Parallel Hill Cipher Encryption Algorithm

Mais Haj Qasem
Computer Science Department University of Jordan
Amman, Jordan

Mohammad Qatawneh
Computer Science Department University of Jordan
Amman, Jordan

ABSTRACT
Cryptography is the discipline of encoding and decoding messages. Cryptography is used frequently in people's daily lives to keep sensitive information, such as credit card information, safe. Many everyday activities can be easily monitored by unintended third parties via Internet. Hill cipher is a classic cryptography based on linear algebra that is simply a linear transformation represented by a matrix. The encoding and decoding process in Hill cipher involves matrix multiplication, which is potentially time consuming, making it one of the most well-studied problems in this field. In this paper, we implement the message passing interface (MPI) and MapReduce methods to demonstrate their effectiveness in expediting Hill cipher algorithm in parallel algorithms on a multi-core system. Simulation results show that the efficiency rates of MPI and MapReduce are 93.71 % and 53.43 respectively, with a multi-core processor on the large file size, indicating better performances compared with sequential methods.

Keywords
Cryptography, Hadoop, Hill Cipher, MPI, MapReduce, Matrix Multiplication.

1. INTRODUCTION
Cryptography is the encoding and decoding secret messages into unreadable form to ensure privacy by keeping information hidden from third parties. Recently, mathematicians and research scientists have worked on cryptography to find the best and suitable algorithm to securely store and transfer sensitive information over the Internet, which can be easily monitored by unintended third parties [6], [17]. Cryptography has been proven a critical factor of success in war and business.

Hill cipher is a classic cryptography that is technically a polygraphic substitution cipher. Hill cipher is one of the first practical applications of linear algebra to polygraphic ciphers. Creator Lester Hill first described ciphers in 1929 in The American Mathematical Monthly [11], and he wrote another article about them in 1931[12].

Hill cipher acts on groups of letters, where plaintext is divided into groups of letters of a fixed size, and each group is transformed to a different group of letters. This transformation is accomplished using matrix multiplication, which is involved in encoding and decoding. A large amount of information is sent over the Internet every second, and one type of information is extremely difficult to break, namely, that which uses matrix to encode a message. In Hill cipher, the first matrix is called the encoding matrix and its inverse is called the decoding matrix.

Using matrix multiplication focuses on computational problems that should be investigated thoroughly to enhance the efficiency of the implemented algorithms. Hence, several parallel and distributed systems for matrix multiplication methods have been proposed over the years to reduce cost and time of matrix multiplication over multiple processors [5], [16],[24].

Parallel and distributed computing systems are high-performance computing systems that spread out a single application over many multi-core and multi-processor computers to rapidly complete the task [25],[26],[27],[29]. Parallel and distributed computing systems divide large problems into smaller sub-problems and assign each of them to different processors in a typically distributed system running concurrently in parallel [28]. MapReduce [19] and message passing interface (MPI) are among these computing systems, which will be discussed in the following section.

In this study, we applied Hill cipher encryption algorithm on different sizes of files by using efficient MapReduce with an optimized mapper set produced by [14] and MPI library. We used this method to demonstrate the performance of Hill cipher encryption algorithm by using parallel computing and compared it with sequential methods.

This paper is organized as follows. Section 2 reviews works that are closely related with using the matrix multiplication in many applications. Section 3 presents hill cipher implementation. Section 4 presents all methods used in this work. Section 5 presents the experimental results. Section 6 gives the conclusion.

2. RELATED WORK
Mathematicians and research scientists have found many matrix algorithms. The advent of personal and large-scale computers increased the use of matrices in a wide variety of applications, such as economics, engineering, statistics, and other sciences.

Traditional sequential algorithms for matrix multiplication consume considerable space and time. Fox [10], and Cannon [4], algorithms have been proposed for parallelizing matrix multiplication to enhance its efficiency. These approaches balance inter-process communication, dependencies, and parallelism level to maximize efficiency. Parallel matrix multiplication relies on the independence of multiplication, which includes multiple independent element-to-element multiplications and multiple aggregations of independent multiplication results.

Zhang et al. [23] presented an outsourcing computation schema in an amortized model for matrix multiplication of two arbitrary matrices that meet the requirements for both security and efficiency. They compared their scheme functionalities with existing works, such as Fiore's [7], Li's [16], and Jia's schema [11]. Zhang et al. [23] proved that their schema is more efficient in terms of functionality as well as computation, storage, and communication overhead.

Kumar et al. [15] proposed a privacy-preserving, verifiable, and efficient algorithm for matrix multiplication in outsourcing paradigm to solve the lack of computing resources, where the client with a large dataset can perform matrix multiplication using cloud server. Kumar et al. [13] evaluated their algorithm on security, efficiency, and variability parameters. With high efficiency and practical usability, their algorithm can mostly replace costly cryptographic operations and securely solve matrix multiplication algorithm.

Acharya et al. [1] proposed a novel advanced Hill (AdvHill) encryption technique to encrypt an image using a technique different from the conventional Hill cipher. This fast encryption scheme overcomes problems of encrypting the images with homogeneous background. Acharya et al. concluded that their proposed AdvHill algorithm is more secure to brute force attacks and quite reliable and robust after a comparative study of the proposed encryption and the existing schemes was done.

Panigrahy et al. [21] proposed an efficient method of generating self-invertible matrix for Hill cipher algorithm for use in image encryption. It is not only limited to this area but can also be widely applied in other information security fields, such as video encryption. They concluded that these methods eliminate the computational complexity as inverse of the matrix is not required, whereas decrypting in Hill cipher involves finding the inverse of the matrix in decryption

AL-Laham [20] enhanced technique of color image encryption-decryption based on random matrix key encoding is proposed, which utilizes matrix multiplication and inverse matrices. They concluded that his proposed technique rapidly increases the image transmission security and enhances the encryption-decryption process by eliminating the mean square error and maximizing the speed of the encryption decryption process.

3. HILL CIPHER IMPLEMENTATION

Hill ciphers apply matrices to cryptography. Ciphers are methods for transforming a secret message called plaintext into a particular form so that only those for whom it is intended and know the key can read and process it. In a cipher, the key transforms the plaintext letters into other characters known as the cipher text. The secret rule, that is, the inverse key, is required to reverse the transformation to recover the original message. Using the key to transform plaintext into cipher text is to encipher the plaintext. Using the inverse key to transform the cipher text back into plaintext is to decipher the cipher text. We need to understand modular arithmetic and multiply invert matrices to understand Hill ciphers.

A common way to send coded messages is to assign numerical values from 1–26 to the alphabet, as shown in Figure 1 below, and send the message as a string of integers. Codes such as these are easily broken using an analysis of the frequency of numbers that appear in the coded messages.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Fig 1: Numerical Values to The Alphabet

The encoder is a matrix and the decoder is its inverse. On the sender side, A is the encoding matrix, B is the message matrix, and C is the encrypted matrix. The sizes of A and B must be consistent and will determine the size of C. Mathematically, the operation is

$$\boldsymbol{mod\ 26\ (AB) = C}$$

On the receiver side, the intended user who wants to recover the original message B must have C and know A. Thus, this situation would be the same as solving the matrix equation for B by multiplying both sides of the equation on the left by A^{-1}. The operation is

$$\boldsymbol{B = A^{-1}C}$$

The following step shows an example of matrix multiplication encryption for plaintext message "hello mohamad" using the keyword "alphabet" and a 3 x 3 matrix.

3.1 Encryption

Step-1: Convert the characters of the key to integers between 1 and 26 using the figure above and fill the matrix. If the keyword is longer than the 9 letters needed, only take only the first 9 letters. Conversely, if it is shorter, fill it with the alphabet in order.

0	**11**	**15**	**7**	**0**	**1**	**4**	**19**
A	**L**	**P**	**H**	**A**	**B**	**E**	**T**

$$\begin{Bmatrix} 0 & 11 & 15 \\ 7 & 0 & 1 \\ 4 & 19 & 0 \end{Bmatrix}$$

Step-2: Convert the characters of the plaintext to integers and fill the matrix with the same size of key row length.

7	**4**	**11**	**11**	**14**	**12**	**14**	**7**	**0**	**12**	**0**	**3**
H	**E**	**L**	**L**	**O**	**M**	**O**	**H**	**A**	**M**	**A**	**D**

$$\begin{Bmatrix} 7 & 4 & 11 & 11 \\ 14 & 12 & 14 & 7 \\ 0 & 12 & 0 & 3 \end{Bmatrix}$$

Step-3: Encrypt the message by multiplying the message matrix by the key matrix to perform the matrix multiplication and obtain the encryption matrix as follows:

$$\begin{Bmatrix} 0 & 11 & 15 \\ 7 & 0 & 1 \\ 4 & 19 & 0 \end{Bmatrix} x \begin{Bmatrix} 7 & 4 & 11 & 11 \\ 14 & 12 & 14 & 7 \\ 0 & 12 & 0 & 3 \end{Bmatrix} =$$

$$\begin{Bmatrix} 154 & 312 & 154 & 122 \\ 49 & 40 & 77 & 80 \\ 294 & 244 & 310 & 177 \end{Bmatrix}$$

$$\begin{Bmatrix} 154 & 312 & 154 & 122 \\ 49 & 40 & 77 & 80 \\ 294 & 244 & 310 & 177 \end{Bmatrix} mod\ 26 = \begin{Bmatrix} 24 & 0 & 24 & 18 \\ 23 & 4 & 25 & 2 \\ 8 & 10 & 24 & 21 \end{Bmatrix}$$

Encrypted message will now be

24	**0**	**24**	**18**	**23**	**4**	**25**	**2**	**8**	**10**	**24**	**21**
Y	**A**	**Y**	**S**	**X**	**E**	**Z**	**C**	**I**	**K**	**Y**	**V**

3.2 Decryption

We will focus on finding the inverse key matrix that is not an easy task, because the majority of the process is the same as encryption. We perform the calculation below equation, where A is the key matrix, D is the determinant of the key

matrix, and $adj(A)$ is the adjugate matrix of A to find the inverse of the key matrix.

$$\boldsymbol{A^{-1} = D^{-1} \times adj\,(A)}$$

Step-1: Find determinant of the key

The determinant is a number that relates directly to the entries of the matrix. For our 3 × 3 matrix example, it is found by multiplying the top left entry by the determinant of the 2 × 2 matrix formed by the entries that are not in the same row or column as that entry. Similar steps are done with the other two elements in the top row, and the middle value is subtracted from the sum of the other two as shown below.

$$\begin{Bmatrix}0 & 11 & 15\\7 & 0 & 1\\4 & 19 & 0\end{Bmatrix} = 0\begin{vmatrix}0 & 1\\19 & 1\end{vmatrix} - 11\begin{vmatrix}7 & 1\\4 & 0\end{vmatrix} + 11\begin{vmatrix}7 & 0\\4 & 9\end{vmatrix}$$
$$= 2039 = 11\ mod\ 26$$

We must now find the multiplicative inverse of the determinant working mod 26, that is, the number between 1 and 25 that gives an answer of 1 when multiplied by the determinant. In this case, we are looking for the number that we need to multiply 11 by to get an answer of 1 mod 26.

$$11 \times x = 1\ mod\ 26$$
$$dd^{-1} = 1\ mod\ 26$$
$$11 \times 19 = 209 = 1\ mod\ 26$$

Step-2: Find the adjugate matrix

$$adj\begin{Bmatrix}0 & 11 & 15\\7 & 0 & 1\\4 & 19 & 0\end{Bmatrix}$$
$$= \begin{pmatrix} + \begin{vmatrix}0 & 1\\19 & 0\end{vmatrix} - \begin{vmatrix}11 & 15\\19 & 0\end{vmatrix} + \begin{vmatrix}11 & 15\\0 & 1\end{vmatrix} - \begin{vmatrix}7 & 1\\4 & 0\end{vmatrix} + \\ \begin{vmatrix}0 & 15\\4 & 0\end{vmatrix} - \begin{vmatrix}0 & 15\\7 & 1\end{vmatrix} + \begin{vmatrix}7 & 0\\4 & 19\end{vmatrix} \begin{vmatrix}0 & 11\\4 & 19\end{vmatrix} \begin{vmatrix}0 & 1\\7 & 0\end{vmatrix} \end{pmatrix}$$

$$= \begin{pmatrix}-19 & 285 & 11\\4 & -60 & 105\\133 & 44 & -77\end{pmatrix} mod\ 26 = \begin{pmatrix}7 & 25 & 11\\4 & 18 & 1\\3 & 18 & 1\end{pmatrix}$$

Step-3: Multiply the multiplicative inverse of the determinant by the adjugate matrix.

We now multiply the inverse determinant (that was 19 in our case) from Step 1 by each of the elements of the adjugate matrix from Step 2 to obtain the inverse key matrix. Next, we take each of these answers' mod 26.

$$19 \quad \times \quad \begin{Bmatrix}7 & 25 & 11\\4 & 18 & 1\\3 & 18 & 1\end{Bmatrix} = \begin{Bmatrix}133 & 475 & 209\\76 & 342 & 19\\57 & 342 & 19\end{Bmatrix} mod\ 26 = \begin{Bmatrix}3 & 7 & 1\\24 & 4 & 19\\5 & 4 & 19\end{Bmatrix}$$

So, if $\begin{Bmatrix}0 & 11 & 15\\7 & 0 & 1\\4 & 19 & 0\end{Bmatrix}$ $then, k^{-1} = \begin{Bmatrix}3 & 7 & 1\\24 & 4 & 19\\5 & 4 & 19\end{Bmatrix}$

Finally, we have the inverse key matrix, and we multiply this by the encrypted message received to get the original message as follows:

$$\begin{Bmatrix}3 & 7 & 1\\24 & 4 & 19\\5 & 4 & 19\end{Bmatrix} \times \begin{Bmatrix}24 & 0 & 24 & 18\\23 & 4 & 25 & 2\\8 & 10 & 24 & 21\end{Bmatrix} = \begin{Bmatrix}7 & 4 & 11 & 11\\14 & 12 & 14 & 7\\0 & 12 & 0 & 3\end{Bmatrix}$$

Encrypted message will be:

7	**4**	**11**	**11**	**14**	**12**	**14**	**7**	**0**	**12**	**0**	**3**
H	**E**	**L**	**L**	**O**	**M**	**O**	**H**	**A**	**M**	**A**	**D**

4. METHODS

MapReduce [19] and MPI are parallel and distributed computing systems with high-performance computing that spread out a single application over many multi-core and multi-processor computers to rapidly complete the task. MapReduce [19] and MPI divide large problems into smaller sub-problems and assign each of them to different processors in a typically distributed system running concurrently in parallel.

- **First Method:** Sequential is accessed code by a single thread. This means that a single thread can only do code in a specific order, hence it being sequential.
- **Second Method:** MPI is a library of routines that can be used to create parallel programs in C, C++, and Fortran77 using commonly-available operating system services to create parallel processes and exchange information among these processes, as shown in Figure 2. The design process of MPI includes vendors (such as IBM, Intel, TMC, Cray, and Convex), parallel library authors (involved in the development of PVM, and Linda), and application specialists. The final version for the draft standard became available in May of 1994 [2].

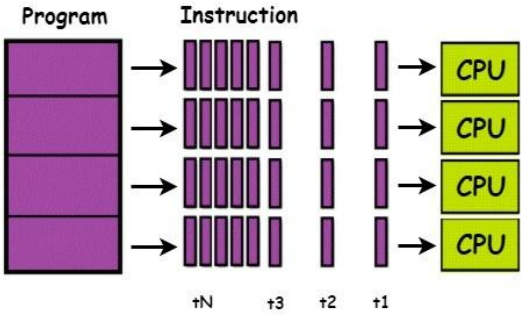


Fig 2: MPI Parallel Processes

MPI is a standardized means of exchanging messages among multiple computers running a parallel program across a distributed memory to improve scalability, performance, multi-core and cluster support, and interoperation with other applications. These programs cannot use any MPI communication routine. The two basic routines are MPI_Send, to send a message to another process, and MPI_Recv, to receive a message from another process.

We run MPI code in IMAN1, Jordan's first and fastest high-performance Computing resource, funded by JAEC and SESAME. It is available for use by academia and industry in Jordan and the region. In our project, we worked in a Zaina server, an Intel Xeon-based computing cluster with 1G Ethernet interconnection as shown in Table 1. The cluster is mainly used for code development, code porting, and synchrotron radiation application purposes. In addition, this cluster is composed of two Dell PowerEdge R710 and five HP ProLiant DL140 G3 server's.

Table 1. Zaina Technical Details

Properties	Details
Server	7 Servers (Two Dell PowerEdge R710 and five HP ProLiant DL140 G3)
CPU per server	Dell (2 X 8 cores Intel Xeon) HP (2 X 4 cores Intel Xeon)
RAM per server	Dell (16 GB) HP (6 GB)
Total storage (TB)	1 TB NFS Share
OS	Scientific Linux 6.4

- **Third Method:** MapReduce is an algorithm design and processing paradigm proposed by Dean and Ghemawat in 2004 [7]. MapReduce enables efficient parallel and distributed computing and consists of two serial tasks, namely, map and reduce. Each serial task is implemented with several parallel subtasks. Specific MapReduce paradigms include MapReduce with expectation maximization for text filtering [4], MapReduce with K-means for remote-sensing image clustering [18], and MapReduce with decision tree for classification [9]. MapReduce has also been used for job scheduling [22] and real-time systems.

Traditional parallel-based matrix multiplication has been recently replaced with MapReduce, a parallel and distributed framework for large-scale data [3]. Typical MapReduce-based matrix multiplication requires two MapReduce jobs:

- **The first job:** A pair of elements is created for multiplication by combining input arrays together during map task. The reduce task of this job is inactive at this point.
- **The second job:** The map task independently implements the multiplication operations on each pair of elements. The reduce job aggregates the results corresponding to each output element.

Hadoop is a Java open-source platform used for developing MapReduce applications. Google developed this platform [8]. Figure 3 illustrates the Hadoop architecture.

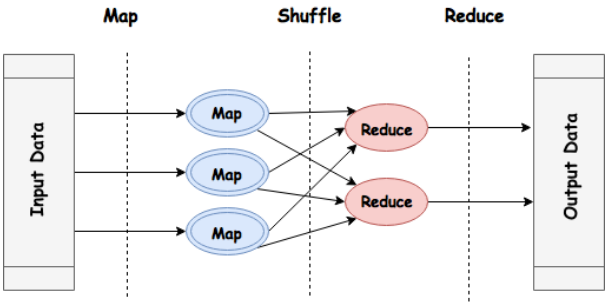


Fig 3: Hadoop MapReduce Architecture

In this study, we used MapReduce-based matrix multiplication proposed by [14], which reduces both time and memory utilization compared with existing schemas [14]. In the proposed technique, matrix multiplication is implemented as an element-to-block schema, as illustrated in Figure 4.

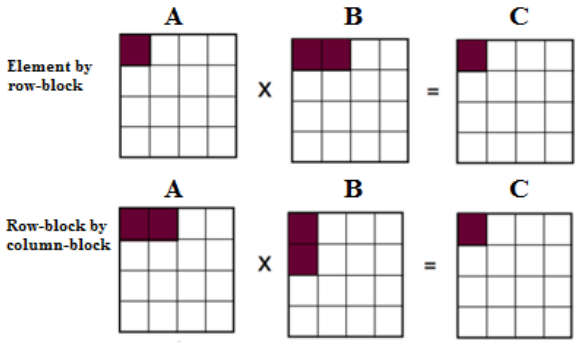


Fig 4: Efficient MapReduce Matrix Multiplication techniques

5. EXPERIMENTS AND RESULTS

In our research, we used different plaintext sizes and two key size, then we ran them in parallel using Hill cipher algorithm coded by MPI and MapReduce. We then compared their efficiencies and time performances in a large plaintext size. The results of the tested methods are discussed below.

- **MPI Result:**

Table 2 presents the results for different numbers of cores. The plaintext which is less than 1.80 KB does not need more than 4 cores, and 100×100 key size, because of the small problem size, whereas the plaintext within the range of 1.80 KB to 7.21 KB does not need more than 8 cores for its problem because it is inefficient and takes more time, where plaintext with size 1.80 KB is fair enough to use 100×100 key size, and 200×200 key size for plaintext 7.21 KB In addition, we concluded that when testing a large plaintext more than 7.21 KB, increasing the number of cores to 32, and using 200×200 key size is more effective and efficient because of the large problem size that needs more parallelism.

Table 2. MPI Run Time Results

Plaintext Size	Core 2	Core 4	Core 8	Core 16	Core 32
Key Size 100 × 100					
Encryption Time					
461 KB	1.88 sec	1.80 sec	1.84 sec	2.00 sec	2.35 sec
1.80 MB	2.59 sec	1.91 sec	1.80 sec	1.81 sec	2.22 sec
7.21 MB	9.09 sec	4.16 sec	3.34 sec	2.98 sec	3.27 sec
28.8 MB	86.21 sec	30.89 sec	23.95 sec	17.54 sec	16.46 sec
Decryption Time					
461 KB	1.98 sec	1.90 sec	1.94 sec	2.10 sec	2.40 sec
1.80 MB	2.60 sec	1.92 sec	1.81 sec	1.82 sec	2.23 sec
7.21 MB	9.29 sec	4.36 sec	3.54 sec	3.18 sec	3.47 sec
28.8 MB	86.22 sec	30.90 sec	23.96 sec	17.55 sec	16.47 sec

Total Time					
461 KB	3.86 sec	3.70 sec	3.78 sec	4.10 sec	4.75 sec
1.80 MB	5.19 sec	3.83 sec	3.61 sec	3.63 sec	4.45 sec
7.21 MB	18.38 sec	8.52 sec	6.88 sec	6.16 sec	6.74 sec
28.8 MB	172.43 sec	61.79 sec	47.91 sec	35.09 sec	32.93 sec
Key Size 200 × 200					
Encryption Time					
461 KB	2.01 sec	1.92 sec	1.96 sec	2.13 sec	2.51 sec
1.80 MB	2.02 sec	1.49 sec	1.46 sec	1.41 sec	1.74 sec
7.21 MB	10.52 sec	4.81 sec	3.86 sec	3.45 sec	3.79 sec
28.8 MB	77.27 sec	27.69 sec	21.47 sec	15.72 sec	14.75 sec
Decryption Time					
461 KB	2.11 sec	2.02 sec	2.06 sec	2.23 sec	5.02 sec
1.80 MB	2.03 sec	1.50 sec	1.47 sec	1.42 sec	1.75 sec
7.21 MB	10.72 sec	5.01 sec	4.06 sec	3.65 sec	3.99 sec
28.8 MB	77.28 sec	27.70 sec	21.48 sec	15.73 sec	14.76 sec
Total Time					
461 KB	4.12 sec	3.94 sec	4.02 sec	4.36 sec	7.53 sec
1.80 MB	4.05 sec	2.99 sec	2.93 sec	2.83 sec	3.49 sec
7.21 MB	21.24 sec	9.82 sec	7.92 sec	7.10 sec	7.78 sec
28.8 MB	154.55 sec	55.39 sec	42.95 sec	31.45 sec	29.51 sec

The speedup is the ratio between sequential and parallel time. The speedup for different numbers of core on different plaintext sizes with different key size are illustrated in Figure 5 for key size 100×100 encryption, and Figure 6 for key size 200×200 encryption, whereas Figure 7 for key size 200×200 decryption, and Figure 8 for key size 200×200 decryption. The results show that MPI achieves the best speedup values.

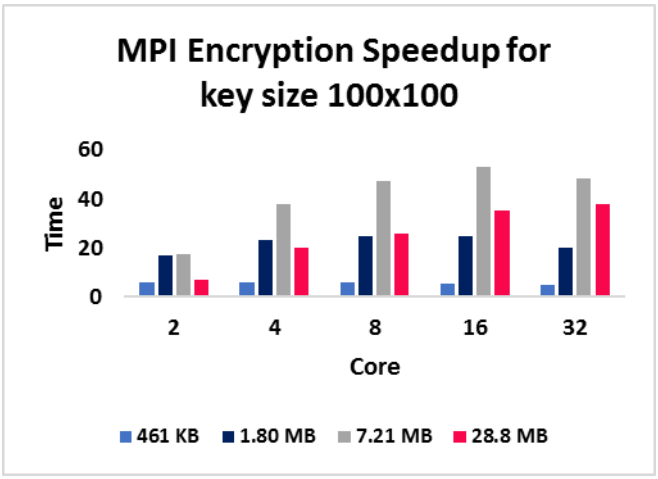
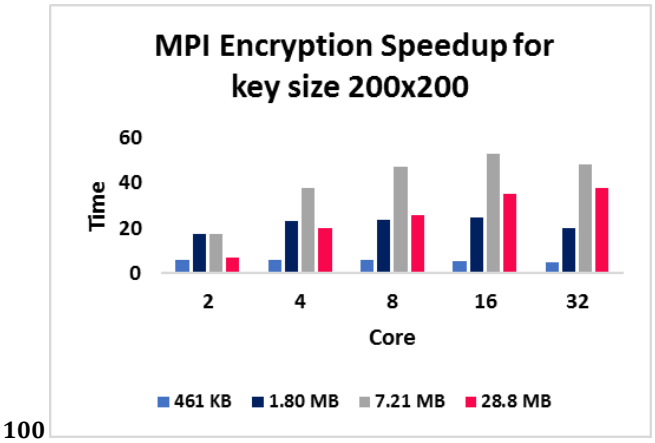


Fig 5: MPI Encryption Speedup plotting for Key Size 100 ×



100

Fig 6: MPI Encryption Speedup plotting for Key Size 200 × 200

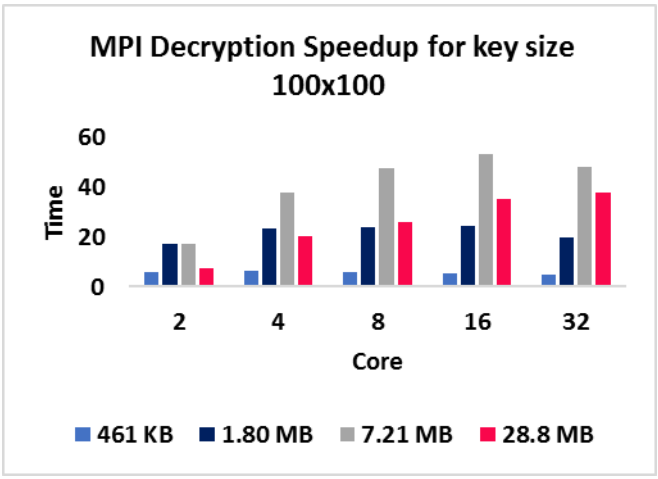


Fig 7: MPI Decryption Speedup plotting for Key Size 100 × 100

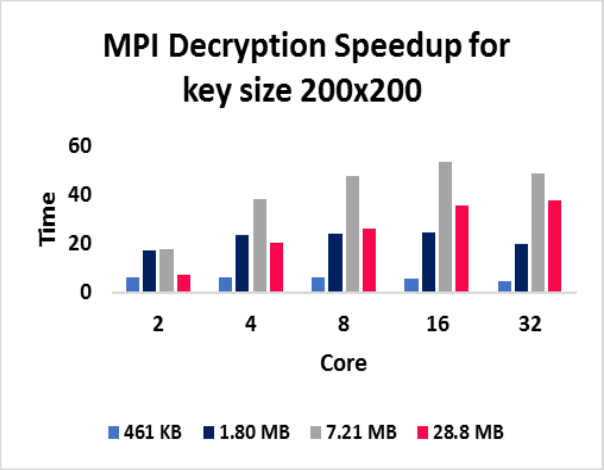


Fig. 8. MPI Decryption Speedup plotting for Key Size 200 × 200

- **MapReduce Result**

The MapReduce results of the Hill cipher using Hadoop for inputs with various plaintext sizes and two key size are presented. The running time was cut down in the proposed schemes, as the sorting process in the shuffling process was reduced. As the plaintext size grows, the stability of the proposed scheme is almost linear. Table 3 provides the results.

Table 3. MapReduce Run Time Result

Plaintext Size	Encryption Time	Decryption Time	Total Time
Key Size 100 × 100			
461 KB	6.1 sec	6.20 sec	12.3 sec
1.80 MB	15.22 sec	15.23 sec	30.445 sec
7.21 MB	114.41 sec	114.61 sec	229.02 sec
28.8 MB	155.11 sec	155.12 sec	310.225 sec
Key Size 200 × 200			
461 KB	6.5 sec	6.60 sec	13.1 sec
1.80 MB	11.89 sec	11.90 sec	23.785 sec
7.21 MB	132.32 sec	132.52 sec	264.84 sec
28.8 MB	139.02 sec	139.03 sec	278.045 sec

The speedup for different plaintext sizes with different key size are illustrated in Figure 9 for key size 100×100 encryption, and Figure 10 for key size 200×200 encryption, whereas Figure 11 for key size 200×200 decryption, and Figure 12 for key size 200×200 decryption. The results show that MapReduce achieves speedup values less than MPI, especially on large number of processors.

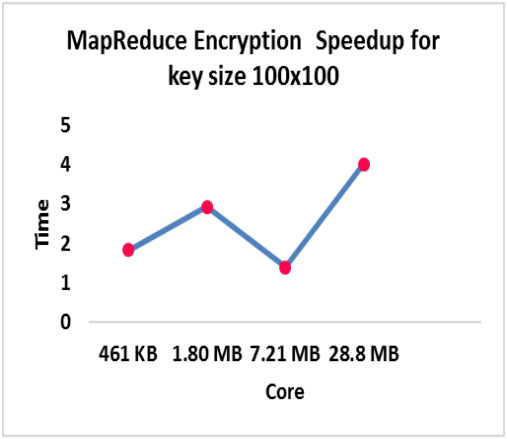


Fig. 9. MapReduce Encryption Speedup plotting for Key Size 100 × 100

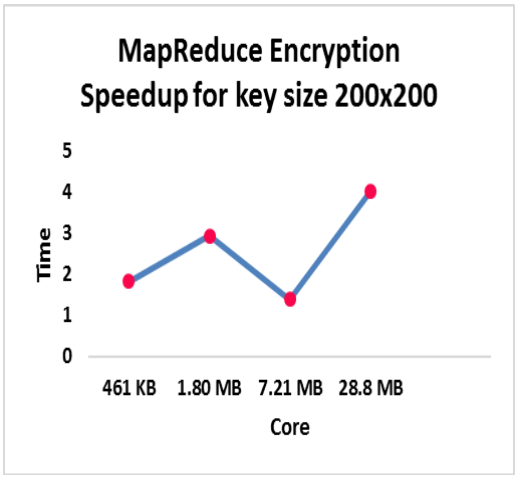


Fig. 10. MapReduce Encryption Speedup plotting for Key Size 200 × 200

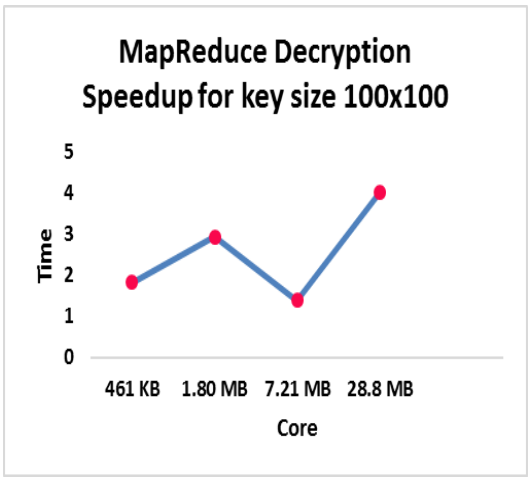


Fig. 11. MapReduce Decryption Speedup plotting for Key Size 100 × 100

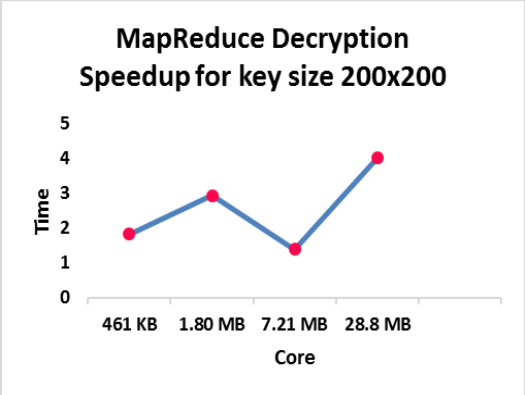


Fig. 12. MapReduce Decryption Speedup plotting for Key Size 200 × 200

- **Sequential Result**

The sequential results of Hill cipher were tested on various plaintext sizes with different key size. The algorithm is written in java and the experimental results are calculated on HP® core™ i7-5500U CPU @ 2.40GHz / 8 GB RAM. The results are given in Table 4.

Table 4. Sequential Run Time Results

Plaintext Size	Encryption Time	Decryption Time	Total Time
Key Size 100 × 100			
461 KB	11.16 sec	11.26 sec	22.42 sec
1.80 MB	44.52 sec	44.52 sec	89.04 sec
7.21 MB	158.32 sec	158.52 sec	316.84 sec
28.8 MB	621.38 sec	621.38 sec	1242.76 sec
Key Size 200 × 200			
461 KB	11.89 sec	11.99 sec	23.87 sec
1.80 MB	34.77 sec	34.77 sec	69.54 sec
7.21 MB	183.09 sec	183.29 sec	366.39 sec
28.8 MB	556.95 sec	556.95 sec	1113.90 sec

- **Comparison**

The comparison between MPI and MapReduce results are always faster and more efficient than sequential methods for the different plaintext size, as shown in the efficiency table 5 below. The MPI outperformed the MapReduce; thus, the research goal is achieved. Comparison are illustrated in Figure 13 for key size 100×100 encryption, and Figure 14 for key size 200×200 encryption, whereas Figure 15 for key size 200×200 decryption, and Figure 16 for key size 200×200 decryption.

Table 5. Time Efficiency Result

Plaintext Size	Encryption Efficiency	Decryption Efficiency
Key Size 100 × 100		
MPI		
461 KB	83.87%	83.12%
1.80 MB	95.96%	95.95%
7.21 MB	98.12%	97.99%
28.8 MB	97.35%	97.35%
MapReduce		
461 KB	45.34%	44.93%
1.80 MB	65.81%	65.80%
7.21 MB	27.74%	27.70%
28.8 MB	75.04%	75.04%
Key Size 200 × 200		
MPI		
461 KB	83.85%	83.15%
1.80 MB	95.94%	95.93%
7.21 MB	98.12%	98.01%
28.8 MB	97.35%	97.35%
MapReduce		
461 KB	45.34%	44.93%
1.80 MB	65.81%	65.80%
7.21 MB	27.74%	27.70%
28.8 MB	75.04%	75.04%

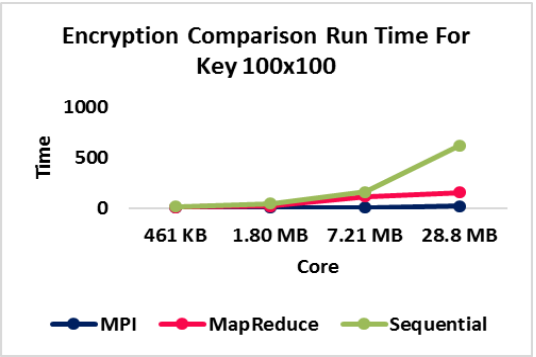


Fig. 13. Encryption Methods Compassion plotting for Key Size 100 × 100

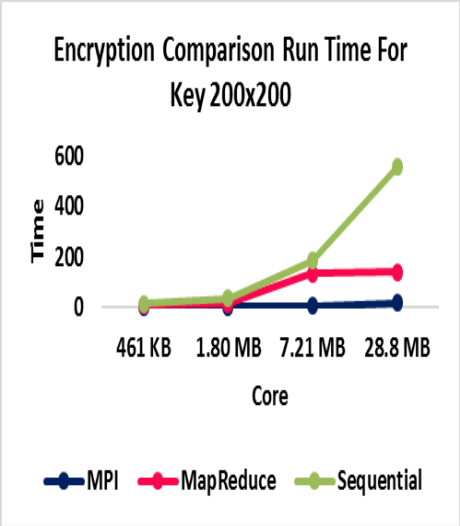


Fig. 14. Encryption Methods Compassion plotting for Key Size 200 × 200

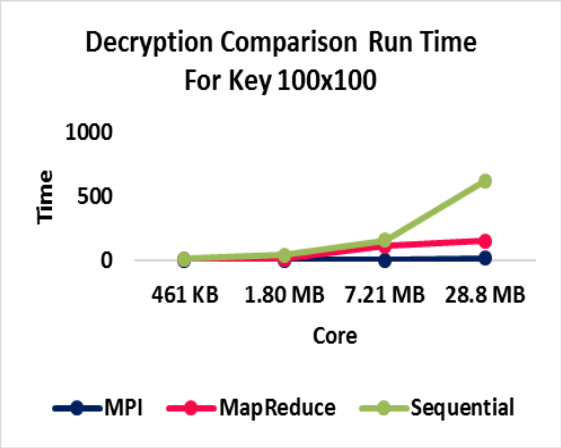


Fig. 15. Decryption Methods Compassion plotting for Key Size 100 × 100

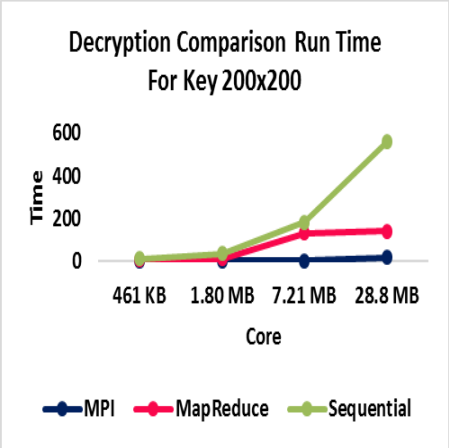


Fig. 16. Decryption Methods Compassion plotting for Key Size 200 × 200

6. CONCLUSION

Using the conducted experimental study as basis, MPI and MapReduce Hill cipher are always faster than the sequential methods, with 93.71 % and 53.43 % efficiency, respectively. Hence, parallel and distributed computing for hill cipher algorithm have been proposed to reduce the cost and time of matrix multiplication over multiple processors. MPI hill cipher is also more efficient than MapReduce hill cipher because its matrix size growth outperforms sequential methods.

10. REFERENCES

[1] charya, B., Panigrahy, S. K., Patra, S. K., & Panda, G. (2009). Image encryption using advanced hill cipher algorithm. International Journal of Recent Trends in Engineering, 1(1).

[2] Burns, G., Daoud, R., & Vaigl, J. (1994, June). LAM: An open cluster environment for MPI. In Proceedings of supercomputing symposium (Vol. 94, pp. 379-386).

[3] Catalyurek, U. V., & Aykanat, C. (1999). Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems, 10(7), 673- 693.

[4] Cannon, Lynn E. A Cellular Computer to Implement the Kalman Filter Algorithm. No. 603-Tl-0769. Montana State Univ Bozeman Engineering Research Labs, 1969.

[5] Coppersmith, D., & Winograd, S. (1987, January). Matrix multiplication via arithmetic progressions. In Proceedings of the nineteenth annual ACM symposium on Theory of computing (pp. 1-6). ACM.

[6] Chatterjee, D., Nath, J., Dasgupta, S., & Nath, A. (2011, June). A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm. In Communication Systems and Network Technologies (CSNT), 2011 International Conference on (pp. 89-94). IEEE.

[7] Dean, G. (2004). J. Dean, S. Ghemawat. Mapreduce: simplified data processing on large clusters, OSDI. USENIX (2004), 10.

[8] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: a flexible data processing tool." Communications of the ACM 53.1 (2010): 72-77.

[9] Dekel, Eliezer, David Nassimi, and Sartaj Sahni. "Parallel matrix and graph algorithms." SIAM Journal on computing 10.4 (1981): 657-675.

[10] Fox, Geoffrey C., Steve W. Otto, and Anthony JG Hey. "Matrix algorithms on a hypercube I: Matrix multiplication." Parallel computing 4.1 (1987): 17-31.

[11] Hill, L. S. (1931). Concerning certain linear transformation apparatus of cryptography. The American Mathematical Monthly, 38(3), 135-154.

[12] Hill, L. S. (1929). Cryptography in an algebraic alphabet. The American Mathematical Monthly, 36(6), 306-312.

[13] H. Li, S. Zhang, T. H. Luan, H. Ren, Y. Dai, and L. Zhou, "Enabling efficient publicly verifiable outsourcing computation for matrix multiplication," in Telecommunication Networks and Applications Conference (ITNAC), 2015 International. IEEE, 2015, pp. 44–50.

[14] Kadhum, M., Qasem, M. H., Sleit, A., & Sharieh, A. (2017, April). Efficient MapReduce Matrix Multiplication with Optimized Mapper Set. In Computer Science On-line Conference (pp. 186-196). Springer, Cham.

[15] Kumar, M., Meena, J., & Vardhan, M. (2017). Privacy preserving, verifiable and efficient outsourcing algorithm for matrix multiplication to a malicious cloud server. Cogent Engineering, (just-accepted), 1295783

[16] Liu, Xiufeng, Nadeem Iftikhar, and Xike Xie. "Survey of real-time processing systems for big data." Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, 2014.

[17] Lee, K. H., & Chiu, P. L. (2012). An extended visual cryptography algorithm for general access structures. ieee transactions on information forensics and security, 7(1), 219-229.

[18] Lv, Zhenhua, et al. "Parallel K-means clustering of remote sensing images based on MapReduce."

[19] Norstad, John. "A mapreduce algorithm for matrix multiplication." 2013-02-19]. http://www. norstad. org/matrix-multiply/index. html (2009).

[20] AL-Laham, M. M. (2015). Encryption-Decryption RGB Color Image Using Matrix Multiplication.

[21] Panigrahy, S. K., Acharya, B., & Jena, D. (2008). Image encryption using self-invertible key matrix of hill cipher algorithm.

[22] Zaharia, Matei, et al. "Job scheduling for multi-user mapreduce clusters." EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55 (2009).

[23] Zhang, S., Li, H., Jia, K., Dai, Y., & Zhao, L. (2016, December). Efficient Secure Outsourcing Computation of Matrix Multiplication in Cloud Computing. In Global Communications Conference (GLOBECOM), 2016 IEEE (pp. 1-6). IEEE.

[24] Azzam Sleit, Wesam AlMobaideen, Mohammad Qatawneh, Heba Saadeh."2008". Efficient Processing for Binary Submatrix Matching. American Journal of Applied Sciences 6 (1): 78-88, 2008, ISSN 1546-9239.

[25] Mohammad Qatawneh, Azzam Sleit, Wesam Almobaideen. "2009". Parallel Implementation of Polygon Clipping Using Transputer. American Journal of Applied Sciences 6 (2): 214-218, 2009. ISSN 1546-9239.

[26] Mohammad Qatawneh. "2011".Multilayer Hex-Cells: A New Class of Hex-Cell Interconnection Networks for Massively Parallel Systems. Int. J. Communications, Network and System Sciences, 2011, 4, 704-708.

[27] Mais Haj Qasem, Mohammad Qatawneh. "2017". Parallel Matrix Multiplication for Business Applications. Proceedings of the Computational Methods in Systems and Software. 2017, 24-36.

[28] Azzam Sleit, Wesam Almobaideen, Mohammad Qatawneh, Heba Saadeh. "2008". Efficient Processing for binary Submatrix matching. American Journal od Applied Science, 2017, 6(1): 78-88.

[29] Ola M. Surakhi, Mohammad Qatawneh, Hussein A. al Ofeishat. "2017". A parallel Genetic Algorithm