Proceedings of Parallel Scientific Computing
ID 5130
Jan-May, 2024, IIT Madras, India

PROJECT FINAL REPORT

PARALLELIZATION OF ADVANCED ENCRYPTION STANDARD(AES)

Srinivasan Nerudtha
ME21B195

Sakthe Balan
ME21B174

ABSTRACT

Cryptography is a method of protecting information such that the information is read by only those to whom it was intended. The techniques for protection are derived from mathematical concepts and a set of rule-based calculations to transform messages in ways that are hard to decipher. Encryption is the process of encoding information, which converts the original representation of the information, known as plaintext, into an alternative form known as ciphertext. A key is used to encrypt and as well as decrypt messages. The ciphertext is unreadable unless one has the respective key to decode it back to the original plaintext.

AES encryption is used in many applications where a large amount of sensitive data needs to be protected from unauthorized access. Hence, it is essential to make the encryption process work faster. Instead of encrypting a large message sequentially, the message is divided and distributed to processors which encrypts the messages in parallel. There is also scope to parallelize within each round of encryption. This project aims to parallelize the AES algorithm uses different parallel paradigms such as OpenMP, and OpenAcc and compares the performance metrics with that of serial code.

INTRODUCTION

The Advanced Encryption Standard (AES) algorithm, a variant of the Rijndael algorithm, was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST in 2001 during the AES selection process. Rijndael is a family of ciphers with different key and block sizes. It is one of the most commonly and widely used symmetric-key block cipher algorithms with a block/chunk size of 128 bits. It converts these individual blocks using keys of 128, 192 or 256 bits. Once it encrypts these blocks, it joins them together to form the ciphertext.

AES is based on a substitution-permutation network, also known as an SP network. It consists of a series of linked operations, including replacing inputs with specific outputs (substitutions) and others involving bit shuffling (permutations), which are done multiple times (in rounds).

DESIGN PRINCIPLES AND STRUCTURE OF AES

AES is a symmetric key algorithm, which means the same key is used for both encryption and decryption. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael, with a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, Rijndael *per se* is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits. AES is much more efficient in both hardware and software.

High-level description of the algorithm

1. Key Expansion
2. Key Addition
3. Successive 9, 11, or 13 rounds of:
   (a) **Sub Bytes**
   (b) **Shift Rows**
   (c) **Mix Columns**
   (d) **Add RoundKey**
4. Final round (making 10, 12 or 14 rounds in total):
   (a) **Sub Bytes**
   (b) **Shift Rows**
   (c) **Add Round Key**

The SubBytes Step

At every step of transformation, the matrix is called a state matrix. At round 0, the state array is the same as the input message or text. A look-up table (usually termed as S-box) contains the respective substitutions for every character in the state array. So, every input character is then replaced by its respective counterpart from the table. This step takes only O(1) time as this is just a look-up step. During decryption, another look-up table known as inv-S-box is used, which is basically the inverse of S-box.

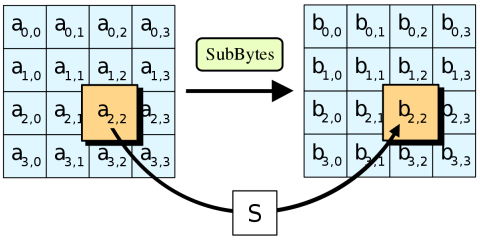

FIGURE 1. SubBytes

The ShiftRows Step

This step shifts the elements by an offset. The offset is equal to the row index (starts from 0). This means elements of row 0 are left unchanged, row 1 elements are offset to the left by 1 position, row 2 by 2 positions, and row 3 by 3 positions. This step is an important one so that the columns won't be ciphered independently as 4 different blocks.

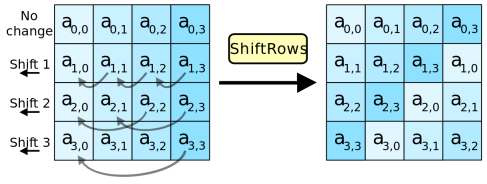

FIGURE 2. ShiftRows

The MixColumns Step

The columns of the current state are modified by multiplying with a given matrix(refer below) and instead of performing standard addition over the elements, an XOR operation is carried out. Along with the above 2 steps, this step also induces mixing and provides diffusion in the cipher.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad 0 \leq j \leq 3$$

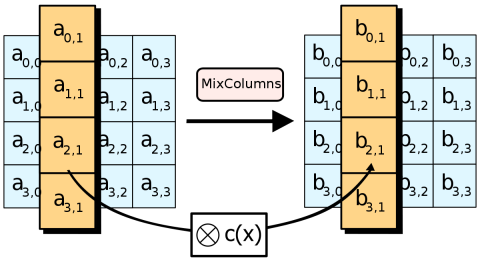

FIGURE 3. MixColumns

The AddRoundKey Step

The key scheduler derives a subkey for every round and this subkey is combined with the state by performing bitwise XOR addition with the corresponding byte.

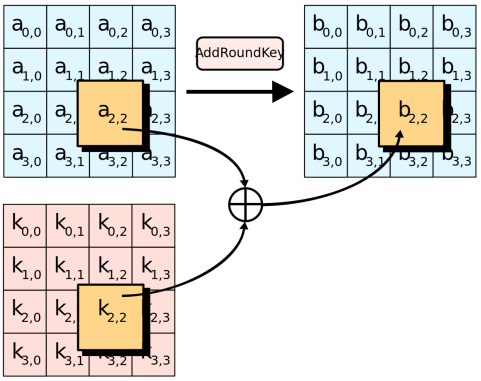

FIGURE 4. AddRoundKey

ENCRYPTION AND DECRYPTION PROCESS

The AES algorithm encrypts 16 bits (as the matrices used are of size 16) of the input message by performing the above-mentioned steps. So if the input is not a block of 16, extra padding is added and the input is fed into the algorithm. The following is the order of encryption:

1. Initial Round - Add Round Key.
2. Iterative Rounds - Sub Bytes, Shift Rows, Mix Columns, Add Round Key.
3. Final Round - Sub Bytes, Shift Rows, Add Round Key.

The initial round is nothing but an Add Round Key round and the final round is the same as the Iterative round without the Mix Columns step. The key size determines the number of iterative rounds the algorithm operates. At every round, the key scheduler generates sub-keys which are used in the respective rounds.

The Decryption process happens in the similar fashion, except that the step orders and the matrices used for the transformation and look-up are all inversed. It is as follows:

1. Initial Round - Sub Round Key, Shift Rows, Sub Bytes.
2. Iterative Rounds - Sub Round Key, Inverse Mix Columns, Shift Rows, Sub Bytes.
3. Final Round - Sub Round Key.

The keys can be of sizes 128-bit, 192-bit or 256-bit as said earlier. The bigger the key, more safer the encryption is, as there are 2^{256} permutations for the key, and brute-forcing one's way to find the key would take millions of years using current computing technology.

PARALLELIZATION

The AES algorithm has 6 different variants to it (Figure 5) and the Electronic Codebook(ECB) mode is parallelizable as, unlike other variants, it doesn't depend on the previous ciphers being encrypted. We have parallelized the ECB variant using OpenMP and OpenACC.

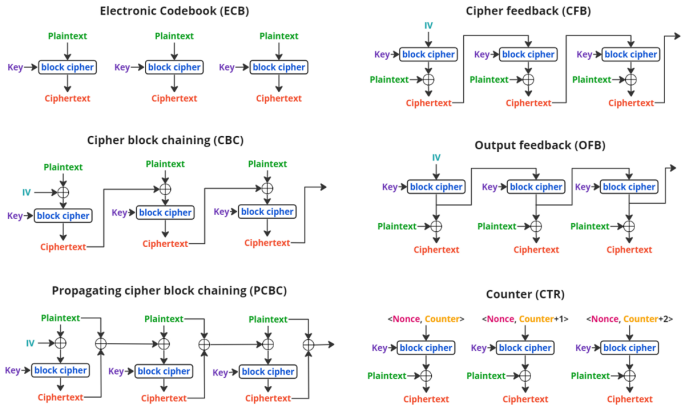

FIGURE 5. Modes of AES

PARALLELIZATION STRATEGY

Since AES operations revolve around working and manipulating a 4x4 matrix, it was decided not to parallelize those steps as parallelizing them would result in high overheads just to spawn and delete threads. Performing those operations in serial would be a better option which reduces overheads and time of execution.
Parallelization is among "encrypting the words" of the input message themselves, while the encryption process takes place serially. This means different words of the sentence are being encrypted in parallel.
OpenMP and OpenACC parallelization were used in the current context. The next step towards parallelization could be using both OpenMP and OpenACC, wherein each process would send its data to different gpus and so the encryption process takes place.

RESULTS

The encryption algorithm reads from a `.txt` document, which is the input file. It then encrypts it to another `.aes` file which is then read by the decryption algorithm which decrypts the message and prints it on screen. Here is a comparison between the serial, OpenMP, and OpenAcc versions of the code.

OpenMP and OpenACC

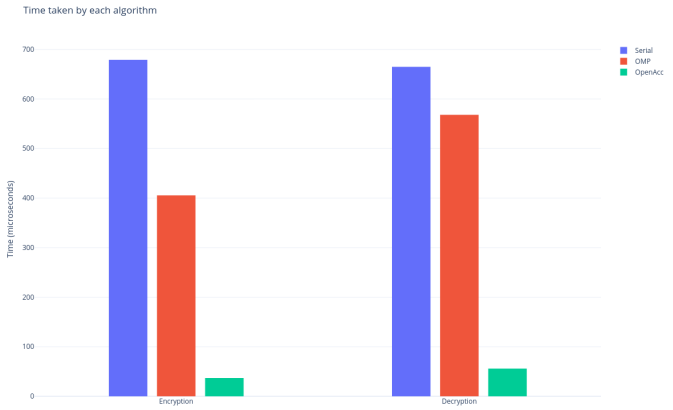

FIGURE 6. Time taken to encrypt and decrypt

It is evident from Figure 6 and Figure 7 that GPU parallelization results in the fastest encryption of the input document. The OpenMP versions provide some faster results but in reality, it doesn't offer that much. This could be because spawning OpenMP threads take a lot of time.

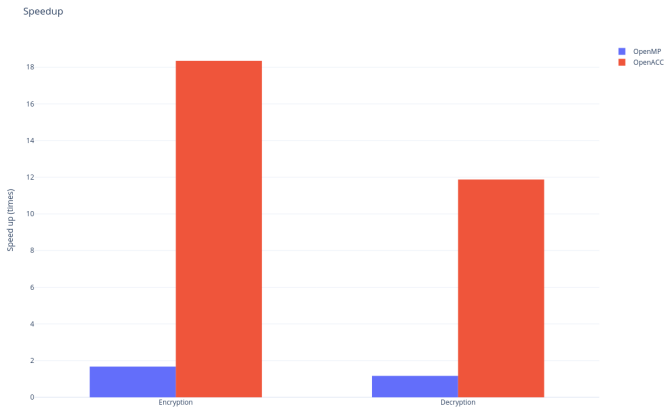

FIGURE 7. Speed up

But the results would improve as the size of document increases. The spawned threads could persist for a longer time and offer an improvement in performance compared to its serial counterpart.

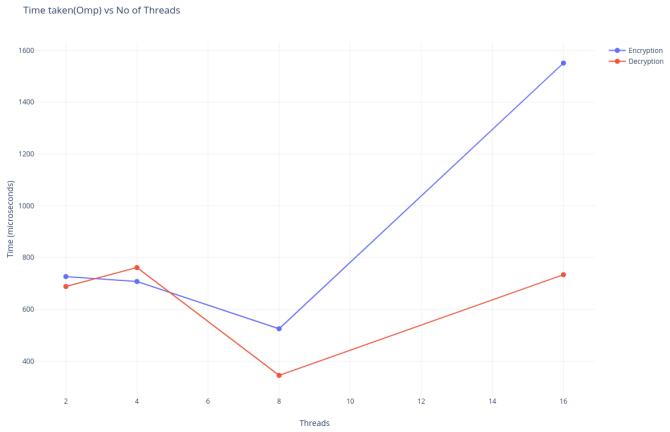

FIGURE 8. Time vs Threads

Figure 8 clearly indicates that there is overhead with spawning more threads and that is what is responsible for the lower speedups of the OpenMP versions.
However, the OpenACC version is way ahead in performance as there is only enqueuing done to the GPU and GPUs are very good at performing SIMD tasks. From Figure 7 it can be seen that OpenACC version offers about $12-20$x speed up, which on average is 15x speed up.

ACKNOWLEDGMENT

Thanks to Prof. Kameswararao Anupindi for his support.

REFERENCES

[1] Wikipedia, 2001. Advanced encryption standard.
[2] Wikipedia, 2002. Block cipher mode of operation.
[3] Wisniewska, C., 2017. aes-github-repository.
[4] Abdullah, A., 2017. "Advanced encryption standard (aes) algorithm to encrypt and decrypt data".
[5] 197, F. I. P. S. P., 2001. Advanced encryption standard (aes).
[6] tozny, 2015. aes-crypto-github-repository.
[7] guru irl, 2018. parallel-aes-github-repository.