ACCELERATION OF AES ALGORITHM BASED ON CUDA- ARCHITECTURE MULTI-THREADED GPU USING IMPROVED T-TABLE AES ALGORITHM

[1]Manjit Jaiswal, [2]Shivam Pandey, [3]Yengkhom Bidyapati Devi
[1]Assistant Professor, [2] Student, [3]Student
[1]Computer Science & Engineering Department,
[1]School of Studies & Engineering Technology Guru Ghasidas Vishwavidhyalaya, Bilaspur, India

Abstract: GPU can be viewed as modern CPU as of its capacity to vastly outperforming CPU while becoming more general purpose The Advanced Encryption Standard (AES) is used as a security mechanism in wide spheres now. However, there is still scope for further improvement of its execution efficiency. Since the graphic processing unit (GPU) with potent ability of parallel computing as it has found application in general purpose of computation, people have tried to use it to faster execution time in various cryptographic algorithms. In order to improve upon the security and throughput aspect of AES algorithm, this paper proposed an improved T-table AES algorithm (ITAES) based on the CUDA architecture and use of T-Box. The proposed ITAES algorithm help us to achieve optimal throughput, increased processing speed and an easy-to-implement CUDA algorithm with minimized lookup time and a parallel approach of encryption algorithm.
Keywords: CUDA, GPGPU, GPU, AES, Electronic Codebook, Cipher Feedback, Improved T-Table AES algorithm (ITAES) parallel computing.

I. INTRODUCTION

The world goes online and so online information is a key to any organization and any of the institution maintaining such information on the cyber world. Furthermore in [7] they have described how security aspect became so important so as to protect the accessibility, authenticity of these data. This was key to development of several security mechanisms like AES, DES, RSA and so on.
As the technology aspect is growing so is our data size and to such a massive volume of data encryption with CPU is not that efficient barring to its limited speed. Here comes the role of GPU as a rescuer.
In [16] they have described that GPU is around 1000 times faster than CPU, if utilized properly, it can be used for encryption of massive volume of data within limited frame of time. In earlier days it was more or like meant for graphic display and graphic optimization but with constant demand it has now been developed along the line of GPGPU that is for the general-purpose computing. In [13] they have described earlier efforts like OpenGL and DirectX which was later followed by NVIDIA CUDA which let programmers ignore the underlying graphical concept in wake of more common general-purpose computing concepts.
In the upcoming sections, we have broadly tried to absorb the in-depth of AES algorithm as sub-parts of this section describes the GPU and its CUDA version as well as give a brief overview of traditional AES algorithm. In section 2 we majorly cover an extensive overview of related works in this field which led us to refine our research. In section 3 we proposed details of our proposed work while section 4 majorly covers the pseudo-code of proposed algorithm.

I.I OVERVIEW OF AES

In [14] they have described AES as Advanced Encryption Standard which is also termed as Rijndael and is a specification of encryption of electronic data and was developed by U.S. National Institute of Standards and Technology (NIST) in 2001. In [8], they have described the traditional AES structure which invariably depicted as Figure 1.

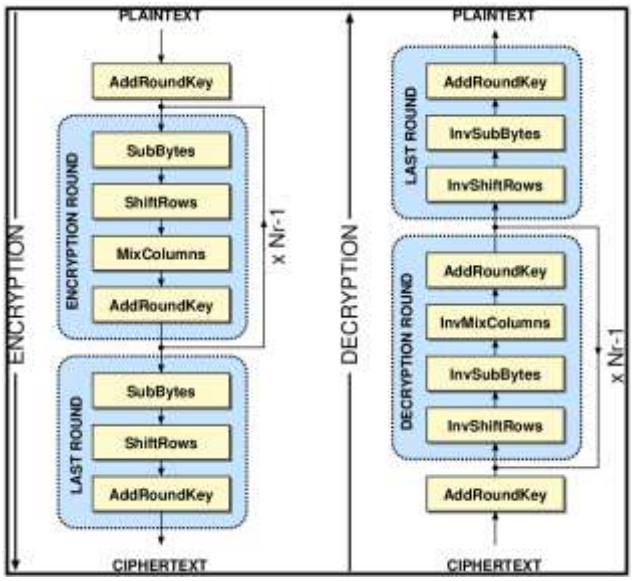

Figure 1: Process in AES algorithm

The various involved steps are:
-Sub-Bytes(): replace one byte to another byte by a S-box.
-Shift-Rows(): a simple substitute.
-Mix-Columns(): an arithmetic substitute using $GF(2^8)$.
-AddRoundKey(): a Round Key is added to the *State* byte simple bitwise XOR operation.

I.II GPU, GPGPU and CUDA

GPU is a graphical processing unit which allow us to display the high-quality graphics on our PC, a much sought after demand of this age. In design it is very similar to CPU (Central Processing Unit), that is it contains a single chip processor. However, the GPU contains a large number of cores in comparison to CPU which is just a bundle of few cores.GPU was initially developed to calculate 3D functions as these types of specific tasks are of heavy load on the CPU, the GPU effectively share the load by doing heavy part of the execution. Initially GPU was designed for graphical work load, it has now evolved into more general purpose with ease to program.
CPU houses only few cores and that too for sequential processing which is waste of resource and time while a GPU houses a number of cores so as in synchronization to carry out processing in a parallel manner in a fast.

The normal architecture of GPU Requires integration and synchronization of various components of computer together like CPU, main memory control transfer in order to move data from GPU to CPU efficiently

Now the major question is how does GPU get an edge over CPU in computing speed. The idea is to use CPU-GPU in a heterogeneous manner where a part of computive intensive code runs on GPU while most of the code whose processing demand serial execution runs on CPU. It seems to user as if the application is faster because it is using the optimal features of the GPU to improve its own performance.

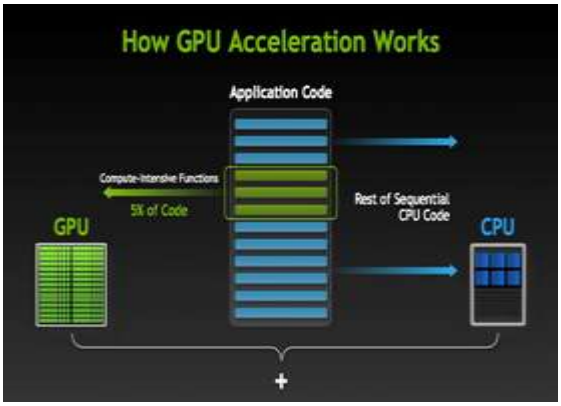

Figure 2: Distribution of code between GPU and CPU

Initially GPU functionality has, traditionally, been very limited in scope for it was a CPU for graphic workload. This result in its use, for many years as an accelerator for graphics pipeline. The processing capacity of the GPU is limited to independent vertices and fragments. But this processing can be done in parallel using the multiple cores being housed by the GPU. This proves to be more effective when the programmer wants to get processed a task involving vertices or fragments in the similar way. In this sense, GPUs are processors that are highly SIMD in nature operating in parallel by running a single kernel code on many data stream at the same time. A stream is simply a set of records or data stream that demand similar computation. Thus data parallelism is invoked. We can have multiple inputs and outputs at the same time, but it demand a well adequate memory organization. This enables the GPU to have Data-parallel processing, as the GPU architectures are ALU-heavy and contain multiple vertex & fragment pipelines. This results in tremendous computational power for the GPU. But this computational capability was unutilized in the earlier evolutionary stages of the GPU.

Using the GPU for processing non-graphical entities is known as the General Purpose GPU or GPGPU. Traditionally GPU was used to provide better graphical solutions for available environments. If we use GPU for computationally intensive tasks, then this kind of work is known as GPGPU. It is used for performing complex mathematical operations in parallel for achieving low time complexity.

As a result of the irresistible qualities of the GPU an increasing number of programmers are trying to make their application more and more GPGPU oriented. Advantages of GPU programmability are so irresistible that many programmers have begun rewrite their applications using GPGPU. In that way they can make use of high computational capacity of GPU. The biggest hurdle that a programmer faces when first programming a GPGPU is learning how to get the most out of a data-parallel computing environment. GPU parallelism is present at more than one level. Parallel execution on multiple data elements is a fundamental design element of GPGPUs.

Parallelism is obtained be rearranging the data from scalar quantities to vector ones. The vector processors can schedule more than one instruction per cycle. And so, the vectors can be computed in parallel if their elements are independent of each other. This results in a chance for instruction level parallelism, which the GPU can utilize.

CUDA (Compute Unified Device Architecture) provides a platform for parallel computing and Application Programming Interface (API) model created by NVIDIA. It has direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. C, C++ and Fortran programming languages are supported in this platform. This accessibility makes it easier for specialists in parallel programming to use GPU resources.

In [13] they have described slightest advantage of CUDA over other GPU enlisted as Scattered reads – code can read from arbitrary addresses in memory, Unified virtual memory (CUDA 4.0 and above), Unified memory (CUDA 6.0 and above), Shared memory-CUDA exposes a shared memory region that can be shared among threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups, faster downloads and read backs to and from the GPU, full support for integer and bitwise operations, including integer texture lookups. While all other GPU do not provide that ease of programming CUDA came as a great rescuer here as its packages and pre-defined library help us to code the GPU in a much more easy way than earlier GPU making it a more stronger platform for GPU programming.

CUDA memory and Cached Architecture:

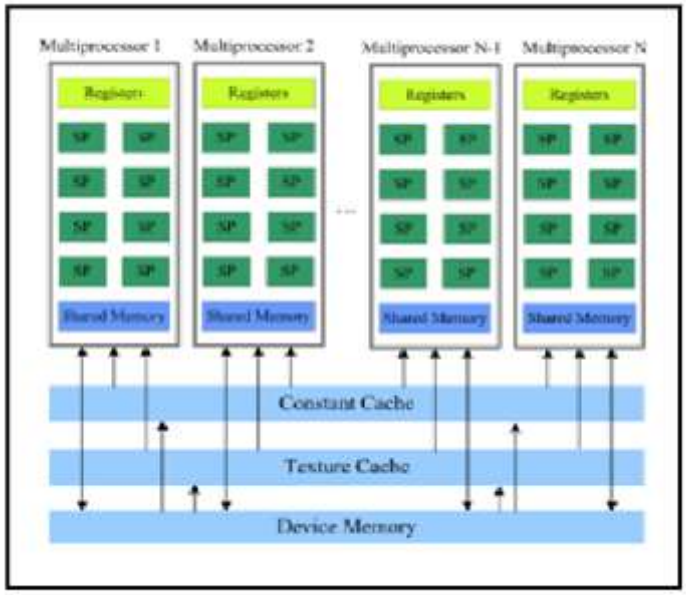

Figure 3: CUDA memory architecture

Global Memory - Global memory is the physical memory on your graphics card. It can be thought of as the amount of memory allotted to the graphics device if we have an integrated Nvidia chipset like the ION. All threads can read and write to Global memory. We can even read and write to Global memory from a thread on the CPU.

Shared Memory- We know that many processors, or multiprocessors are there in GPU, each multiprocessor has a small amount of Shared memory, on the order of about 16KB of memory. Generally it is used as a very quick working space for threads within a block. It is allocated on a block by block basis. For example, you may have three blocks running consecutively on the same multiprocessor. This means that the maximum amount of shared memory of the blocks can be reserved to 16KB/s. Threads within the same block can quickly and easily communicate with each other by writing and reading to the shared memory. Shared memory is 100 times faster than global memory, so it's very advantageous if we can use it correctly.

Texture Memory – A GPU also has texture units and memory which can be taken advantage of in some circumstances. Generally it is read only and cached unlike global memory. If we expect threads to access memory addresses which have some coherence, we might want to consider using texture memory to speed up those memory accesses.

Constant Memory- The CUDA language makes available another kind of memory known as constant memory. As the name may indicate, we use constant memory for data that will not change over the course of a kernel execution. NVIDIA hardware provides 64KB of constant memory that it treats differently than it treats standard global memory. In some situations, using constant memory rather than global memory will reduce the required memory bandwidth.

Thread hierarchy:

Half-Warp- It is a group of 16 consecutive threads. Generally, these threads are executed together. They are aligned.

Warp- A warp of threads is a group of 32 consecutive threads

Block- A block is a collection of threads.

Grid- A grid is a collection of blocks.

II. RELATED WORK

In [5] as described in this paper, AES is implemented based on optimized ANSI C source code provided as a part of OpenSSL, the open source toolkit for SSL/TLS[12]. Its algorithm defines round processes combined into a transformation simply using a lookup table called "T-box" and exclusive-or operation. Letting *a* be the round input, which is divided into each 32 bits, the round output *e* is represented as
$ej = T0[a0;j] \oplus T1[a1;j+1] \oplus T2[a2;j+2] \oplus T3[a3;j+3] \oplus kj$; where *T*0, *T*1, *T*2, and *T*3 are lookup tables and *kj* is the j-th column of a round key. This algorithm includes only four lookup table transformations and four exclusive-or operations.

In [6], they have improved the efficiency of AES algorithm by variable s-box, 200-bit data block and key. The images in jpeg format were also encrypted into the text format. So finally it was concluded that the new improvised algorithm saves a lot of encryption time and provides more security to the encrypted document

In [8] they proposed two schemes for parallel AES encryption implementation with off-line key expansion on shared-memory multi-core architecture. The tasks are balanced partitioned in a cluster.

In [10] a new AES design methodology is proposed through integrating pipeline technology with dynamic reconfiguration on the basis of partial modules similarity in AES encryption/decryption. This dynamic reconfigurable AES design method can process AES encryption/decryption simultaneously in real time and with a high throughput of chip, it enhances the efficiency of hardware resources.

In [11] their implementation is based on optimized ANSI C source code for AES, which is provided as a part of OpenSSL the open source toolkit for SSL/TLS[6]. Its algorithm defines round processes combined into a transformation using look up table called "T-box" and ex-or operation simply.

In [9] the key expansion is done on CPU and the encryption is done on the GPU. The key expansion is done serially and it will slow the GPU down, so that is why we choose to do it on the CPU. To try optimization they set all the threads to use the global memory. In doing this they group all the access to memory, all data will coalesce to permit faster memory read/writes. Access to global memory is done in the initial phase, before processing data. The data is moved into the shared memory where it can be accessed faster.

In [12] although several proposals have stated many advantages of mapping the full round computation (AES block) into each GPU core, these are just a fine-grained approach parallelism optimization.

III. PROPOSED AND IMPROVED T-TABLE AES (ITAES) ALGORITHM:

Based on the above study we now present our improved T-Table AES algorithm which is improved version of previous works and solely based on running it on GPU.

III.I Getting the T-Table from S-Box and constant Matrix

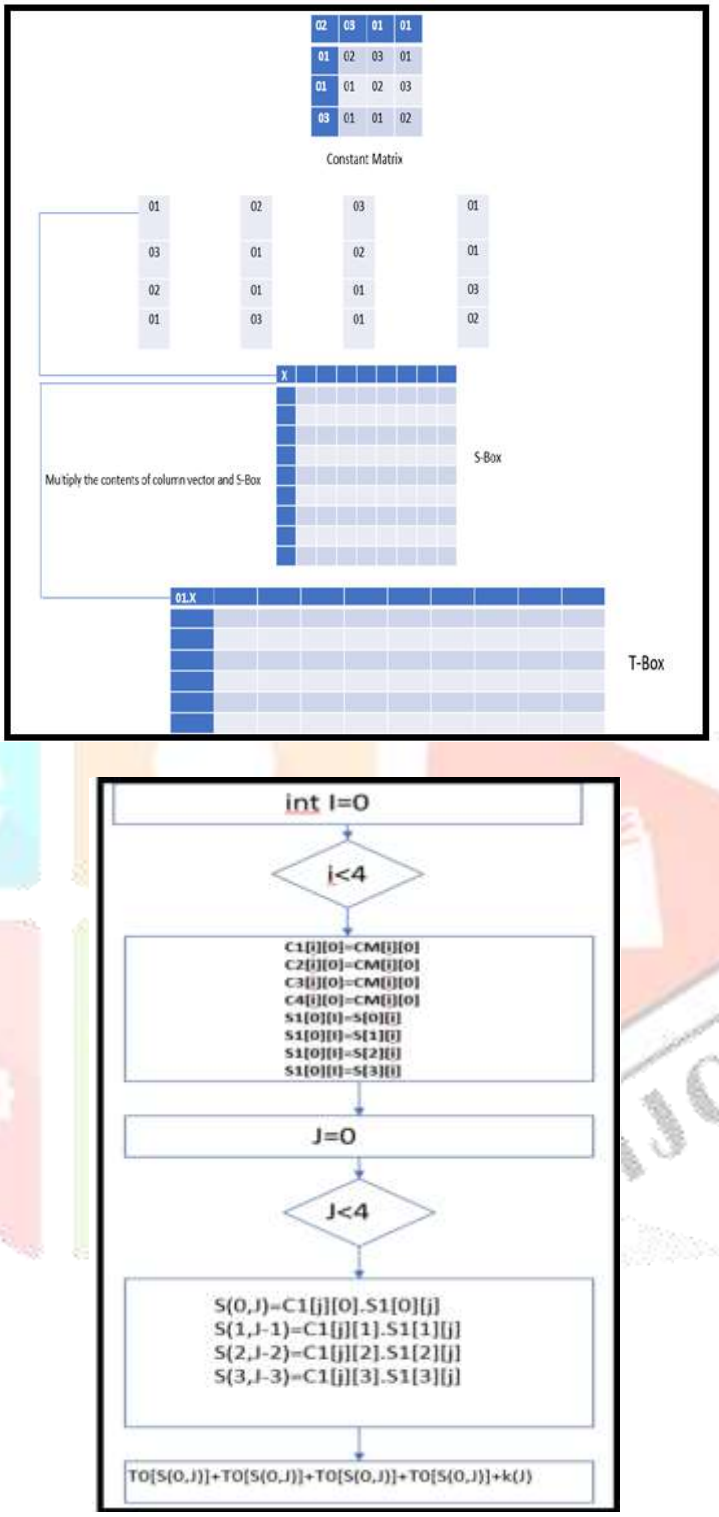

Figure 4: Flowchart to obtain T-Table

III.II Improved T-Table AES (ITAES) algorithm:

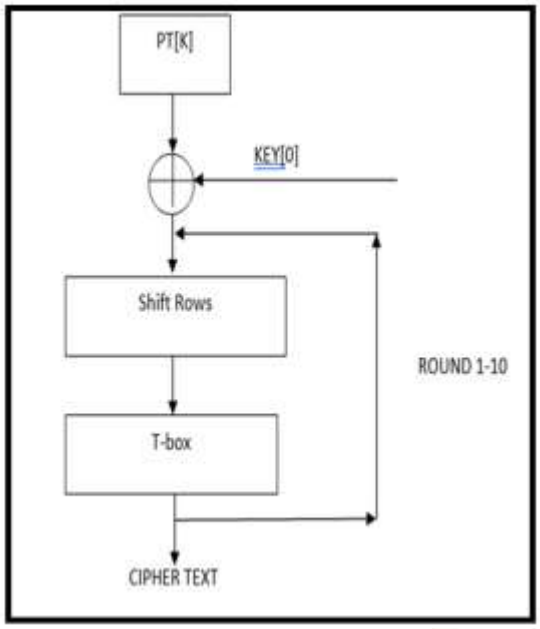

Figure 5: Flowchart of proposed Improved

T-Table AES Algorithm (ITAES)

III.III Acceleration of AES with proposed Imp roved T-Table AES Algorithm (ITAES):

The T-box which is used in the algorithm is eventually derived from multiplying the S-box contents with the constant matrix used in mix- column step used in the traditional AES algorithm. Hence it essentially combines the two steps and help us to accelerate the algorithm. The T-box generation process creates a lookup-table of size 1 kb each and there are 4 T-boxes in total which take up space of 4kb.

Now $a_{i\ j}$ represents the state matrix, $k_{i,j}$ represents round key matrix. . Then we combine the four steps to one expression, which is illustrated as formula 2; Thus, we can represent matrix's multiplication by vector's linear combination. As is showed in formula 3, we define the four tables in the front of formula 2 as T-Box which contains 256 words. Then the expression can be presented as formula 4. When one column of the state is operated, each byte of the column looks up one T-box and is combine into one column of a new state.

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} =$$
$$\left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right) \oplus$$
$$\left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{0,j-2}] \right) \oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{0,j-3}] \right)$$
$$\oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Figure 6: Combining various operations in one instruction

As a result, the implementation of AES only needs 4 T-Box lookups, 4 XORs per round per column and 4KB storage storing these tables; This is very suitable for paralleling on GPU which avoids a large of logical computing. This is about encryption and decryption is similar to it which we do not present here.

$$T_0[x] = \begin{pmatrix} \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[x] \end{pmatrix} T_1[x] = \begin{pmatrix} \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \end{pmatrix} \quad (3)$$

$$T_2[x] = \begin{pmatrix} \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \end{pmatrix} T_3[x] = \begin{pmatrix} \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x] \end{pmatrix}$$

$$\begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus \quad (4)$$

$$T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Each Look-up in the T-box will give us the result of three state in traditional AES in combined form, these are mix-columns, substitute byte and round key addition. This algorithm is further highly suitable to SIMD structure since only one instruction is needed to operate on various data sets.

Further in [1] the 10 round they have removed T-Box since 10^{th} round do not contain mix-column steps, hence for 1-9 rounds they use T-box and for final round S-box is used. How-ever in order to keep the encryption process to a steady rate we really do not need to use a S-Box later as for the 10^{th} round can directly be performed with the help of pre-calculated T-Box. It helps to save a lot of execution time which otherwise could have lost due to control transfer from Host to Device.

III.IV Sample round transform

A sample example has been given in order to clarify the working of the proposed algorithm in much better way. It gives insight into how T-Table works and calculate elements of round 1 as well as the result of one round transform.

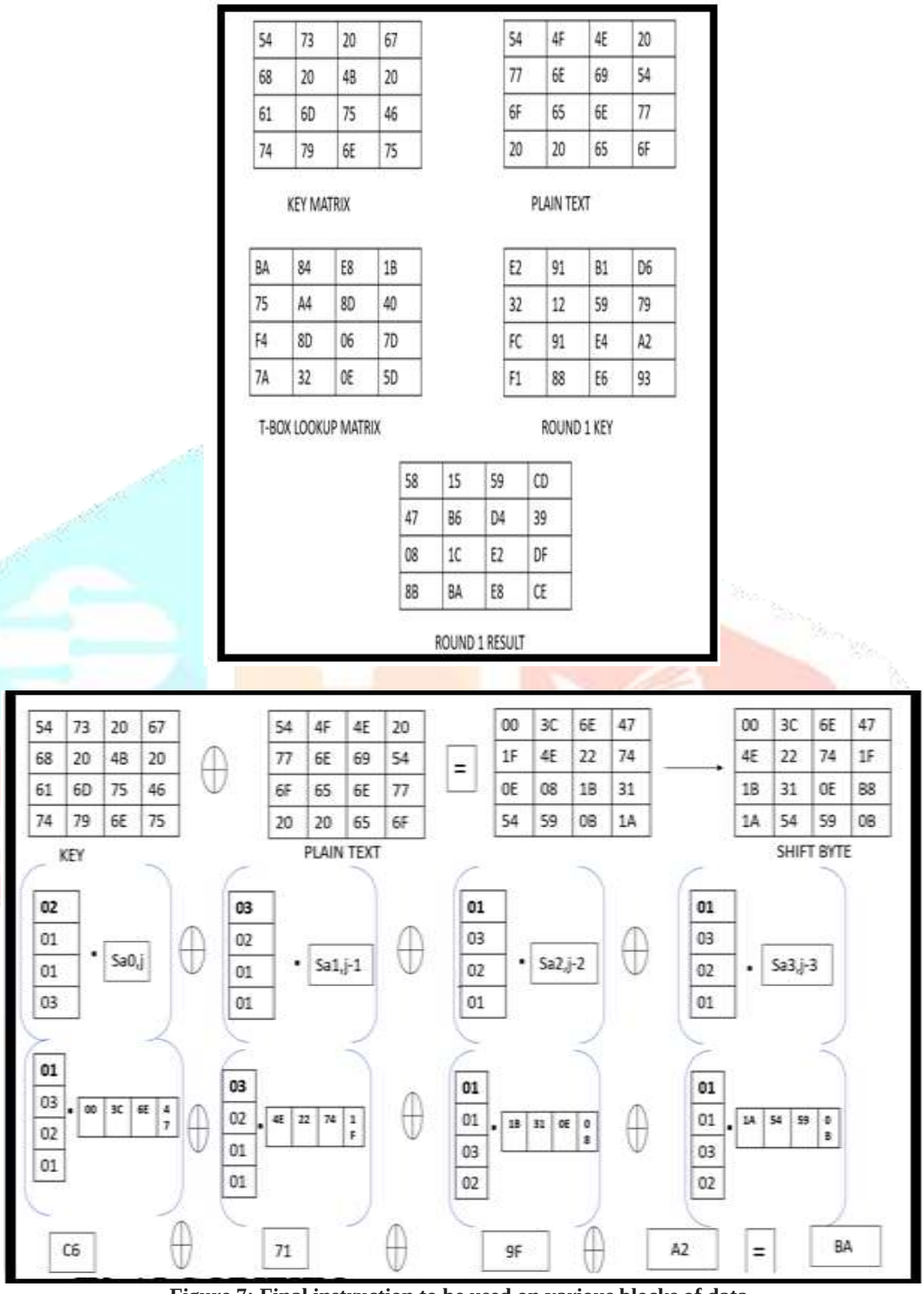

Figure 7: Final instruction to be used on various blocks of data.

IV. ALGORITHM

The algorithm presented there essentially gives you a glimpse of how to divide the blocks, allocate various device memories and host memories, load these memories and invariably call the kernel function to be launched from main() function of the host, do the processing and return the processed data back to the Host for further I/O operations to be performed.

Step 1: Input plaintext of n bit in an array PT[k]

Step2: int Nb = Ceil Of(n/128),k=0;

Step 3:int i, j, PE[20][20]

Step4: for (i=0; i<nB; i++)

Step4.1: for (j=0; j<128; j++)

Step4.1.1: PE[i][j] = PT[j+k]

Step4.1.2: k++

Step 5: Allocate memories to various device arrays.

Step 6: All the Arrays are inserted as input into cores of GPU including T-boxes through cudaMemCopy in-built function with direction HostToDevice.

Step 7: Write a kernel program for AES that will be replicated to all the cores.

Step 8: Call the kernel program as AESROUND <<<8, 16, 16 >>>(pl_size);

Step 9: Through this kernel program AES is applied on all cores in parallel.

Step 10: Synchronise the data using cuda Device Synchronise function.

Step 11: The data is retrieved using CudaMemCopy function but this time direction parameter changes to DeviceToHost.

Step 12: Then display the output using main function on Host.

V. CONCLUSION:

The proposed AES Algorithm ITAES improves the efficiency of AES Algorithm by using T-box and highly parallel CUDA architecture. The proposed Encryption Algorithm ITAES as per expectation will produce best results in terms of Encryption time and Security when compared to the Original AES Algorithm and Enhanced AES Algorithm. It helps us to ensure that the data encrypted can withstand brute force attacks. The presented the most efficient currently known approaches in encryption and decryption of messages with AES on programmable graphics processing units. While the study suggested that the traditional graphics hardware architectures could now be compared with optimized sequential solutions on the CPU, it definitely proved that the novel unified architecture of GPU can now be used a cryptographic acceleration board.

Since here we are majorly concerned with the acceleration of AES algorithm, this led to propose our future work, which works on the scope of further improving security aspect through increased key-size as well as working on granularity in order to further increase its throughput rate. This led us to encryption technique that possess the characteristics of high throughput rate as well as improved security aspect and thus making it immune to attacks.

REFERENCES

[1] Jianwei Ma, Xiaojun Chen, Rui Xu, Jinqiao Shi "Implementation and Evaluation of Different Parallel Designs of AES Using CUDA", 2017 IEEE Second International Conference on Data Science in Cyberspace.

[2] Ahmed Awadalla Abdelrahman, Mohamed Mahmoud Fouad, Hisham Mohamed Dahsan "Digital Image Processing and Computer Graphics volume: 15 j number: 3 j 2017 j September.

[3] Chinnakandukuri Paul Pramod, Manjit Jaiswal "An Advanced Sequential & Parallel AES Algorithm Using Swap Method on Multicore CPU and GPU in Network Security" International Conference on Computing Communication and Automation (IEEE) 2017, Galgotias University, Greater Noida, 5th – 6th May 2017.

[4] Ritambhara, Alkagupta, Manjit 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(ICCCA), IEEE Conference, 2017.

[5] VikasKaula, Bhushan Nemade, Dr. Vinayak Bharadi, Dr. S. K. Narayan khedkar, "Next Generation Encryption using Security Enhancement Algorithms for End to End Data Transmission in 3G/4G Networks", 7th International Conference on Communication, Computing and Virtualization 2016.

[6] ShivedraShekhar, Pushkar Singh, Manjit Jaiswal "An Enhanced AES Algorithm Based on Variable SboxAnd 200 Bit Data Block" International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization) Vol. 4, Issue 4, April 2016

[7] Richa Gupta, Sunny Gupta, Anuradha Singhal, "Importance and Techniques of Information Hiding: A Review", International Journal of Computer Trends and Technology (IJCTT) – volume 9 number 5– Mar 2014

[8] K. Rahimunnisa, P. Karthigaikumar,N. Anitha Christy, S. Suresh Kumar, J. Jayakumar, "Parallel Sub-Pipelined Architecture for high Throughput", Central European Journal of Computer Science, December 2013, Volume 3, Issue 4, pp 173–186.

[9] Ali A. Abed, Ali A. Jawad, "FPGA Implementation of a Modified Advanced Encryption Standard Algorithm", The First International Conference of Electrical, Communication, Computer, Power and Control Engineering ICECCPCE'13/December17-18, 2013

[10]Zhiyong Guo, Guangjun Li, Yang Liu, "Dynamic Reconfigurable Implementations of AES algorithm using pipeline structure".

[11]Keisuke Iwai, Naoki Nishikawa, TakakazuKurokawa, " Acceleration of AES encryption on CUDA GPU",International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2839 (print) ISSN 2185-2847 (online),Volume 2, Number 1, pages 131–145, January.

[12] Chakchai So-In, SarayutPoolsanguan, ChartchaiPoonriboon, KanokmonRujirakul, ComdetPhudphut,"Performance Evaluation of Parallel AES Implementations over CUDAGPU Framework", Maung, KhonKaen, Thailand, 40002.

[13] https://en.wikipedia.org/wiki/General purpose computing on graphics processing units.

[14] https://developer.nvidia.com/cuda-gpus.

[15] https://en.wikipedia.org/wiki/AdvancedEncryptionStandard

[16] Jayshree Ghorpade , Jitendra Parande , Madhura Kulkarni , Amit Bawaskar," Gpgpu processing in CUDA architecture", Advanced Computing: An International Journal (ACIJ), Vol.3, No.1, January 2012

[17]Ijaz Ali Shoukat, Kamalrulnizam Abu Bakar and Mohsin Iftikhar, "A Survey about the Latest Trends and Research Issues of Cryptographic Elements", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011.