GPU File System Using AES Algorithm

Bhushan Marathe

Abstract

Computer systems' information security is a key concern, and its significance has grown recently. The performance of systems with several users running concurrent applications is, however, constrained by the computing demands of cryptographic methods, which can be particularly taxing on machines without accelerators. On the other hand, Graphics Processor Units (GPUs) have progressed from specialized computer graphics accelerators to generally available co-processors [1]. The amount of data generated every minute has increased significantly because of technological improvements; this data needs to be stored or transported while maintaining a high level of security. Today's military and armed forces significantly rely on computers to store enormous amounts of crucial and secret data that are crucial to the country's security. Although it provides a high level of security and is at the core of practically all applications today, the traditional standard AES encryption technique is time-consuming when used in a sequential fashion. This paper presents a GPU-accelerated implementation of a data encryption and decryption algorithm. The algorithm uses the concepts of a modified XOR cipher to encrypt and decrypt the data, with an encryption pad, generated using the shared secret key and some initialization vectors. It uses a genetically optimized pseudo-random generator that outputs a stream of random bytes of the specified length. The proposed algorithm is subjected to several theoretical, experimental, and mathematical analyses, to examine its performance and security against several possible attacks, using the following metrics - histogram analysis, correlation analysis, information entropy analysis, NPCR and UACI. The performance analysis carried out shows an average speedup-ratio of 3.489 for encryption, and 4.055 for decryption operation, between the serial and parallel implementations using GPU. The algorithm aims to provide better performance benchmarks, which can significantly improve the experience in the relevant use-cases, like real-time media applications.

1) INTRODUCTION

Computer users are becoming more security conscious, and one of their top worries is maintaining the privacy of their personal information. Encrypting the data, especially that which is kept on the hard drive, is the most popular and practical method for achieving this. Encryption's fundamental tenet is to change data so that an attacker cannot access it. Symmetric encryption techniques like the Advanced Encryption Standard (AES) perform this transition, particularly when working with huge amounts of data [3]. The secret key, which is required to encrypt and decode the data, is the foundation of these techniques. This part describes `briefly the motivation behind the research work, aim and objectives, contributions of

the research and introduction of the paper. Since ancient times, research on cryptography has dominated the field of information security. After the development in mechanics, researchers have been driven to create solutions for greater performance in terms of speed and security, starting with the ancient Caesar's cipher that required manual encryption of data and progressing to the enigma machine. The era of digital automation began with the introduction of transistors, which gave rise to microprocessors. A new generation of encryption technology was created because of this new development and modern CPUs. Different encryption techniques have been created and used since then. In practically every industry where data privacy is an issue, encryption is used to assist secret communication. This includes scientific institutes, corporate offices, social networking sites, and more particularly, military and government affairs. The military and armed forces are one of the most crucial sectors to adopt encryption technology since they possess a significant amount of sensitive information pertaining to the protection of the nation and its citizens.

2) Overview

The U.S. National Institute of Standards and Technology (NIST) created the Advanced Encryption Standard (AES) in 2001[25] as a specification for the encryption of electronic data. Today, practically every industry uses this standard, including government agencies, the military, educational institutions, ATM cards, computer password security, and online commerce. The exponential growth of confidential data produced by today's state-of-the-art technology requires that it first be encrypted before being stored or transported to a designated location. Because encryption relies so largely on complicated mathematical calculations, it takes time [4]. The importance of fast encryption has long been a topic of research and debate. To provide improved speedups, a variety of implementation strategies and optimization approaches have been used. However, only a partial solution to the question "Which method can produce the best solution?" has been found [19]. The performance of these algorithms is directly impacted by developments in the computer industry. This is a good indication for researchers to concentrate on enhancing processor utility to obtain faster processing rates.

Since many years ago, microprocessors with a single central processing unit (CPU) have increased performance significantly. The processor's speed has been increased via a variety of methods, including increasing the clock speed. Due to problems including high power consumption, heat dissipation, and current leakage, the culture of increasing clock speed has plateaued. Additionally, extreme heat increases the requirement for costly cooling equipment, driving up system costs. The single core microprocessors provided significant functionality and user interfaces for decades. The users themselves yearn for more advancements and faster processing, as the rule of science dictates. Multi-core processors, which use many processing units or processing cores on a single chip, have replaced single-core processors in the computer

industry. The multicore processor concept has had a significant influence on the software industry. A dense integrated circuits transistor count doubles roughly every two years, according to Moore's law [37]. In other words, every new CPU generation doubles the number of processing cores. However, since a sequential program can only operate on one of the processor cores, an increase in the number of cores has no effect on such computers.

3) MOTIVATION

There have been numerous studies done in the past to examine how well CPU and GPU perform while implementing different encryption methods, including AES. Where a few studies assert that GPUs perform better than CPUs, the implementation method employed is either flawed or insufficient. Additionally, the performance advantage is very slight or almost nonexistent, making it impossible to guarantee that GPUs will always outperform CPUs. The GPUs used are older models with the Tesla and Fermi architecture, and there hasn't been any research on the newest GPUs with the Kepler architecture.

This research seeks to evaluate various AES algorithms (AES-128, AES-192, and AES-256) at various levels of optimization on modern CPU and GPU hardware to determine the optimum implementation method for achieving the quickest execution. It focuses on comprehending how efficiently numerous threads in a GPU may be used to produce the highest algorithmic performance. It measures the performance of a parallel CPU implementation utilizing Pthreads in comparison to a single threaded program. To comprehend its effects on the performance of the algorithm, it also attempts to differentiate the performance of the optimized version of GPU programming using CUDA STREAMS. Finally, this research suggests methods that future researchers might consider when they develop an algorithm on the GPU. This study presents useful concepts and methods that have not before been effectively described, and as an active addition to the field of parallel computing, it makes a significant contribution to the field.

3.1 Research Questions

RQ1: a) In the implementation of the AES algorithm, does the GPU outperform the single-core CPU?

b) Does the GPU perform better than a multi-core CPU while running the AES algorithm?

RQ2: Does the usage of CUDA STREAMS improve the effectiveness of the AES algorithm?

RQ3: Which implementation offers the fastest AES execution?

3.2 Contribution to Armed Forces

The military and defense industries use GPUs for a variety of tasks, including image and video processing, cockpit and commander displays, video tracking, digital mapping, radar processing, and data protection tasks like encryption and compression. The quick execution of these apps is directly supported by finding a better approach to utilize GPU parallelism. Due to its well-known security and reliability, the AES encryption algorithm is frequently employed by the military and armed forces. The time it takes to encrypt the vast amounts of sensitive data that are always growing can be decreased with an increase in AES's execution speed.

4) Approach

The design of the AES algorithm is briefly described in the first part. In the second section, the idea of parallel computing is introduced. Then, the GPU is described using the CUDA programming model. The use of CUDA STREAMS is then shown. When encrypting and decrypting files, I tried to utilize both the CPU and GPU for the greatest performance. Choosing which one to use when the difficult part was. Balancing latency and throughput proved difficult. Sending data to the GPU for modest read/write operations takes a lengthy time since copying to the GPU happens across a sluggish bus. However, the parallelized AES implementation performed noticeably better if the read/writes were of a greater size. Considering this, I considered which systems would profit most from this methodology. I found that a GPU-based version could be especially useful for PCs with low-end processors.

4.1 Advanced Encryption Standard

A symmetric block cipher is the Advanced Encryption Standard (AES). A symmetric block cipher encrypts and decrypts data using the same secret key, or cipher-key. The AES creates a 128-bit output sequence after receiving a 128-bit input sequence. This series is also known as a block. The length of a block is determined by the number of bits inside. The AES method has three different cipher-key length options: 128 bits, 192 bits, and 256 bits. The AES version is referred known as AES-128, AES-192, or AES-256, depending on the cipher-key size chosen. The cipher-key size affects a lot of variables. The number of transformations rounds that would be utilized for the encryption or decryption of the data depends on the cipher-key size used for an AES cipher. The number of rounds or cycles of repetition in encryption and decryption is 10 when the cipher-key size is 128 bits, compared to 12 rounds and 14 rounds for 192 bits and 256 bits, respectively. This section provides a detailed explanation of each of the AES computation rounds' numerous operations. The fixed input block size for this technique is 128 bits, while the key sizes are 128, 192, and 256 bits. As seen in figure below, the input's array of bytes A0 through A15 is copied into the state array.

A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15

Since AES is a byte-parallel algorithm by nature, the following strategies are used to parallelize it.

- As previously noted, AES is a byte-parallel algorithm, thus we allocate n threads to each byte statically.
- Therefore, we will have 16 blocks with 16 threads, each working on a byte, for a 16*16-byte example of data.
- Each thread will perform all required operations on a byte, including various iterations of AddRoundKey, ShiftRows, and MixSubColumns.
- Cache integer stage in shared memory
- Constant memory for S tables.
- Since it is a one-time computation, the CPU will perform the Expanded Key computation from a 128-bit key for each round of AES.

AES operates on an input data of 128-bits i.e., 16-bytes. These bytes can be represented as finite field elements in the form of polynomial representation [25]:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

For example: A set of data {10011100} will be represented as

$$x^7 + x^4 + x^3 + x^2$$

4.2 Input/Output

The input data (plain-text) and the cipher-key are the two fundamental components on which the AES algorithm functions. The input matrix, also known as the input data, is a 4 by 4 column-major order matrix that contains the 16-byte input data. It is designated as "in matrix" as shown below.

$$\begin{bmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{bmatrix}$$

The columns of the matrix are arranged so that they are stored one after the other. The first column of the 4 by 4-byte matrix is occupied by the first four bytes of the 128-bit input block. The second column is home to the following four bytes, and so forth. Any matrix that follows is organized using the same technique. This matrix is subsequently replicated into the input state, a second two-dimensional array of bytes that serves as the AES algorithm's input. The input state, state array, or simply state refers to this two-dimensional array of bytes, which is designated as "in state." The in state, which is also known as state, passes through different transformation rounds and produces following outputs after each operation. A 16-byte, 4 4 column-major matrix called the output matrix marked as "out matrix" receives the final output of the method, also known as the output state denoted as "out state," when all rounds have been completed. The necessary cipher-text is contained in the data in the output matrix. Below are some graphs illustrating GPUFS design.

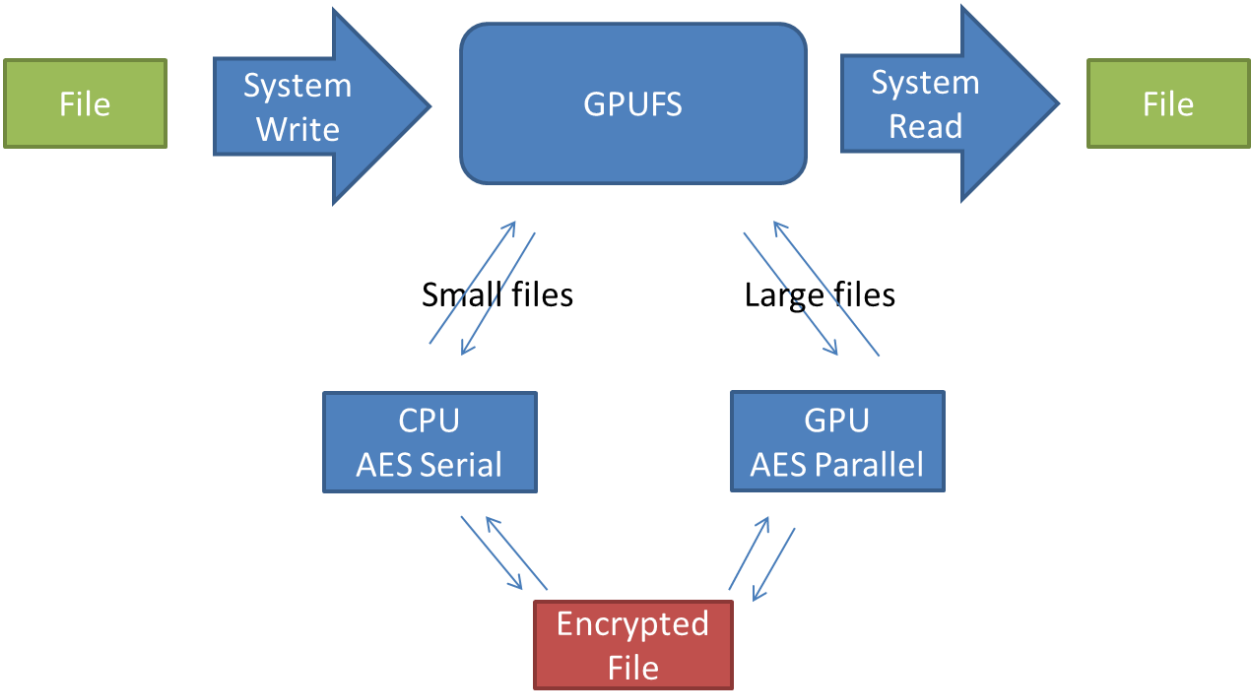
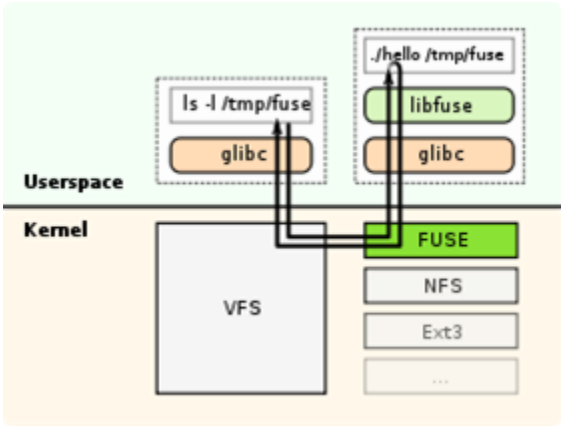


Figure 1.0



Fuse Layer

4.3 Encryption Process

The following steps are used in an encryption process to transform plain text into a secure cipher-text.

1. Initial round
2. Rounds
    i. Substitute byes or Sub Bytes
    ii. Shift Rows
    iii. Mix Columns
    iv. Add Round Key
3. Final round (No Mix Columns)
    i. Sub Bytes
    ii. Shift Rows
    iii. Add Round Key

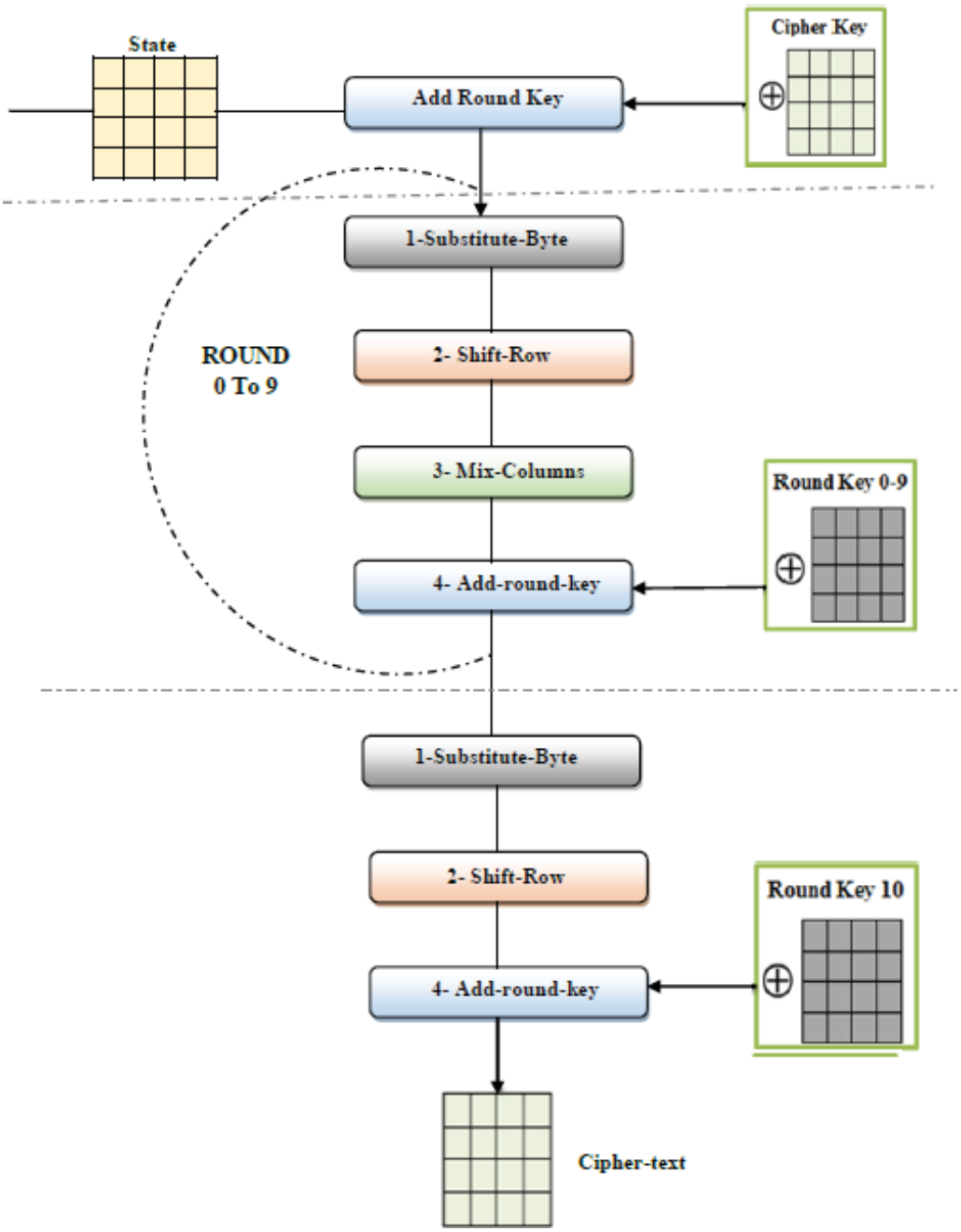


Figure 1.1

The encryption procedure is well illustrated in Figure 1.1 above. The figure shows that the encryption process is composed of 10 rounds, the first 9 of which are made up of four separate transformations. One of the four transformations, the mixing of columns, is not used in the final round. (As was previously established, there are 10, 12, and 14 rounds for 128-bit, 192-bit, and 256-bit long key sizes, respectively.)

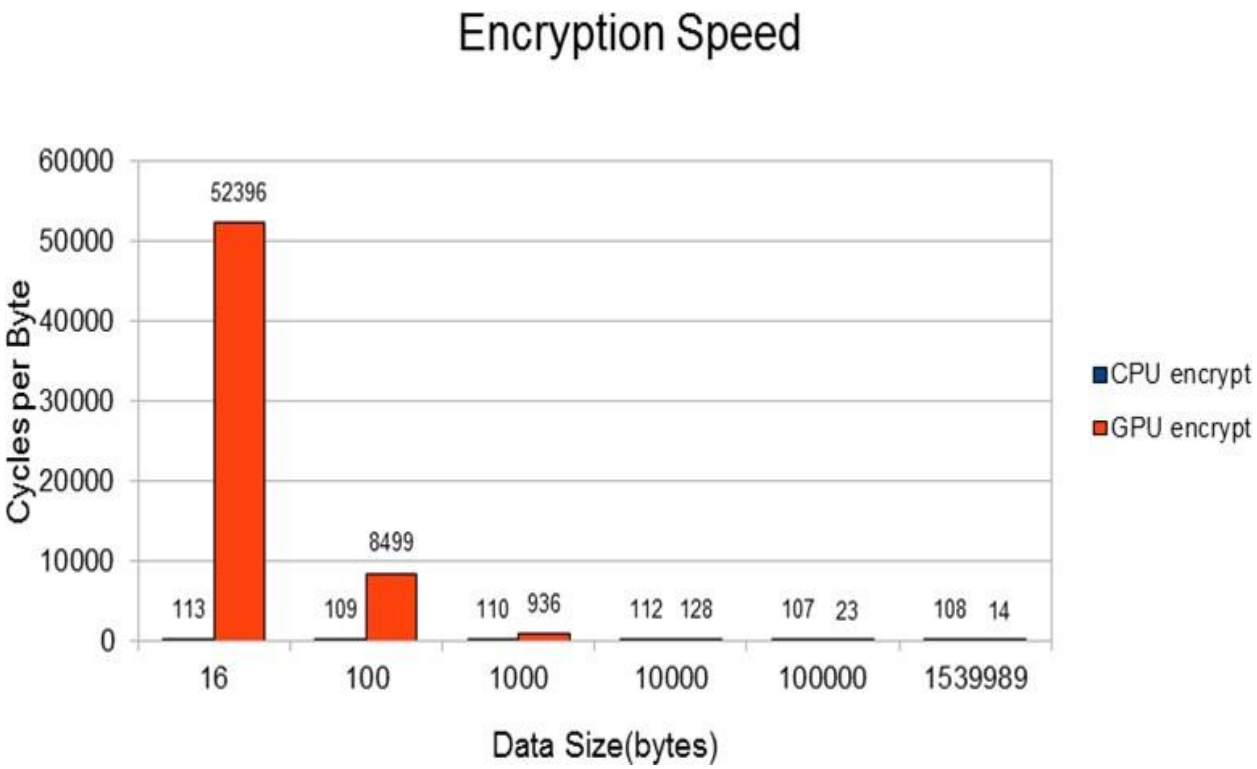


Figure 2.0

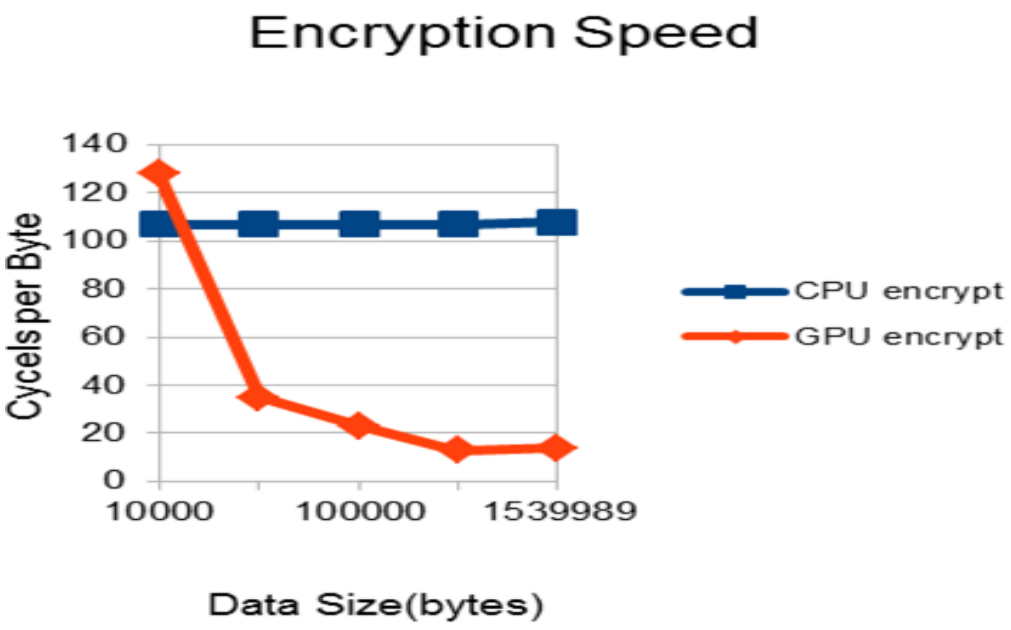


Figure 2.1

The two graphs up above show that there is initially a significant overhead when using GPU when the data size is extremely tiny. This is because data is sent to and from the GPU across a

bus that is relatively sluggish. But as the data size grows, a very clear pattern emerges, and by 16KB, we can clearly see that the GPU implementation is faster than the CPU version for encryption and decryption. The GPU implementation is around 6 times quicker than the CPU counterpart at this point, where both implementations essentially plateau at little over 1MB.

4.4 Decryption Process

The key that is received from the sender determines how AES is decrypted. For data encryption and decryption, the sender and the receiver share the same key. Although it is done in the opposite direction, this procedure is comparable to encryption. An initial round, nine iterations of a "inverse normal round," and a "AddRoundKey" are the steps in the decryption process. The procedures "AddRoundKey," "InvMixColumns," "InvShiftRows," and "InvSubBytes" make up a "inverse normal round," respectively. With the exception of the "InvMixColumns," the final round has the identical procedures as a "inverse normal round."

The figure below demonstrates the process for decryption

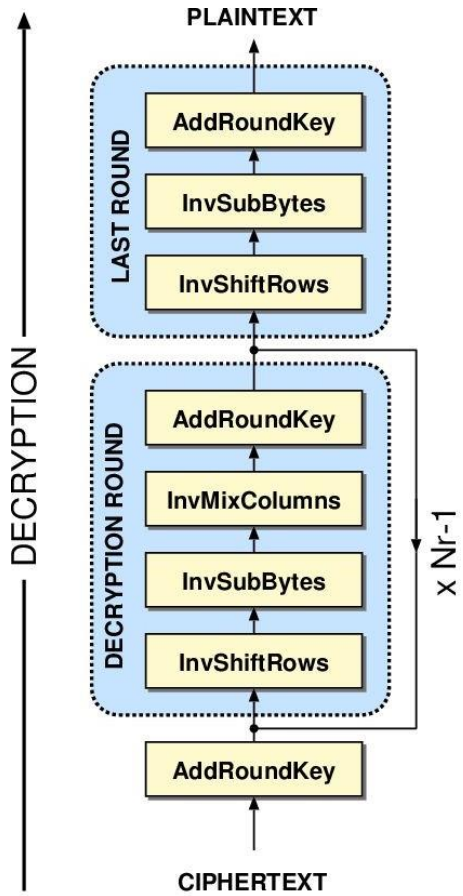


Figure 3.0

Using a process map, this picture also graphically explains the decryption procedure.

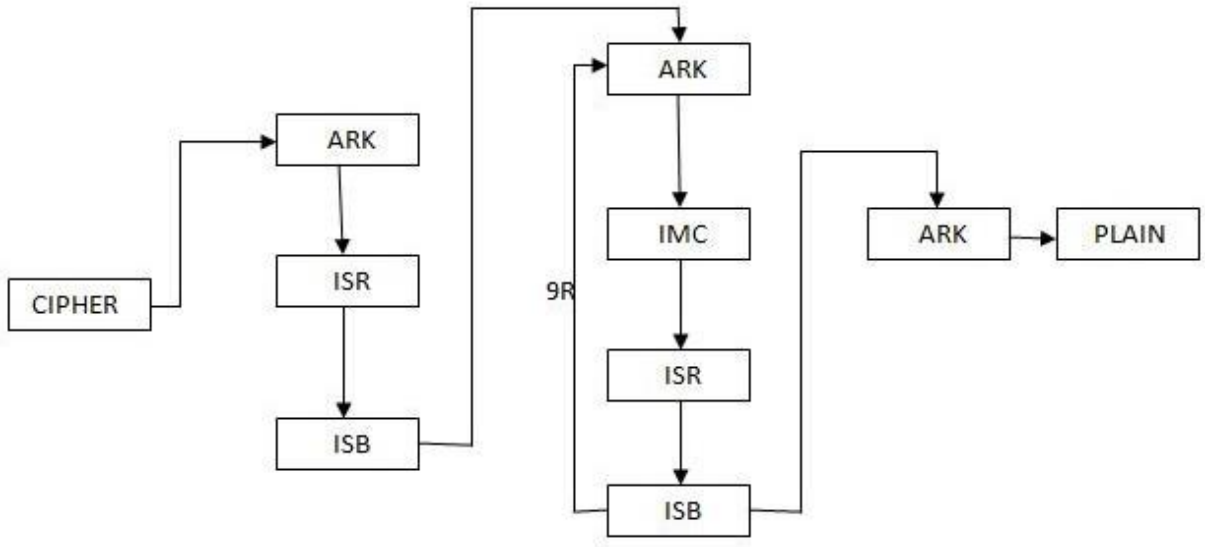


Figure 3.1

4.5) AES Applications

Numerous applications exist for AES encryption and decryption. It is employed when the information is too sensitive for anybody but the authorized parties to have access to. The many applications are as follows:

- Discreet Communications
- Intelligent Cards
    - RFID.
    - Networks for ATMs.
    - Image security
- Discreet Storage
    - Documents for Confidential Cooperation
    - Governmental Records
    - FBI Records
    - Personal Electronic Storage
    - Protection of Person Information
    - Wi-Fi (can be used as part of WPA2)
    - Mobile apps (such as WhatsApp and LastPass)

5) PARALLEL Computing

5.1) Why Parallel Computing?

One can tackle difficulties involving complex computations more quickly with a faster computer. It can offer higher response in the case of interactive apps. Obtaining better answers in the same amount of time is the second crucial action one may take. It enables the process to become more sophisticated and aids in improving model resolution. The goal of parallel computing is to accelerate the completion of a given task using several strategies, one of which is the simultaneous execution of all of the subtasks that make up the original task. Since years, microprocessors built around a single central processing unit (CPU) have accelerated performance improvement. To achieve speedier performance, numerous hardware and software advancements have been done. Clock performance enhancements, hardware optimization techniques including instruction prefetching, reordering, pipelined functional units, branch predictions, and hyperthreading, among other things, are included in these advances. Herb Shutter is credited with the famous phrase "the free lunch is ended."

This indicates that the clock speed will no longer increase exponentially as it did in the past, when Moore's Law predicted that it would double every 18 to 2 years. A new strategy for enhancing speed was required because the computer industry could no longer rely solely on

advancements in hardware and software. One such strategy was to switch to multi-core CPUs, which led to the development of parallel computing.

5.2) Architectural Difference in GPU and CPU

CPU and GPU have fundamentally different design philosophies as illustrated in figure 3.0.

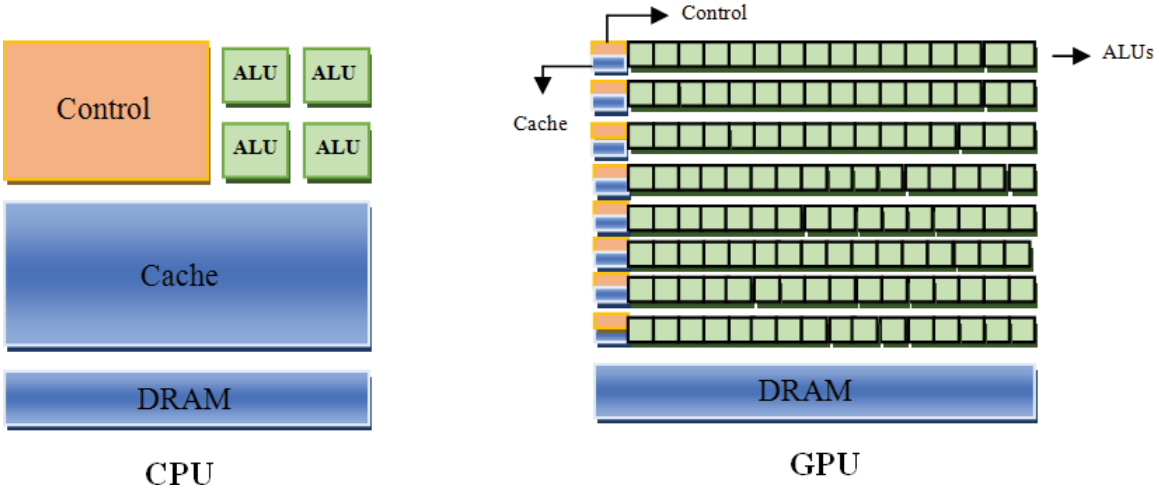


Figure 4.0: CPU and GPU architectural difference.

The CPU is configured for the quickest switching between operations with the least amount of latency. To push as many processes through the device as possible, GPUs are tuned for throughput. There is a lot of infrastructure on the chip, such as big caches, to ensure that one has access to a huge amount of data quickly to achieve low latency on the CPU. A lot of control flow silicon is present. There aren't many components specifically for computing. However, the balance has changed in the GPU, where many arithmetic and logic units are required. Because it is tailored for computation-intensive, highly parallel tasks, more transistors are allocated to processing than to memory and control. If there is enough work in the application, the length of time a GPU takes to receive data from the DRAM is not a huge problem, therefore the L2 cache can get smaller. To disguise the latency that is introduced while retrieving data from the global memory or when processing a complex operation, there must be enough data. So, in order for a GPU to endure latencies, a huge number of threads must be used.

The traditional GPU-based OpenGL platform is not appropriate for AES computation because it is limited to floating point data and lacks bitwise logical operations, in contrast to AES, which uses bitwise logical operations during each round of transformation and involves significant mathematical computation [9]. AES encryption can be put into use on a solid platform like OpenCL.

On the other hand, CUDA is tailored for NVIDIA GPUs. The most recent NVIDIA GPU, which supports the CUDA platform, is selected in this study since it is adaptable and effective for AES encryption.

6) GPU Computing with Compute Unified Device Architecture (CUDA)

A new set of instructions and a new parallel programming model are features of CUDA, a new parallel computer architecture announced by NVIDIA in 2006. C can be used as a programming language for the GPU thanks to a new software environment provided by CUDA. When utilizing CUDA, the GPU is viewed as a computing device that can run many threads in parallel. The GPU serves as the CPU's coprocessor. Supporting heterogeneous computations—in which sequential portions of an application are executed on the CPU and parallel portions on the GPU—is one of CUDA's primary design objectives. GPUs are employed in calculations in the computer game business in addition to rendering graphics (physical effects like debris, smoke, fire, fluids, etc.) Both a lower-level API and a higher level API are offered by CUDA.

6.1 CUDA Architecture

NVIDIA created the parallel computing framework and programming model known as CUDA. By utilizing the graphics processing unit's power, it offers significantly higher computational performance (GPU). NVIDIA GPUs can be programmed using the compiler and tools CUDA. The

C programming language is enhanced with CUDA API. It is a scalable paradigm that uses hundreds of threads to run.

- Support for C, C++, and OpenCL.
- Compatibility with Windows, Linux, and OS X.

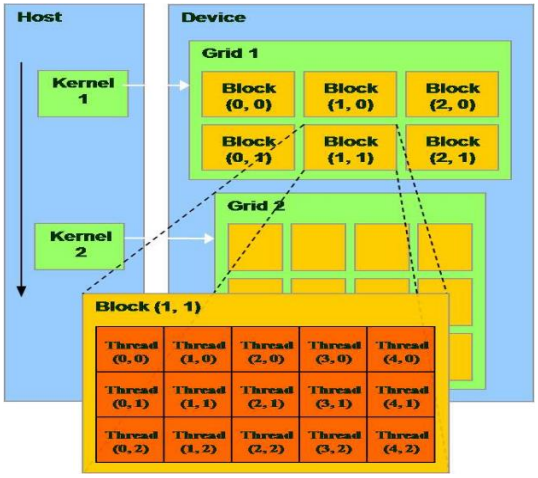


Figure 5.0: CUDA Architecture

6.2 Advantages Of CUDA

- Flexibility for the supplier.
- Makes latency invisible and aids in maximizing GPU efficiency is transparent to programmers.
- Deadlocks are avoided thanks to automatic thread management and the provision of a small amount of synchronization between threads.

6.3 CUDA Program Structure

A CUDA program is divided into phases that can be carried out on either the host (CPU) or the target (device) (GPU).

The phases that do not exhibit parallelism are implemented in the host code, and the phases that do exhibit data parallelism are implemented in the device code, according to the NVIDIA C compiler, NVCC. The device code is created in C augmented with keywords for identifying kernels, which are data-parallel functions. Host (CPU) execution gets the process started. A lot of threads are created on the device when a kernel function is called, moving the execution there. A grid of threads is created. The corresponding grid finishes after all of a kernel's threads have finished running; however, host execution continues until a different kernel is launched.

7) Limitations Of AES

While studying I have encountered several restrictions that prohibited us from attaining the ideal theoretical speedup. One significant one was that we couldn't complete all the processes on the VFS layer. This requires us to make nine different copies of the file, for instance, for every write, which adds a lot of overhead:

- User space to VFS layer
- VFS Layer (Kernel space) back to FUSE (User space)
- FUSE layer to a buffer (our implementation bad)
- CPU to GPU global memory
- Global memory to shared memory (bottleneck)
- shared memory to Global memory (bottleneck)
- Global memory to CPU
- GPUFS to Kernel space
- Kernel space to user space

As was already said, one drawback of larger speedup is that we were unable to employ the hardware that, in our opinion, would gain the most from a CPU+GPU hybrid solution rather than just a CPU implementation (slow CPU computers, such as netbooks).

Several problems could have an impact on our analysis. One is that, even though we are using a widely used AES implementation, there is a chance that the serial implementation of AES could be enhanced, which would have an impact on the GPUFS threshold and speed.

8) Improvements

Encryption is one answer to the growing problem of cyber security and is crucial to the protection of data. Both symmetric and asymmetric encryption techniques are frequently employed in information security. One of the most popular and safe symmetric encryptions available today is called Advanced Encryption Standard (AES). AES encryption, however, has a poor processing speed. According to the experiment, combining two techniques takes 3.045 milliseconds for 1024 bytes of data and increases to 3–4 milliseconds for 2048 bytes, and so forth. The AES algorithm is improved using Shift Row and S.Box modification for Mix Column transformation within that paper[10]. Performance is tested in most studies by running parallel and sequential versions of the same code. When processing massive amounts of data utilizing a CPU and GPU combination, a significant performance difference might be noticed. To facilitate parallelization, parallel structures like parfor and gpuarray are employed. Vector addition is used in the experiment sequentially and concurrently. The execution time for vector addition when done consecutively is 8.96s. However, by running

the identical program in parallel, the execution time is drastically decreased to 0.00018 seconds.

9) Future Work

Due to numerous limitations, one of which is time, this thesis work is constrained. The following are some potential directions for further research to expand the findings about the utilization of parallelism utilizing GPU.

- To see if the findings hold true for other implementations, this research can be done on other cryptographic ciphers including blowfish, serpent, and Salsa20.
- Utilizing task parallelism to speed up the AES algorithm would be beneficial.
- To compare and determine absolute outcomes, similar algorithm models can be applied on various GPUs.
- It would be fascinating to explore how the AES algorithm's performance could be accelerated by using a heterogeneous system, or the combination of CPU and GPU.

9.1 Major Challenges in real-world Adoption:

To normally establish a threshold on when to route data to the GPU based on the CPU speed, GPU speed, and the available GPU, a hybrid technique is used when deciding whether to use CPU or GPU for AES. As a result, the GPU will only be used if the amount of data being read or written is more than the threshold. When using programs like cp, vi, etc. to copy or read a huge file in its entirety, the program attempts to read the disk 4096-byte blocks at a time.

Even though it is designed to read/write more than the threshold, all AES data will be transmitted to the CPU if the threshold occurs to be above 4096 bytes (in the case of fast CPUs). Applications that deal with images, such photo viewers, frequently read more than one block of data at once, crossing the barrier and assisting us in testing our system.

This problem can be overcome by: -
1) Buffering data in memory with optimism up until the threshold is crossed and then sending it to the GPU.

Determine the time by which we will determine if the threshold has been crossed because this parameter influences the file system's latency.

2) Faster data copying to the GPU as it is read or written to hide memory transfer latency Although CUDA has this feature, we have not yet used it.

3) Continually encrypting files: - We must put the streams for every file into practice to allow File I/O in tandem with GPU encryption. CUDA's Streams feature allows for mutual exclusion

amongst CUDA kernels that have been run for various Files. We still haven't completed this task.

9.2 Answers to Research Questions

RQ1.

a) **In the implementation of the AES algorithm, does the GPU outperform the single-core CPU?**

Answer:

In the implementation of the AES algorithm, the GPU performs better than a single core CPU. It speeds up the execution of AES 128, AES 192, and AES 256 by factors of 22, 21, and 12 for a data size of 32000 bytes, respectively. It speeds up the execution of AES 128, AES 192, and AES 256 by a factor of 13, 12 and 12 times, respectively, with a data size of 32000*5 bytes in which data is separated into chunks of 32000 bytes each to be processed by the GPU in sequence.

b) **Does the GPU perform better than a multi-core CPU while running the AES algorithm?**

Answer:

In the implementation of the AES algorithm, the GPU performs better than the multi-core CPU. For the implementation of AES 128, AES 192, and AES 256, the GPU performs 6, 5, and 3 times quicker than multi core CPU for a data size of 32000 bytes, correspondingly. For the implementation of AES 128, AES 192, and AES 256, the GPU is 3, 3, and 2.5 times faster than the CPU for data of size 32000*5 bytes, where the data is consecutively processed in chunks of 32000 bytes each.

RQ2: Does the usage of CUDA STREAMS improve the effectiveness of the AES algorithm?

Answer:

For a data size of 32000 bytes, the usage of CUDA STREAMS has no beneficial effects on the performance of the AES algorithm. However, it exhibits improved performance for data quantities greater than 32000*5 bytes. Consequently, the CUDA STREAMS can benefit the AES method for larger data sizes with significant data in each stream.

RQ3: Which implementation offers the fastest AES execution?

Answer:

The CUDA program on the GPU with granularity 2 performs the fastest for AES 128, AES 192, and AES 256. This program processes 32 bytes of input data per thread, which is internally executed in chunks of 16 bytes, with a balanced thread/block distribution. According to Granularity 2, each thread processes 32 bytes of input data, which are internally broken up into 16-byte chunks. The task is divided between the threads and blocks in a way that the GPU effectively uses parallelism by using an efficient number of blocks with an efficient number of threads in each block.

10) Conclusion

The use of GPUs to improve encryption and decryption is suggested in this research. This paper outlines the fastest methods for text encryption and decryption using the AES algorithm on programmable graphics processing units, with up to a considerable deal of speed on a similar CPU. If it runs in a graphics processing environment, the amount of time needed for encryption and decryption is drastically decreased when the amount of data is huge.

More apps are starting to employ Advanced Encryption Standard (AES) instead of Data Encryption Standard (DES) to safeguard their data and security because of recent advancements in cryptanalysis. The AES algorithm is currently implemented, although it has a low throughput and consumes a lot of CPU resources. The parallel GPU (Graphical Processing Unit) computer technology and its cryptography-optimized design were researched by the authors. They then created and constructed a quick data encryption system based on GPU using CUDA features for AES parallel encryption. The testing showed that their strategy considerably increased the speed of AES encryption.

References

[1] Performance Analysis of GPGPU and CPU On AES Encryption https://www.diva-portal.org/smash/get/diva2:831353/FULLTEXT01.pdf - September 2014

[2] GipherFS: a GPU-accelerated ciphered file system

[3] Optimization of Advanced Encryption Standard on Graphics Processing Units - https://eprint.iacr.org/2021/646.pdf

[4] ACCELERATING ENCRYPTION/DECRYPTION USING GPU'S FOR AES ALGORITHM - https://www.ijser.org/researchpaper/ACCELERATING-ENCRYPTION-DECRYPTION-USING-GPUS-FOR-AES-ALGORITHM.pdf

[5] GPUfs: Integrating a File System with GPUs - https://www.cs.utexas.edu/users/witchel/pubs/silberstein13asplos-gpufs.pdf

[6] Implementation of Advanced Encryption Standard Algorithm M.Pitchaiah, Philemon Daniel, Praveen – https://www.ijser.org/researchpaper/Implementation-of-Advanced-Encryption-Standard-Algorithm.pdf

[7] Report on the development of the Advanced Encryption Standard(AES) - https://www.ijser.org/researchpaper/Implementation-of-Advanced-Encryption-Standard-Algorithm.pdf

[8] Advanced Encryption Standard(AES) Algorithm to Encrypt and Decrypt Data - (PDF) Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data (researchgate.net)

[9] NVIDIA CUDA Software and GPU Parallel Computing Architecture - https://www.researchgate.net/publication/221032946_NVIDIA_CUDA_software_and_GPU_parallel_computing_architecture

[10] Improvement Of Advanced Encryption Standard Encryption With Shift Row and S. Box Modification Mapping in Mix Column - https://www.sciencedirect.com/science/article/pii/S1877050917321294?ref=pdf_download&fr=RR-2&rr=7762c74b6e320d1a

[11] Speculative Encryption on GPU Applied to Cryptographic File Systems - https://www.usenix.org/system/files/fast19-eduardo.pdf