Vishal Pachori, Gunjan Ansari, Neha Chaudhary / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 1,Jan-Feb 2012, pp.967-971

Improved Performance of Advance Encryption Standard using Parallel Computing

Vishal Pachori*, Gunjan Ansari, Neha Chaudhary*****

*(Department of Computer Science, JSSATE, Noida, UPTU, India)
** (Department of Computer Science, JSSATE, Noida, UPTU,India)
*** (Department of Computer Science, JSSATE, Noida, UPTU,India)

ABSTRACT

This paper presents the implementation of Advance encryption (AES) algorithm using parallel computing. Most of the research for improving performance of AES is based on hardware implementation. This paper presents the parallel implementation of AES using JPPF (Java Parallel Programming Framework) which provides flexibility & performance improvement in terms of speed-up. In this implementation there are two approaches data parallelism and control parallelism. In Data parallelism, xn plain text is taken where x is 128 bits and n is any integer. Further xn data will be divided into n parts and each part will be send to independent processing units. Value of n can be increased depending on processing units available. In control parallelism, four round functions of AES Subbyte , ShiftRow , Mixcolumn. AddroundKey are divided into two independent parts for which structure of round functions has been modified. These two independent parts are assigned to different processing units and these processing units will exchange their intermediate result for further processing. Finally, results are conducted on single processing unit and multiple processing units on different sets of 256 bits of data and the performance analysis shows improvement in terms of execution time.

Keywords – **AES, Data Parallelism Control Parallelism, Parallel computing**

1 INTRODUCTION

For the protection of data transmission over insecure channels two types of cryptographic systems are used: Symmetric and Asymmetric cryptosystems[1]. Symmetric cryptosystems such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES) uses an identical key for both to encrypt the plain text and decrypt the cipher text. Asymmetric cryptosystems such as Rivest-Shamir-Adleman (RSA) & Elliptic Curve Cryptosystem (ECC) uses different keys for encryption and decryption.Symmetric cryptosystem is more suitable to encrypt large amount of data with high speed.

To replace the old Data Encryption Standard, in Sept 12 of 1997, the National Institute of Standard Technology (NIST) required proposals to what was called Advanced Encryption Standard (AES)[2]. Many algorithms were presented originally with researches from 12 different nations. On October 2nd 2000, NIST has announced the Rijndael algorithm is the best in security, performance, efficiency, implement ability, & flexibility. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Rijmen of Katholieke University at Leuven. It became effective as a Federal government standard on May 26, 2002 after approval by the Secretary of Commerce. It is available in many different encryption packages. AES is the first publicly accessible and open cipher approved by the National Security Agency (NSA) for top secret information. So, it has broad applications, such as smart cards and cell phones, WWW servers and automated teller machines, and digital video recorders. Numerous architectures have been proposed for the hardware implementations of the AES algorithm.

2 AES ALGORITHM

AES, i.e. Rijndael algorithm is a symmetric key cryptography. In 1998 Rijndael cipher developed by the two Belgian cryptographers, John Daemen and Vincent Rijmen was published. This cipher was selected later on by the NIST as the Advanced Encryption Standard to supersede the old Data Encryption Standard. The AES standard has a constant block size of 128 bits with 3 different key sizes of 128 bits, 192 bits and 256 bits, where 10, 12 and 14 encryption rounds will be applied for each key size, respectively. During the encryption and decryption processes, the 16 bytes of data will form a changeable (4*4) array called the state array. During the encryption process, the state array consists initially of the input data, this array will keep changing until reaching the final enciphered data. In the decryption process the

state array will start by the enciphered data and will keep changing until retrieving the original data. The encryption of AES is carried out in blocks with a fixed block size of 128 bits each. The AES cipher calculation is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of cipher text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform the cipher text back into the original plaintext using the same encryption key.

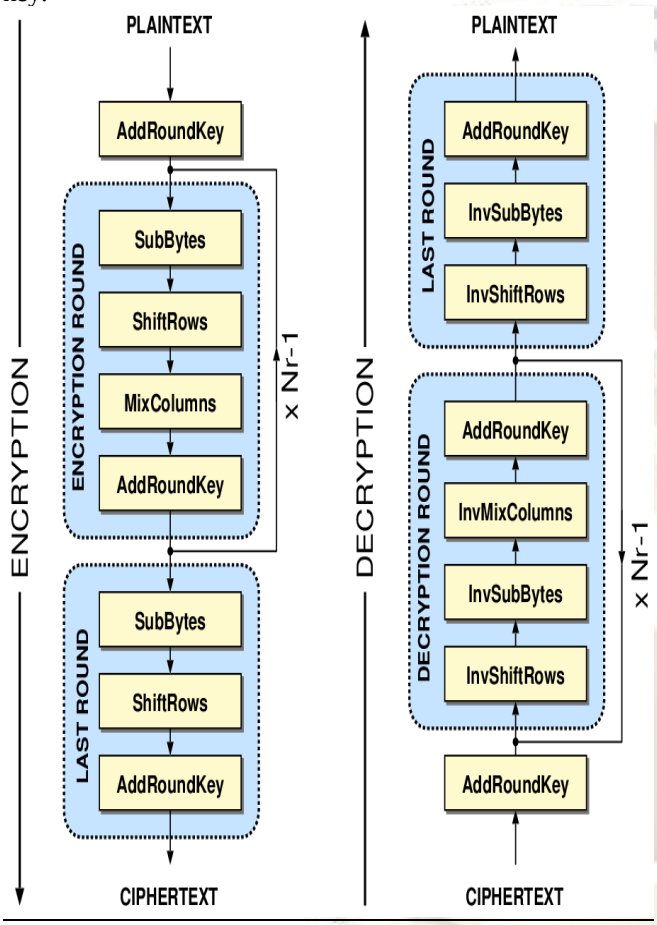

figure 1 AES Cipher

3 RELATED PREVIOUS WORK

Various attempts are made to increase the performance of AES algorithm as it is widely used, some of them are as follows:-

In March 11, 2010, Piotr Bilski *et al* (2010), proposed "The multi-core implementation of the symmetric cryptography algorithms in the measurement system [1]. They proposed three different approaches of implementing AES algorithm on different processors. The first modification of the original algorithm does not require any changes in the algorithm itself. The procedure involves dividing the input, plaintext bytes into blocks that can be encrypted and decrypted independently. This way multiple blocks can be processed simultaneously. The first operation here is separating the plain text or cipher blocks into independent streams and then applying the AES encryption or decryption procedures. The expected increase of the computational efficiency depends on the number of the applied copies of the procedure, determined by the number of the processing units (processors or cores). The second modification is more efficient approach as the modification made inside the algorithm to make every subprocedure inside the AES schedule to be able to run on the independent processor cores. This might also be necessary when the plain text or cipher blocks are processed in relation to the values obtained in the previous iterations. . The decryption and key expansion algorithms are performed analogously. Note that the parallelism of the whole scheme requires firstly row, then column operations. Firstly, the steps of the algorithms that are in a sequence can not be modified to be run simultaneously, as their results depend on the results obtained from the preceding operations. Therefore, the steps inside the ''for" loop (SubBytes, ShiftRows, MixColumns and AddRoundKey) must be processed sequentially, and no concurrent execution is possible. However, the analysis of the operations inside of these steps shows that the matrix operations can be broken into independent parts. The SubBytes and ShiftRows are the operations that transform the individual rows in the state array – each row can be processed separately. Moreover, as these operations are put one after another in the algorithm, there is no need to treat them as the separate subprocedures. They can be used inside one program function. Similarly, MixColumns and AddRoundKey can be decomposed into the simultaneously processed parts, but this time the latter operate on columns. Third approach depend upon modes of operation of AES algorithm. As the AES algorithm is the public key system, it is vulnerable to the mode of the operation. As all data are processed using the same key, identical plaintexts will be transformed into identical ciphers. This might facilitate the intruder to obtain useful information about the transmitted data without breaking the cipher. Therefore it is important to incorporate into the system block processing modes. To increase security of the encryption and decryption operations, the dependency between the subsequent data blocks must be introduced. The most popular safe modes that can be used in the MS are Cipher-Block Chaining (CBC), Propagating Cipher-Block Chaining (PCBC) and Cipher Feedback (CFB). The modification of all three safe modes includes separating them into independent streams. Limitations of

the proposed solution lie in the computer configuration, although it also relies on the specific software (Lab VIEW environment and RTOS). It is possible to make this implementation in other environments which are more efficient (such as Agilent VEE), or generic programming language and use library other then CRYPTO-G library.

Hua li *et al* (2008), proposed "A new compact dual-core architecture for AES encryption and decryption" in which they present a new compact architecture, consisting of two independent cores that process encryption and decryption simultaneously [3]. They unroll one iterative round loop with a 32-bit data path, which means that four clock cycles are required to complete one transformation round of a 128-bit block of data. They also present a new method for implementing ShiftRows/InvShiftRows, which are critical operations that directly affect the architecture of the compact 32-bit data path design. In addition, a compact key generation unit that generates subkeys for encryption and decryption on the fly is proposed. The proposed architecture can be easily extended to 192- and 256-bit cipher key implementations. . This approach is useful when both plaintext and cipher text is available to process as in the case of real time server. The architecture is based mainly on new optimized ShiftRows and InvShiftRows operations which avoid complex multiplexers and reduce the critical path of the design. The compact implementation can perform encryption and decryption in parallel, and it is most suitable for applications such as real-time dual-duplex wireless communication. In this approach, both hardware and software both needs to be modified. So, this approach is not cost effective and it can be used only where there is large amount of data for encryption and decryption both.

4 THE AES MODIFICATION

To improve the performance of AES algorithm using parallel computing there are two major approaches DATA Parallelism and Control Parallelism.

In Data Parallelism the data is divided into more than one part and send different part to different nodes for execution. Each node is executing the same procedure or function but on different data. This approach is very effective when there is large data to process. AES can be implemented in the following manner using DATA parallelism. Server sends Plaintext with the Key on node 1 and it will compute the cipher text by running the AES algorithm and finally sends the result back to the Server. Node 2 follows the same procedure as shown in Fig. 2. The number of nodes can be increased according to our requirement and number of processing units available. This approach is implemented using JPPF framework by passing different data sets to different JPPF nodes (node 1 and node 2).

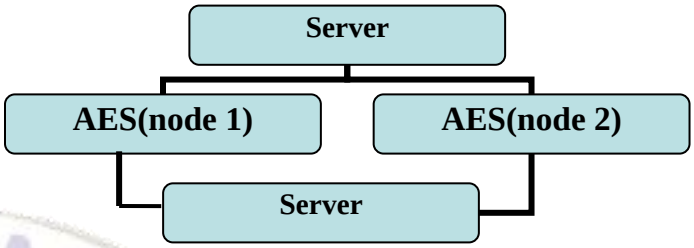

figure 2 Data Parallelism

In Control Parallelism the operation or function is divided instead of data. The different operation or function is assigned to different nodes and then finally the output is send to the server for final processing, as shown in Fig 3. Although it is less scalable then data parallelism but more speed up can be achieved by this approach.

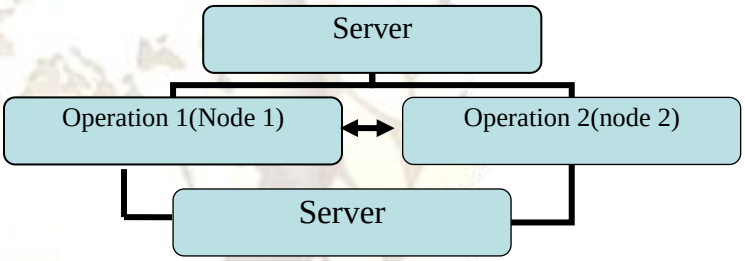

figure 3 Control Parallelism

Here,

Operation 1 = combination of (SubByte and Shiftrow operation)

Operation 2 = combination of (Mixcolumn and Addround key)

In control parallelism approach, the four main operations in AES algorithm are divided into two parts and combination of these operations is Operation 1 and Operation 2. Node 1 will execute only operation 1 and Node 2 will perform only operation 2. Nodes will communicate the result of each other when needed.

The Fig. 3 is System architecture of control parallelism approach. It shows the entire process of proposed control parallelism approach.

In this approach first of all Server fetches Plaintext (or message to encrypt) and round key from two different files then server will perform the initial function which is Add Round Key in which server simply e-xor the plaintext with

the input round key. Then server divides the resultant into two parts ,say A and B, and sends these parts to Node 1(First processing unit) then Node 1 will perform operation 1 which is combination of two round functions namely SubByte and Shiftrow on data set A and then Node 1 will send its computed result to Node 2 (second processing unit).
Node 2 will perform the operation 2 which is combination of other two round functions namely Mixcolumn and Addround key on data set A at the same time Node 1 is performing operation 1 on data set B. Then Node 2 and Node 1 will exchange their result and both wait for other processing unit to compute the result.

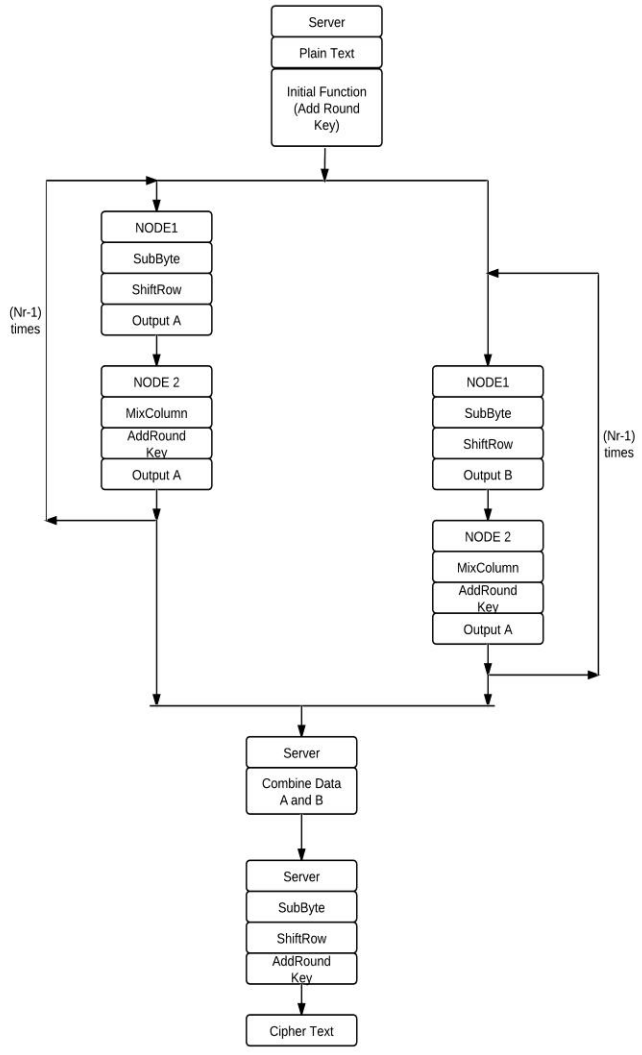

figure 4 Proposed AES Modification

As operation 1 and operation 2 approximately takes equal time to execute so the waiting time is negligible. At the same time, Node 1 will perform operation 1 and Node 2 will perform operation 2 on different data sets and then they exchange their results and this process is repeated (N_r - 1) times where N_r is number of rounds. The value of N_r will depend upon the length of the input key for 128bit key the value of N_r is 10. Then, Node 2 will send the computed data after (N_r -1) rounds to the server and server will combine that computed data.

Finally, server will perform the last round which contains SubByte, ShiftRow and AddRoundKey round function. These Operations are performed on server because if the data is sent to Node 1 and it will compute the last round then it will increase the overall execution time so it's better to execute it on the server as these operations are performed serially so there is no need to send data to other processing unit and it will save the communication time or delay.

5 IMPLEMENTATION

Both the approach can be implemented using JAVA Parallel Programming Framework (JPPF). JPPF works on any system that supports Java[5]. There is no operating system requirement; it can be installed on all flavors of UNIX, Linux, Windows, Mac OS, and other systems such as zOS or other mainframe systems. A JPPF grid is made of three different types of components that communicate together:

- **clients** are entry points to the grid and enable developers to submit work via the client APIs
- **servers** are the components that receive work from the clients, dispatch it to the nodes, receive the results fom the nodes, and send these results back to the clients
- **nodes** perform the actual work execution

To understand how the work is distributed in a JPPF grid, and what role is played by each component, the following two units of work that JPPF handles should be defined.

A **task** is the smallest unit of work that can be handled in the grid. From the JPPF perspective, it is considered *atomic*.

A **job** is a logical grouping of tasks that are submitted together, and may define a common service level agreement (SLA) with the JPPF grid.

For establishing communication between nodes "Hazelcast" Framework is used. Hazelcast framework is used only to create a distributed data structure on which node 1 and node 2 can communicate by putting their computed value.

The **Data parallelism** approach can be written as:

AES (data parallelism)

Initial Condition: Plain Text (p bits) and key (k bits) where p is multiple of 128 and n number of processors

Final Condition: p bits of encrypted text (cipher text)

Begin:

Spawn (P_0, P_1 …….P_{N-1})

For all P_i where 0< i <n-1

A_i = Plaintext [(i*128), (i*128) + 1, (i*128) +2… (i*128)+127]

Call AES(A_i, k)

End for

End

The **Control Parallelism** approach can be written as:

Initial Condition: Plain Text (p bits) and key (k bits) where p is multiple of 128 and n number of processors

Final Condition: p bits of encrypted text (cipher text)

Begin:

Spawn (P_0, P_1)

A_0 = PlainText [0,127]

A_1 = PlainText [127,255]

For all P_i, 0<= I <= 1

For j = 0 to 9

Call Operation1 on A_i

Wait for Operation1 to complete

Call Operation 2 on A_i

End for

Call LastRound on [A_0 + A_1]

End for

End

Here,

Last Round contains SubByte , ShiftRow and AddroundKey Operation as shown in Fig. 1.

6 PERFORMANCE ANALYSIS

The performance of proposed architechture is measured in terms of execution time. The performance is measured on 256 bits of data and on two nodes or processing units. The following Table gives the execution time of converting 256 bits plain text into cipher text on JPPF framework using two nodes. The time taken by single core to encrypt 256 bits of data is 14, 15 and 13 seconds in different run. The time taken by the 1st run is more than the time taken in the subsequent run because in the first run the Hazelcast Framework is loaded which takes time to load. In the subsequent runs the time taken by the modified AES algorithm is almost same i.e. execution time gets stable. Speed up of the modified AES algorithm is shown below.
The configuration of the system on which these results are taken is as follows:
Processor: Intel Core(TM)2Duo E7300@ 2.66 Ghz 3MB L2 cache memory
Ram: 2GB

Speed Up = (Time taken by the serial algorithm) / (Time Taken by the Parallelism algorithm)

Speed up For Data Parallelism for 256 bits of Data:

Speed up for Data parallelism (1st run) = 15/10 = 1.5

Speed up for Data parallelism (2nd run) = 14/7 = 2.0

Speed up for Data parallelism (3rd run) = 13/6 = 2.16

Speed up for Data parallelism (4th run) = 13/7 = 1.85

Speed up For Data Parallelism for 256 bits of Data:

Speed up for Control parallelism (1st run) = 15/11 = 1.36

Speed up for Control parallelism (2nd run) = 14/7 = 2.0

Speed up for Control parallelism (3rd run) = 13/6 = 2.16

Speed up for Control parallelism (4th run) = 13/6 = 2.16

	Data Parallelism (Seconds)	**Control Parallelism(Sec)**
1st Run	10	11
2nd Run	7	7
3rd Run	6	6
4th Run	7	6

7 CONCLUSION

Both the proposed approach of AES algorithm is successfully implemented in JPPF framework and performance is measured in terms of execution time. It can be seen that the experiment shows a significant improvement in terms of execution time. For the Future work other cryptography algorithm can also be implemented using parallel computing in order to increase their performance. In this paper, AES is implemented using JPPF framework it can also be implemented using other fast framework available. The network traffic can affect the performance of this implementation.

REFERENCES

[1] Piotr Bilski , Wiesław Winiecki, 2010 ,Multi-core implementation of the symmetric cryptography algorithms in the measurement system, Measurement 43 (2010) 1049–1060, Elsevier. 10.1016/j.measurement.2010.03.002

[2] Behrouz A. Forouzan, De Anza College *Cryptography and Network Security* (McGraw-Hill,2007)

[3] Hua li and Jianzhou li 2008 proposed, A new compact dual-core architecture for AES encryption and decryption ,Lethbridge,Alberta. *Electrical and Computer Engineering, Canadian Journal* . 10.1109/CJECE.2008.4721627

[4] Zhiyong Guo, Guangjun Li, Yang Liu,2010, Dynamic Reconfigurable Implementations of AES Algorithm Based on Pipeline and Parallel Structure,China. *Computer Engineering and Technology (ICCET),* 10.1109/ICCAE.2010.5451864.

[5] Online forum of JPPF Team available at: http:// www.jppf.org/forum.