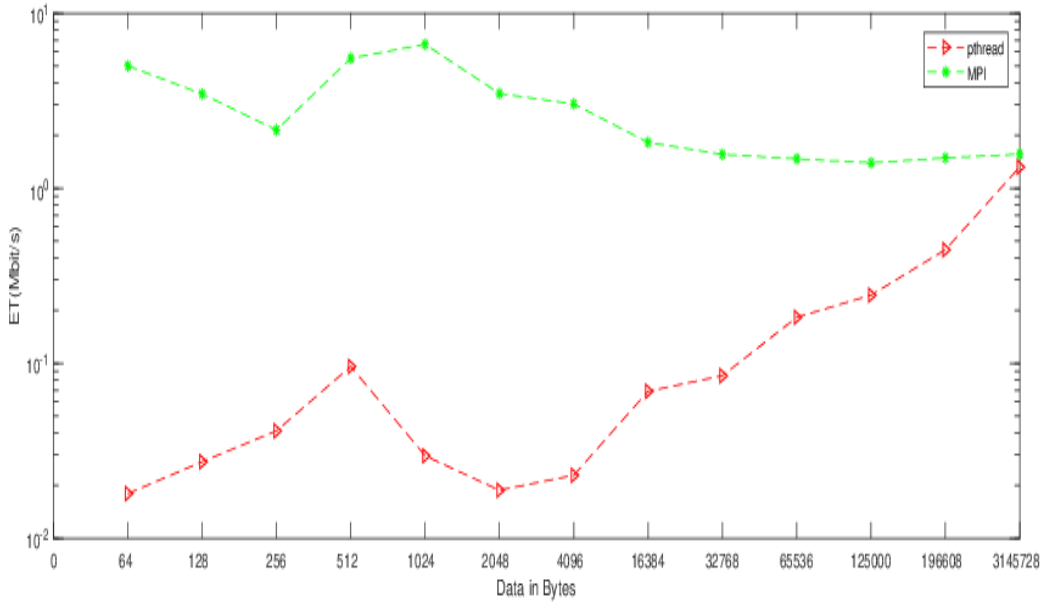


Figure 26 : NCpB Enhancement on Two Cores

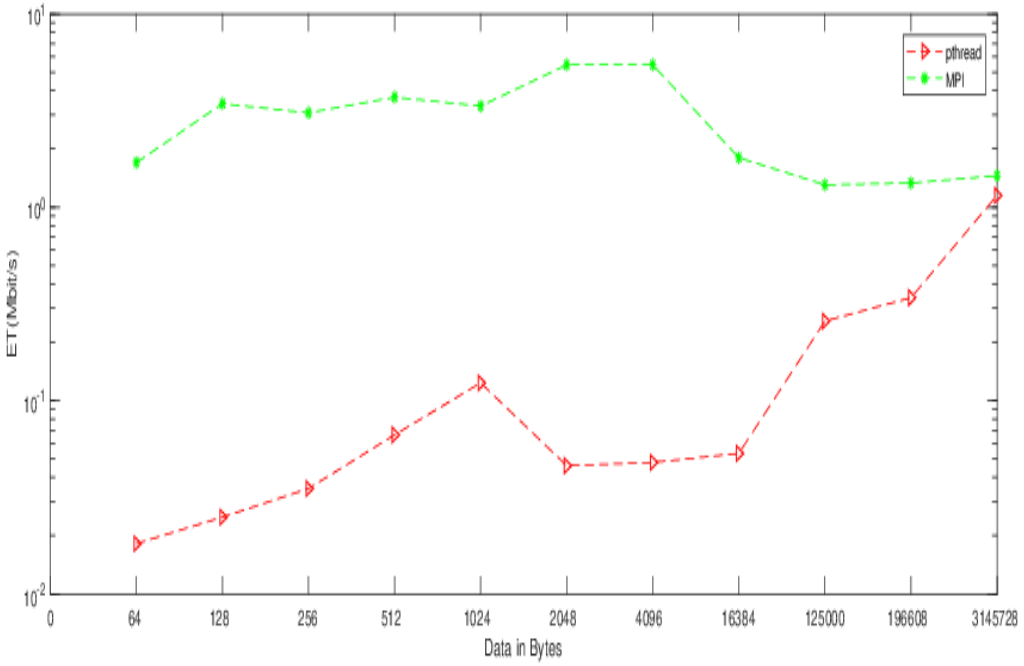


Figure 27: NCpB Enhancement On Four Cores

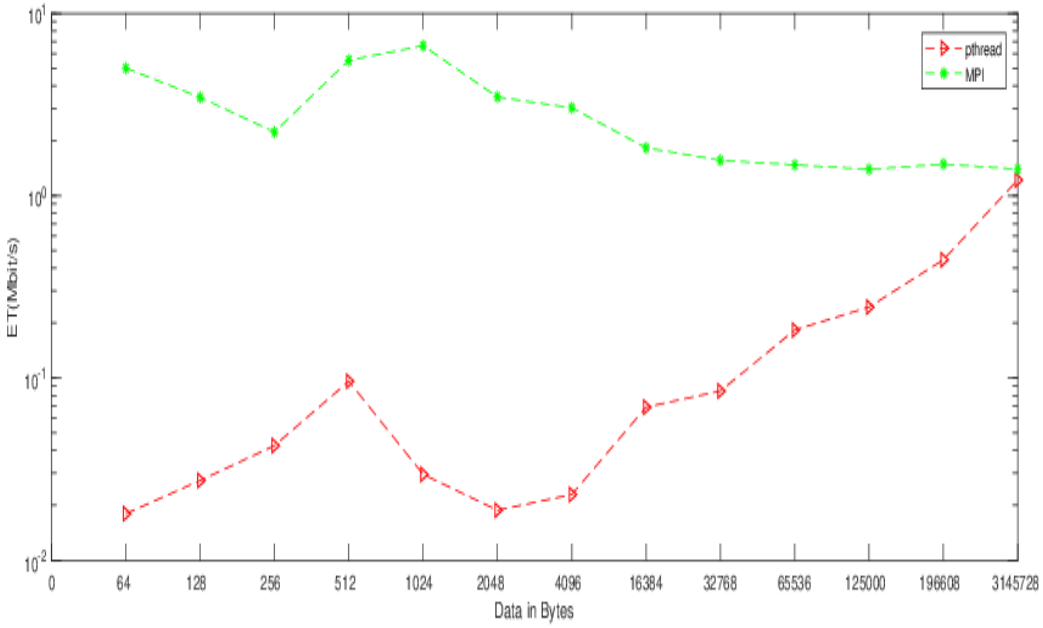


Figure 28: Generation time enhancement On Two Cores

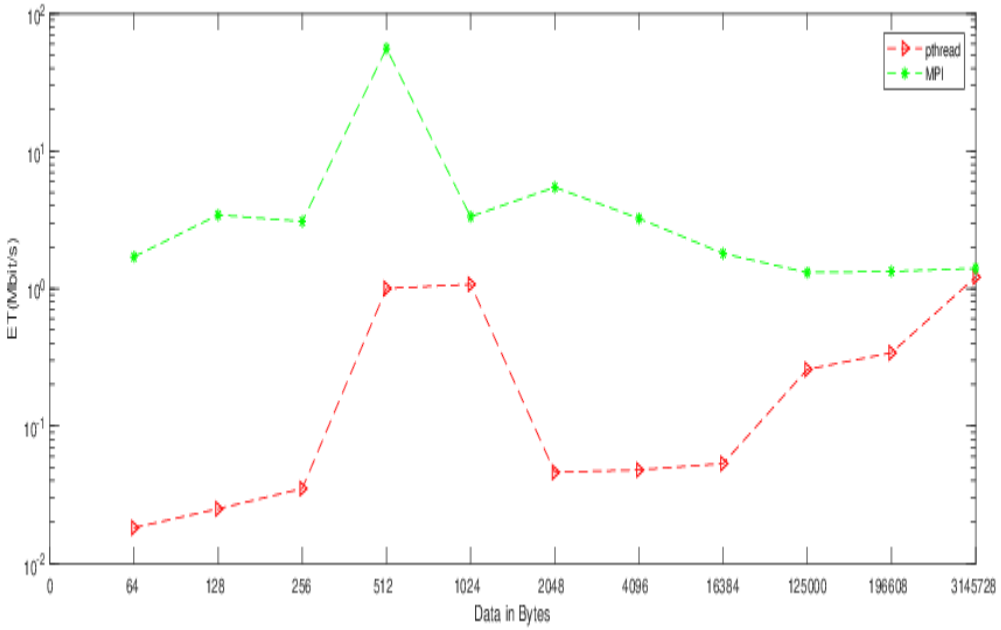


Figure 29: Generation time Enhancement on Four Cores

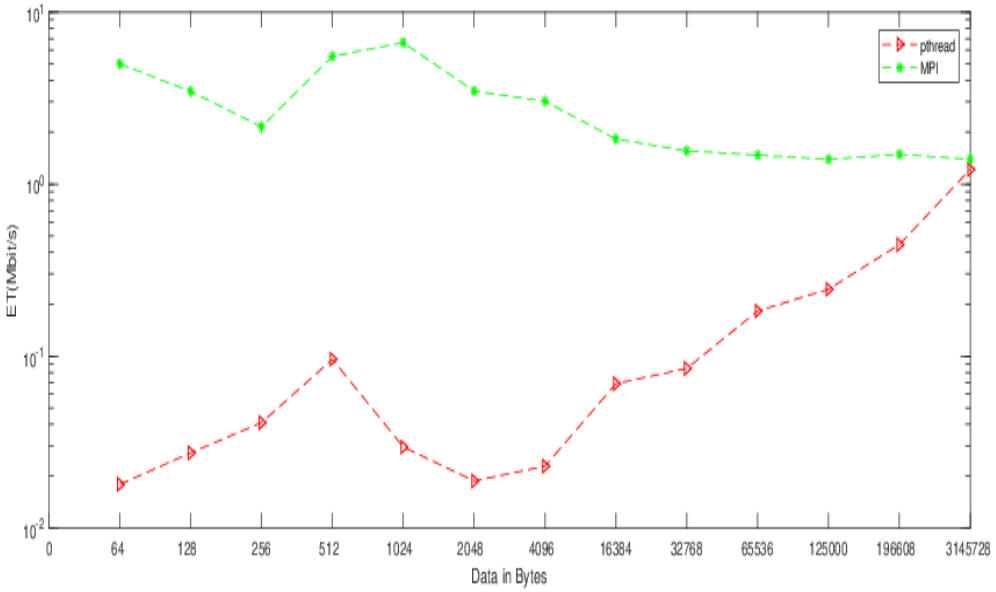


Figure 30 : bit rate Enhancement On Two Cores

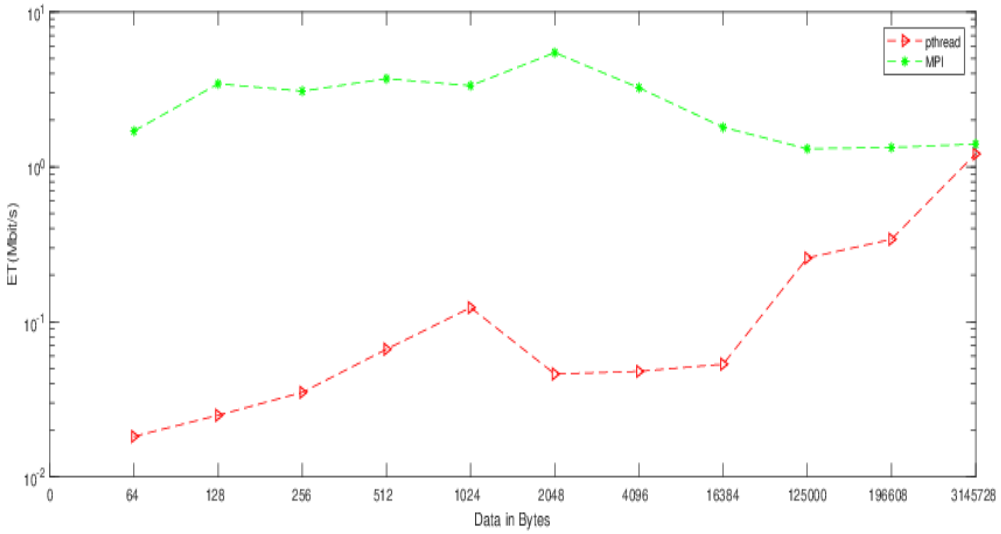


Figure 31: bit rate Enhancement on four cores

4.5 Security

In this section, we will discuss the security analysis of the parallel proposed cryptosystem against cryptanalytic and Statistical attacks.

Key Space

As we know in encryption, the larger keyspace algorithm has a strong ability to resist a brute force attack in contrast to algorithms that have a small keyspace suffered from lacking sequence randomness. The proposed cryptosystem has different keyspace according to the selected delay. The keyspace in delay =1 is nearly 299-bit however in delay = 3 the keyspace reach to 555-bits. This numerous keyspace make the proposed cryptosystem has immunity against brute force attacks. **Table 15** list the size of keyspace of similar chaotic algorithms. As clarified in **Table 15** that our algorithm keyspace in delay 3 is 555 bit. It seems the largest one comparing with the other listed algorithms.

Table 15: keyspace comparison of similar Algorithms

Encryption Algorithm	Key Space
Proposed Algorithm	2^{555}
Wange et. al Algorithm[26]	2^{149}
Guesmi et. al Algorithm[27]	2^{256}
Li et. al Algorithm[28]	2^{299}
Li et. al Algorithm[29]	2^{375}
Curiac et. al Algorithm[30]	2^{128}
Curiac et. al Algorithm[31]	2^{357}
Zhu et. al Algorithm[32]	2^{339}

Key security and sensitivity attack

In order to check key sensitivity of our proposed cryptosystem, we apply two important measurements which are Number of pixel change rate (NPCR) and the unified Average changing intensity (UACI) which has shown that the proposed cryptosystem is very sensitive to one-bit

change that appears when we encrypt “Lena " image more than 100 times using 100 secret key that different in LSB bit

$$NPCR = \frac{1}{L \times C \times P} \times \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{j=1}^{C} D(i,j,p) \times 100\% \quad (8)$$

Where

$$D(i,j,p) = \begin{cases} 0, & if\ C_1(i,j,p) = C_2(i,j,p) \\ 1, & if\ C_1(i,j,p) \neq C_2(i,j,p) \end{cases} \quad (9)$$

$$\text{UACI} = \frac{1}{L \times C \times P \times 255} \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{j=1}^{C} |C_1(i,j,p) - C_2(i,j,p)| \times 100\% \quad (10)$$

In the prior equations L, C, and P are the length, width, and plane sizes of the image respectively. However i, j and p are the rows, column, and plane index respectively. As the result shown in **Table 16** that NPCR, UACI of the proposed cryptosystem is close to the optimal NPCR and UACI values 99.61 and 33.46 respectively.

Table 146: NPCR, UACI and HD measurements

Cryptosystem	NPCR	UACI
Proposed Cipher Cryptosystem	99.665	33.459

Information entropy

The entropy $E(X)$ is statistical measure of uncertainty in information theory [40]. It is defined as follows:

$$E = -\sum_{i=0}^{256} P(x_i)\, log_2\, P(x_i) \quad (11)$$

Where X is a random variable and $p(x_i)$ is the probability mass function of the occurrence of the gray value x_i. Let us consider that there are 256 states of the information source in red, green,

and blue colors of the image with the same probability. We can get the ideal $E(X) = 8$, corresponding to a truly random source. As noted from **Table 17** that illustrate information entropy of some Cipher Images such as Sharukhan, Titanic, Photographer, Manhattan, Camera-man, Lena and Boat is closer to 8. The entropy of all ciphers are closer to 8 and it proves that the cipher image is random dataset of pixels.

Table 17: information entropy of some Cipher Images

Cipher Image	Sharukhan	Titanic	Photographer	Manhattan	Camera-man	Boat	Lena
entropy	7.9999	7.9999	7.9999	7.9999	7.9998	7.9997	7.9999

As shown in **Table 18** that provides the contrast data with other advanced schemes by comparing information entropy in our proposed scheme with other advanced algorithms. The result has shown that our proposed Scheme has greater superiority in information entropy comparing with other algorithms.

Table 18: Information entropy comparison

Encryption methods	Information entropy
Proposed algorithm	7.9999
Ref. [35]	7.9973
Ref. [36]	7.9975
Ref. [37]	7.9977
Ref. [38]	7.9973
Ref. [39]	7.9982

Statistical analysis

NIST test

NIST is a statistical test suite which proposed by National Institute of Standards and Technology. NIST includes 15 tests that concentrate on different types of non-randomness that could be in a sequence. These tests are Cumulative sums (forward), the Longest run of ones, Cumulative sums (reverse, Non overlapping templates, Block-frequency, Frequency (mono bit), Runs, Rank, FFT,

Overlapping templates, Random excursions variant, Approximate entropy, Serial 1, Serial 2, Universal .In order to investigate the randomness of binary data, we apply NIST [33] to many ciphered text and the result shows that cipher text has a high rate of randomness as shown in the **Figure 32**[34]**.**

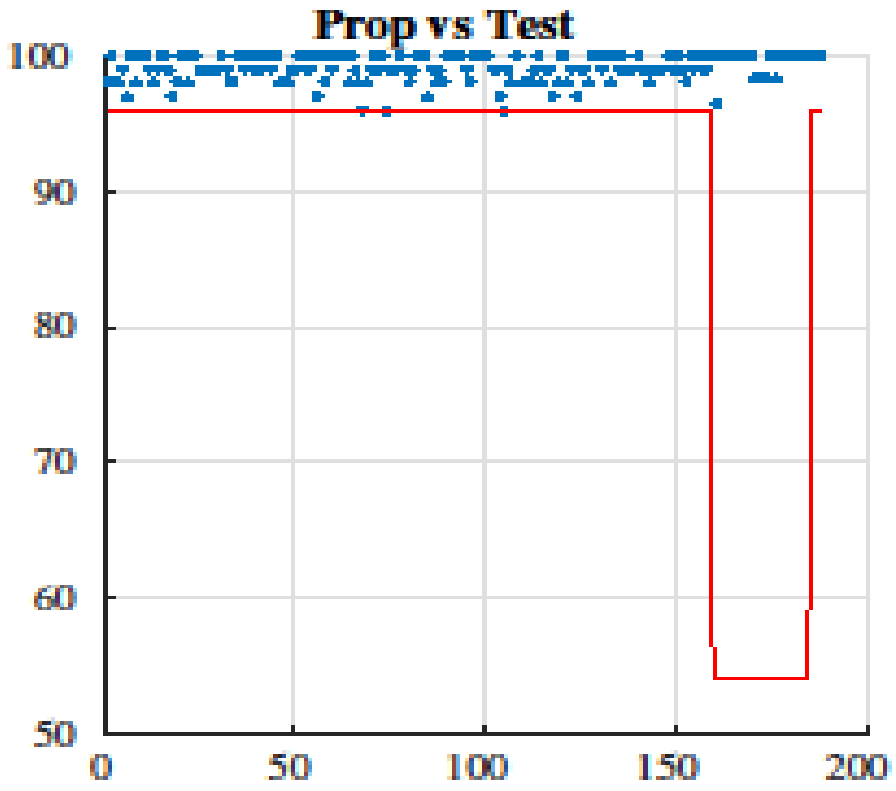
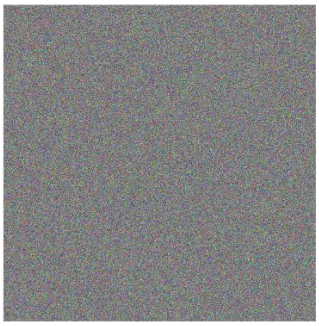


Figure 32 :NIST Test

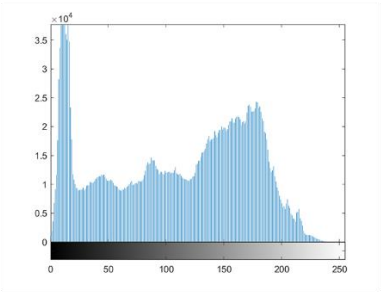
Chi-square test and histogram

In order to resist potential and statistical attacks we apply histogram which represents the distribution of pixel in specific images. It is a measure to test the uniformity of CipherImage. Whenever the histogram in CipherImage is uniform that means having more immunity against statistical stacks. For this purpose we apply the histogram for many pictures and as seen from the **Figure 33, 33, 35 and 36** that the histogram of CipherImage is uniform comparing with their PlainImage.

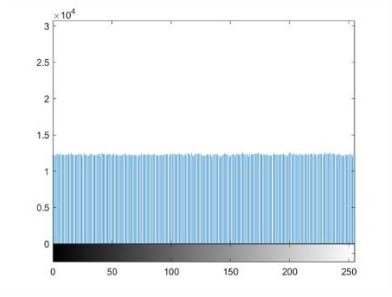
A) Titanic PlainImage



B) Titanic CipherImage



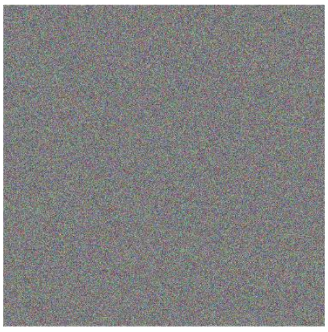
C) Titanic PlainImage Histogram



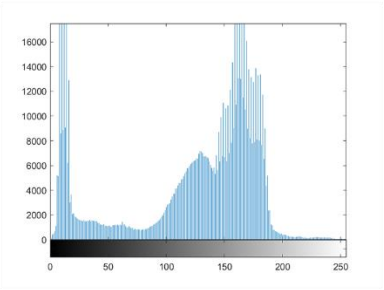
D)Titanic CipherImage Histogram

Figure 33 : Histogram of the Titanic plainImage and its CipheredImage

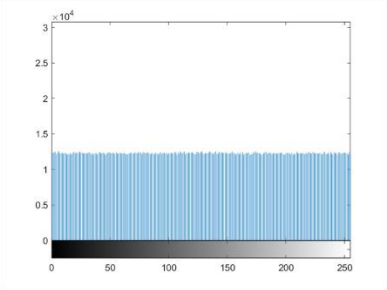
A) Photographer PlainImage



B) Photographer CipherImage



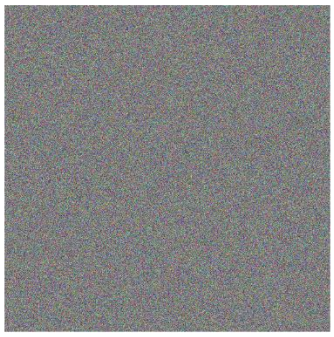
C) Photographer PlainImage Histogram



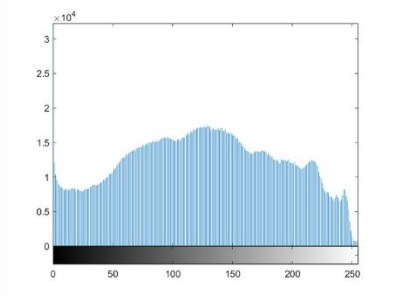
D) Photographer CipherImage Histogram

Figure 34: Histogram of the Photographer plainImage and its CipheredImage

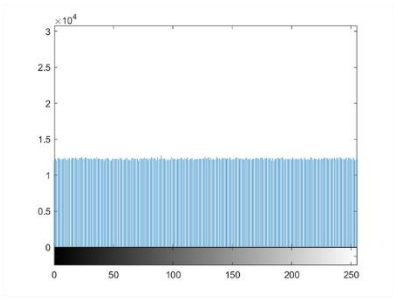
A) Manhattan City PlainImage



B) Manhattan City CipherImage



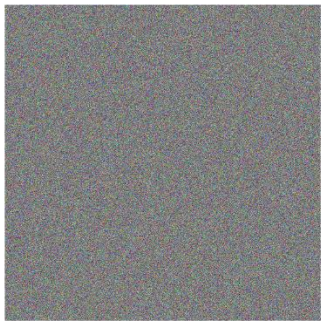
C) Manhattan City PlainImage Histogram



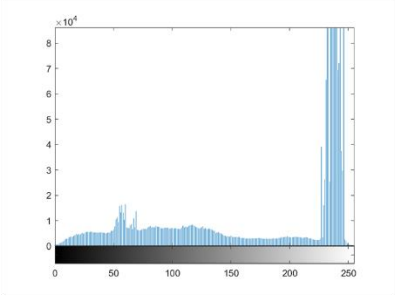
D) Manhattan City CipherImage Histogram

Figure 35: Histogram of the Manhattan plain image and its ciphered image

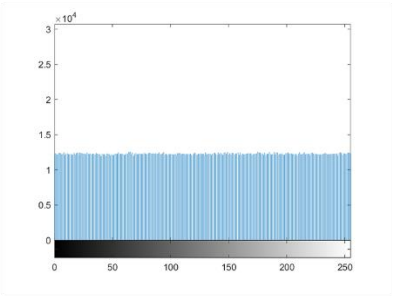
A) Sharukhan PlainImage



B) Sharukhan CipherImage



C) Sharukhan PlainImage Histogram



D) Sharukhan CipherImage Histogram

Figure 36: Histogram of the Sharukhan plain image and its ciphered image

Table 19: Chi-square value of histograms for different ciphered/plain images with different sizes

Image	Experimental value	Theoretical Value
Titanic 256×256×3	245.8750	293.247835
Titanic 512×512×3	279.1621	293.247835
Titanic 1024×1024×3	283.5923	293.247835
Photographer 256×256×3	252.1406	293.247835
Photographer 512×512×3	243.8066	293.247835
Photographer 1024×1024×3	251.6162	293.247835
Manhattan 256×256×3	264.6719	293.247835
Manhattan 512×512×3	254.6660	293.247835
Manhattan 1024×1024×3	257.3975	293.247835
Sharukhan 256×256×3	252.9531	293.247835
Sharukhan 512×512×3	228.9316	293.247835
Sharukhan 1024×1024×3	245.7544	293.247835

$$\chi^2 = \sum_{i=0}^{K-1} \frac{(O_i - E_i)^2}{E_i} \qquad \textbf{(12)}$$

The **Equation 12** is used to perform the Chi test with the following parameters: K is the number of levels (here K = 256), Oi is the observed occurrence frequency of each color level (0-255) on the histogram of the ciphered image, and E_i is the expected occurrence frequency of the uniform distribution, given hereby$\mathbf{E_i} = \frac{\mathbf{L \times C \times P}}{\mathbf{K}}$. For a secure cryptosystem, the experimental chi-square value must be less than the theoretical chi-square one, which is 293 in the case of α = 0.05 and K = 256. In **Figures 33, 33, 35 and 36** we give the histograms for the plain/cipher images for Titanic, Photographer, Manhattan City and Sharukhan in size 1024×1024×3. As we can see, the histogram of the ciphered image seems to be uniform. To assess the uniformity, we performed the chi-square test. The experimental value obtained is less than the theoretical one at 293. This means that the histograms are uniform (see **Table 19**).

Correlation analysis

The correlation test is an important test especially in encryption field according to the importance of hiding information form attacker to know some information of the plaintext from the ciphered text so in order to achieve security the correlation of proposed cryptosystem should be as lowest as possible. We apply this test by taking 10000 adjacent pixels from plane Image and ciphered image as input to the **Equations 13, 14, 15.**

$$P_{xy} = \frac{Cov(x,y)}{\sqrt{D(x)}\sqrt{D(y)}}. \quad \textbf{(13)}$$

$$Cov(x,y) = \frac{1}{N}\sum_{i=1}^{N}([x_i - E(x)\,]\,[y_i - E(y)]) \quad \textbf{(14)}$$

$$D(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - E(x))^2 \quad \textbf{(15)}$$

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i) \quad \textbf{(16)}$$

In the previous equations $\mathbf{P_{xy}}$ is the correlation coefficient of two sequences x and y . X_i and Y_i are the values of x and y respectively. For proving that cipher image is completely different from the original image we used correlation analysis of adjacent pixels for both cipher and the original image. As the result clarified in **Figure 37** that adjacent pixels in the plain image are redundant and correlated, however, adjacent pixels in cipher image nearly completely different and seem to have redundancy and correlation as low as possible as shown in **Figure 38** . This another proof that shows our proposed system has immunity against statistical attacks.

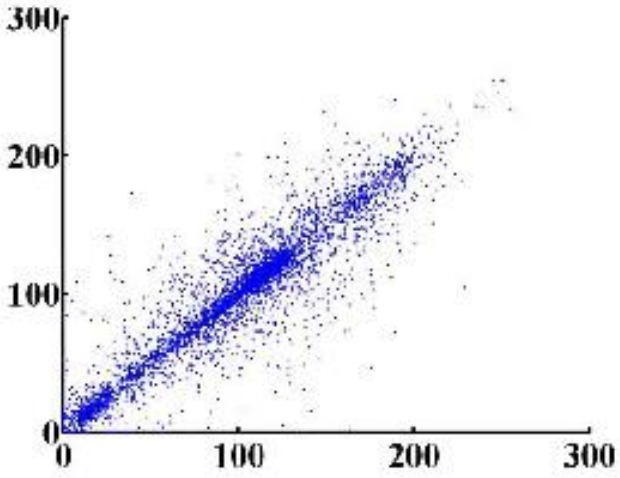


Figure 37 : Adjacent Pixels for Plain Image

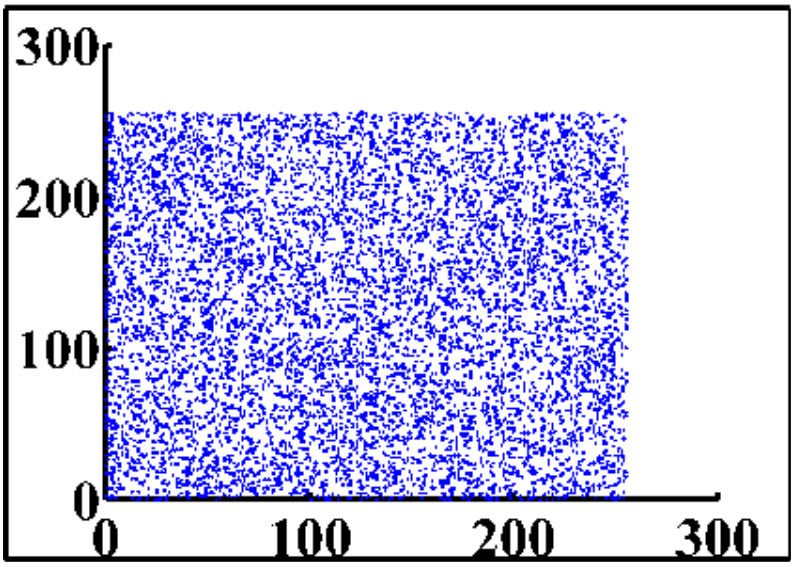


Figure 38 : Adjacent Pixels for ciphered Image

4.6 Conclusion

We have designed a parallel chaos generator based on the message passing interface consider as the fastest chaos stream cipher model its structure generic, randomness, and non-periodic. Furthermore, we apply many tests on the cryptosystem to measure its efficiency, speed, and security such as Chi-square and histogram, NIST, correlation analysis, and information entropy tests. All of the tests proved that the proposed cryptosystem has immunity against brute force and statistical attack. In addition to the previous characteristics and advantages the result shows that the sequence was generated from the parallel proposed cryptosystem has a high degree of randomness or uncertainty. In the end, we have achieved the research objectives by building a system that combines both security and good performance.

References

[1] Boris V.Protopov, A. S. (1997). A MultiThread Message Passing Interface(MPI) architecture : Performance and program issue . *parall and Distributing Comuting* , 30.

[2] gropp, R. T. (2007). Open Issue In MPI Implementation . *Springer*, 12.

[3] gropp, R. T. (September 2007). Test Suite for Evaluating Performance of MPI Implementations That Support MPI THREAD MULTIPLE. *Conference: Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 10 .

[4] Jesper Larosson, W. G. (2009). Self-Consistent MPI Performance Requirements . 10.

[5] Jian, H. P. (2018). Research on digital image encryption algorithm based on double logistic Choatic map. *EURASIP*, 10.

[6] Taha, M. A. (2017). Design and Efficent Implementation OF Choas-based Stream Cipher. 16.

[7] zue, S. (2018). A New Image Encryption Algorithim Based on Choas and Secure Hash SHA-256. 18.

[8] Diab. (2018). Secure image crypto system with unique key streams via hyper-choatic system . 53-68.

[9] zhang. (2011). A novel image encryption method based on total shuffling scheme. 2775–2780.

[10] Pacheco, P. (2011). *An introduction to parallel programming*. Elsevier.

[11] Forouzan, B. A. (2007). *Cryptography & network security*. McGraw-Hill, Inc..

[12] Heys, H. M., & Tavares, S. E. (1994, September). On the security of the CAST encryption algorithm. In *Canadian Conference on Electrical and Computer Engineering* (pp. 332-335).

[13] Mandal, P. C. (2012). Superiority of Blowfish algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(9).

[14] Weerasinghe, T. D. B. (2013, December). An effective RC4 stream cipher. In *2013 IEEE 8th International Conference on Industrial and Information Systems* (pp. 69-74). IEEE.

[15] Bernstein, D. J. (2008). The Salsa20 family of stream ciphers. In *New stream cipher designs* (pp. 84-97). Springer, Berlin, Heidelberg.

[16] Basu, S. (2011). International Data Encryption Algorithm (Idea)–A Typical Illustration. *Journal of global research in Computer Science*, *2*(7), 116-118.

[17] Stallings, W. (1995). *Network and internetwork security: principles and practice* (Vol. 1). Englewood Cliffs, NJ: Prentice Hall.

[18] Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., & McDonald, J. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.

[19] Lewis, B., & Berg, D. J. (1996). PThreads Primer. *Sun Microsystems Inc*.

[20] Boesgaard, M., Vesterager, M., & Zenner, E. (2008). The Rabbit stream cipher. In *New stream cipher designs* (pp. 69-83). Springer, Berlin, Heidelberg.

[21] Milanov, E. (2009). The RSA algorithm. *RSA Laboratories*, 1-11.

[22] Carts, D. A. (2001). A review of the Diffie-Hellman algorithm and its use in secure internet protocols. *SANS institute*, 1-7.

[23] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), 1645-1660.

[24] Çavuşoğlu, Ü., & Kaçar, S. (2019). A novel parallel image encryption algorithm based on chaos. Cluster Computing, 22(4), 1211-1223.

[25] Shi, Y. (1996). Reevaluating Amdahl's law and Gustafson's law. Computer Sciences Department, Temple University (MS: 38-24).

[26] Wang, X., Zhu, X., Wu, X., & Zhang, Y. (2018). Image encryption algorithm based on multiple mixed hash functions and cyclic shift. Optics and Lasers in Engineering, 107, 370-379.

[27] Guesmi, R., Farah, M. A. B., Kachouri, A. & Samet, M. A novel chaos-based image encryption using dna sequence operation and secure hash algorithm sha-2. Nonlinear Dyn. 83, 1123–1136 (2016).

[28] Li, S., Chen, G. & Mou, X. On the dynamical degradation of digital piecewise linear chaotic maps. Int. journal Bifurc. Chaos 15, 3119–3151 (2005).

[29] Li, S., Chen, G., Wong, K.-W., Mou, X. & Cai, Y. Baptista-type chaotic cryptosystems: problems and countermeasures. Phys. Lett. A 332, 368–375 (2004).

[30] Curiac, D.-I. & Volosencu, C. Chaotic trajectory design for monitoring an arbitrary number of specified locations using points of interest. Math. Probl. Eng. 2012(2012).

[31] Curiac, D.-I., Iercan, D., Dranga, O., Dragan, F. & Banias, O. Chaos-based cryptography: end of the road? In The International Conference on Emerging Security Information, Systems, and Technologies (SECUREWARE 2007), 71–76 (IEEE, 2007).

[32] Zhu, S., Zhu, C., & Wang, W. (2018). A new image encryption algorithm based on chaos and secure hash SHA-256. Entropy, 20(9), 716.

[32] Wu, Y., Noonan, J. P., Agaian, S.et al.Npcr and uaci randomness tests for image encryption.Cyber journals: multidisci-plinary journals science technology, J. Sel. Areas Telecommun. (JSAT)1, 31–38 (2011).

[33] Barker, E. B. & Kelsey, J. M. Recommendation for random number generation using deterministic random bit generators (revised) (US Department of Commerce, Technology Administration, National Institute of ..., 2007).

[34] Rukhin, A., Soto, J., Nechvatal, J., Smid, M. & Barker, E. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. Rep., Booz-allen and hamilton inc mclean va (2001).

[35] Rehman, A.U., Liao, X., Kulsoom, A. & Ullah, S. A modified (Dual) fusion technique for image encryption using SHA-256 hash and multiple chaotic maps. Multimed. Tools Appl. 75, 1-26 (2016).

[36] Chen, G., Mao, Y. & Chui, C.K. A symmetric image encryption scheme based on 3D chaotic cat maps. Chaos Solitons Fractals 21, 749-761 (2004).

[37] Belazi, A., ElLatif, A. & Belghith, S. A novel image encryption scheme based on substitution-permutation network and chaos.Signal Process. 128, 155-170 (2016).

[38] Song, C.Y., Qiao, Y.L. & Zhang, X.Z. An image encryption scheme based on new spatiotemporal chaos. Optik 124, 3329-3334(2013).

[39] Hua, Z. & Zhou, Y. Image encryption using 2D Logistic-adjusted-Sine map. In. Sci. 339, 237-253 (2016).

[40] Shannon CE. A mathematical theory of communication. *Bell System Technical Journal*. 1948;27:379–423, 623–656.