PIPELINED DATA PARALLEL MODEL OF ADVANCED ENCRYPTION STANDARD ALGORITHM

Mohammed A. Mikki

Professor, Computer Eng. Dept., IUG, Palestine, mmikki@iugaza.edu.ps

ABSTRACT:

The Advanced Encryption Standard (AES) was officially adopted in 2002 as the new encryption standard algorithm. AES specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. It is a symmetric block cipher that can encrypt and decrypt information. This paper develops a pipelined data parallel model of AES. The parallelism in the algorithm is two dimensional. The first dimension is AES enter-stage (pipelining) and the second dimension is data parallelism. Pipelining parallelism exploits the availability of several processes to execute different stages of different data blocks in parallel. The data parallelism exploits data independence among data blocks to implement data level parallelism. The parallel implementation of AES decreases the time needed for encryption and decryption processes. We use the ECB mode in encryption/decryption algorithm in our parallel implementation of AES to implement the parallelization at data level where data blocks are encrypted and decrypted in parallel. We also develop an MPI-based algorithm to be used with a cluster of workstations (COW). We validate the approach by simulating the model with various input parameters (input data file size, number of processes, communication/computation operation execution time, etc.) and measuring the corresponding performance. Performance metrics include speedup, communication to computation ratio and efficiency. Results show that performance obtained by the developed model is superior to parallel implementations of AES which include only data parallelism or pipelining.

KEYWORDS: Advanced encryption standard, cipher, transformation, performance, speedup.

I. INTRODUCTION

Advanced Encryption Standard (AES) [1][2] was selected by the National Institute of Standards and Technology (NIST) as a new encryption standard to replace the Data Encryption Standard (DES) in 2000. AES algorithm is a symmetric block cipher that processes data blocks of 128 bits. The data is operated by 10, 12 or 14 rounds of transformations with key length of 128, 192 or 256 bits. A lot of hardware implementations of AES algorithm have already been proposed. They can be classified into two types: high speed designs and low- cost designs.

Hardware implementation of AES uses a data input length of 128 bits with a key length of 128 bits. A block of data is placed into a 16-byte array, and proceeds through 10 rounds of encryption. Basic operations include byte substitutions, independent row byte shifts, column Galois field multiplications, and key additions. AES can encrypt one block (128 bits) of data at a time. However, there are a lot of applications that need to encrypt the data blocks in parallel form such as Data Authentication Algorithm and Whirlpool, etc. Implementing AES in hardware is a complex design, and has a higher cost. Fig. 1 shows how a plaintext can be encrypted using the encryption algorithm of AES cipher to produce the ciphertext and how we can get the same plaintext from the ciphertext using the decryption algorithm of AES.Gaza Strip is 360km^2 with a high density population of about 4,118 persons/km^2 [2], so Gaza Strip represents one of the most densely populated areas in Middle East. As the population in Gaza Strip increases (population growth rate 3.349%/year [3]), the consumption of water and energy will increase; leading to significant rise in unacceptable levels of air pollution, and the defect in water supply and energy sources will increase; leading to severe economical crisis that will result in a significant rise in the probability of an outbreak of warfare.

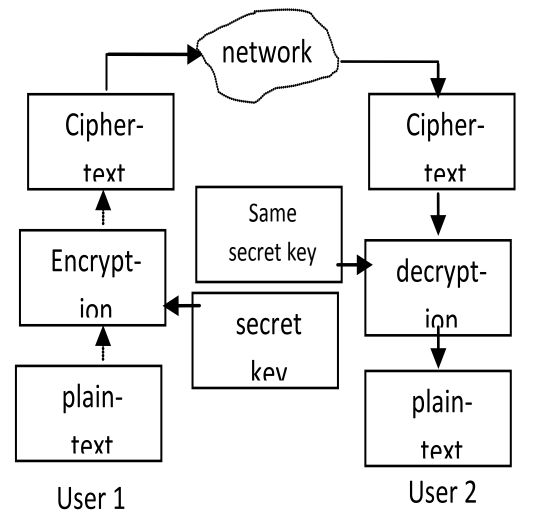


Figure 1: Symmetric key encryption and decryption using AES

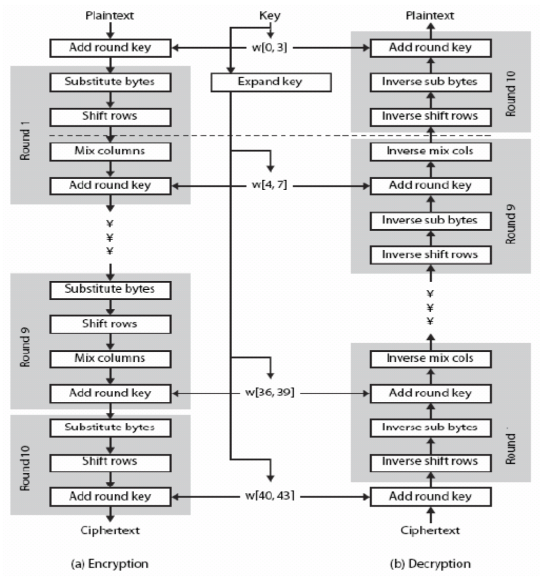


Figure 2: AES architecture

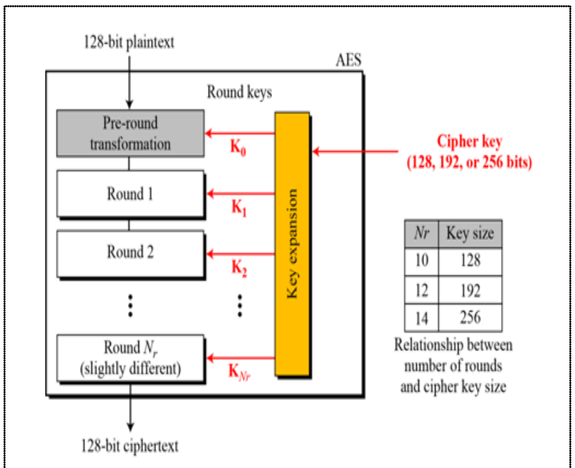
This paper develops a pipelined data parallel model of AES. The parallelism in the algorithm is two dimensional. The first dimension is AES enter-stage (pipelining) and the second dimension is data parallelism. Pipelining parallelism exploits the availability of several processes to execute different stages of different data blocks in parallel. The data parallelism exploits data independence among data blocks to implement data level parallelism. We use the ECB mode in encryption/decryption algorithm in our parallel implementation of AES to implement the parallelization at data level where data blocks are encrypted and decrypted in parallel. We also develop an MPI-based algorithm to be used with a cluster of workstations (COW).

The rest of the paper is organized as follows: Section II presents the related work. Section III presents an overview of AES Architecture. Section IV presents the pipelined data parallel model of AES approach developed in this paper. Section V presents experimental results. Finally, section VI concludes the paper.

II. OVERVIEW OF AES ARCHITECTURE

AES is a new encryption standard to replace the existing Data Encryption Standard (DES), which has been in place for more than two decades. See Fig. 1 for the AES block cipher. AES cipher is a “Block Cipher” with multiple options for its block and key sizes [3]. The NIST approved AES is a subset of these options; the block size is fixed at 128-bits, but the key may be either 128, 192 or 256-bits in length. Internally, in common with many block ciphers, AES consists of a complex non-linear core function [4], which is iterated multiple times on the incoming plaintext data block. The number of times this iteration is needed (the number of rounds required), depends on the selected key size. For 128-bit key AES, there are 10 rounds. The round function is slightly different for the final round, and an initial pre-processing function is also required at the start. Each round of AES requires a unique 128-bit Roundkey to be fed in to the complex round function. This series of 128-bit Roundkeys are generated from the supplied 128-bit, 192-bit or 256-bit AES key using a specified key expansion algorithm. This expansion yields exactly the right number of Roundkeys to feed the single pre-process step and the multiple rounds [5]. Fig. 2 shows the architecture of the AES block cipher. The internal complex round function for encryption is effectively inverted for use in the decryptor. The Roundkeys are identical, but are required in reverse order [1]. The inverse round function for decryption is significantly more complex than that for encryption, and so an AES decryptor will always be both bigger and slower in hardware than its matching AES encryptor [2].

Fig. 3 shows the AES rounds and their key sizes. AES uses several rounds in which each round is made of several stages [6]. Fig. 4 shows the structure of each round where a data block is transformed from one stage to another. To provide security, AES uses four types of transformations: substitution (SubBytes), permutation, mixing, and key-adding.


128-bit plaintext
AES
Round keys
Pre-round transformation
K_0
Round 1
K_1
Round 2
K_2
Round N_r (slightly different)
K_{Nr}
Key expansion
Cipher key (128, 192, or 256 bits)
Relationship between number of rounds and cipher key size
128-bit ciphertext

Nr	Key size
10	128
12	192
14	256

Figure 3: AES rounds

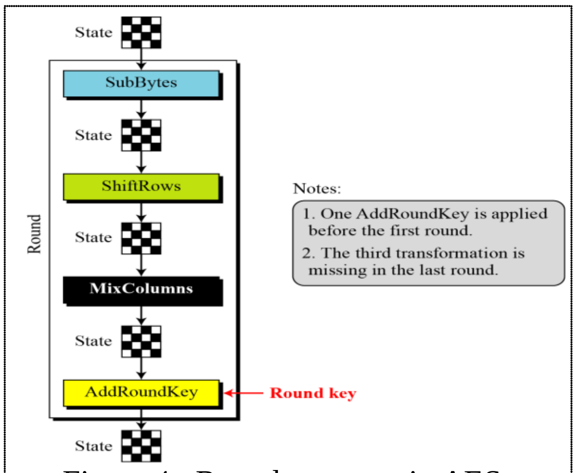

State
SubBytes
State
ShiftRows
State
MixColumns
State
AddRoundKey
Round key
State
Round
Notes:
1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

Figure 4: Round structure in AES

Modes of operation have been devised to encipher text of any size using AES. AES may use any of Electronic Codebook mode (ECB), Cipher Block Chaining (CBC) or Counter (CTR) [7] modes of operation. In CBC mode, AES encrypts data by splitting it up into 128-bit blocks, and putting each block in turn directly through the algorithm [8][9].

III. PROPOSED PIPELINED DATA PARALLEL MODEL OF AES APPROACH

In this section we present the pipelined data parallel model of the AES approach. We also develop an MPI-based algorithm to be used with a cluster of workstations (COW). The parallelism is two dimensional. The first dimension is AES enter-stage (pipelining/intra-pipeline parallelism) and the second dimension is at data level (data-level parallelism). Data level parallelism is an inter-pipeline level parallelism. The parallel implementation of AES decreases the time needed for encryption and decryption processes. Fig. 5 shows the proposed pipelined data parallel model of AES approach. It implements the AES architecture shown in Fig. 2 but in parallel.

A. Data Parallelism

The proposed approach exploits data independence among data blocks to implement data level parallelism. In data level parallelism data is divided into N blocks which are encrypted/decrypted in parallel. To implement the software parallelization of AES we choose the ECB mode of the AES because ECB uses one block at a time with no feedback, no preserved state for previous block are needed, and repetitive plaintext blocks produce repeated cipher blocks. In Fig. 5, data parallelism is implemented by scattering the data to k pipelines (processor). Hence, each pipeline executes the same tasks but on different data (Single Instruction Multiple Data – SIMD parallel architecture).

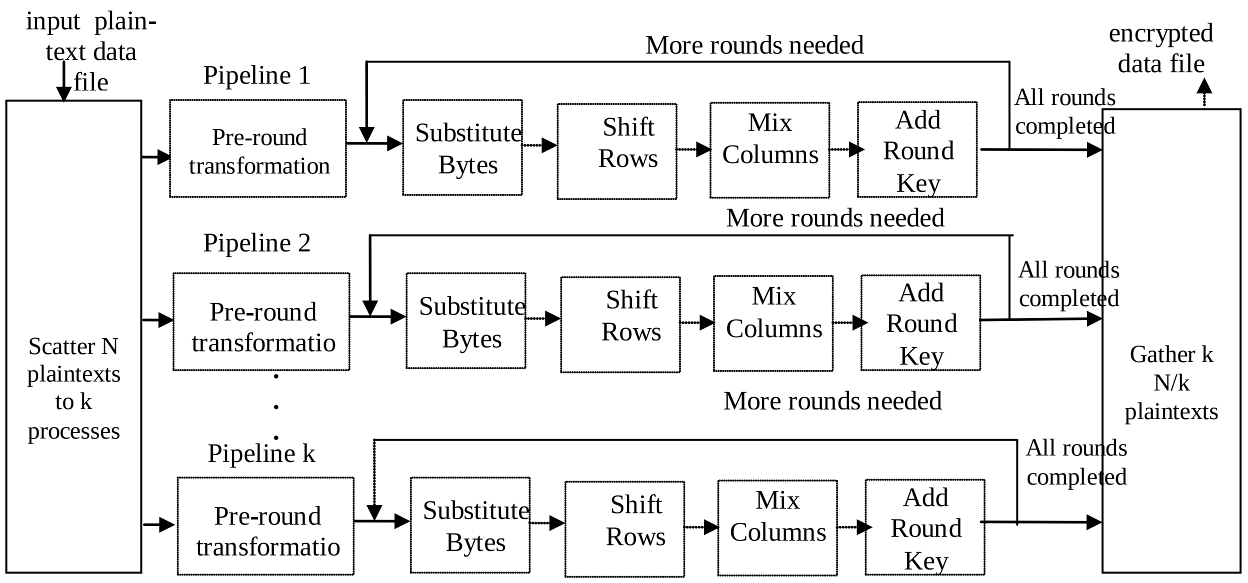


Figure 5: The proposed pipelined data parallel model of AES approach

B. **Pipelining**

Pipelining parallelism exploits the availability of several processes to execute different stages of different data blocks in parallel. Fig. 5 (the proposed approach) shows k 5-stage pipelines. The stages of the pipeline are:

- Pre-round transformation (initializes the rounds)
- Substitute Bytes (the first AES transformation)
- Shift Rows (the second AES transformation)
- Mix columns (the third AES transformation)
- Add round key (the fourth AES transformation)

Each stage in each pipeline processes a plaintext and forwards the result to the next stage. The last stage checks if the plain text has been processed the required number of rounds. If the plaintext has been processed the required number of rounds then it is done. Otherwise, the plaintext is fed back to the second stage. The pipeline is simulated using processes. Each stage in the pipeline is one process in the cluster of workstations. As the figure shows, each pipeline processes N/k plaintexts. The first pipeline processes the first N/k plaintexts, the second pipeline processes the second N/k plaintexts, etc. where N is the total number of plaintexts to be encrypted/decrypted.

C. **Performance Evaluation of the Proposed Model.**

In order to evaluate the performance of the proposed parallel architecture of AES, the following assumptions are made:

- number of plaintexts is N
- number of rounds is NR
- time to scatter one plaintext from a process to a process is $T_{scatter}$
- time to gather a plaintext by a process from a process is T_{gather}
- time of pre-round transformation is T_{pre}
- time for one round is T_{round}

Then the sequential time of AES is:

$$T_1 = N*T_{pre} + N*NR*T_{round} \tag{1}$$

The parallel time of the algorithm (assuming all pipeline stages take the same time) is:

$$T_{par} = (m + k)*T_{scatter} + (N*T_{pre} / k) + N*NR*T_{round} / (4*k) + (m + k)*T_{gather} \tag{2}$$

where m is the size of the message sent/received by the scatter/gather operations to/from each pipeline. Here m = N/k.

In Equation 2, the term $(m + k)*T_{scatter}$ represents the time needed to scatter N plaintexts by a process to all processes. The first stage of each pipeline receives N/k plaintexts. The cost of scatter operation is proportional to the size of the message (measured by the number of plaintexts, N). Similarly, the term $(m + k)*T_{gather}$

represents the time needed to gather N plaintexts by a process from all processes. The last stage of each pipeline sends N/k plaintexts. The cost of the gather operation is proportional to the size of the message (measured by the number of plaintexts, N). Equation 2 assumes that all the last four stages of the pipeline take the same tame.

The speedup (for very large N >> k) is:

$S_p = T_1/T_{par} = 4*k$ (3)

This is an ideal speedup, i.e., when the sequential part of the program is negligible compared to the parallel part. In this case, the program is perfectly parallel. The speedup in case of using one single pipeline (k=1) is:

$S_p = 4$ (4)

The efficiency (for very large N >> k) is:

$E_p = S_p/p = 1$ (5)

Where p is the number of processor in the parallel machine. Here p = 5k since each pipeline has 5 stages and we have k pipelines.

The communication to computation overhead is:

$T_{comm/comp} = ((m+k)*T_{scatter} + (m+k)*T_{gather}) / ((N*T_{pre} / k) + N*NR*T_{round}/(4*k))$ (6)

D. MPI-based Algorithm of the Pipelined Data Parallel Model of AES

Algorithm 1 shows the MPI-based algorithm of the proposed pipelined data parallel model of AES. The algorithm takes the data file to be encrypted “data_file” as input and outputs the encrypted data file “encrypted_data_file”. Step 1 of the algorithm partitions the data file into N 128-bit plaintexts (data blocks). Step 2 configures the cluster of workstations into a pipelined data parallel architecture as shown in Fig. 5, i.e., k pipelines with 5-stages each. Each process of the architecture is identified as process (i,j) where i is the pipeline index and j is the stage index in that pipeline. Step 3 scatters (distributes) the N plaintexts to the k pipelines. Each pipeline is assigned N/k plaintexts. In step 4 each pipeline encrypts its assigned plaintexts in a pipelined fashion. The while loop at step 4.2 implements the encryption of the assigned N/k plaintexts the required number of rounds (required number of rounds). Step 4.2.1 ensures that each stage of the pipeline executes its assigned task. The first stage (pre-round transformation stage) reads a plaintext from its input buffer, does a pre-round transformation to the plaintext, and then sends the plaintext to the input buffer of the next stage. Each of stages 2, 3, 4, and 5 reads a plaintext from its input buffer, executes its corresponding transformation to the plaintext, and then sends the plaintext to the next stage. Stage 5 checks if the plaintext has been processed the required number of rounds. If so, the plaintext is stored in the output buffer of the stage. Otherwise, the plaintext is sent to the second stage. Finally, step 5 of the algorithm gathers all encrypted plaintexts from the output buffers of the fifth stage of all pipelines into the encrypted data file.

```
Algorithm Encrypt ( Input: File: data_file,
                    Output: File: encrypted_data_file )
Begin
1.    Partition data_file into N 128 bit plaintexts (data blocks);
2.    Create k pipelines (each pipeline is 5 stages);
3.    // Scatter the N plaintexts to K pipelines, input buffer of pre-round
      // transformation stage receives N/k plaintexts
      process (0,0) sends N/k blocks to process (i,0)  where 0< i < k;
4.    // All pipelines encrypt k plaintexts in parallel
     Parallel for pipeline i where   0< i < k
     4.1 number of processed rounds = 0;
     4.2  while ( pre-round transformation stage input buffer is not empty &&
                number of processed rounds != required number of rounds)
      4.2.1 parallel case (pipeline stage){
            pre-round transformation stage:
                read a k plaintext from pre-round transformation stage input buffer;
                do pre-round transformation to the plaintext;
                Send the plaintext to Substitute Bytes stage input buffer;
            Substitute Bytes stage:
                Receive a plaintext from pre-round transformation stage;
                Do substitute bytes transformation to the plaintext;
                Send the plaintext to Shift Rows stage input buffer;
            Shift Rows stage:
                Receive a plaintext from Substitute Bytes stage;
                Do Shift Rows transformation to the plaintext;
                Send the plaintext to Mix Columns stage input buffer;
            Mix Columns Stage:
                Receive a plaintext from Shift Rows stage;
                Do Mix Columns transformation to the plaintext;
                Send the plaintext to Add Round Key stage input buffer;
            Add Round Key Stage:
                Receive a plaintext from Substitute Bytes stage;
                Do Shift Rows transformation to the plaintext;
                 if (number of processed rounds !=  required number of rounds)
                     Send the plaintext to Mix Columns stage input buffer
                 else
                     Store plaintext in output buffer of Add Round Key Stage
          } // end parallel case
5.    // Gather N plaintexts from k processes into encrypted_data_file
      Process (i,4) sends N/k blocks to process (0,0)  where 0< i < k
End algorithm
```

Algorithm 1: MPI-based algorithm of the pipelined data parallel model of AES

IV. PERFORMANCE MEASUREMENTS

In this section we measure the performance of the proposed model using the performance metrics and their corresponding equations discussed in section III.C. We vary various input parameters including:

- input data file size (number of plaintexts (N))
- number of created pipelines (k)
- cost of various communication/computation operations ($T_{scatter}$, T_{gather}, T_{pre}, T_{round})
- number of rounds (NR)

We measure the performance of the parallel model and compare it with the performance of using a single processor. Performance metrics which are used include speedup, efficiency, and communication to computation time ratio.

The first measurement studies the effect of varying the input file size (number of plaintexts N) on the speedup for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 1. In Table 1, both $T_{scatter}$ and T_{gather} are 1 unit of time each. This value is comparable to the cost of the first stage of the pipeline. This case is reasonable for simulating networks of workstations (NOW) clusters with fast communication systems. Fig. 6 shows the results. As the results show, speedup increases as number of plaintexts increase. This is a result to an increased available data parallelism in the application. In addition, speedup in very low compared to the theoretical speedup (5k) for small N. This is due to not enough data parallelism in the application which leads to some of the pipelines being idle. Also, for very large values of N, speedup reaches a maximum value (upper bound value of speedup) that is not exceeded even if we increase N. This is due to the fast that the maximum theoretical speedup is 5k. But due to communication overhead, the proposed architecture does not reach this value.

Table 1: Value settings of input parameters used in first, third and fifth experiments

Input parameter	Value
$T_{scatter}$	1
T_{gather}	1
T_{pre}	1
T_{round}	10
NR	10

Table 2: Value settings of input parameters used in second, fourth and sixth experiments

Input parameter	Value
$T_{scatter}$	10
T_{gather}	10
T_{pre}	1
T_{round}	10
NR	10

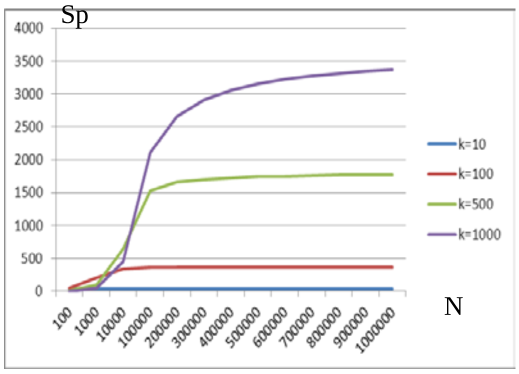


Figure 6: Speedup as a function of number of plaintexts for high speed communication COWs

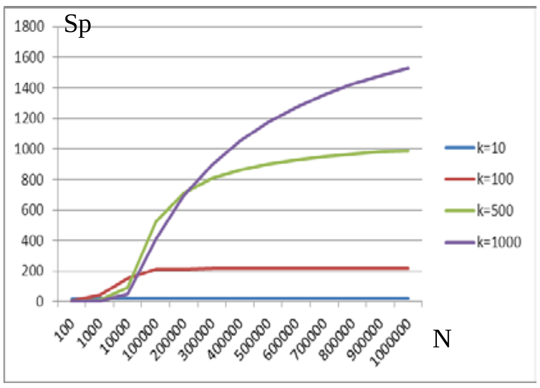


Figure 7: Speedup as a function of number of plaintexts for low speed communication COWs

The second measurement is similar to the first measurement. But we increased the cost of communication to be significantly large compared to cost of transformations (both $T_{scatter}$ and T_{gather} are 10 units of time each). This case is reasonable for simulating networks of clusters (NOW) with slow communication systems. The measurement studies the effect of varying the input file size (number of plaintexts N) on the speedup for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 2. Fig. 7 shows the results. The results show same conclusions reached in the first measurement. In addition, results show that the speedup is less than the speedup of first measurement. This is due to increased communication overhead. This increase is due to the high latency in communication operations as reflected by the large values of $T_{scatter}$ and T_{gather} . There is no communication overhead in case of a single processor system. In parallel systems, the parallel time increases as the communication overhead increases which leads to a decreases in the speedup.

The third measurement studies the effect of varying the input file size (number of plaintexts N) on the efficiency for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 1. This case is reasonable for simulating networks of workstations (NOW) clusters with fast communication systems. Fig. 8 shows the results. As the results show, efficiency increases as N increases. This is due to the increase in speedup as N increases as explained in the first experiment. In addition, for small N, efficiency increases as number of pipelines decreases for the same value of N. This is due to the smaller utilization of pipelines due to the lack of data parallelism. For large values of N, efficiency is approximately same for different values of K for same N. This is due to the full utilization of pipelines due to the increased availability of data parallelism.

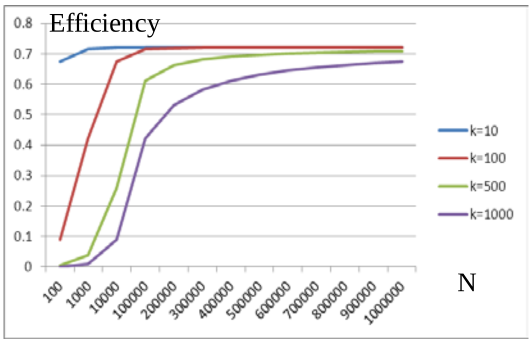


Figure 8: Efficiency as a function of number of plaintexts for high speed communication COWs

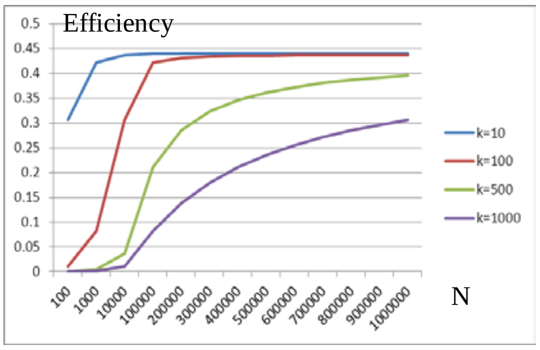


Figure 9: Efficiency as a function of number of plaintexts for low speed communication COWs

The fourth measurement studies the effect of varying the input file size (number of plaintexts N) on the efficiency for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 2. This case is reasonable for simulating networks of workstations

(NOW) clusters with slow communication systems. Fig. 9 shows the results. The results show same conclusions reached in the first measurement. In addition, results show that the efficiency is less than the efficiency of third measurement. This is due to the smaller speedup in case of low speed communication networks of workstations clusters than speedup in case of high speed communication networks of workstations clusters where efficiency = speedup/number of processors.

The fifth measurement studies the effect of varying the input file size (number of plaintexts N) on the communication to computation ratio for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 1. This case is reasonable for simulating networks of workstations (NOW) clusters with fast communication systems. Fig. 10 shows the results. As the results show, communication to computation ratio decreases as N increases. Both communication and computation times increase as N increases. But computation increases faster than communication. Communication increases with a factor of (K/ N + k) while computation increases with a factor of max(N/k , N*NR/4k). Also, communication to computation ratio is higher for larger values of k for same N. This is due to more data parallelism availability in case of smaller K. For small N there is not enough computations to be assigned to pipelines. As N increases the difference in communication to computation ratio for different values of k becomes less significant. This is due to increased available data parallelism for larger values of N which is used by the increased number of pipelines.

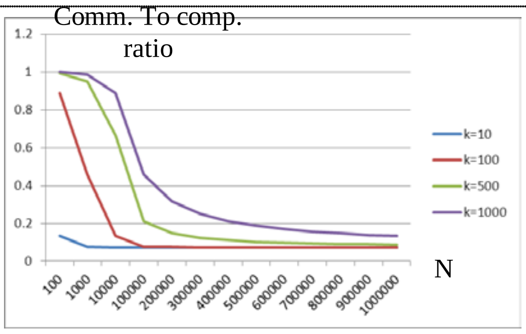


Figure ١٠: Communication to computation ratio as a function of number of plaintexts for high speed communication COWs

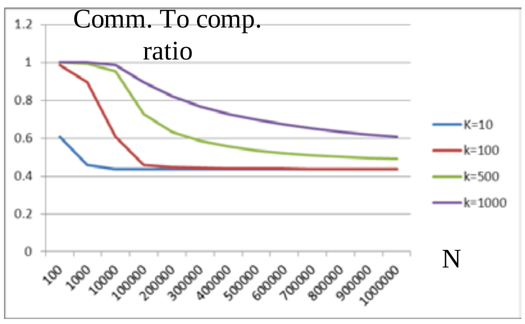


Figure ١١: Communication to computation ratio as a function of number of plaintexts for low speed communication COWs

The sixth measurement studies the effect of varying the input file size (number of plaintexts N) on the communication to computation ratio for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 2. This case is reasonable for simulating networks of workstations (NOW) clusters with slow communication systems. Fig. 11 shows the results. The results show same conclusions reached in the first measurement. In addition, communication to computation ratio in this measurement is higher than that in the fifth measurement. This is a straight forward result of the increased communication overhead in slower communication systems of networks of workstations (NOW) clusters than communication overhead in faster communication systems of networks of workstations (NOW) clusters.

The last measurement studies the effect of varying the cost of $T_{scatter}$ and T_{gather} (speed of the communication network of the network of workstations) on the speedup for different number of pipelines (k). We use the values of different input parameters and communication and computation operations costs as shown in Table 3. Fig. 12 shows the results. As the results show, speedup decreases as the speed of communication systems in networks of workstations (NOW) clusters decreases (cost of $T_{scatter}$ and T_{gather} increases). This is due to increased communication overhead when cost of $T_{scatter}$ and T_{gather} increases. In addition, speedup of smaller k is smaller than speedup of larger k for same values of $T_{scatter}$ and T_{gather}. As $T_{scatter}$ and T_{gather} increase, the difference in speedup for different values of K becomes less significant.

Table 3: Value settings of input parameters used in last experiment

Variable	Value
T_{pre}	1
T_{round}	10
N	500000
NR	10

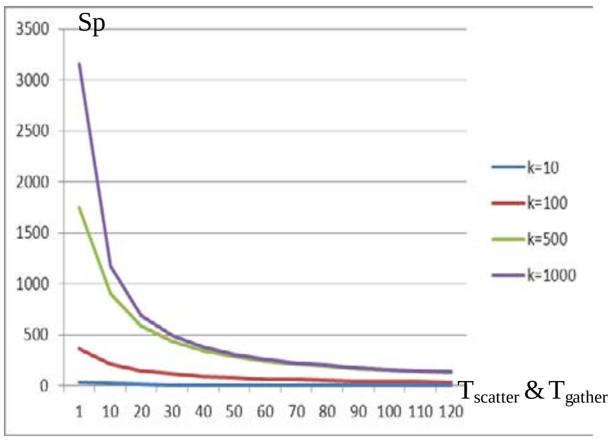


Figure ١٢: Speedup as a function of cost of $T_{scatter}$ and T_{gather}

V. CONCLUSION

This paper developed a pipelined data parallel model of AES. The parallelism in the algorithm is two dimensional. The first dimension is AES enter-stage (pipelining) and the second dimension is data parallelism. The model used the ECB mode in encryption/decryption algorithm to implement the parallelization at data level. The paper also developed an MPI-based pipelined data parallel model of AES algorithm to be used with a cluster of workstations (COW).

We measured the performance of the proposed model by simulating the model with various input parameters (input data file size, number of processes, communication/computation operation execution time, etc.) and measuring the corresponding performance. Performance metrics included speedup, communication to computation ratio and efficiency. Results show that performance obtained by the developed model is superior to parallel implementations of AES which include only data parallelism or pipelining.

VI. REFERENCES

[1] Advanced encryption standard, National Institute of Standards and Technology Std., Nov. 2001.
[2] Elbirt J., Yip W., Chetwynd B., and Paar C.: An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 9, No. 4, pp. 545–557, Aug. 2001.
[3] Advanced Encryption Standard (AES), Federal information processing standards publication 197, Nov. 26, 2001.
[4] Chodowiec P., Khuon P., and Gaj K.: Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining, Proceedings of the 2001 ACM/SIGDA ninth international symposium on field programmable gate arrays, pp. 94-102.
[5] Alam M., Badawy W., and Jullienn G.: A Novel pipelined threads architecture for AES encryption algorithm, Proceedings of the IEEE international conference on application-specific systems, architectures, and processors, 2002.
[6] Rodriguez-Henriquez, Saqib N., and Diaz-Perez A.: A 4.2 Gbit/s single-chip FPGA implementation of AES algorithm, Electronics letters, Jul. 2003, pp. 1115- 1116.
[7] Rijmen V.: Efficient implementation of the Rijndael S-box. [Online]. Available at: www.esat.kuleuven.ac.be/~rijmen/rijndael/sbox.pdf
[8] Jarvinen K., Tommiska M., and Skytta J.: A Fully pipelined memoryless 17.8 Gbps AES-128 encryptor, Proceedings of the 2003 ACM/SIGDA eleventh international symposium on field programmable gate arrays, pp. 207-215.
[9] Specification for the Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, Nov. 26, 2001.

ACKNOWLEDGMENT

This research was partially funded by the deanery of scientific research at the Islamic University of Gaza in Gaza, Palestine.