VOL. 14, NO. 11, JUNE 2019 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

ENHANCING PERFORMANCE OF AES ALGORITHM USING CONCURRENCY AND MULTITHREADING

Hasanain Ali Al Essa and Asmaa Shaker Ashoor
College of Education for Pure Science, University of Babylon, Iraq
E-Mail: asmaa_zaid218@yahoo.com

ABSTRACT

The Advanced Encryption Standard (AES) algorithm can encrypt and decrypt information and it's necessary to the security of the electronic information like the malware authors, unauthorized users and information leakage. However, AES is a complicated algorithm that imposes a major number of mathematical calculations to be done. Thus, a traditional execution lead to low throughput and consumption of CPU resources. In some application data flow needs faster rate of computation for encryption/decryption and traditional execution of AES algorithm may not be appropriate with fast transactions of electronic data. This paper focuses on increasing the performance time of AES algorithm to speed up the encryption/decryption procedure by using multicore processors efficiently. The AES algorithm has been executed and parallelized using C++17 Standard language programming and the results shown that our parallel design improve the throughput over the traditional approach.

Keywords: advanced encryption standard (AES), shared-memory multiprocessors, multithreading, parallelization.

1. INTRODUCTION

Multicore desktop computers, more than one core on a single chip, are now increasingly popular. Increasing computing power of these computers does not from execution a single task faster, but from executing multiple tasks in parallel.

In the past, developers, monitors their projects get faster with each new releasing of processors, without any labour on their part. But now, the projects must be developed to execute multiple tasks concurrently to take advantage of this increased computing power. Nowadays, Advanced Encryption Standard (AES) Algorithm [1] is the most popular executed by computers. AES is based on the block cipher Rijndael [2, 3] to encrypt and decrypt data for sending it through dangerous environment like web. But AES algorithm has many performance limitations such as memory requirement and execution time [4]. The AES algorithm is slow, although it's very secure, as it include dense mathematical computations. However, enhancing the performance can be important for sending and receiving messages through the internet. One of the best approaches used to enhance the performance of AES algorithms is to utilize parallel programming to solve the problem. The approaches of the parallel programming provide a cheap alternative to design a dedicated piece of hardware to enhance the performance and also it's much easier to scalability, especially when the design of parallel algorithm is well. Moreover, a multi-core processor nowadays supports most of the computer systems some has eight cores or more.

This paper focuses on increasing the performance time of AES algorithm to speed up the encryption/decryption procedure by using multicore processors efficiently. C++17 Standard [5] is one of the language programming that supported by multicore architectures to provide multithreaded shared memory parallelism and make the developers writing multithreaded code without relying on platform-specific extensions and these significant new features in the C++17 allow the developers writing portable parallel code with ensured behavior.

The result of the parallel AES algorithm will be evaluated by measuring the scalability and speedup features, moreover, our parallel algorithm will be able to indicate the number of threads that can truly run concurrently for a given execution at runtime.

This paper is organized as follows: Second section shows of the AES Rijndael design. Third section describes the proposed of AES parallel algorithm. The main results of execution of AES parallel algorithm are presented and discussed in the fourth section. Finally, Fifth section presents the conclusion of this work.

2. RELATED WORK

There are master papers published on the amendment of AES as it is the standard for encrypting.

- 2017 Ritambhara; Alka Gupta; Manjit Jaiswal, suggested An enhanced AES algorithm using the cascading method on 400 bits key size used in enhancing the safety of next generation internet of things (IOT). We take symmetrical block encryption using 200 bit sequential plain text and 400 bit key. The algorithm uses 400 bit key which is divided into 2 parts of 200-200 bits providing different keys for each round of cascading which will increase the security. Here AES-AES cascading is done, which consist of 5 rounds instead of 10, thus it will omit the mix column twice from the original AES Encryption technically, it takes half time than a simple AES algorithm to encrypt the block data [6].

-2017 C.P. Pramod, M. Jaiswal, used an advanced AES algorithm using swap and 400 bit data block with flexible S-Box in Cloud Computing. New Algorithm uses 400 bit block encryption scheme and a key dependent rotated S-Box which varies according to the 200 bit key provided by the user. The fixed S-box allows hackers to study S-box and find weaker points, whereas using key-dependent S-Box; it is harder for an attacker to do any offline analysis of an attack on one particular set of

S-boxes. This helps to store data in cloud securely and to transfer data without any obstruction or modification in Cloud Computing and big data [7].

- 2017 P. Dixit, J. Zalke, S. Adman, prepared the FPGA technology has ability to design, efficient and high performance embedded systems which are based on soft core processors, embedded memories and the IP core. In the design of such

FPGA based systems with soft-core processor, to gain and improve the performance, the Hardware-Software Co-design is generally used. Thus the proposed work is based on the study of effect of Hardware-Software Co-design on performance parameter. An AES cryptography algorithm is taken as an application, which is implemented on a soft core processor using Microblaze. AES (Advanced Encryption Standard) is a viable encryption algorithm used in application like Internet applications, RFID devices etc. to provide digital security. Cryptography plays an important role for end-toend safe communication [8].

-2012 Angelo Barnes, Ryan Fernando, Kasuni Mettananda and Roshan Ragel, We propsed a sequential program that implements the AES algorithm and convert the same to run on multicore architectures with minimum effort.

We implement two different parallel programmes, one with the fork system call in Linux and the other with the pthreads, the POSIX standard for threads. Later, we ran both the versions of the parallel programs on different multicore architectures and compared and analysed the throughputs between the implementations and among different architectures. The pthreads implementation outperformed in all the experiments we conducted and the best throughput obtained is around 7Gbps on a 32-core processor (the largest number of cores we had) with the pthreads implementation [9].

3. A- OVERVIEW OF AES RIJNDAEL DESIGN

Advanced Encryption Standard (AES) is a new block cipher by Rijndael [2, 3] and it has been dependent in a large number of encryption/ decryption modules worldwide since 1977. The finite field operation is the core of AES algorithm [10].

The majority of the AES algorithm is consists of four functions Add Round Key, Sub Byte, Mix Columns and Shift Rows, as shown in Figure 1, these functions execute iteratively as ten, twelve and fourteen rounds depending on the key length 128, 192 and 256 bits respectively. In order to reduce the conditions to switch between the key lengths, we focus on key length of 128 bits (16 bytes) and it's suitable for most purposes. The implementation of 192 or 256 bit key lengths can surely perform by changing the corresponding constants. The State matrix is a 4×4 bytes square matrix represent the plaintext block of 128 bits.

3.1 Encryption

The AES encryption process responsible for doing the actual encryption of plaintext consist of the block size of 16 bytes into the output cipher text vector.

To transform the plain text to cipher text, the AES algorithm performs ten rounds of substitution and permutation this is in the case of 128 bit key. The operations of Sub Byte, Shift Row, Mix Column, Add Round Key are used in the first 9 rounds of encryption and the algorithm is skipped the Mix column operation for the final round to achieve the encryption. Plaintext is often reshaped from 16 byte vector into 4 * 4 byte array and it is termed as ‘‘state”.

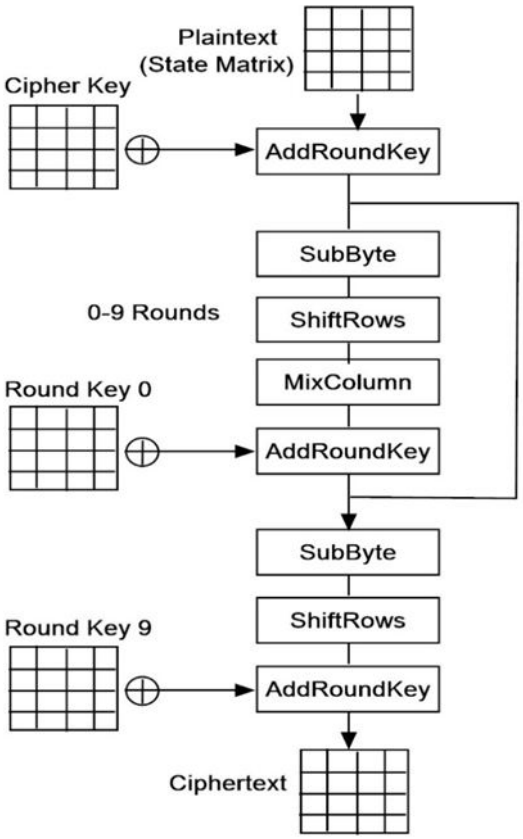

Figure-1. Block diagram for the AES algorithm.

From the Figure-1, we can see different operations added through an iterative process to "state" to perform the cipher text. Below is the brief discussion on the steps, shown in Figure-1.

Add Round Key performs a bitwise XOR of the current state array and the current round key array [11]. In our work, we replace bitwise XOR operation with addition lock up table this make our implementation faster and portable.

Sub Byte performs substitution each byte of the state array with a value of s-box. The s-box is applied in just one line code as shown in the snippet code below.

```
state [row][col] = s_box[state[row][col]];
```

The multiplicative inverse in the finite field GF(2^8) is used to create s-box table [11].

Shift Rows cyclically performs shifts the rows of the state array to the left, as shown in Figure-2.

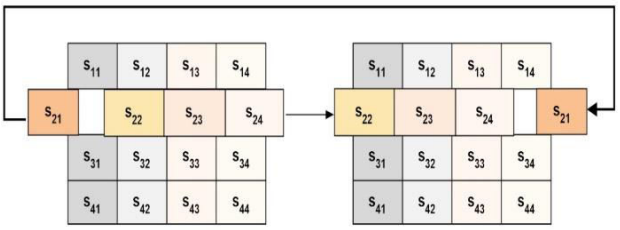

Figure-2. Shifting rows operation.

Mix Column this function is applied in a column-by-column manner to the state matrix. Each column in state matrix is treated as a four polynomial in GF(2^8) and then multiplied by the constant polynomial value generate by the equation 1 and modulo (x^4+1). The operation shown in Figure-3.

$$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \quad (1)$$

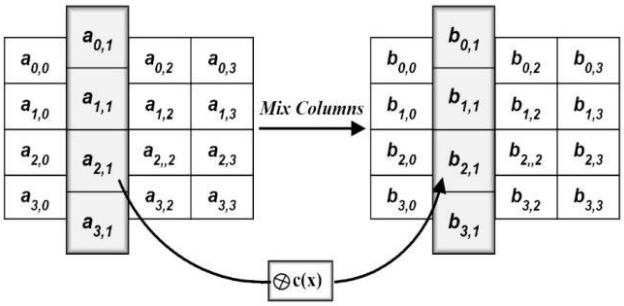

Figure-3. The Mix Columns operation.

3.2 Decryption

A decryption is the process to reverse the transformations of the encryption process. In other words, it's the process to decrypt cipher text back into plaintext. There is the similarity between encryption/decryption functions, as in encryption, but in the opposite order it performs nine rounds of Shift Row, Sub Byte and Mix Columns and a final tenth round (again with a missing Mix Columns) to end up with the reshaped plaintext.

Add Round Key the last round key that has been used in cipher be used in this step. As a consequence, state array directly added to the last round key using an addition table that has been created in real time rather than using bitwise XOR operation as mentioned before.

Inverse Shift Rows the effect of the inverse shift rows function is to reverse the effect of the shift rows in the encryption process by shifting right (back) all rows of the state array as shown in Figure-4.

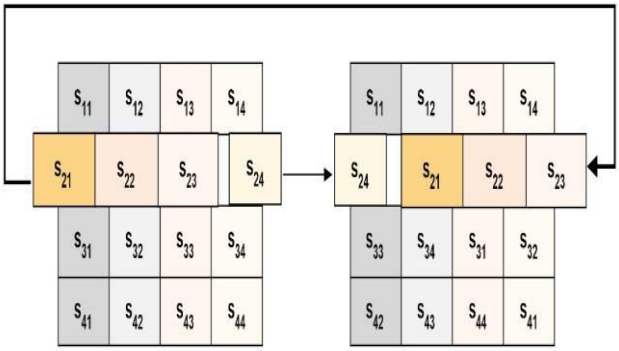

Figure-4. Inverse shift rows.

Inverse sub bytes the Sub Bytes employs inverse s_box table [11] for reversing the substitution in the cipher.

Mix columns the Mix Columns is also employ the inverse polynomial array [12] to compensate the corresponding call in cipher which is define by the following equation:

$$a(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\} \quad (2)$$

4. PROPOSED AES ALGORITHM IMPLEMENTATION

In this section, the AES algorithm will be developed using C++17 concurrency and the parallel version will be implemented directly using multithreading tasks. The analysis of the sequential algorithm will be provided with problem identification to provide solutions in the parallel mode.

4.1 Parallel design

The operation modes determine the role of encrypted [13, 14]. However, not all these modes can be parallelized or can be parallelized in encryption but not in decryption. In our work we will implement AES algorithm in parallel using ECB mode [15] because this mode can paralyze in encryption and decryption and it's the most understandable. In this mode, data are broken into independent blocks which are encrypted each block is encoded independently of the other blocks as Eq. (3), Eq. (4) respectively.

$$C_i = E_k(P_i) \quad (3)$$

$$P_i = D_k(C_i) \quad (4)$$

Where ***E*** is the Encryption, ***D*** Decryption, ***Pi*** plaintext block *i*, ***Ci*** Cipher text block *i*, and ***k*** Secret key. From the figure 5 we show that the mode produces the same cipher text for the same input text.

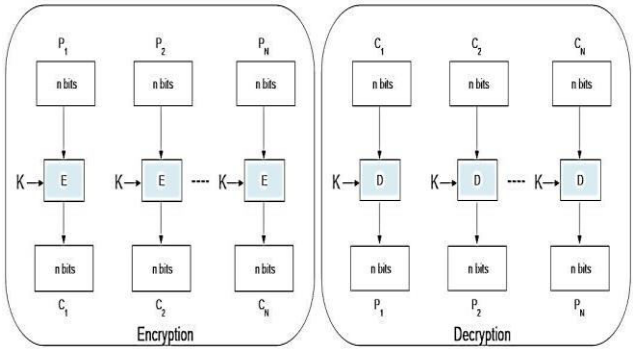

Figure-5. Electronic codebook (ECB) mode.

4.2 Parallel implementation

In this section, we discuss the implementation of the AES algorithm and the optimization performance for dividing the data file. Snapshot code in Figure-6 shows a simple implementation of a parallel release of AES algorithm. It depends heavily on Ref. [16] for implementation accumulator with the number of threads at runtime. It partition the data file among the threads. For avoiding the overhead of too many threads, the algorithm chooses a minimum number of chunks per thread which it is equal to block size (block size is 16, 32, 64 or 128 byte), Line 5.

The C++ Standard Library has one feature can be a useful guide for partition a data file into chunks is std::thread::hardware_concurrency(). This function returns the number of CPU cores and therefore the number of threads that can truly run concurrently for a given execution of a program. This avoids our implementation the oversubscription (running threads more than hardware supporting), and therefore avoids the context switching (more threads will decrease the performance), Line 7.

```
1.  template<typename Iterator>
2.  vec2d parallel_plaintext(Iterator first, Iterator last, unsigned long const block_size)
3.  {
4.  unsigned long const length = std::distance(first, last);
5.  unsigned long const min_per_thread = block_size;
6.  unsigned long const max_threads =
    (length + min_per_thread - 1) / min_per_thread;
7.  unsigned long const hardware_threads =
    std::thread::hardware_concurrency();
8.  unsigned long const num_threads =
    min(hardware_threads != 0 ? hardware_threads : 2, max_threads);
9.  unsigned long const Chunks = length / num_threads;
10. vec2d results(num_threads, vector<int>(Chunks));
11. std::vector<std::thread>  threads(num_threads - 1);
12. std::atomic<int> i;
13. Iterator block_start = first;
14. for (unsigned long i = 0; i<(num_threads - 1); ++i)
15. {
        16. Iterator block_end = block_start;
        17. std::advance(block_end, Chunks);
        18. threads[i] = std::thread(
            Chipher<Iterator>, block_start, block_end, std::ref(results),i);
        19. block_start = block_end;
    }
    20. Chipher<Iterator>(block_start, last, results, (num_threads - 1));
    21. std::for_each(threads.begin(), threads.end(),
        std::mem_fn(&std::thread::join));
    22. return results;
}
```

Figure-6. Snapshot code of the divided data.

We choose the number of threads at runtime by return the minimum of the maximum of threads and the hardware threads, Line 8. The number of chunks for each thread to implement is the length of the plaintext divided by the number of the threads, Line 9. We created two vectors for the result and the number of threads, Line 10. Simple loop for launching the threads, Line 14. Advance the iterator of the block_end with the size of chunks on the current block, Line 17. Launching a new thread to find the cipher text for this block, Line 18. Once we've ciphering all the blocks, spawning threads should wait to finish the jobs, Line 21. We return the result by reference to the relevant entry in

the calling function. C++ futures provide other ways for returning the results from the threads through the use of futures provider.

5. RESULTS AND DISCUSSIONS

To prove the accurate and speedup of our algorithm, we have tested the algorithm with diverse plaintext sizes from 64 bytes to 16kilobytes on 2, 4 and 8 single processor-multi core systems. Usually, the behavior of the AES algorithm encrypts 16 Kbyte with each status. This behavior breakdown with our algorithm which the block size is depends on the user's choice (generally 16-128 KB) and at runtime the block size is divided between the physical cores as we mentioned above.

Finally, the executing time from repeated same process in three different processors are analyzed and compared as the following:

Comp. 1: Windows 8.1 pro, Core i5-3210M CPU 2.50GHz, and 4GB RAM. (with dual cores).

Table-1, displays the execution times for AES serial and AES parallel in milliseconds (ms) required by different entire file size (plaintext) for encryption and decryption processes.

Comp. 2: Windows 10, i7-4500U CPU 1.80GHz, 2401 MHz, with (4 cores).

Table-2 displays the execution times for AES serial and AES parallel in (ms) required by different entire file size (plaintext) for encryption and decryption processes.

Comp. 3: Windows 10 pro, Core i7-3210M CPU 2.50GHz, and 4GB RAM. (with 8 cores).

Table-3 displays the execution times for AES serial and AES parallel in (ms) required by different entire file size (plaintext) for encryption and decryption processes.

Table-1. Execution time with dual cores.

Entire file (byte)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Decryption Time (ms)	AES Parallel Decryption Time (ms)
64	57.1	40.2	65.3	50.5
128	130	75.4	159.1	58.7
192	180.2	84.8	201.6	72.2
256	212.5	139.3	236.2	127.6
320	278.3	145	331.1	135.8
384	336.1	186.3	354.3	194.1
448	376.3	216.8	423.7	229
512	415.5	236.4	516.6	280.5
576	473.7	268.3	553.4	299.8
640	504.5	290.9	636.9	318.1
704	566.6	322	698.5	354.1
768	619.7	350.1	715.4	392.4
832	632.4	365.3	759.1	439
896	663.2	399.6	775.9	463.2
960	823.1	422	899.4	501.9
1024	912.2	454.4	964.5	537.4
2048	1801.7	778.3	1942.6	880.4
4096	4571.3	1079.9	5784.1	2117.2
8192	5611.7	3721.8	6917.7	3924.8
16384	6853.2	5033.2	7720.3	6540.5

Table-2. Execution time with 4 cores.

Entire file (byte)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Decryption Time (MS)	AES Parallel Decryption Time (ms)
64	29.2	18.9	31.1	27.6
128	37.2	21.3	50.5	41.3
192	55.4	31.2	75.1	61.8
256	72.7	39.2	102.1	82.6
320	90.2	49.1	128.5	103.8
384	108.6	56	153.2	123.5
448	127.5	65.4	178.4	145.2
512	145.3	75	202.3	163.6
576	162.5	84.1	226.1	182.2
640	180.4	93.6	240.6	205.3
704	200	102.2	278.4	225.1
768	218	114	302.3	245.6
832	234.4	126.5	311.4	258.7
896	252.3	135.6	351.8	283.5
960	270.1	141.5	377.2	305.1
1024	288.5	149.7	402.3	324.6
2048	575.6	294.8	801.2	651.3
4096	987.1	601.2	1121.1	997.3
8192	2346.1	1183.2	3107.2	2522.3
16384	4653	2359.6	5774.3	3240.2

Table-3. Execution time with 8 cores.

Entire file (byte)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Decryption Time(MS)	AES Parallel Decryption Time (ms)
64	19.4	15.5	25.5	21
128	20.8	17.2	46.2	32.4
192	21.2	18.9	68.4	40.2
256	25.1	20.6	89.8	51.7
320	34.6	27.3	111.3	59.7
384	40.3	35.2	136.5	73.1
448	52.1	42.7	156.8	84.4
512	67.1	50.2	180.1	95.8
576	75.3	61.3	201.4	107.5
640	83.7	69.8	224.2	119.9
704	98.6	80.1	245.3	141.8
768	101.8	90.4	268.5	146.8
832	109	103.4	290.5	155.4
896	127.6	111.2	313.5	170.3
960	138.9	119.7	335	181
1024	154.3	127.4	358.3	194.6
2048	285.8	251.3	715.3	381.2
4096	578.6	521.6	1003.5	770.5
8192	1157.2	1020.1	2891.7	2091.3
16384	2109.5	1769.7	3052.1	2712.5

Table-4. Comparison between AES serial and the enhancing algorithm.

Entire file (plaintext) (byte)	Comp.1		Comp.2		Comp.3	
	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)
64	57.1	40.2	29.2	18.9	19.4	15.5
128	130	75.4	37.2	21.3	20.8	17.2
192	180.2	84.8	55.4	31.2	21.2	18.9
256	212.5	139.3	72.7	39.2	25.1	20.6
320	278.3	145	90.2	49.1	34.6	27.3
384	336.1	186.3	108.6	56	40.3	35.2
448	376.3	216.8	127.5	65.4	52.1	42.7
512	415.5	236.4	145.3	75	67.1	50.2
576	473.7	268.3	162.5	84.1	75.3	61.3
640	504.5	290.9	180.4	93.6	83.7	69.8
704	566.6	322	200	102.2	98.6	80.1
768	619.7	350.1	218	114	101.8	90.4
832	632.4	365.3	234.4	126.5	109	103.4
896	663.2	399.6	252.3	135.6	127.6	111.2
960	823.1	422	270.1	141.5	138.9	119.7
1024	912.2	454.4	288.5	149.7	154.3	127.4
2048	1801.7	778.3	575.6	294.8	285.8	251.3
4096	4571.3	1079.9	987.1	601.2	578.6	521.6
8192	5611.7	3721.8	2346.1	1183.2	1157.2	1020.1
16384	6853.2	5033.2	4653	2359.6	2109.5	1769.7

Table-5. Comparison between AES serial and the enhancing algorithm for the decryption operation.

Entire file (plaintext) (byte)	Comp.1		Comp.2		Comp.3	
	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)	AES Serial Encryption Time (ms)	AES Parallel Encryption Time (ms)
64	65.3	50.5	31.1	27.6	25.5	21
128	159.1	58.7	50.5	41.3	46.2	32.4
192	201.6	72.2	75.1	61.8	68.4	40.2
256	236.2	127.6	102.1	82.6	89.8	51.7
320	331.1	135.8	128.5	103.8	111.3	59.7
384	354.3	194.1	153.2	123.5	136.5	73.1
448	423.7	229	178.4	145.2	156.8	84.4
512	516.6	280.5	202.3	163.6	180.1	95.8
576	553.4	299.8	226.1	182.2	201.4	107.5
640	636.9	318.1	240.6	205.3	224.2	119.9
704	698.5	354.1	278.4	225.1	245.3	141.8
768	715.4	392.4	302.3	245.6	268.5	146.8
832	759.1	439	311.4	258.7	290.5	155.4
896	775.9	463.2	351.8	283.5	313.5	170.3
960	899.4	501.9	377.2	305.1	335	181
1024	964.5	537.4	402.3	324.6	358.3	194.6
2048	1942.6	880.4	801.2	651.3	715.3	381.2
4096	5784.1	2117.2	1121.1	997.3	1003.5	770.5
8192	6917.7	3924.8	3107.2	2522.3	2891.7	2091.3
16384	7720.3	6540.5	5774.3	3240.2	3052.1	2712.5

Tables 4 and 5 display the different execution time periods when executed on different cores. Based on the results of the AES serial and AES parallel algorithm for the encryption and encryption processes, we can see the performance is not constant for all plaintext sizes and the time taken to the encryption and decryption operations decreases when use more than one core, for example, for the file size of (16384B) for the improved AES parallel algorithm, the improvement in execution time using 4 and 8 cores is ranging between (2359.6 - 1769.7 ms). In the encryption process. Whereas they are (3240.2 - 2712.5 ms) in the decryption process for the same algorithm. It is clear that the execution time needed for the encryption and decryption operations approximately reduced.

Figure-7 and Figure-8 show graphically the enhanced performance of the execution time for the AES parallel algorithms.

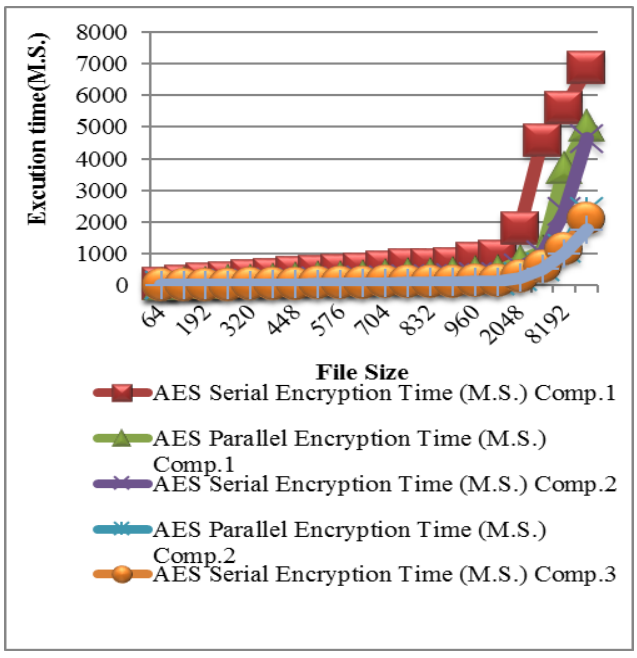

Figure-7. The execution time of the serial and parallel algorithm for the encryption operation using multiple cores on a different entire file size.

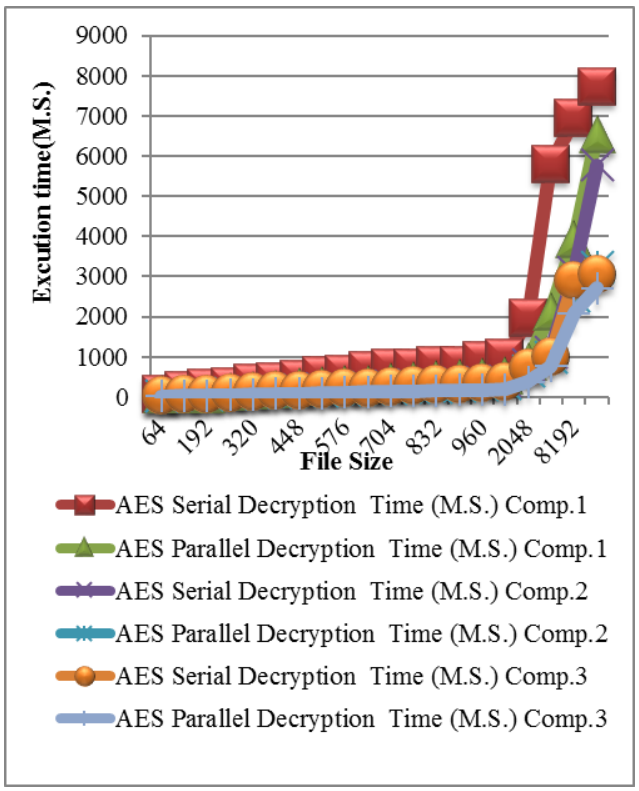

Figure-8. The execution time of the serial and parallel algorithm for the decryption operation using multiple cores on a different entire file size.

6. CONCLUSIONS

The AES algorithm can be developed using C++17 concurrency and the parallel version can be implemented directly using multithreading tasks. Roles of the encryption can be determined by the operating modes, but not all the modes can be parallelized. We implemented the AES algorithm using ECB mode because this mode can be paralyzing in encryption and decryption. Comparing the sequential algorithm with the parallel version we showed that the sequential algorithm works well yet, the performance, reduced alarmingly after we reach 1MB of the file size. For the big data, partition the data file among the threads at runtime achieved high performance and efficiency in memory usage, of course the speedup increase as the number of cores increase.

REFERENCES

[1] R. Wobst. 2007. Cryptology Unlocked, Chichester: John Wiley & Sons Ltd.

[2] V. R. Joan Daemen. 2002. The Design of Rijndael, AES the Advanced Encryption Standard. Springer, Berlin, 2002.

[3] J. J. G. Savard. 2012. John Savard. Studio Foglio LLC. [Online]. Available: http://www.quadibloc.com/crypto/co040401.htm.

[4] R. L. H. Y. Wei Liu. 2009. Cryptography Overhead Evaluation and Analysis for Wireless Sensor Networks. in CMC '09. WRI International Conference, Yunnan, China.

[5] N. M. Josuttis. 2017. C++17 - The Complete Guide, New Jersey: Addison-Wesley.

[6] Ritambhara A., Gupta M. Jaiswal. An Enhanced AES Algorithm Using Cascading Method On 400 Bits Key Size Used In Enhancing The Safety Of Next Generation Internet Of Things (IOT). International Conference on Computing, Communication and Automation (ICCCA2017).

[7] C. Paul Pramod, M. Jaiswal. An Advanced AES Algorithm using Swap and 400 bit Data Block with flexible S-Box in Cloud Computing. International Conference on Computing, Communication and Automation (ICCCA2017).

[8] P. Dixit, J. Zalke, S. Admane. Speed optimization of AES algorithm with Hardware-Software Co-design. 2017 2nd International Conference for Convergence in Technology (I2CT).

[9] A. Barnes, R. Fernando, K. Mettananda and R. Ragel, Improving the Throughput of the AES Algorithm with Multicore Processors, Industrial and Information Systems (ICIIS), 2012 7th IEEE International Conference.

[10] Barnes A. P. P. D. a. K. V. Nalini. 2007. Optimized S-BOX Design for AES Core. in IET-UK International Conference on Information and Communication Technology in Electrical Sciences, India.

[11] S. W. 2012. Cryptography and Network Security Principles and Practice. Fifth Edition, Prentice Hall.

[12] M. R. 2004. A VHDL Implemetation of the Advanced Encryption Standard-Rijndael Algorithm. in University of South Florida, College of Engineering, Ms. Thesis.

[13] W. contributors. 2017. Block cipher mode of operation--Wikipedia {,} The Free Encyclopedia. Wikimedia Foundation, Inc. [Online].Available: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation. [Accessed 14 December 2017].

[14] A. R. R. Mohan H.S. 2012. Revised AES and its Modes of Operation. International Journal of

Information Technology and Knowledge Management. 5(1): 31-36.

[15] M. Dworkin. 2013. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Computer Security Division's (CSD) Security Technology Group (STG).

[16] A. Williams. 2012. Managing threads. in C++ Concurrency in Action Practical Multithreading, Shelter Island, Manning Publications. pp. 28-30.

[17] M. C. S. Mithila Nagendra. 2014. Performance Improvement of Advanced Encryption Algorithm using Parallel Computation. International Journal of Software Engineering and Its Applications. 8(2): 287-296.

[18] M. P. S. Y. A. R. Vandan Pendli. 2016. Algorithm, Improvising performance of Advanced Encryption Standard algorithm. In Mobile and Secure Services (MobiSecServ), Second International Conference, Gainesville.

[19] V. H. S. a. T. Diwan. 2017. Parallel Computation of Advance Encryption Standard Algorithm for Performance Improvement. in International Conference on Recent Trends in Engineering Science and Technology, Maharashtra.

[20] A. Y. Ebrahim, Asmaa Shaker Ashoor. 2018. Tumor classification using enhanced hybrid classification methods and segmentation of MR brain images. ARPN Journal of Engineering and Applied Sciences. 13(20).